Inner Ensembles: Using Ensemble Methods in Learning Step

by

Houman Abbasian

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the degree of Doctor of Philosophy in Computer Science

Ottawa-Carleton Institute for Computer Science
School of Electrical Engineering and Computer Science
University of Ottawa

Abstract

A pivotal moment in machine learning research was the creation of an important new research area, known as Ensemble Learning. In this work, we argue that ensembles are a very general concept, and though they have been widely used, they can be applied in more situations than they have been to date. Rather than using them only to combine the output of an algorithm, we can apply them to decisions made inside the algorithm itself, during the learning step. We call this approach *Inner Ensembles*. The motivation to develop Inner Ensembles was the opportunity to produce models with the similar advantages as regular ensembles, accuracy and stability for example, plus additional advantages such as comprehensibility, simplicity, rapid classification and small memory footprint. The main contribution of this work is to demonstrate how broadly this idea can be applied, and highlight its potential impact on all types of algorithms. To support our claim, we first provide a general guideline for applying Inner Ensembles to different algorithms. Then, using this framework, we apply them to two categories of learning methods: supervised and un-supervised. For the former we chose Bayesian network, and for the latter K-Means clustering. Our results show that 1) the overall performance of Inner Ensembles is significantly better than the original methods, and 2) Inner Ensembles provide similar performance improvements as regular ensembles.

Acknowledgements

First of all, I would like to thank the omnipresent God, who answers my prayers; the Lord who give me the strength to plod through the life harshness; thank you so much Dear God. It would not have been possible to write this doctoral dissertation without the support, help and guidance of the kind people around me who in one way or another contributed and extended their valuable assistance; the people to only some of whom it is possible to give particular mention here. I would like to thank my parents for their support and inspiration during these many years of grad school as an International student; My father **Bahman Abbasian** who has been my inspiration and supportive as I hurdle all the obstacles since my childhood and especially for his moral support during my PhD program; My lovely mother **Maryam** who always motivated me during my PhD program. I would like to express my sincere gratitude to my three supervisors Professor, **Stan Matwin**, **Nathalie Japkowicz** and **Christopher Drummond** for their valuable guidance, consistent encouragement and most importantly being supportive throughout this long journey. **Prof. Stan Matwin**, a great man who is like father to his students, whom there is no need to mention about his unsurpassed knowledge and insight. He always granted me unlimited support, advice and friendship, not only by words also by his great heart, for which I am extremely grateful. I would like to thank **Prof. Nathalie Japkowicz** for her kindness, support and shared valuable insights in the relevance of the completion of this dissertation; Many thanks to **Prof. Christopher Drummond** from whom I have learnt many things and for his help during my PhD. He has been always very helpful and motivated during my study. I would also like to express my special thank to **Dr. Jelber Sayyad Shirabad** for his motivations and discussions. I express my thank to my brother **Reza** for his care and his try to be helpful. Finally I would like to acknowledge the financial, academic and technical supports of the University of Ottawa especially at the school of information technology that provided mean appropriate condition during my PhD program.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In today's complex world, we encounter many different types of decision making. The decisions can be simple or complicated, general or specific, mundane or critical. We usually make non-critical decisions alone, and spontaneously consult with others about the critical ones, to decrease the potential for mistakes. One example is elections, in which a decision is made according to the majority of opinions. In this type of decision making each person has one vote, and the final decision is made by choosing the collective opinion of the majority of the participants. This type of decision making is often called decision making by committee, or the *wisdom of the crowd*. The main motivation of the *wisdom of the crowd* is that decisions made by groups of people are more robust than those made by individuals. For example, suppose people are asked to guess the weight of an ox. To Francis Galton's surprise, the median of the results was closer to the actual weight of the ox than most of the individual guesses (Galton, 1907). A more modern example is guessing the number of jelly beans in a jar. If enough people are asked, the average guess will be close to the actual number (Surowiecki, 2004). As we increase the size of the ensemble the average guess is closer to the actual number, thus increasing the ensemble size raises the accuracy. However, at some point increasing the size will not affect the accuracy, since it will decrease the diversity. This point is different for every problem, so we cannot determine a single ensemble size that is best for all problems.

An important sub-field of machine learning, ensemble methods, has effectively exploited this idea, and produced substantial performance gains. Ensemble methods generally work by combining the output of learned models, using different techniques such as majority vote, averaging, maximum and weighted majority vote (Kuncheva, 2004). However, we argue that this idea has been applied in a limited way, and there is con-

siderable potential to extend it. Here, instead of combining the output of the models, we apply ensemble methods to the decisions used to generate the models. We call this idea *Inner Ensembles* as we are exploiting the *wisdom of the crowd* during the learning step. Similar to more traditional ensemble methods, Inner Ensembles define a broad framework that has the potential to impact all types of algorithms. To clarify the use of ensembles during the learning step, consider a company deciding if it should collaborate with a foreign one. For this collaboration a group of representatives must travel abroad. To send a group of people, the first company will have an internal meeting to discuss the decisions they will make when they meet with the foreign company. This can be considered a learning step in which each individual learns. Then this group travels abroad for the meeting, and to make a decision about business strategy in that meeting, they vote. This step can be considered as the deployment step, and is analogous to a traditional ensemble which uses voting in the testing step. However, it would be difficult and expensive for a group of people to travel multiple times. So, the company decides to select one person to travel and represent the company. To do this, the company holds another internal meeting to determine whose opinion best represents the full group, and this is who they send abroad to make decisions on behalf of the company. In this case, an ensemble is used to select a person in the company's internal meeting (training), rather than during the meeting with the foreign company (test).

Using ensembles during the learning step results in creating the better model. For example, consider the different sections within a company, such as advertising, sales, finance and human recourses. These sections are not independent, as they must work together in a coordinated manner to achieve the company's objectives (which can be considered a model). Each section has its own manager, and assigning a manager to each section can be considered a learning step. Now, if a committee decides about assigning the manager for each section of the company, instead of one person, we expect that the assigned managers of each section will be more suitable for that section. This is another example of Inner Ensembles.

Although two methods (Geurts and Wehenkel, 2000; Prez et al., 2004) that apply to decision trees fit within the framework, it is actually far more general, with broader benefits than previously thought. Here we argue that, like more traditional ensemble methods, Inner Ensembles define a broad framework that has the potential to impact all kinds of algorithms, not just decision trees. More specifically, Inner Ensembles work on *Decision Makers* inside the algorithm, by applying ensemble methods to make more robust decisions at multiple stages throughout the learning algorithm. As the structure

of every algorithm is different, it is difficult to detail the steps to apply Inner Ensembles in every case. However, with the knowledge of how each algorithm works, we can follow some general guidelines. The core idea of Inner Ensembles is shown in algorithm 1. According to this algorithm, the first step is to find a decision maker inside the algorithm (line 1). This decision maker makes choices based on a measure (line 2). The measures guide the search through the decision space, and because different decisions result in different hypotheses, the output of the measures indirectly guides the search through the hypothesis space to a single hypothesis. If the output of the measure is more robust, we expect to find a better hypothesis. Therefore, we have to work on the input and output of these measures (lines 3,4). By manipulating the input, by sampling for example, we can produce different outputs (lines 5,6). Then we combine these outputs and apply the results to the decision being made at that point inside the algorithm (line 7).

In Chapters 3 and 4, we will discuss two different algorithms for supervised and unsupervised learning, to illustrate how they follow the general guidelines. The results show that Inner Ensembles perform statistically better than the original algorithm, and comparably to ensemble methods. Inner Ensembles have different advantages over regular ensembles, such as faster classification time, memory efficiency (because it only needs to store one model, while ensembles needs to store all of the modes), and that the final model is interpretable, which it is not for the regular ensembles. In the following chapters, we will discuss in detail how the general guidelines can be applied to other popular algorithms, thus demonstrating the generality of this framework. One way to explain the success of ensemble methods in general is to consider bias and variance decomposition of the error. Overall, it is accepted that Bagging reduces the variance of the error but does not change the bias, and boosting affects bias at the early iterations and affects variance at the later iterations (Kuncheva, 2004). We expect that since Inner Ensembles are ensemble methods, albeit which work inside the models, a similar effect is at work. Another way to show how ensemble methods converge to a correct decision is to use the well-known Condorcet's Jury Theorem (Rokach, 2010). To illustrate this theory, assume that a group of voters want to make decisions on a binary problem. If p is the probability that each voter is correct and M is the probability that the majority of voters are correct, if $p > 0.5$ then the probability that the majority of voters are correct is larger than p, and if $M > p$, the probability approaches 1 as the number of voters approaches infinity. This theorem can be applied for Inner Ensembles as well. According to algorithm 1 a decision maker works based on a measure. We can consider the measure with different inputs as different voters (line 5 and 6). Then the probability that each voter is correct

Algorithm 1 General guidelines for using Inner Ensembles

1: **Locate a decision maker inside the algorithm.**
2: **Find a measure based on which that decision maker works.**
3: **Indicate the input and output of the measure.**
4: **Apply the ensemble on the measure:**
5: Change the input of the measure in different ways. {It can be sampling of data or feature based modifications or other methods.}
6: Generate an output based on each input.
7: Combine the outputs
8: **Apply the output of the ensemble for decision making.**

is analogous to the probability that the measure with specific input is correct, and the probability that the majority of voters are correct is equivalent to the probability that combination of different measures with different inputs is correct (line 7). Increasing the ensemble size could have a similar effect on Inner Ensembles for each decision maker. Though increasing the size of the ensemble will raise the accuracy of each decision maker inside the algorithm, at some point this will not impact the accuracy, and might even reduce it by decreasing the diversity. On the other hand, increasing ensemble size could have a different effect on Inner Ensembles for the model itself. Models have different decision makers, and increasing the ensemble size for the entire model might have a positive effect on some decision makers and a negative effect on others, as it could decrease their diversity. Thus, increasing the ensemble size for the entire model will not necessarily raise the accuracy of the model.

1.1 Novelty

The main novelty of this thesis is viewing the application of ensemble methods from a new perspective. Ensemble methods in machine learning are typically used to combine the output of models in different ways. Ensemble methods can be used to combine the outputs of classifiers, the output of clustering algorithms (partitioning of the space), the feature subsets selected by different feature selection algorithms, or even the output of semi-supervised learning. However, this is the view from outside the algorithms; we are focused inside the algorithms, and we use ensemble methods for building decision makers there. We propose general guidelines to apply Inner Ensembles to different algorithms with the assumption that we have some knowledge about how each algorithm works.

The motivation for doing this is presented in the next section.

1.2 Why Use Inner Ensembles?

Using the Inner Ensembles framework, we can retain many benefits of the traditional ensemble methods, while restoring more intuitive models produced by the base model. At a more practical level, Inner Ensembles produce models with a number of clear advantages, which can be divided into two groups. The first group includes the benefits of the traditional ensembles, such as stability, while the second group has the advantages gained by using a simpler model, namely comprehensibility, simplicity, faster classification and a smaller memory footprint. In this section, we discuss the main motivation of using Inner Ensembles as another way of using *wisdom of the crowd*. We explain the shortcomings of the traditional ensemble methods, and how Inner Ensembles do not have these drawbacks. We describe how we can match the benefits of traditional ensemble with Inner Ensembles. We recognize that these advantages over traditional ensembles must often be traded off against absolute performance; that is the cost of using Inner Ensembles. However, our work in this thesis shows that much of the improved performance can be maintained, though we do expect that continued refinement of the approach will lead to further improvement.

1.2.1 Advantages Over Traditional Ensembles

Comprehensibility is an important concept in knowledge discovery and concept acquisition, with the main interest being to derive human-oriented descriptions that are understandable by humans. The importance of comprehensibility was recognized in the early days of machine learning research, and the idea was argued for by Michalski (1983) in his comprehensibility postulate:

"*The results of computer induction should be symbolic descriptions of given entities, semantically and structurally similar to those a human expert might produce observing the same entities. Components of these descriptions should be comprehensible as single 'chunks' of information, directly interpretable in natural language, and should relate quantitative and qualitative concepts in an integrated fashion.*"

Comprehensibility is essential in many real-world applications, including medicine, fraud detection for insurance companies and loan concession in financial environments (Prez et al., 2004). If the hypothesis is comprehensible, then the findings can be reviewed

by humans (Hunter and Klein, 1993). In some domains, such as medical diagnosis, it is important that the user is able to inspect the learned hypothesis, to build trust in the performance of the model (Wolberg et al., 1994). Also, in scientific discovery many important hidden feature relationships can be revealed.

In many domains, explanation of the classification of the individual examples is important. In such domains, if the model is understandable it is possible it can be used to produce explanations of the classifications of the examples (Gallant, 1993). One of the important issues that affects the generalization power of the learning algorithm is feature representation. If a model is comprehensible, it may be helpful to suggest better feature representation (Craven and Shavlik, 1995). According to Craven and Shavlik (1995), approximate domain theory is defined as an incomplete description of solving a problem. Learning models, if they are comprehensible, may help to refine approximately-correct domains.

There are several applications that used machine learning techniques and were effective because the models were comprehensible. An example of this is the work of Drummond et al. (2006), who focused on the linguistic ability of people living in Canada before the late twentieth century. The authors had two goals: to infer theories from the data, and to test the existing theories to determine if they are confirmed or contradicted by the data. Machine learning techniques were used to address these goals. Although logistic regression is usually applied for this type of research, decision trees were used because their comprehensibility allows them to extract rules and help identify interesting population subgroups with different linguistic ability than the dominant group. In this application, finding relationships among the population in history is what matters, not accuracy. For this reason, in this application comprehensibility is the main reason to select the decision tree.

Another example is the work by Alexopoulos et al. (1999), in which the author presents the application of inductive machine learning in the medical diagnosis of stroke. It is based on the extended version of C4.5 to extract IF/THEN rules from a decision tree. The extracted rules are tested for perfect classification rate and very high generalization power, and examined by the expert for comprehensibility, complexity and usefulness. This example requires both comprehensibility and accuracy. The expert cares about the comprehensibility, the meaningfulness of the rules extracted from decision tree, as much as the accuracy.

Another application is troubleshooting, which is modeled by Bayesian network. Troubleshooting a man-made device can be very complex, but it can be modeled more easily

with Bayesian network based on the relationship among three variables: 'Faults' that show the different issues of the system, 'Questions' that need to be asked to help find the relevant fault, and 'Actions' that must be performed in order to fix the fault (Vomlel, 2002). Actions and Faults are related to each other, as are Questions and Faults, while Questions and Actions are independent. For each action and fault, a probability must be calculated by an expert to show if the action actually fixes the corresponding fault. Similar calculations are made with the questions and faults, to show if a question helps find a corresponding fault. With this system, the troubleshooting should be minimized and the results could be either fixing the problem or giving up. Because the modeled system itself is what is most important, the comprehensibility and understanding the relationships between the actions, questions and faults is paramount. Indeed, ensemble methods cannot be used here because the system would be eliminated. The goal is modeling, not accuracy, so the only important factor in this example is the comprehensibility of the model.

The effectiveness of traditional ensemble methods can be enhanced by increasing the number and diversity of the models. Although traditional ensemble methods significantly improve the accuracy, they also have drawbacks, an important one being the loss of comprehensibility of the models, which makes them unusable if the main goal is knowledge discovery (Ferri et al., 2002). With Inner Ensembles, we use ensemble methods to generate models that ultimately result in one single model. In this way, we can use ensemble methods while maintaining the comprehensibility of the model.

Simplicity is an important property in its own right, and is typically motivated by Occam's razor (Blumer et al., 1987). There are two different interpretations of Occam's Razor in the machine learning domain (Domingos, 1998a). The first is that if two different classifiers have the same generalization error the simpler classifier should be used, because simplicity is always preferable. The other interpretation is that if two classifiers have the same error on a training set the simpler one should be used, because it is likely to have a lower generalization error. However, according to Rokach (2010), there is a possibility that the generalization error could continue to improve by increasing the size of the ensemble, even after the training error is zero. However increasing the size of ensemble improves the generalization error while the training error is zero, and the complexity of ensemble increases because the number of classifiers increases. But, if we bring ensembles inside the classifier during the training for decision making, then increasing the size of the ensemble does not affect the number of classifiers, because it remains at one, yet it may continue to decrease the generalization error.

Simplicity, in terms of the actual features used, leads to two other desirable properties: fast classification and small memory footprint. One of the drawbacks of traditional ensemble methods is the requirement for large amounts of memory to store the hypothesis (Ferri et al., 2002). Using Inner Ensembles, we need to store only one model in memory. In many on-line applications, the speed of determining the membership, either in classification or clustering, is an important practical consideration (Williams et al., 2006). For ensemble method classification, time means to classify the examples using all of the ensemble members, but for Inner Ensembles it means to classify for just one model, which is clearly much faster.

1.2.2 Shared Advantages with Traditional Ensemble Methods

Stability is the property of being robust to small changes in the underlying data (Dwyer and Holte, 2007). Robust models are important, because they evoke more confidence that the underlying concept has been truly captured, and their accuracy is less susceptible to noise. One of the advantages of using ensemble methods is that they are robust to variance and, based on this, we expect that using Inner Ensemble will result in more robust decisions that are less sensitive to variation. Thus we expect that Inner Ensembles are generally more robust to noise. Each algorithm has different decision makers whose outputs are making decisions about something, such as assigning an instance to the cluster center, selecting an attribute to split, or calculating the error. Noise in the data will affect the output of these decision makers, and impact the entire model. Thus, using ensemble methods inside the models will result in more robust decision makers that are less sensitive to noise, and models that are more robust to noise. Robustness to noise is the result of using ensemble methods inside the models, and this causes the training or running time of the algorithm to increase; that is the cost we pay for robustness to noise.

1.2.3 Significance

Significance means how broadly an idea can be applied. Inner Ensembles can generally be applied to the decision makers inside an algorithm. The algorithms can be supervised, un-supervised learning, search algorithms, feature selection or different algorithms not related to machine learning such as graph partitioning. One application is when using ensemble methods is problematic, like when the comprehensibility of the resulting model is very important, such as with Bayesian Network, decision rules, or hierarchical clustering such as COBWEB. A further application is when we are unable to use regular ensemble

methods in algorithms with outputs that are not labeling the data, such as association rule mining algorithms. Another useful application for Inner Ensemble is online clustering, since using cluster ensemble is very time consuming. Inner Ensembles can be used to obtain the benefits of cluster ensemble from inside the clustering algorithm. Depending on the application, different ensemble methods inside algorithms can be similar to Bagging, Boosting and other kinds of ensembles. Thus, Inner Ensembles can be applied broadly for different purposes and in different ways than regular ensembles.

1.2.4 Feasibility

Feasibility means why we expect an idea to work. To address this, we discuss the rationale of why ensemble methods work based on the reasoning by Dietterich (2000b), and we explain how these reasons can be extended to Inner Ensembles. According to Dietterich (2000b), there are three different reasons why an ensemble of classifiers is better than a single classifier: *Statistical, Computational and Representational*.

Statistical Reason There is an issue when the number of training samples is very small compared to the number of hypotheses. In this situation, many hypotheses will have very good accuracy for training instances, but their generalization error will be different, which raises the question of which hypothesis should be used. By constructing an algorithm that averages the votes of the classifiers, the risk of choosing the wrong classifier is decreased. Though the final classifier may not be as accurate as the best single classifier, the probability of choosing a bad classifier is reduced. This is illustrated in Figure 1.1 where H_1 and H_2 are the good hypothesis with the same performance on the training sets. H^* represents the best classifier, and the approximation of H^* can be found by averaging the classifiers Ha.

- **How do statistical reasons relate to Inner Ensemble:**

  When using Inner Ensembles, the output of the ensemble is a decision maker that decides about the parts of the model to be created. Therefore, we have the space of decision makers, and by using ensemble methods on this space we can average them. Because decision makers are components of the hypothesis, the averaging over the space of decision makers can be considered another type of averaging of the hypotheses in hypothesis space, which gives us another approximation of the best hypothesis H^*.

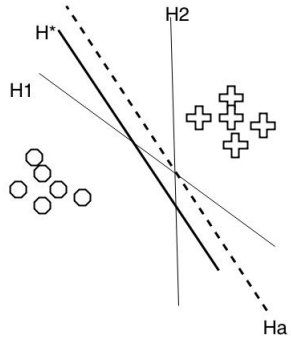


Figure 1.1: Statistical Reason (Dietterich, 2000b).

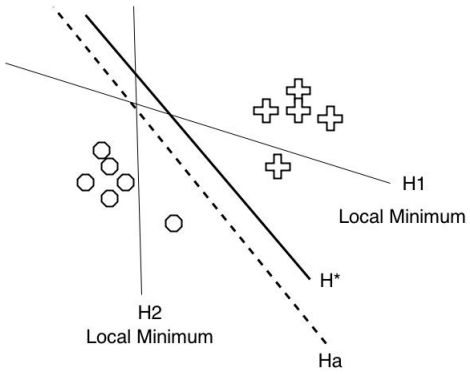


Figure 1.2: Computational Reason (Dietterich, 2000b).

Computational Reason Though many classifiers can conduct a search through the hypothesis space to find the best hypothesis, this local search can be trapped in local minima/maxima (H_1 and H_2). For this reason, an ensemble that searches through the hypothesis space from different starting points could give a better approximation of the best hypothesis. This is illustrated in Figure 1.2.

- **How does the computational reason relate to Inner Ensemble:**

  Having a space of classifiers is similar to having a space of components of the classifiers, and searching for the best classifier is similar to searching for the components in the component space that belong to the best classifier. Inner Ensembles search the component space for each component by starting the search from different starting points, to get a better approximation of each component, with the expectation that the resulting components will be the components of the best classifier.

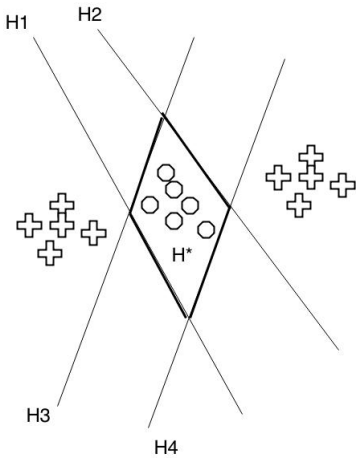


Figure 1.3: Representational Reason (Dietterich, 2000b).

Representational Reason The final reason is representational. In many machine learning problems, the learning algorithms may not include all the hypotheses, which means that the hypothesis space is limited for some cases. In these cases, it is possible that the best hypothesis is not in the search space (linear hypothesis H_1, H_2, H_3, H_4). Using an ensemble of classifiers could find a hypothesis that is not in the search space, and it could be a better estimate of the best hypothesis (H^*). This is illustrated in 1.3.

- **Is it possible to relate the representational reason to Inner Ensemble:**

  This reasoning cannot be extended to Inner Ensembles completely. This is because Inner Ensembles are similar to finding the components of a classifier in the space of all components; the resulting classifier cannot be outside the space of the classifiers. Although this reason may not be valid externally, it can be extended internally. This means that, if we assume that we have a space of decisions inside each algorithm for each decision maker, then the best decision may not be found on that space if we use the measure for decision making. By using an ensemble of decision makers, there is a possibility to make decisions that are not in that space, and which might be better decisions. It should be noted that, for Statistical, Computational and Representational reasons, using Ensembles does not guarantee the best answer. However, it does guarantee that we will not get the worst answer.

1.3 Hypotheses of the Thesis

In this thesis, we propose and test different hypotheses. The main goal is to introduce a new framework for applying Inner Ensembles to different algorithms. We apply Inner Ensembles to test the following hypotheses:

1. We expect the output of the Inner Ensembles will be more comprehensible than regular ensembles because it is only one model. Regular ensembles create multiple models which makes them more difficult to interpret.

2. Inner Ensembles only work on the same datasets as regular ensemble methods, and we do not expect regular ensembles to work for every dataset. Since Inner Ensembles can be considered regular ensembles which work inside the algorithm, we believe they improve the performance of the datasets on which ensemble methods work in general. Thus, if regular ensembles do not improve the performance we exclude the dataset from the experiments, because we presume that Inner Ensembles will also not work on that dataset. In addition, we needed to compare Inner Ensembles to both regular ensembles and the original algorithm, to test the hypothesis that Inner Ensembles work significantly better than the original algorithm, and not significantly differently than regular ensembles. The comparisons are done on multiple domains, and we used statistical testing to determine the significance. The suitable statistical tests for multiple comparisons on multiple domains are ANOVA or Freidmans test Japkowicz and Shah (2011). Since ANOVA uses assumptions, we used Friedmans test in all our experiments, as it is non-parametric and does not use assumptions.

3. We expect that Inner Ensembles will perform better than the original method, because we use ensemble methods for decision making during the training step of the algorithm. This results in more robust decision makers and, ultimately, more robust models.

4. We expect that Inner Ensembles will have significantly faster classification times than regular ensembles as the output is only one model, while the output of regular ensembles is multiple models. We also expect that the classification time of Inner Ensembles will not be significantly different than the original methods, as both return a single model.

5. Finally, we expect Inner Ensembles to be more robust with respect to noise than regular methods. This is because regular ensembles are robust to variations, and noise is a type of variation. Since Inner Ensembles are inside the algorithms, we expect that they will be more robust for noisy training data.

In addition to the hypothesis we mentioned, Inner Ensembles have disadvantages and limitations related to the no-free-lunch theorem, which states that there is no model that works best for every problem, unless assumptions are made related to the model. The assumption here is that our method works on the same datasets for which regular ensembles improve the performance. One disadvantage of this method is that the training time is very slow. We believe that classification time is more important than training time, and our method causes slow training due to using ensembles during the learning step. Another disadvantage is that Inner Ensembles cannot improve performance as much as regular ensembles. Assessing the advantages and disadvantages of Inner Ensembles, we believe that the higher comprehensibility and faster classification time surpasses the disadvantages. Other advantages include higher robustness to noise over regular ensembles, and better performance than the original method. We argue that a slow training step is usually not as important as a faster classification step.

1.4 Contributions of this Thesis

This research addresses the problem of using ensemble methods for building models, a process we called as Inner Ensembles. The key contribution is to show that ensemble methods can be applied more generally than they have been. We provide general guidelines for using ensembles inside different algorithms. To show how broadly this idea can be applied, we have applied Inner Ensembles to two categories of learning: supervised and unsupervised. For the former we use Bayesian networks, and for the latter K-Means clustering. In later chapters, we will discuss how we can extend this idea to other algorithms, and how it is possible to apply different kind of ensemble methods inside models.

1. **Applying Ensembles More Broadly**

   One contribution of the thesis is to show that ensemble methods have been used in a limited manner, and that there is considerable latitude to extend this. In this thesis, we detail one way that this can be achieved. In particular, we provide a general

guideline to using Inner Ensembles on different algorithms. More specifically, we first have to be able to find a decision maker inside each algorithm. Then, we provide a general method of using ensemble methods on the decision maker inside the algorithm. This guideline is based on the measure on which that decision maker works. Applying Inner Ensembles involves changing the input of the measure in different ways, and for each changed input computing the output based on the measure, then finally combining the generated output to get the final output.

2. **Structure of the Bayesian Network:**

   For supervised learning, we chose Bayesian network to apply Inner Ensembles. Specifically, a novel method of learning of the structure of the Bayesian network based on the Inner Ensembles is introduced. Despite other methods for learning the structure of the network which either do not use ensembles or they only combine different structures based on voting, this method works based on using ensembles during the learning of the structure to decide about the parents of each node in the network. This method is applied to the algorithm for building the structure of a network K2, and the resulting algorithm is known as Inner Ensemble K2 (IEK2). This method has some advantages over ensemble of Bayesian network such as comprehensibility of the network, fast classification time and smaller memory footprint while it has the benefit of the ensemble methods such as better performance in comparison to the original network.

3. **K-Means Clustering:**

   For unsupervised learning we chose K-Means clustering and we introduced a novel way of clustering by applying Inner Ensembles on this clustering method. Despite cluster ensembles that generally partition the space using different clusterings and then combine the results of each clustering method, we uses ensemble methods in each iteration of K-Means to find the best cluster center. We use a random subset of features for each ensemble member inside the clustering. Based on the feature subset we find the best cluster center, and the best cluster center is determined by voting on the ensemble members in each iteration of the clustering; finally, every instance is assigned to the best cluster center. We call the resulting clustering Inner Ensemble K-means (IEKM). This method has several advantages over regular cluster ensembles such as keeping the prototypes of each cluster, smaller memory foot print while it performs significantly better than regular K-Means. In addition

this method has the potential to be applied in online clustering, which is very difficult or impossible for cluster ensembles.

4. **Inner Ensembles on Other Algorithms**

   To show how we can apply Inner Ensembles on algorithms other than Bayesian networks and K-Means clustering, we used the general guideline for eight more algorithms from different categories with the aim that the user can find similar ways to apply Inner Ensembles on the algorithms similar to what we applied Inner Ensembles for. The eight algorithms come from the following groups: supervised learning, unsupervised learning, online clustering, algorithms for pruning the decision tree, and an associate rule mining algorithm. We also discuss this idea in detail for these algorithms. Finally, in order to explain the decision makers inside the similar algorithms from these categories, we provide a table in which we can find a method, a decision maker and how to use ensemble methods for that decision maker. The balance of the thesis is based on this general approach.

1.5 Thesis Organization

The reminder of the thesis is organized as follows: In Chapter 2, we discuss related works in detail, including how previous methods are related to Inner Ensembles and how they are different. In Chapter 3, we apply Inner Ensembles to build the structure of the Bayesian network, and compare the results to Bagging, Boosting and a single Bayesian network in terms of accuracy, classification time and size of the network. Chapter 4 is dedicated to applying Inner Ensembles for K-Means clustering, a very popular clustering method, and comparing the results with different cluster ensembles and original K-Means clustering. In Chapter 5, we suggest the possibility of extending the Inner Ensemble framework to other algorithms, and provide a table that can be used as a guide to apply Inner Ensembles for other methods. The table includes the name of the algorithm, the decision maker inside each algorithm, and potential ways of using Inner Ensemble for that algorithm. Finally, Chapter 6 presents the conclusion and potential future work.

Chapter 2

Related work

In this chapter, we review research related to Inner Ensembles, and position our work among them. The main motivation behind the wisdom of the crowds is that a decision made by a group of people is more robust than one made by an individual. This idea has been explored in an important sub-field of machine learning, known as ensemble methods. Although ensemble methods produce substantial performance gains they also have disadvantages, including lack of comprehensibility, slow classification and a large memory footprint. To solve these issues, researchers have explored a single model that is similar to ensemble in terms of performance. We can divide this research into two general directions, each of which aims to address the disadvantages of ensemble methods. However, they usually fail to deal with the problems completely, or they introduce new problems. In the first direction, instead of using an ensemble to decide about the parts of a model, as our method does, an ensemble itself is used as part of the model to gain comprehensibility through the simplified high level structure (Zimmermann, 2008). However, they could not totally satisfy the lack of comprehension because there are ensembles albeit inside the model which still need interpretation. For example Zimmermann (2008) uses an ensemble of rules inside a decision nodes of the tree, for which the decision about splitting the data is made using voting. Therefore, although the high level structure is easy to interpret, the final decision tree is hard to interpret. In addition, this method cannot be used for other classifiers, only for decision trees.

The second direction uses the standard ensemble as a guide to grow a new, simpler model that is comprehensible. For this direction, the important methods are Van Assche and Blockeel (2008); Domingos (1998b); Ferri et al. (2002). These methods are less related to Inner Ensembles, because in most of them ensemble methods are not used

for decision making in models directly. The common characteristic of these methods is that they are based on creating an ensemble separate from learning step which is done usually based on generating artificial data, and using that ensemble as a guide to grow a simpler model. Although there is a framework for these methods that works for supervised learning methods, it is far from clear that it can be applied to other types of algorithms, such as un-supervised learning. Another method in this direction is by Liu et al. (2007). This method does not rely on artificial data, but the ensemble is created outside the algorithm and then members of the ensemble are combined to generate a single model, which is used as a base to create the final model. This method also does not work on all algorithms.

We argue that there is a clear need for a framework to improve the original algorithm, so that the resulting model is comprehensible and efficient in terms of time and memory. In addition, this framework should have the potential to be applicable to any algorithm, and it should not be based on generating artificial data. Generating artificial data is not always straightforward, the distributions can be very complex, and even examples may not have the similar number of attributes, as in relational datasets (Van Assche and Blockeel, 2008). Our method does not dismiss generating artificial data, it simply eliminates the need to do so. However, if generating artificial data is useful, it is possible to do it during the Inner Ensembles training set. As far as we know, there are two pieces of work that can be considered Inner Ensembles (Prez et al., 2004; Geurts and Wehenkel, 2000), and we explain these two algorithms in detail later in this chapter. However, they were focused solely on decision trees.

To explain the related work, and position our approach with respect to closely related research, we use Table 2.1, which shows the characteristics of each group of methods, and how they fill the gap that is introduced by ensemble methods. In the table we divide the algorithms into four groups. The first group are the regular ensembles, and in this group we focus on the basic ensemble algorithms for supervised learning, Bagging and Boosting, and discuss why these algorithms work. In addition, we explain the basic ensemble methods for cluster ensembles, which include graph based and hierarchical methods. The second group is the one that uses ensembles as part of the model, and includes the work by Zimmermann (2008). The third group includes the methods that use the standard ensemble as a guide to grow a simpler model: (Van Assche and Blockeel, 2008; Domingos, 1998b; Ferri et al., 2002; Liu et al., 2007). And the last group is the Inner Ensemble group, which includes the methods by Prez et al. (2004) and Geurts and Wehenkel (2000).

Table 2.1: Comparison of Related Methods with Inner Ensembles.

	First Group Ensemble Methods	2nd Group Ensemble as Parts of the Model	3rd Group Ensemble as A Guide	Fourth Group Inner Ensembles
Comprehensible	✖	✖	✔	✔
Applicability	✔	✖	✖	✔
Does Not Need Artificial Data	✔	✔	✖	✔
Memory Efficiency	✖	✔	✔	✔
Classification Time Efficiency	✖	✔	✔	✔
Not Using Ensembles as the Part of the Model	✔	✖	✔	✔
Not Training Ensembles Outside the Algorithm	✖	✔	✖	✔

According to the table, ensemble methods are not comprehensible, memory or time efficient, and they do not use an ensemble during the learning step. In contrast, the methods of the second and third groups are memory and time efficient. However, as shown in the table, none of these methods have all the other characteristics only Inner Ensembles do and the results are comprehensible and memory efficient. As the result is one model, it is also efficient in terms of classification time. Inner Ensembles do not rely on generating artificial data, or use an ensemble of decision makers as part of the model, as this makes the model hard to interpret.

The rest of this chapter is divide into three parts. In the first part we explain work related to regular ensembles, in the second we discuss work related more closely to Inner Ensembles. In the third part we explain the characteristics of regular ensembles that may be applied to Inner Ensembles, such as robustness to noise and the effect of ensembles on bias and variance. For the related work, we explain the methods from each group, and highlight their similarities and differences with respect to Inner Ensembles, as well as their shortcomings. Figure 2.1 shows the hierarchy of the organization of this chapter. For the rest of the chapter we follow Figure 2.1 step-by-step, to review the methods of each group.

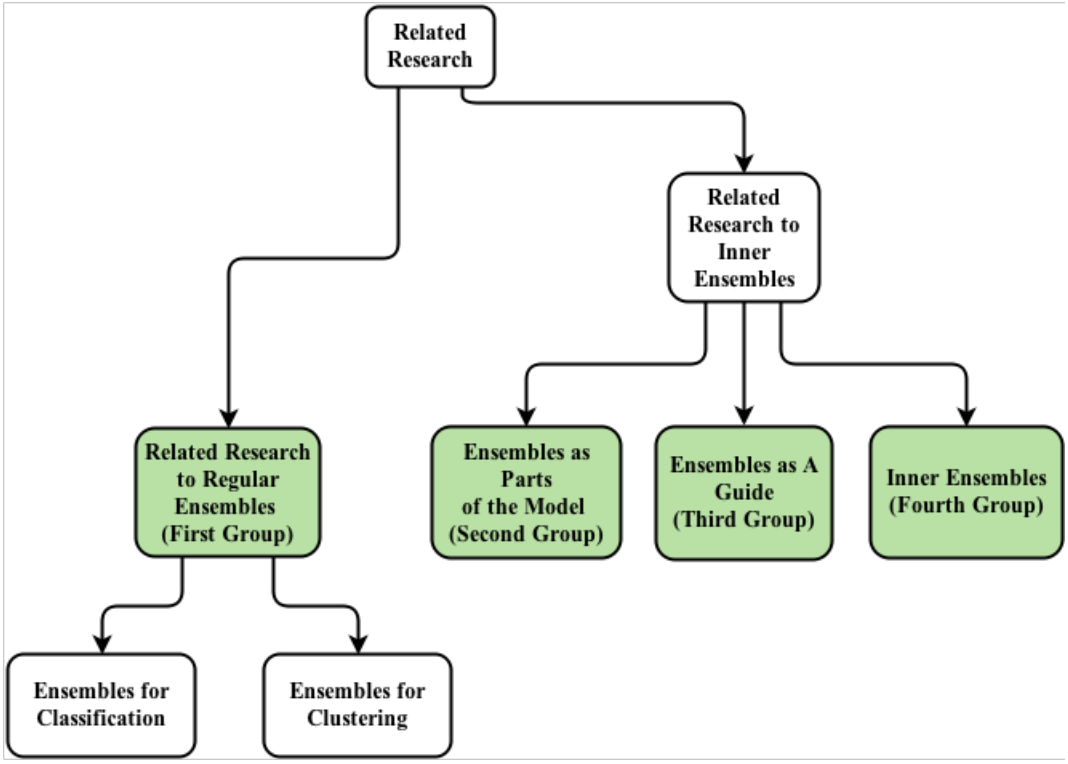


Figure 2.1: Related Research Chapter Organization

2.1 Related Research to Regular Ensembles

In this section, we review the methods from the first group according to Figure 2.1, which include the ensemble methods for classification and clustering. We first explain the basic ensembles for classification, namely Bagging and Boosting, then we review two groups of cluster ensemble methods, Graph-Based methods and Hierarchical Agglomerative methods.

2.1.1 Ensembles for Classification

In this section, we explain two well-known basic ensemble methods, Bagging and Boosting. We also review the reasons why these methods work, because we expect that using Bagging and Boosting inside the algorithms will work for similar reasons.

Bagging

Bagging is one of the general approaches (Breiman, 1996a). It typically works by generating bootstrap samples of the data to generate different training sets, then using the training sets to train different classifiers and, finally, aggregating the predictions of all the trained classifiers to give the final prediction. Assume that the training set is L, and it is defined by $L = \{(x_n, y_n), n = 1..N\}$. In regression problems y is a numerical response, and in classification problems y is a class label. Assume that a classifier is trained using L, and the output of that classifier for test instance x is $\varphi(x, L)$. Assume that $\{L_k\}$ are the sets of N independent observations from the original dataset L, with the same distribution as L. The goal of Bagging is to use $\{L_k\}$ to generate different classifiers, and combine the predictions to get a better result than a single predictor$\varphi(x, L)$ (Breiman, 1996a). There are two basic situations: regression and classification. The first is when y is numeric, in which case the result of combining predictors is the average of the predictions. In this case, $\varphi(x, L)$ is replaced by $\varphi_A(x) = E_L(\varphi(x, L))$, for which E_L is the expectation over all the training sets, and L, and φ_A is the aggregation. The other situation is classification.

For classification, a predictor predicts a class value $j \in \{1, ..., J\}$, and the predictions of learners $\varphi_A(x, L_k)$ are combined by voting. Voting is defined as $\varphi_A(x) = argmax_j N_j$, for which $N_j = nr\{k, \varphi(x, L_k) = j\}$. As explained, for Bagging there should be different learning sets $\{L_k\}$, but in reality there is only one, L. Therefore, the rest of the learning sets are generated using bootstrap sampling of the original learning set (Efron and

Algorithm 2 Bagging algorithm.

1: **TRAINING:**
2: D: Ensemble of Classifier.
3: L: Ensemble Size.
4: **for** k=1 to L **do**
5: $S_k = bootstrap(Z)$
6: $Build(D_k, S_k)$
7: $D = D \cup D_k$
8: **end for**
9: **return** D
10: **TEST:**
11: Classify x using trained classifiers $D_1, ...D_L$.
12: Assign the class that has maximum number of votes to x.

Tibshirani, 1993). Suppose that the generated bootstrap samples are $\{L^{(B)}\}$, then the predictors that are trained using the generated bootstrap samples are $\{\varphi(x, L^{(B)})\}$. The aggregation prediction is then denoted by $\varphi_B(x)$, and calculated as the average of all predictions for numerical problems, and by voting for classification problems. According to Breiman, this procedure is called as **B**ootstrap **agg**regat**ing**, or Bagging. Algorithm 2 shows the Bagging procedure (Kuncheva, 2004). In this algorithm, the bootstrap function generates bootstrap samples from dataset Z, and $Build(D_k, S_k)$ builds and trains a classifier D_k on a bootstrap sample S_k.

Why Does Bagging Work?

There are two different lines of reasoning to explain why Bagging works, including Breiman's notion of an order-correct learner[1] (Breiman, 1996a), and reasoning using Bayesian learning theory (Domingos, 1997).

Breiman's reasoning The predictor $\varphi(x, L)$ predicts one of the labels $j \in \{1, ..., J\}$ for an input x. This label is defined as

$$Q(j|x) = P(\varphi(x, L) = j) \tag{2.1}$$

[1] "A learner is order-correct for an example x if, given many different training sets, it predicts the correct class for x more often than any other" (Domingos, 1997)

for which $Q(j|x)$ is the relative frequency of times that predictor $\varphi(x, L)$ predicts j for input x over different training sets L. If $P(j|x)$ is the probability that x has the class label j, then the probability that the predictor gives the correct class to input x is:

$$\sum_j Q(j|x)P(j|x) \tag{2.2}$$

and the probability of correct classification for all inputs is given by:

$$r = \int \left[\sum_j Q(j|x)P(j|x)\right] P_X(dx) \tag{2.3}$$

The Bayesian classifier is $\varphi^*(x) = argmax_j P(j|x)$. In equation 2.2, the maximum value of $\sum_j Q(j|x)P(j|x)$ is equal to $max_j P(j|X)$. By replacing $max_j P(j|X)$ in equation 2.3, we get the classification accuracy for all inputs (Breiman, 1996a):

$$r^* = \int max_j P(j|x) P_X(dx) \tag{2.4}$$

This classification accuracy is the best trainable classification accuracy, because the predictor is the Bayesian predictor. Using the above definitions, Breiman defined an order-correct predictor φ at x if:

$$argmax_j Q(j|x) = argmax_j P(j|x) \tag{2.5}$$

the aggregated predictor is defined as $\varphi_A(x) = argmax_j Q(j|x)$, and the probability that aggregated predictor classifies input x correctly is

$$\sum_j I(argmax_i Q(i|x) = j) P(j|x) \tag{2.6}$$

in which I is an indicator function. Now, if C is the set of all inputs x for which predictor φ is order-correct, then the accuracy of the aggregated predictor for all inputs x will be (Breiman, 1996a) :

$$r_A = \int_{x \in C} max_j P(j|x) P_X(dx) + \int_{x \in C} \left[\sum_j I(\varphi_A(x) = j) P(j|x) P_X(dx)\right] \tag{2.7}$$

This equation shows that if the predictor φ is order-corrected for most of the inputs, the second term in the equation will be small, and the equation will be very close to an optimal predictor.

Domingo's Reasoning Domingos claims that Breiman's reasoning is limited, because it is not known if a learner is order-correct for an input x (Domingos, 1997). Domingos tested two alternative hypotheses to explain why Bagging works. The first was based on Bayesian Learning theory, and the second on changing the model space, or the prior distribution. Accordingly, Domingos two hypotheses are:

1. "Bagging reduces a classification learners error rate because it more closely approximates the optimal procedure of Bayesian model averaging than the single model output by the learner."

2. "Bagging reduces a classification learner's error rate because it changes the learner's model space and/or prior distribution to one that better fits the domain."

Domingos conducted different experiments to validate these two hypotheses, and his results indicated that it is unlikely the first hypothesis is correct, while the second hypothesis is confirmed (Domingos, 1997).

Boosting

Another algorithm that uses resampling to generate members of an ensemble is known as Boosting. As discussed above, Bagging works by generating bootstrap samples of the data, which is sampling with replacement and equal weights for each sample. Bagging works in parallel, each classifier can be trained independent of other classifiers. With Boosting, the ensemble is trained in serial; that is, the next classifier is trained based on the error of the previous classifier during several iterations. In each iteration the weights of the examples are changed based on the error of the trained classifier, and more weights are assigned to the misclassified data. Then, in the next iteration, a weighted sample of the data is extracted based on the weights of the data that have been assigned in the previous iteration, and a new classifier is trained on the samples. The final label of the data is calculated by weighted averaging of different classifiers' outputs.

The origins of Boosting can be found in the framework of PAC learning (Valentini and Masulli, 2002; Valiant, 1984). Boosting is an answer to the question whether it is possible to boost a weak learner, that performs slightly better than random guessing in a PAC framework, to become a strong model (Kearns and Valiant, 1994).

Schapire (1990) developed the first Boosting method; Freund (1995) developed a more efficient version; Adaptive Boosting (AdaBoost) was jointly developed by Freund and Schapire (1997). There are different names for Boosting algorithms in the literature. For

example, Breiman (1996b) called the family of Boosting algorithms "adaptive resample and combining", or *arcing*. He also called Adaboost *arc-fs*.

Adaboost has been investigated by many researchers. Bauer and Kohavi (1999) conducted a large empirical comparison of different variants of Adaboost and Bagging when they are applied to decision trees and Naive Bayes classifiers, to improve the understanding of why and when these algorithms affect the error. Their study showed that Bagging generally reduces the variance for an unstable classifier, while Adaboost reduces both variance and bias. Dietterich (2000a) compared the effectiveness of three methods, Bagging, Boosting and randomization, for generating ensembles of decision trees in the presence of noise . The study showed that without noise, Bagging and randomization are comparable but less effective than Boosting. However, in the presence of noise Bagging is much better than either of the others. Maclin and Opitz (1997) evaluated Bagging and Boosting on neural networks and decision trees. The results showed that, although it is generally accepted that Bagging works better than the base classifier, it is not better than the base neural network. The results also showed that Boosting is a better classifier than Bagging in terms of accuracy, though it is more susceptible to noise. Quinlan (1996) also applied Bagging and Boosting on decision trees, and tested the predictive accuracy on several datasets. The results showed that, while both methods improved the accuracy, and Boosting provided greater improvement overall, although Boosting also decreased the accuracy on some of the datasets. Schwenk and Bengio (1998) applied different training methods based on sampling the training set, thereby re-weighting the cost function on Adaboost to improve the performance of neural networks for character recognition. There are many applications for Adaboost. For example, it has been used for text categorization (Schapire and Singer, 2000), text filtering (Schapire et al., 1998b), ranking problems (Freund et al., 2003) and in natural language processing (Abney et al., 1999). In this section of the chapter, we explain the Adaboost algorithm in detail and discuss why Boosting works, then introduce some variants of Boosting.

The original Adaboost algorithm, Adaboost.M1, was intended for two-class classification. Then Adaboost was extended to multi-class classification, and renamed Adaboost.M2 (Freund and Schapire, 1995). Algorithm 3 shows the Adaboost.M1 algorithm (Kuncheva, 2004). It starts by giving the initial weight (usually equal weight) to every sample. Thus, if the number of samples is N, the weight of each sample will be $\frac{1}{N}$ (line 2). According to the algorithm, w^k is the distribution that is used for sampling in iteration k (lines 2, 14). In each iteration, the distribution w^k is changed, then used to sample the data to train a new classifier (line 14). The number of classifiers that should

be indicated is L (line 4). In each iteration k, first the data Z is sampled according to the distribution w^k (line 6). It is called S_k, and is done using function $Sample(Z, w^k)$. Then a classifier D_k is trained, using S_k (function $Build(D_k, S_k)$) (line 7). The weighted error is calculated in step k, and defined by the equation in line 8. In this equation, w_j^k is the weight of instance z_j at iteration k, and l_k^j is the zero one loss function, which is equal to 1 if D_k misclassifies the input instance z_j, otherwise it is zero. If the classifier is good its error is zero; if the classifier is weak its error is larger than 0.5, and classifier D_k will be ignored (line 11). Otherwise, the weights of the instances will be updated using the error function, and more weight will be assigned to the misclassified instances so that the classifier in the next iteration focuses on them more (line 14). In addition, a weight β_k is assigned to each classifier according to its error (line 13). This procedure repeats for all L classifiers, and classifiers D and their weights $\beta_1, ..., \beta_L$ are returned for the classification step (line 17). In the classification step, the weighted majority vote is used to determine the class for each test instance x, then, for each class, the degree of support will be defined as the equation in line 20. Finally, the class with the most support will be selected as the label of instance x (line 21) (Kuncheva, 2004).

Why Does Boosting Work?

As reported by Kuncheva (2004), the success of Adaboost can be explained in two ways, according to either the upper boundary of the training error, or the upper boundary of the testing error. For the training error, Freund and Schapire showed that increasing the number of classifiers causes the training error of the ensemble to approach zero. On the other hand, the upper boundary of the testing error depends on the margins. Here, we briefly explain the upper boundary of the training error and the margin theory.

Analysis of the Upper Bound on The Training Error The upper boundary of the training error has been proven for the Adaboost algorithm with regard to two-class classification (Freund and Schapire, 1995). As a result, if two classes are $\{w_1, w_2\}$ and $\epsilon_i, i = 1, .., L$ are the weighted training errors of the classifiers given by $\epsilon_k = \sum_{j=1}^{N} w_j^k l_k^j$ in the Adaboost algorithm, then for the ensemble training error ϵ, the following equation is proven:

$$\epsilon < 2^L \prod_{i=1}^{L} \sqrt{\epsilon_i(1-\epsilon_i)} \tag{2.8}$$

Algorithm 3 AdaBoost.M1 algorithm.

TRAINING:
$w^1 = [w_1^1, ..., w_N^1], w_j^1 \in [0,1], \sum_{j=1}^{N} w_j^1 = 1, w_j^1 = \frac{1}{N}$.
D: Ensemble of Classifier.
L Ensemble size.
for k=1 to L **do**
    $S_k = Sample(Z, w^k)$
    $Build(D_k, S_k)$
    $\epsilon_k = \sum_{j=1}^{N} w_j^k l_k^j$
    $l_k^j = 1$ if D_k misclassifes z_j otherwise $l_k^j = 0$
    if the classifier is perfect ($\epsilon_k = 0$) or it is too weak ($\epsilon_k \geq 0.5$) then ignore D_k
    Otherwise
    $\beta = \frac{\epsilon_k}{1-\epsilon_k}$ where $\epsilon_k \in [0, 0.5]$
    $w_j^{k+1} = \frac{w_j^k \beta_k^{1-l_k^j}}{\sum_{i=1}^{N} w_i^k \beta_k^{1-l_k^i}}$, $j = 1, ..., N$.
end for
return D and $\beta_1, ..., \beta_L$.
CLASSIFICATION:
Compute the sum of weights corresponding to classifiers that predict class t:
$W_t(x) = \sum_{D_k(x)=t} Ln(\frac{1}{\beta_k})$
Lable for x is chosen as: $argmax_t W_t(x)$.

In the case of multiple-class classification, the error boundary is the same if the error of each individual classifier in the ensemble is less than 0.5 (Freund and Schapire, 1995). This equation shows that increasing the number of classifiers causes the training error of the ensemble to approach zero, which is one of the reasons for the success of the Adaboost algorithm. More information about the proof of the equation 2.8 can be found in Freund and Schapire (1995), and in the Appendix of Kuncheva (2004).

Margin Theory Another reason for the success of Boosting can be explained by the upper boundary of the testing error of the ensemble. Margin is related to the concept of VC- dimension (Vapnik, 1995). VC-dimension gives the upper boundary of the classification ability of the classifier (Kuncheva, 2004), and an upper boundary of the testing error. Thus, the testing error for a classifier is related to the concept of margins. According to Kuncheva (2004), the margin of an object is related to the certainty of the

classification, and can be negative or positive. Negative margins indicate misclassifications, and positive margins indicate correct classifications. Positive margins can be certain or uncertain, thus correctly classified instances with high certainty have large margins, and correctly classified instances with low certainty have small margins. For a c class problem the margin of object x is calculated as:

$$m(x) = \mu_k(x) - max_{j \neq k}\{\mu_j(x)\} \tag{2.9}$$

where $\mu_j, j = 1, ..., c$ is the degree of support for each class, and $\sum_{j=1}^{c} \mu_j(x) = 1$. According to equation 2.9, misclassified instances have negative margins and correctly classified instances have positive margins. Schapire et al. (1998a) proved an upper boundary for the testing error of ensembles that depends on the margins. The following theorem is for two class classification, that is $c = 2$:

"Suppose that H is the finite space of the classifiers. For any $\theta > 0$ and $\delta > 0$, with probability at least $1 - \delta$ over the random choice of the training set Z , any classifier ensemble $D = \{D_1, ..., D_L\} \subseteq H$ combined by weighted average satisfies: (Kuncheva, 2004)"

$$P(error) \leq P(training_margin \leq \theta) + O\left(\frac{1}{\sqrt{N}}\left(\frac{logNlog|H|}{\theta^2} + log(\frac{1}{\delta})\right)^{\frac{1}{2}}\right) \tag{2.10}$$

In this equation, $P(error)$ is the probability of the error of the ensemble for a randomly drawn point x, $P(training_margin \leq \theta)$ is the probability that the margin of the randomly drawn training instance is less than or equal to θ, and $|H|$ is the number of classifiers in the classifier space [2].

Comparing to Inner Ensembles and Limitations

There are some limitations related to ensemble methods. The first and most important one is lack of comprehensibility. When we apply ensemble methods, we train several classifiers and combine their outputs, thus the results are very hard to interpret. An example is the ensemble of the decision trees. It is possible to understand single decision tree but when we have an ensemble, it is very hard to understand the output. With Inner Ensembles, we use ensemble methods to generate models that finally result in one single model. In this way, we can use ensemble methods while maintaining the

[2] More details can be found in (Kuncheva, 2004; Schapire et al., 1998a)

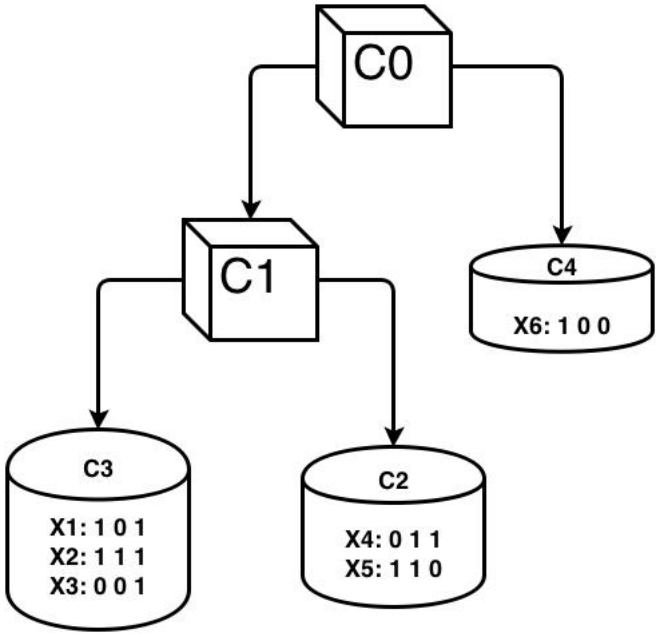


Figure 2.2: An Example of COBWEB Hierarchy.

comprehensibility of the model. Another drawback of traditional ensemble methods is the need for large amounts of memory to store the hypothesis. As we increase the ensemble size the memory requirement grows linearly, but this is not usually an issue for most computers. However, if we use devices with limited memory, such as mobile phones, the memory requirements become very important. For example, if we have a speech recognition system installed on a mobile phone, the ensemble of this recognition classifier would need a great deal of memory. In addition for ensemble methods, classification time means to classify the examples using all of the ensemble members. Thus ensemble methods have slow classification time which is important in many on-line applications. However, using Inner Ensembles, we only need to store one model in memory which needs less amount of memory and its classification time is much faster.

2.1.2 Ensembles for Clustering

Cluster ensemble methods have similar characteristics to ensemble methods in Table 2.1. Some clustering methods, including K-Means, use prototypes to represents the clusters. One example of prototypes is the cluster center, and another is a single data point from the cluster that is representative of the cluster. For example, with respect to medical data, a patient with special features, such as test results, age or sex, can be representative

of a group of specific diseases. Prototypes are useful in data summarization, compression, or even finding the nearest neighbors efficiently (this is discussed in detail in chapter 4). Another important clustering method is conceptual clustering, such as COBWEB. A characteristic of conceptual clustering is that it represents different concepts using the hierarchy, which is an important way of summarizing data understandably (Fisher, 1987). COBWEB clustering incrementally organizes observations into a conceptual tree, in which each concept (cluster) contains several sub concepts. A clustering hierarchy example can be seen in figure 2.2. According to the Figure, C_0, C_1, C_2, C_3, C_4 are the concepts (clusters), and C_3, C_2 and C_4 have 3,2 and 1 data instance respectively. A new incoming data point is either placed in each of the concepts according to the measure of similarity, or a new concept is created. The figure indicates that the hierarchy shows two things: the partitioning of the data, and the hierarchy itself that represents the relationship among the concepts.

Cluster ensembles create different partitions of the data, and combines these in a single partition by first transforming them to other structures, such as a graph or matrix. During the transformation of the partitions to the new space the prototypes are eliminated, because this step only works on the different partitionings of the data, and the new structure represents the relationships among the data only using their labels[3]. For conceptual clusterings such as COBWEB, each member of the cluster ensemble is a hierarchy, similar to Figure 2.2. When using cluster ensemble, the labels of the data is only information needed, and during transformation of the partitions the hierarchy for each clustering will be lost. Thus, the output of the cluster ensemble may not be comprehensible.

In addition, clustering ensemble may not be memory efficient, because it needs extra memory for the co-association matrix and hyper-graph representation. Clustering time is also long, it needs to cluster the data several times. Extra time is needed for building the co-association matrix, which has polynomial time complexity. However, similar to ensemble methods, cluster ensembles are applicable on all clustering algorithms, and they do not need to generate artificial data. Here, we explain two groups of cluster ensemble methods, graph-based methods and hierarchical agglomerative clustering consensus (HAC), because these are widely used for cluster ensembles (Ghaemi et al., 2009).

The cluster ensemble is more difficult than the classifier ensemble problem, because for cluster ensemble we need to solve the correspondence problem[4] as well. In addition,

[3]During the clustering a "label" is assigned to each partition and the data instances associated to.

[4]Because the clustering labels are symbolic, for the same clustering we may have different labels for

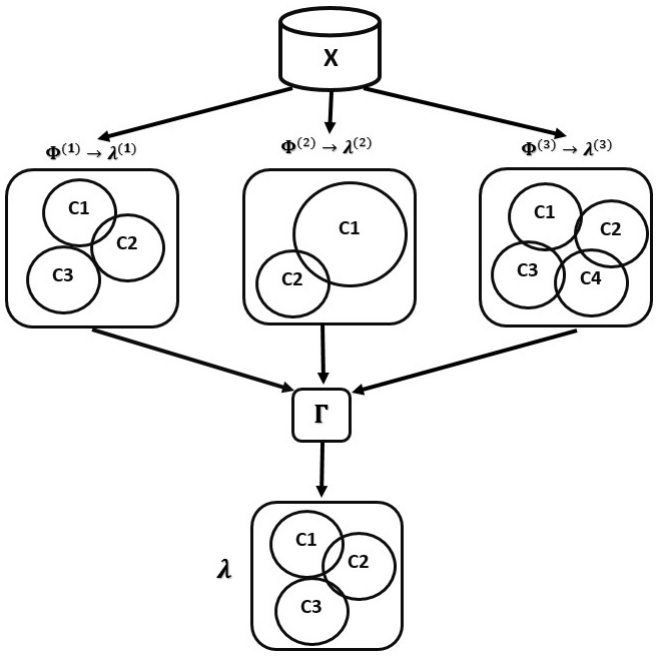


Figure 2.3: Basic Concept For Cluster Ensemble (Strehl et al., 2002).

each solution of the cluster ensemble is different in terms of the number and shape of the clusters. The diversity for cluster ensembles can be created in different ways (Strehl et al., 2002):

1. Using different features: For example different feature subsets.
2. Using different number of clusters.
3. Varying the order of the data in some clustering
4. Using different clustering algorithms

Notation $X = \{x_1, x_2, , x_n\}$ is a set of data objects. A partitioning of n objects into k clusters can be represented as a label vector λ. Φ is a clustering function that generates the labels for a given dataset, and Γ is consensus function that combines the labels. Figure 2.3 shows the basic concept for cluster ensemble (Strehl et al., 2002).

In this section, we focus on Γ, the combining function for different clustering labels, and we explain six different combining schemes from two different families of cluster ensembles.

each data instance; this is called the cluster correspondence problem (Strehl et al., 2002).

Graph-Based Methods

In this section, we explain three different graph-based methods: Cluster-based Similarity Partitioning Algorithm (CSPA), Hyper-Graph Partitioning Algorithm (HGPA) and Meta-Clustering Algorithm (MCLA). All these algorithms approach the problem of combining the outputs of different clustering by first transferring the set of clusterings into a hyper-graph representation. We first briefly explain the representation of the sets of clusterings as a hyper-graph.

Representing Sets of Clusterings as a Hyper-Graph

A graph consists of a set of vertices and a set of edges. Each edge in a graph connects two vertices. But a hyper-graph consists of vertices and a set of hyper-edges, and each hyper-edge can connect any set of vertices (Strehl et al., 2002). For each label vector $\lambda^{(q)}$, a binary membership indicator $H^{(q)}$ is constructed. The number of rows of this matrix is n, the number of data instances, and the number of columns of the matrix is k, the number of clusters in label vector $\lambda^{(q)}$. Then $\lambda^{(q)}(i) = j$ means that the label for data instance i is j. Thus, the entry (i, j) of matrix $H^{(q)}$ should be set to 1. If the label of instances i is missing, then the ith row of matrix $H^{(q)}$ should be 0.

Then the connected block matrix $H = H^{(1,,r)} = (H^{(1)}...H^{(r)})$ is the adjacency matrix with n vertices and $\sum_{q=1}^{r} k^{(q)}$. Each column h_a indicates a hyper-edge, for which 1 indicates the vertices corresponding to that hyper-edge. Thus, this is a way to map each cluster to a hyper-edge, and set of clustering to the hyper-graph (Strehl et al., 2002). Figure2.4 shows the simple mapping from three different clusterings to hyper-graph representation.

Cluster-Based Similarity Partitioning Algorithm (CSPA)

The Cluster-Based Similarity Partitioning Algorithm (CSPA) works by using graph partitioning algorithms on the similarity matrix. The similarity matrix for each cluster, sometimes called the co-association matrix, is an $n \times n$ matrix whose entry (i, j) is 1 if data instance i and j are in the same cluster, and 0 otherwise. The overall similarity matrix S is created by entry-wise averaging of r different similarity matrices, for which r is the ensemble size and n is the number of instances. The matrix can be calculated using matrix H, which is the hyper-graph representation of r different clusterings as $S = \frac{1}{r} H H^T$.

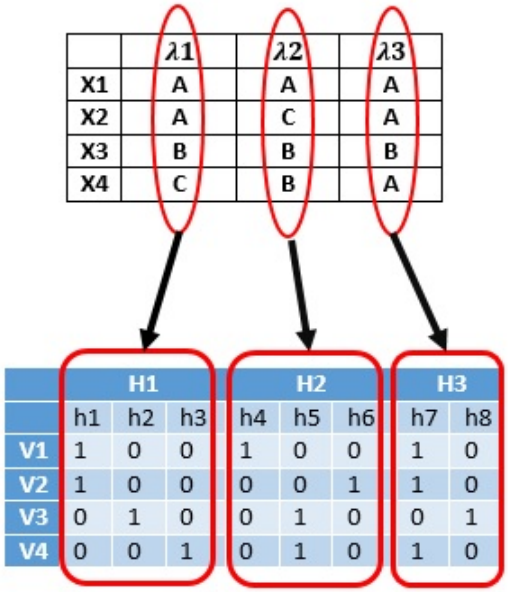


Figure 2.4: Simple Mapping from Original Label Vectors (top) to Hyper-Graph Representation (button) (Strehl et al., 2002).

In the next step, this similarity matrix is used to re-cluster the objects. To do this, CSPA uses the similarity matrix as a graph, where each of n data objects is one vertex, and entry (i, j) shows the edge weight between vertex i and j. It then uses a graph partitioning method called METIS to partition the graph (Karypis and Kumar, 1998). METIS is a multilevel graph partitioning algorithm that is basically quite simple. The graph G is first coarsened down to a sequence of smaller graphs, $G_1, G_2, ...G_m$, such that $|V_1| > |V_2| > ... > |V_m|$. Then the last graph in the sequence, G_m, is partitioned into two parts, each of which contains half the vertices of the original graph G_0. This partition is called P_m. Finally, the partition P_m is projected back to the G_0 through intermediate partitions $P_{m-1}, P_{m-2}, ..., P_1, P_0$. This is the refinement step, and such refinements usually decrease the edge-cut (Karypis and Kumar, 1998).

Hyper-Graph-Partitioning Algorithm (HGPA)

HGPA is a direct approach to using clustering ensembles to re-partition the data. The difference between this method and CSPA is that the CSPA first creates a similarity matrix and represents that matrix using hyper-graph representation H as a single graph, then uses the graph partitioning algorithm on the similarity matrix. HGPA does not

create a similarity matrix. It works directly on hyper-graph representation H and, instead of a graph-partitioning algorithm, it uses equivalent to a hyper-graph partitioning algorithm called HMETIS (Karypis et al., 1997). The cluster ensemble then partitions the hyper graph H by cutting the minimal number of hyper- edges. In this method, all hyper-edges and vertices have the same weights. CSPA works with a graph instead of a hyper-graph, and only considers the relationships between pairs of vertices. HGPA considers the relationships between sets of vertices, then looks for cutting hyper-edge separators that partition the hyper-graph into k unconnected components of approximately the same size, for which k is the number of clusters. This means that HGPA tries to find similar size clusters, since comparably sized partitions is a standard technique in graph partitioning clustering methods. Therefore, if a dataset contains very imbalanced clusters, HGPA does not work correctly on it (Karypis and Kumar, 1998).

Meta-CLustering Algorithm (MCLA)

The idea of MCLA is to group and collapse hyper-edges, and assign each object to the most related hyper-edge. In each step, the related hyper-edges must be found, which can be done by clustering the hyper-edges. Each cluster of hyper-edges is called a meta-cluster $\zeta^{(M)}$. Thus, the first step is to construct a meta-cluster, a clustering of the hyper-edges (Karypis and Kumar, 1998).

Construct Meta-Graph. From figure 2.4 we know that we have $\sum_{q=1}^{r} k^{(q)}$ hyper-edges h. These can be considered as vertices of another undirected graph, and the weights of the edges are proportional to the similarity between the hyper edges. The similarity between two hyper-edges h_a and h_b is calculated by the Jaccard measure (Karypis and Kumar, 1998):

$$w_{a,b} = \frac{h_a^T h_b}{\|h_a\|_2^2 + \|h_b\|_2^2 - h_a^T h_b} \tag{2.11}$$

Then the related labels for the hyper-edges are found by partitioning the meta-graph. The METIS algorithm is used to do this, and it results in the clustering of h vectors.

Collapse Meta-Clusters. For each of k meta-clusters, hyper-edges are collapsed into a single meta-hyperedge. For example, in figure 2.4, assume that the first meta-cluster is $\zeta_1^{(M)} = \{h_2, h_5, h_8\}$. Then, collapsing the hyper-edges results the meta-hyperedge $h_1^{(M)} = \{v_3, v_4\}$. Each meta-hyperedge has an association vector, which contains the level

of association of an object with the corresponding meta-cluster. The level of association is also computed by averaging all indicators h of a particular meta-cluster. For example, for meta-hyperedge $h_1^{(M)}$ the association vector is defined as $(0, 0, 1, 1/3)$. For data instance x_4 in figure 2.4, we can see that it is just a member of h_5, and not a member of h_2, h_8. Thus, its association to $h_1^{(M)}$ is $1/3$.

Compete for Objects. In the final step, each data object is assigned to the most suitable meta-cluster, which is the one with the highest entry in the association vector. For instance , in the previous example x_6, x_7 is assigned to meta-cluster $\zeta_1^{(M)}$, because the association level for x_6, x_7 for this meta-cluster is 1, which is the highest among all the other meta-clusters.

Hierarchical Agglomerative Clustering Consensus (HAC)

This method for cluster ensemble works on the co-association matrix. As explained, the co-association matrix has an entry (i, j) equal to 1 if i and j are in the same cluster, and 0 otherwise. This matrix is calculated for each clustering member $\{C_1, C_2, ..., C_k\}$ of the ensemble with k as the ensemble size, and the final co-association matrix C_F is calculated by averaging all the matrices. Then different hierarchical clusterings are applied to this matrix to extract the final clusters. The difference between this method and CSPA, is that CSPA first converts the final matrix to the graph, while HAC uses the final matrix as a similarity matrix, and an input to the hierarchical clusterings. Here we explain three different hierarchical agglomerative clustering algorithms: Single Linkage, Average Linkage and Complete Linkage. The general algorithm for these three methods is the same, but the methods are different with respect to the distance measure they use for the clustering.

Different Linkage Methods

Hierarchical clustering algorithms is divided in to two distinct approaches. The first approach is called *Agglomerative* (bottom-up) which starts with the data instances as a single cluster, then iteratively clusters them in to larger groups. The other method is *Divisive* (top-down) which starts with all samples as one cluster and iteratively split the clusters into smaller ones. The best known way of representing the hierarchy is by tree or dendrogram (Duda and Hart, 1973). The finest groups at the bottom of the dendrogram are data instances, which can be considered single groups, and the coarsest groups are

Algorithm 4 General Agglomerative Clustering.

1: **Input:** $X_i, i = 1, 2, .., n$: Data instances
2: Begin with n clusters each containing on single instance.
3: **for** m=1 to n-1 **do**
4: Find the most similar clusters C_k and C_j
5: Merge C_k and C_j.
6: **end for**

at the top of the dendrogram, which contains all of the data instances. The intermediate levels between the finest and coarsest groupings are different clusters in each level. If two data instances are not in the same cluster, they may belong to the same cluster at a higher level of the hierarchy. For cluster ensembles, agglomerative methods are applied on the final co-association matrix C_F. The main steps of agglomerative clustering are shown in algorithm 4 (Gose et al., 1996).

Different hierarchical algorithms are obtained by using different similarity measures to find the closest clusters. The best known distance for agglomerative clustering is Euclidean distance. The **Single Linkage** algorithm, known as **nearest neighbour**, is obtained by calculating the distance of two clusters as the smallest distance between two points with each point in different cluster (Gose et al., 1996). Equation 2.12 shows the distance measure for Single Linkage, where $d(a, b)$ is the distance between a and b.

$$D_{SL}(C_i, C_j) = \min_{a \in C_i, b \in C_j} d(a, b) \tag{2.12}$$

The **Complete Linkage** algorithm, also known as the **farthest neighbour**, is obtained by calculating the distance of two clusters as the largest distance of two points with each point in different cluster. Equation 2.13 shows the distance measure for Complete Linkage.

$$D_{CL}(C_i, C_j) = \max_{a \in C_i, b \in C_j} d(a, b) \tag{2.13}$$

The single linkage algorithm allows long, thin clusters, while complete linkage tends to create more compact clusters. However, both methods are sensitive to outliers. The **Average Linkage** method attempts to overcome this problem with single and complete linkage methods. Average linkage method, also known as **unweighted pair-group method using arithmetic averages**,is one of the best known hierarchical clustering methods, and is obtained by calculating the distance of two clusters as the average

distance of all of the points of the clusters. Equation 2.14 shows the distance measure for Average Linkage where n_i and n_j are the number of instances in clusters C_i and C_j respectively.

$$D_{AL}(C_i, C_j) = \frac{1}{n_i \times n_j} \sum_{a \in C_i, b \in C_j} d(a, b) \tag{2.14}$$

Comparing to Inner Ensembles and Limitations

There are several limitations related to the cluster ensemble methods. The first is that for cluster ensembles the labels are symbolic, and combining different partitions can be difficult because similar partitioning involve different labels. This problem is not an issue for Inner Ensembles, because the ensemble method is used inside the clustering, and thus it works with a single label. Another problem with cluster ensembles is the co-association matrix, which is $O(N^2)$ for N data instance, and requires a huge amount of memory for large datasets. For Inner Ensembles, there is no need to store the co-association matrix. In addition, to the best of our knowledge cluster ensemble seems to be extremely difficult to apply for online clustering for each incoming data example clustering has to be run at a different time, then the co-association matrix must be updated and, finally, another algorithm must be applied to the co-association matrix. However, using Inner Ensembles makes it possible to run everything online, because during each iteration of clustering the incoming data can be assigned to the clusters using ensembles, which does not need to calculate the co-association matrix. Plus, using Inner Ensemble retains the comprehensibility of the method, which would be lost with cluster ensembles.

2.2 Related Research to Inner Ensembles

In this section we review the related research to Inner Ensembles. Specifically, we review the methods from the second groups which is using ensembles as parts of the model then we explain different methods of the third group (using ensembles as a guide) and finally we explain the methods which are considered as Inner Ensembles (the fourth group).

2.2.1 Ensembles as Parts of the Model

This group of methods uses ensemble methods as parts of the model in order to gain comprehensibility through the simplified higher level structure (Kohavi and Kunz, 1997; Murthy, 1997; Geamsakul et al., 2003; Zimmermann, 2008). However, the resulting model may not be comprehensible, because there are small ensembles inside the model that make it difficult to interpret. In addition, we do not know of any framework to apply this idea to different types of algorithms. The group is related to Inner Ensembles because it uses ensemble inside the model. For this group, in this section we explain the work by Zimmermann (2008).

Ensemble Tree

Zimmermann (2008) discussed the benefits of decision trees. He argued that they are affected by two disadvantages: over-fitting, and the fact that small changes in the decision tree can result in very different trees. He indicated that, although ensemble methods can be used to solve these problems, they cause the model to become incomprehensible. To maintain the benefits of using ensemble methods in decision trees, and solve the comprehensibility problem, he induced small ensembles of rules as tests inside the node of the decision tree. This method is called Ensemble-Tree.

Zimmermann declared that an ensemble of conjunctive rules performs better than decision lists, and if the stabilizing effect of using a small ensemble of rules is brought inside the node of the decision tree, it is expected that the splits by the ensemble of rules will be less sensitive to changes in the data than splits by a single attribute.

Thus, the first step is to find the best k conjunctive rules, which is done using a branch-and-bound search for convex interestingness measures, such as information gain. This technique is based on calculating the upper bound for each rule, and is done using the method developed by Morishita and Sese (2000).

Algorithm 5 shows the pseudo-code of the Ensemble Tree, and algorithm 6 shows the sub-algorithm for creating the node of the tree (Zimmermann, 2008). In algorithm 6, the k best rules according to the interesting measure ϕ are found in line 2. Note that in line 3, if the algorithm cannot find k conjunctive rules, then that node is converted to a leaf node with the majority class in its data, which is a different stopping criteria than the original decision tree. Then, in lines 6 and 7, the algorithm decides to split the data based on the majority vote of the rules.

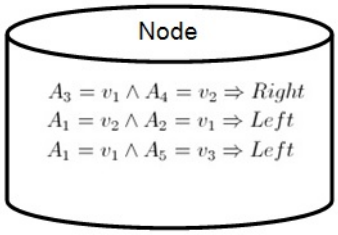


Figure 2.5: A Node of the Ensemble Tree.

Algorithm 5 The Ensemble Tree Algorithm.

1: Input: dataset D, interesting measure ϕ, minimum leaf size m, number of rules k.
2: Output: Ensemble Tree T.
3: $T = CreateNode(D, \phi, m, k)$
4: **return** T

Comparing to Inner Ensembles and Limitations

The work by Zimmermann (2008) is related to Inner Ensembles because its motivation is to have a single decision tree that is comprehensible and it uses ensemble methods to create a decision tree. However it is different from Inner Ensembles because the entire ensemble of rules is placed inside the nodes of the decision tree. For Inner Ensembles, ensembles are used during the learning of the decision tree not inside the model. There are also some limitations to this work. The first limitation is that, although it is more interpretable than the ensemble of decision trees, it is still difficult to interpret. For example, assume that at one node the algorithm finds three rules for binary prediction. Therefore, at that node the data is split by voting for the output of the three rules. Figure 2.5 shows the node of the tree, with the three rules inside the node. In this figure, A_i is the ith attribute and v_j is the jth value of the attribute. As can be seen when interpreting the tree, understanding each node can still be difficult, because there is a small ensemble of rules inside each node. The second limitation is that it is based on a special type of ensemble which is finding k conjunctive rules. These rules cannot be generated by ensembles. Thus, different ensembles cannot be replaced inside the nodes of the tree to find different rules. In addition, the ensemble size in this method cannot be arbitrarily large, because it is possible that we can not find any K best conjunctive rules. Also, this method only allows binary splits, since using conjunctive rules as tests always results in binary splits.

Algorithm 6 Create the Node of Ensemble Tree.

1: $makenode(D, \phi, m, k)$
2: Find k best conjunctive rules $R = \{rule_1, ..., rule_k\}$ according to the measure ϕ
3: **if** Not enough rules ($|R| < k$) **then**
4: **return** majority class in D is used.
5: **end if**
6: $Data_{left}$ is a subset of $d \in D$ for which majority of rules vote them to the left side
7: $Data_{right}$ is a subset of $d \in D$ for which majority of rules vote them to the right side
8: **if** Sizes of $Data_{left}$ and $Data_{right}$ are larger than or equal to m **then**
9: $Node_{left} = CreateNode(D_{left}, \phi, m, k)$
10: $Node_{right} = CreateNode(D_{right}, \phi, m, k)$
11: **return** node(R,n_{left},n_{right})
12: **else**
13: **return** leaf(majority class(D))
14: **end if**

2.2.2 Ensembles as a Guide

This group uses standard ensemble methods as a guide to growing a comprehensible model. The general method is to first create an ensemble through an off-line procedure, and then use that ensemble to grow a new model, with the aim of having a model that works comparably with the generated ensemble. For this group, generating an ensemble could rely on generating artificial data (Domingos, 1998b; Ferri et al., 2002), or only rely on creating ensembles using the original data (Van Assche and Blockeel, 2008; Liu et al., 2007). From this group, only Domingos (1998b) works with different algorithms. However, relying on an artificial dataset is an important limitation with this method. We explain these methods in the following.

Combined Multiple Models (CMM)

The most well-known method of this group is the work by Domingos (1998b). He stated that, in addition to accurate models, data mining searches also extract useful knowledge from the databases, which means the comprehensibility of the algorithms is very important. He maintained that, although decision trees are comprehensible, the disadvantage is they allow a learner to be very sensitive to changes in the training set. Ensemble methods usually solve the problem of instability of the decision tree, and usually pro-

duce highly accurate outputs. However, they significantly decrease the comprehensibility, because there is more than one model.

To solve this problem, he proposed a framework to extract a single model that has performance comparable to that of the ensemble of models. This framework is based on generating artificial data. Because an ensemble of models is more complex than a single model, the single model that is generated using this framework is expected to be more complex than the single model. As well, because this framework outputs an estimate of the ensemble of the classifier, the output of the framework is likely less accurate than the ensemble of classifiers, but more accurate than a single classifier.

Algorithm 7 shows the pseudo-code of this framework, which is called **Combined Multiple Models** (CMM) (Domingos, 1998b). In the algorithm, each model M_i is produced first. Then a number of samples $\vec{x}$ are randomly generated, and labels are assigned to the samples by applying a combination of M_is. The newly labeled samples are then added to the training samples, $(S = S \cup \{(\vec{x}, c)\})$, and a new single classifier is trained on the combination of generated and original samples.

Comparing to Inner Ensembles and Limitations

This work can be considered related to Inner Ensembles because its output is comprehensible, and it uses ensemble methods to create new models. However, the ensemble methods are not used directly inside the algorithms, so in this sense it is different than Inner Ensembles. Although Domingos introduced this work as a general framework, it has only been applied on C4.5RULES. Another important limitation is that this method relies on generating artificial data, offline, and not during the training step of the algorithm. Finally, this method only works for supervised learning, while Inner Ensemble can be applied to both supervised and un-supervised learning methods.

From Ensemble Methods to Comprehensible Models

Ferri et al. (2002) argued that the comprehensibility of a model is an important characteristic, and though non-comprehensible methods can be useful for predictions, they do not provide insight about how the predictions are made, which is vital in many applications. Ensemble methods are usually used to improve accuracy, but they are not comprehensible. In this paper, Ferri produced a single hypothesis that is most similar to combined hypotheses, and called it *archetype or representative* of the ensemble. More specifically, a single hypothesis h_i from the ensemble E is selected, such that h_i is the most similar to

Algorithm 7 Combined Multiple Models (CMM).

Inputes:
S :the training set, **L**:the learning algorithm,
C : A procedure for combining models for example bagging.
m: the number of models to generate, **n** :the number of examples to generate.

m variations of S are generated: $S_1, ..., S_m$
Models $M_1, ..., M_m$ are produced by training of L to each S_i.

for j=1 to n **do**
  $\vec{x}$ is a randomly generated example.
  c is the class assigned to $\vec{x}$ by $C_{M_1,...,M_m}(\vec{x})$.
  $S = S \cup \{(\vec{x}, c)\}$.
end for
M is the model produced by applying L to S.
Return M.

the ensemble E. This means that the representative of the ensemble is selected from the ensemble members. Because a statistical problem arises when the amount of the training data is too small, the first step is to generate unlabeled datasets based on the training data. Then ensemble is used to label the randomly generated datasets. Each hypothesis that is a member of the ensemble is then compared to the ensemble with respect to the generated dataset, and a similarity measure and best hypothesis is returned as the solution.

In this work, the author used a special type of ensembles called *shared* ensemble. Shared ensemble is a multi-tree, which is a data structure that helps learn the ensembles of the trees that share some parts of their branches. This data structure is different than the forest, because forest is a collection of trees, while multi-tree combines the nodes of several trees at each level of the tree. For example, for a node to split it must have the nodes of several trees at that level combined by OR. This means that the node to split at each level of the tree is the node of the first tree, or the second tree, or... at that level. Thus, multi-tree combines different parts of several decision trees at each level into a single decision tree. The author used this type of ensemble because the number of hypotheses is exponential with respect to the number of iterations and, thus, there are more ensemble members from which the representative can be found. The main steps of

Algorithm 8 Shared Ensemble.

1: Multi-tree generation.
2: Generate random dataset.
3: Labelling the generated dataset using the Multi-tree.
4: Calculation and propagating of contingency matrices.
5: Selection of a solution.

the method based on the shared ensembles are shown in algorithm 8 (Ferri et al., 2002).

Line 4 of this algorithm is the main step. For each leaf of the multi-tree a matrix called a contingency matrix is calculated, using generated labelled datasets. Then the matrices are propagated from the leaves of the multi-tree to the root of the tree, using similarity measure to choose the best part of the multi-tree to be used as the part of the final single hypothesis. In this way, the best parts of the tree have been flagged, from the leaves to the root. An archetype hypothesis is extracted from the multi-tree, by descending the final contingency matrix at the root through the flagged nodes of the multi-tree (Line 5). For more details of each step refer to (Ferri et al., 2002).

Comparing to Inner Ensembles and Limitations

The only similarity between this work and Inner Ensembles is that their outputs are comprehensible. An important difference is that ensembles are not used in this method, even for growing the new methods. Instead, ensemble is used to find the best model that has already been generated and is a member of ensemble itself. In fact, unlike Inner Ensembles, this method does not use ensemble during the learning process of the single hypothesis. The method also has several limitations compared to Inner Ensembles, one of which is that it only works for decision trees, which means it is not applicable to other algorithms. In addition, similar to CMM, this method relies on generating artificial data to select the best single hypothesis.

Interpretable Single Model (ISM)

Van Assche and Blockeel (2008) mentioned the importance of comprehensibility of models, particularly in inductive logic programming (ILP). In ILP, comprehensibility is more important than in propositional learning. For the problems addressed by ILP algorithms, such as medical domains, the end-user cares not only about the accuracy of the model, but also the comprehensibility, and accepts the model only if it can explain why it makes

the decisions. Although ensemble methods usually increase the accuracy of the learner, they are not commonly used in ILP because they damage comprehensibility. He also discussed the problems of the previous methods that attempt to extract comprehensible models from the ensembles without excessively sacrificing accuracy.

The main problem of the previous methods is that they rely on generating artificial data. The author mentions that generating relational artificial data for ILP is not straightforward, because the data distributions are too complex. Instead, this method aims to learn a single interpretable model from an ensemble, without generating artificial data. Here the class distributions are estimated directly from the different models in the ensemble.

This work focuses on first order decision tree,TILDE, (Blockeel and De Raedt, 1998) which is similar to C4.5. For this tree induction method, first a set of candidate queries are generated using refinement operator ρ. Then, from the generated queries the optimal split procedure finds the best query by using a heuristic, such as information gain. The queries are tests in the internal nodes of the decision tree that are conjunctions of the first order literals (Van Assche and Blockeel, 2008). Now, assume the heuristic is information gain and the optimal split is needed to calculate according to the ensemble in a certain node of the new tree. And also assume that B is the conjunction of the test from the root of the tree to the node n, and T is a certain test. Then information gain for test T is calculated as:

$$IG(T|B) = entropy(B) - P(T|B)entropy(T,B) - P(\neg T|B)entropy(\neg T,B) \quad (2.15)$$

Where

$$entropy(A) = \sum_{i=1}^{c} -P(C_i|A)log_2 P(C_i|A) \quad (2.16)$$

c is the number of classes and C_i is the ith class. Then the distribution $P(C_i|A)$ needs to be estimated using ensemble methods. Suppose we select a decision tree DT_k from an ensemble E. It is possible to estimate $P_k(C_i|A)$ for this decision tree by propagating through the DT_k. Then we can get (Van Assche and Blockeel, 2008):

$$P_k(C_i|A) = \sum_{l_{kj} \in DT_k} P(C_i|Y_{k,j}, A)P(Y_{kj}|A) \quad (2.17)$$

where Y_{kj} is the conjunction of the tests from the root of the DT_k to the leaf l_{kj}. The probability is then estimated for each ensemble member, and the final probability,

which is an estimate of the probability distribution by the ensemble, is calculated by the average over the class probability estimates of the N tree in ensemble E. This estimated probability is then is used in equation 2.15 to find the best split for the new tree.

Comparing to Inner Ensembles and Limitations

There are two similarities between this work and Inner Ensembles. The first is that the output of both methods is comprehensible, and the second is that both methods use ensemble methods to grow a new model. However, this method does not use ensemble methods directly during the learning step. Instead, it first creates an ensemble of trees, and from that constructs a single tree, while with Inner Ensembles everything is done during the training step. Although this method tries to eliminate the need for generating artificial data, it still based on generating candidate queries. Another important limitation is that this method only works for decision tree in Inductive Logic Programming (ILP).

Bayesian Network Structure Ensemble Learning

Liu et al. (2007) introduced new learning of Bayesian networks using ensemble methods. Their method is based on subsampling the data. First they sample the dataset, and for each sample they learn a BN. Then they combine the networks to create a single network. This work is intended to avoid the local maximum in learning the network structure, and to learn a single network that is better than the networks created using the regular methods. They used a new dataset sampling technique called Root Nodes based Sample Decomposition (RNSD). This sampling method is based on the values of the root nodes (nodes that do not have parents) in the Bayesian network that is learned using the training set. Thus, the first step is to find the root nodes, then sample the dataset based on the values of these nodes. In the next step, for each sampling a Bayesian network is learned using the optimal reinsertion method (Andrew Moore, 2003). Finally, the resulting networks are integrated into a single network, using a two-step integration procedure. The first step is called intra group integration, for which, in the Bayesian networks $BN_1, ..., BN_L$ in the ensemble, a voting method calculates the probability of each edge, and the edges with a probability of less than 0.5 are pruned. The next step is refinement of the resulting network, in which another search or heuristic method with a Bayesian score function is used to refine the network that resulted from the previous step, by adding, deleting or reversing the arcs. The performance of this method has

been evaluated on three datasets, Alarm, Barley and Insur, using the BDeu score for the network structure. The results showed that the performance of this method is better than the performance of each single BN in the ensemble.

Comparing to Inner Ensembles and Limitations

This method cannot be considered Inner Ensembles, because networks are again created during the offline procedure rather than the training step, and then a single network is created using a special combining method. In addition, the final network is not created based on voting; instead, the network that was created using voting among the other networks is used as the starting point of another search to create the final network. Also, the main intent of this work was to create a network whose network quality is better than the original network when evaluated by BDeu, rather than creating a network that works similar to ensemble of networks.

2.2.3 Inner Ensembles

To the best of our knowledge, there are two pieces of work that can be considered Inner Ensembles (Prez et al., 2004; Geurts and Wehenkel, 2000). In these works, instead of using ensemble methods as parts of the classifiers, or as a guide to grow a new model, ensemble methods are used during the learning step. These methods do not rely on generating artificial data. The major drawback of these methods is that they are applicable to decision trees, but not to different types of algorithms.

Consolidated Trees Construction

In this section, we explain the *Consolidated Trees Construction* (CTC) algorithm (Prez et al., 2004), which improves the behavior of C4.5, and reduces the error and complexity of the induced tree. It uses ensemble methods to decide which attribute to split inside each node of the tree. The algorithm first generates subsamples, then generates a tree for each subsample, and at each step it combines the trees at each node. This procedure is similar to using ensemble methods for each decision making event in each node of a decision tree, since, ultimately, only a single decision tree will be generated. The authors' main goal was to maintain the explanation of the CTC algorithm. Indeed, the advantage of this algorithm over Bagging and Boosting is that the final classifier is based on a single tree, rather than groups of trees. To address the instability of the single decision tree, and the comprehensibility problem of ensembles of decision trees, Prez et al. (2004)

developed CTC, a new algorithm for building decision trees. This algorithm improves the performance of the decision tree while reducing the complexity, and maintains the explanation capacity of the algorithm.

Algorithm 9 shows the CTC algorithm, which is based on resampling techniques (Prez et al., 2004). First, different subsamples are generated based on the original data, then each training subsample is used to build a decision tree, though ultimately only one tree is built. In algorithm 9, N_S is the number of subsamples S^i, S is the training data, and LS^i is the set of subsamples. In the first loop of the algorithm, the best split, $(X, B)^i$, will be indicated for each subsample. X is the attribute to split, and B represents the branches to create the new subsamples. For example, if X is the attribute "Sex", then B is "m" and "f". Using a voting procedure with all Xs, the final variable to split, X_c, will be selected. Then, if X_c is numeric B_c is the median, and if X_c is discrete B_c is all possible values of X_c. After making decisions about X_c, all trees are forced to use X_c to make the split to generate new subsamples. These subsamples are then added to the set of subsamples LS^i, and the procedure is repeated until the set of subsamples LS^i is empty.

The resampling method used in this algorithm could be different types, and it could also vary according to size of the subsamples, with or without replacement. The performance of the algorithm depends significantly on the resampling mode, and there is potential for the algorithm to be extended. Since the resampling method can be any type, we can apply different methods, such as that used in Adaboost, to focus more on the hard part of the data. In this way, we can generate a single decision tree that uses Boosting in each node to select the best attribute to split. Another possible direction is to analyze the error using bias and variance decomposition, to determine how this method affects bias and variance. The authors of the paper mentioned this as a future research direction, and in the following chapters we will examine the potential. Different research that has been done on the CTC algorithm can be found in the literature: (Prez et al., 2007; Prez et al., 2006; Gurrutxaga et al., 2006b; Prez et al., 2005; Gurrutxaga et al., 2006a, 2007). In the following, we briefly discuss this research, because we believe we can perform similar analysis on our idea when we apply it to other algorithms.

Compare the performance of *CTC* and *CMM*: Papers by Gurrutxaga et al. (2006a, 2007) focused on comparing the performance of CTC with that of the Combined Multiple Models (CMM) algorithm. The performance of CTC and CMM for decision tree classifiers has been compared from three points of view: *accuracy*, *complexity* and

stability of explanation. According to Gurrutxaga et al. (2007), complexity is defined as how simple the given explanation is, and the error is the quality of the explanation. Complexity is measured by the number of internal nodes of the tree, and the structural stability is measured using the *common* parameter method (Gurrutxaga et al., 2007; Prez et al., 2006).

$$Common(T_{set}) = \frac{2}{m(m-1)} \sum_{k,l=0,k<l}^{m-1} Similarity(T_k, T_l) \tag{2.18}$$

In this equation, T_{set} is the set of trees with m trees, and *Similarity* counts the number of common nodes between two trees. Starting from the root of the tree, if two nodes have the same level and feature they are common nodes. Otherwise, if they are not common nodes, the sub tree under them has not been processed any further. Thus, the larger the common parameter, the more stable the trees in T_{set} are.

Experiments show that in terms of accuracy, the behaviour of the CTC and CMM algorithms is similar, though CTC is slightly better. As well, experiments based on statistical analysis confirmed that the difference between the performance of the Bagging and CTC algorithms is not statistically significant, but the differences between the performance of the Bagging, CMM and C4.5 is significant. Also, the comprehensibility of the CTC method has been measured with respect to complexity and stability (Gurrutxaga et al., 2007). The analysis of complexity showed that the trees that are obtained using CTC are much simpler than those obtained using CMM. The common parameter was higher for CTC than CMM for most of the datasets used in the experiments.

Different studies have been done on the CTC algorithm, including work to compare the structural convergence of CTC and C4.5 (Prez et al., 2006). The results showed that CTC converges into a single tree when the number of samples is larger than a certain number, and that it has smaller errors than C4.5. Thus CTC builds more stable trees, and the trees also have better explanation quality. Another study has been done on the effect of resampling and the number of subsamples on CTC by Gurrutxaga et al. (2006b), which shows that when the number of subsamples increases the error of CTC decreases, but the complexity of the tree increases. In addition, other work that investigated the effect of changing the distribution of the training data showed that using optimal distribution of the data enables more accurate classifiers to be built, and that the accuracy can be further improved if CTC is applied (Prez et al., 2007).

Comparing to Inner Ensembles and Limitations

This method can be considered as Inner Ensembles, since all the output is comprehensible and it uses ensemble methods during the learning step. However there is still difference between this work and Inner Ensembles. In this work, different decision trees are created, and then a new tree is created using voting to find the best node to split at each level of the tree. Conversely, with Inner Ensembles everything is done inside the learning algorithm. One important limitation of this work is that it does not introduce a framework to apply the method to any algorithm. As well, this method can be extended in several ways, including applying different ensembles as Inner Ensemble instead of using Bagging, and bringing ensemble methods inside the decision tree for each node.

Reduction of Discretization Variance in Decision Tree Induction

Geurts and Wehenkel (2000) focused on the variance introduced in the induction of the decision trees, stating that there are two different sources for the variance. The first is related to the need for discretizing continuous attributes by choosing thresholds. Numeric attributes in each node of the decision tree must be split into two parts, according to the threshold. In local discretization, these thresholds are determined on the subset of instances for each node of the decision tree. Since there are many nodes with a small number of instances, the variance of these thresholds is expected to be high.

Another source of the variance is related to the variability of the tree structure choosing the attribute to split inside the nodes of the decision tree. In fact Prez et al. (2004) focused on this source of variance, and used Inner Ensembles to reduce it. But here, this work mainly focuses on the first source of the variance: choosing the correct threshold for splitting the numerical attributes.

In the literature, there are two different ways to reduce the variance of the decision tree, pruning and ensemble methods (Geurts and Wehenkel, 2000). Pruning reduces the complexity and variance, but it increases the bias. Therefore, using pruning only improves the interpretability, and the accuracy of the learner slightly. Ensemble methods improve the accuracy significantly, but they destroy the comprehensibility of the learner. The general procedure to find a threshold is to calculate a measure (e.g. Shannon Information) based on different threshold values, and select the threshold value with the maximum measure. Geurts and Wehenkel (2000) proposed three different methods to reduce the variance of selecting of the threshold.

Smoothing In this method, before selecting the threshold value with the maximum measure, a moving average filter with a fixed window size is applied to the curve (measure vs different threshold value).

Averaging

1. The score curve and the optimal threshold are calculated. Assume the estimated score is C^*, and its variance is σ_{C^*}.
2. Then the smallest and the largest threshold values $\underline{a}_{th}$ and $\overline{a}_{th}$ for which the corresponding score is larger than $C^* - \lambda\sigma_{C^*}$ are calculated, where λ is a tunable parameter.
3. The final discretization threshold is calculated as $\overline{a}_{th} = (\underline{a}_{th} + \overline{a}_{th})/2$.

Bootstrap This method uses Inner Ensembles to find the threshold to split the numerical attributes. Although it is based on Inner Ensembles, it is different from our method because it only applies to decision trees, and it is used for datasets with numerical attributes. The procedure of finding the threshold using Inner Ensemble is as follows (Geurts and Wehenkel, 2000):

1. Draw by bootstrap L different samples of the data for which L is the ensemble size.
2. For each sample of the data, using the classical procedure determine L different threshold values.
3. The descritization threshold is calculated by averaging L different thresholds.

Comparing to Inner Ensembles and Limitations

This method is also considered Inner Ensemble, because the decision maker is the threshold value inside the node of the decision tree, and it is calculated inside each node using ensemble methods during the training of the decision tree. This work also results in a comprehensible model, which is another similarity to Inner Ensembles. However, this method only works for decision trees, while Inner Ensembles has the potential of being applicable to any type of method. Also, similar to Prez et al. (2004), this work can be extended to apply to different kinds of sampling or ensemble methods inside the node of the tree. In contrast to Prez et al. (2004), here the ensemble methods are used inside the node of the tree during the training step.

2.3 Characteristics of Regular and Inner Ensembles

In this section, we discuss the characteristics of the traditional ensemble methods that may be applied to Inner Ensembles as well. The first one is the effect of the ensemble methods on bias and variance of the error. We argue that using Ensemble methods for decision makings inside the algorithms, could affect the bias and variance of decisions similar to traditional ensembles. Thus in this part we explain the bias and variance decomposition and the effect of ensembles on them. In addition another characteristic of traditional ensemble is robustness to noise. We expect that using Inner Ensembles will have similar effect on noise. Therefore here in this part, we review the literatures about the performance of ensembles on noisy data.

2.3.1 Bias and Variance Decomposition

In the simplest case, a decision maker can be considered a classifier that returns true if it makes a correct decision, and false if it doesn't. Therefore, ensemble methods can work for the decision makers inside the algorithms as well. We argue that, for the same reasons that ensemble methods work, Inner Ensemble could result in more robust decision making inside the algorithms. More specifically, using ensembles for decision making could result in reducing the variance of different decisions, similar to Bagging. As the robustness of the ensembles can be explained based on bias and variance decomposition, we will explain bias and variance in this chapter. However, there are different decompositions of the errors based on bias and variance, such as (Kohavi and Wolpert, 1996; Breiman, 1996c; Tibshirani, 1996). As well, there is research in which the decomposition of the error is given for K-Means clustering (Telgarsky and Vattani, 2010). In this chapter, we explain one of the well-known methods by Domingos (2000).

Domingos Definition

According to Domingos, a training set is defined as $\{(x_1, t_1), ..., (x_n, t_n)\}$. The classifier that is trained with this training set is defined as $y = f(x)$, where y is the label predicted by the classifier. And the correct label of instance x is defined by t, and a general loss function is $L(t, y)$. The optimal prediction y_* is defined as the prediction that minimizes $E_t[L(t, y_*)]$. The error is $E_{D,t}[L(t, y)]$, where D is the set of training sets, and expectation is determined with respect to t and D. This is expected to decompose into three different entities: bias, variance and noise. In order to define bias and variance

for the arbitrary loss function, Domingos first defined the main prediction, and based on that he defined bias, variance and noise. According to Domingos, "the main prediction" for a loss function L and a set of training sets D is defined by:

$$y_m^{L,D} = argmin_{y'} E_D[L(y, y')] \tag{2.19}$$

and $y_m^{L,D}$ is represented by y_m.

Based on the definition of the main prediction, the bias of the learner on an example x is defined by:

$$B(x) = L(y_*, y_m) \tag{2.20}$$

and the variance of the learner on an example x is defined by :

$$V(x) = E_D[L(y_m, y)] \tag{2.21}$$

Finally the noise of an example x is defined by:

$$N(x) = E_t[L(t, y_*)] \tag{2.22}$$

According to these definitions, bias is independent of the training set, and variance is independent of the true class of the instance x. Also, noise only depends on the true class t and the optimal prediction y, which means that it is independent of the classification algorithm, and only depends on the problem. The following is the decomposition of the error to bias, variance and noise:

$$E_{D,t}[L(t, y)] = c_1 \times N(x) + B(x) + c_2 \times V(x) \tag{2.23}$$

In two class classification problems, this equation is valid for any real valued loss function for which the following conditions are satisfied:

$$\begin{aligned} \forall y, L(y, y) = 0 \\ \forall y_1 \neq y_2 L(y_1, y_2) \neq 0 \end{aligned} \tag{2.24}$$

$$c_1 = P_D(y = y_*) - \frac{L(y_*, y)}{L(y, y_*)} P_D(y \neq y_*) \tag{2.25}$$

$$c_2 = \begin{cases} 1, & y = y_* \\ -\frac{L(y_*,y_m)}{L(y_m,y_*)}, & Otherwise \end{cases} \tag{2.26}$$

for which $P_D(y = y_*)$ is the probability that the classifier predicts the same as the optimal classifier over all training sets. In multiclass classification problems, this equation is valid for a zero-one loss function for which the following conditions are satisfied:

$$c_1 = P_D(y = y_*) - P_D(y \neq y_*)P_t(y = t|y_* \neq t) \tag{2.27}$$

$$c_2 = \begin{cases} 1, & y_m = y_* \\ -P_D(y = y_*|y \neq y_m), & Otherwise \end{cases} \tag{2.28}$$

2.3.2 Robustness to Noise

One of the advantages of ensemble methods is their robustness to variation. For example, Bagging works by reducing the variance while keeping the bias unchanged. One of the sources of variation is the noise in the data, and some learning algorithms have a mechanism for handling noise. The problem of learning from noisy data is a major issue in the machine learning community (Zhu and Wu, 2004). Some researchers have suggested that handling the noise from the data before training the classifier will prevent noisy data from affecting the training (Gamberger et al., 2000). Here we review a number of experiments from the literature about the effect of noise on classifiers in general. We then review the work about the effect of noise on ensemble methods, particularly Bagging and Boosting.

For data, noise is defined as anything that deteriorates the correlation between features and class (Zhu and Wu, 2004). Different types of noise are defined in the literature. For example, Hickey (1996) classifies noise as (1) inappropriate or insufficient description for the attribute or class, (2) distortion of the attribute values in the training set or (3) misclassification of the training instances. However, the first type of noise is very difficult to define for real world datasets, because with these datasets finding a sufficient description for the attributes and classes can be problematic. In addition, even if it is possible, it would be very difficult to measure the sufficiency of the description. Thus noise is usually classified as attribute noise and class noise (Zhu and Wu, 2004).

There is much research in the machine learning field dealing with class and attribute noise that demonstrates the importance of learning in the presence of noise, and one of

the most cited papers is the work by Zhu and Wu (2004). In the case of class noise, it is suggested that eliminating some of the noisy instances can improve the accuracy (Brodley and Friedl, 1999; Gamberger et al., 2000; Zhu et al., 2003). However, handling the attribute noise is much more difficult (Zhu et al., 2004), because eliminating the instance with attribute noise will eliminate other attributes of that instance, and that could have a strong impact on classification. Zhu and Wu (2004) state that, though attribute noise is as important as class noise, less attention has been paid to it. They differentiated between attribute and class noise, and analysed the impact on system performance separately, focusing on the attribute noise. Abbasian et al. (2010) conducted experiments to determine which classifier is more robust to changes in the environment. The environmental changes were simulated by adding noise to decrease the influence of the most significant attributes on the class attribute.

Considering the importance of learning in the presence of noise, we need to know how robust ensemble methods are. Several papers address this, and compare the performance of ensemble methods in presence of both class noise and feature noise (Dietterich, 2000a; Maclin and Opitz, 1997; Melville et al., 2004). However, as far as we know there is no work that analyses this issue for ensemble methods in general. Instead, they compare the performance of the specific ensemble methods of Bagging and Boosting in the presence of noise.

Dietterich (2000a) argued that large ensembles might be able to overcome the problem of noise. The author considered the problem of class noise by conducting experiments on nine datasets with different levels of classification noise, and comparing the performance of Bagging and Boosting on C4.5. The results showed that Boosting is generally superior to Bagging and C4.5 in terms of accuracy when the noise level is 0%. However, after adding noise the accuracy of Boosting is less than that of Bagging and C4.5, and the accuracy of Bagging is higher than C4.5. The reason for this is that noise improves the diversity in Bagging, and damages the accuracy of Boosting. The poor performance of Boosting in the presence of noise can be explained by the very high weights that are applied to mislabelled instances. The improvement in diversity of Bagging is because there are some mislabelled training instances among the omitted instances in bootstrap, and their elimination causes large changes in the learned decision trees (Dietterich, 2000a).

Maclin and Opitz (1997) presented a comprehensive evaluation of the Bagging and Boosting algorithms for two base classifiers, neural network and decision tree. This work evaluated the hypothesis that Boosting might overfit due to the noisy instances. Two reasons are given for this, (1) Ada-Boost overemphasizes noisy instances for updating the

weights, and (2) with Ada-Boost, classifiers are combined using a weighted majority vote that could result in overfitting (Sollich and Krogh, 1996). Different levels of attribute noise were added to the datasets, and the results of Bagging, Arcing and Ada-Boost were compared. These results showed that the advantages of Bagging increase in the presence of noise, while the advantages are much smaller for the Boosting family algorithms, particularly Ada-Boost. This means that, though Boosting is still sensitive to attribute noise, it is not as sensitive to class noise.

Melville et al. (2004) argued that, in addition to good performance, ensemble methods are more robust in the presence of noise. The author compared the performances of the ensemble methods of Bagging, Boosting and DECORATE (Diverse Ensemble Creation by Oppositional Relabeling of Artificial Training Examples) (Melville and Mooney, 2003b) in the presence of feature noise and classification noise. The results showed that for classification noise, both Bagging and DECORATE outperform the base classifier, and in higher levels of noise Bagging performs slightly better than DECORATE. Boosting, however, was very sensitive to classification noise, which confirms the findings of other published studies. Boosting performance was significantly better than the base classifier when there was no noise, but in the presence of noise its performance decreased rapidly, and for higher levels of noise it performed significantly worse than the base learner. Thus, for classification noise, it is better to use Bagging or DECORATE, rather than AdaBoost. On the other hand, in the case of feature noise, Bagging, AdaBoost and DECORATE all outperform the base learner.

In conclusion, two different types of noise have been introduced in the literature, classification noise and feature noise, and a number of research papers have investigated the effect of these on classifiers in general. As multiple-classifier systems were expected to be more robust in the presence of noise, some researchers conducted experiments to explore the effect of noise on different ensemble methods, particularly Bagging and Boosting. The results showed that Bagging generally outperforms the base classifier in the presence of noise. A possible explanation for this is that Bagging uses bootstrap sampling, and a number of instances with noisy data could be among the omitted instances (Dietterich, 2000a), which could cause the increase in diversity. In noise-free situations, Boosting outperforms Bagging and the base classifier, but its performance degrades rapidly when the noise level increases, and it performs worse than Bagging and the base classifier in noisy situations. The sensitivity of Boosting to noise could be due to assigning too much weight to noisy instances. We expect that using Inner Ensemble will have similar effects on noise. Specifically, we believe that if we use Bagging for decisions inside the algorithm,

the resulting classifier will be more robust to noise, whereas if we use Boosting it will be sensitive to noise.

2.4 Summary

In this chapter, we positioned our work among related work with Inner Ensembles, by dividing it into four main groups. The first group considers regular ensembles, which we further divided into ensemble methods for classification, and cluster ensembles. For ensemble methods for classification we explained two basic algorithms, Bagging and Boosting, and for cluster ensemble we focused on two groups of cluster ensemble, Graph-Based and Hierarchical based methods. The second group considers the methods for using ensemble of decision makers inside the model, instead of the results of ensemble of decision makers made during the training step. An example of this group is the work by Zimmermann (2008). The third group considers methods that are similar to ensemble methods that use standard ensemble as a guide to growing a new model. Some examples of this group are (Van Assche and Blockeel, 2008; Domingos, 1998b; Ferri et al., 2002; Liu et al., 2007). The fourth group are methods that can be considered as Inner Ensembles; to the best of our knowledge there are the only two of these (Prez et al., 2004; Geurts and Wehenkel, 2000). We argued that, although the first group (regular ensembles) are applicable to different algorithms and do not rely on generating artificial data, their outputs are not interpretable, and they are not memory and time efficient. In addition to supervised learning we discussed cluster ensemble methods. We showed that they have several disadvantages, including lack of comprehensibility, excessive memory usage, and clustering time, that make it very difficult to apply them to online clustering. The problem with the second group is that they are not fully interpretable, and are not applicable to different types algorithm. The third group either rely on generating artificial data, which can be very complicated, or they are not applicable to some algorithms, such as supervised and unsupervised learning. Finally, although the last group are considered as Inner Ensembles, they only work for decision tree and are not extendable to other algorithms. Comparing Inner Ensembles to these algorithms showed that Inner Ensembles: 1) has the potential to be applied to any kind of algorithm, 2) maintains the comprehensibility of the model, 3) does not rely on generating artificial data or generating ensembles through offline processes, 4) is memory efficient, and 5) is test time efficient, which makes it particularly useful for online clustering. We also expect that using Inner Ensembles will have similar effect as traditional on noise and finally Inner

Ensemble could affect the bias and variance of decisions similar to traditional ensembles.

Algorithm 9 Consolidated Trees Construction Algorithm.

Inputes:
S: the Dataset, N_S: the number of subsamples.

N_S samples $S^1, ..., S^{N_S}$ are generated using some *Resampling Method*.
$LS^i = \{S^1, ..., S^{N_S}\}$.
repeat
  for each S^i in LS^i **do**
    $CurrentS^i$ = the first variation of data in LS^i.
    Remove $CurrentS^i$ from LS^i.
    Find the best split $(X, B)^i$ for subsample S^i.
  end for
  X_c is obtained by voting $X_1, ..., X_{N_S}$.
  if X_c is Numeric **then**
    B_c is the median of X_c
  else
    B_c is all possible values of X_c.
  end if
  if $(X_c, B_c) \neq Not - Split$ **then**
    for i= to N_S **do**
      Divide $CurrentS^i$ is divided based on (X_c, B_c) to get new subsamples.
      $\{S_1^i, ..., S_n^i\}$.
      Add new subsamples $\{S_1^i, ..., S_n^i\}$ to LS^i.
    end for
  end if
until LS^i is empty

Chapter 3

Inner Ensemble Bayesian network

3.1 Introduction

In this chapter, we apply Inner Ensembles to supervised learning. We decided to use the Bayesian network algorithm, due to its modeling power (comprehensibility), reasoning ability and accurate results. Real world problems are often so complex that they are very difficult for humans to comprehend and analyze. An important characteristic of the Bayesian networks is its ability to model a problem so that humans can understand it (Vomlel, 2002). The key strength of this graphical model is not only that it helps humans comprehend the domain model, it also enables "efficient uncertainty reasoning with hundreds of variables" (Vomlel, 2002). Bayesian networks has been used in diverse areas, including medical diagnosis (Pourret et al., 2008), expert systems (Desmedt, 1989), and pathfinders (Heckerman and Nathwani, 1992). In addition to accuracy, in domains such as medical diagnosis it is important for the user to be able to inspect the learned hypothesis, in order to trust the performance of the model (Wolberg et al., 1994). Thus, it is essential to have a model that is comprehensible and accurate.

Using ensemble of Bayesian networks also has several disadvantages. First of all, though it is accurate, the comprehensibility of the network would clearly be affected. We illustrate this point using the example of Bayesian network of two diseases which is shown in Figure 3.1. For example, the network shows that *Alcohol Abuse* and *Obesity* are risk factors for *Disease A*. *Disease A* itself can produce *Chest Pain*, and it is linked to *Disease B*. When using an ensemble of Bayesian networks, we would have many networks with different numbers of arcs which could represent quite different relationships, and this model would be difficult for a clinician to understand. On the other hand, though a single

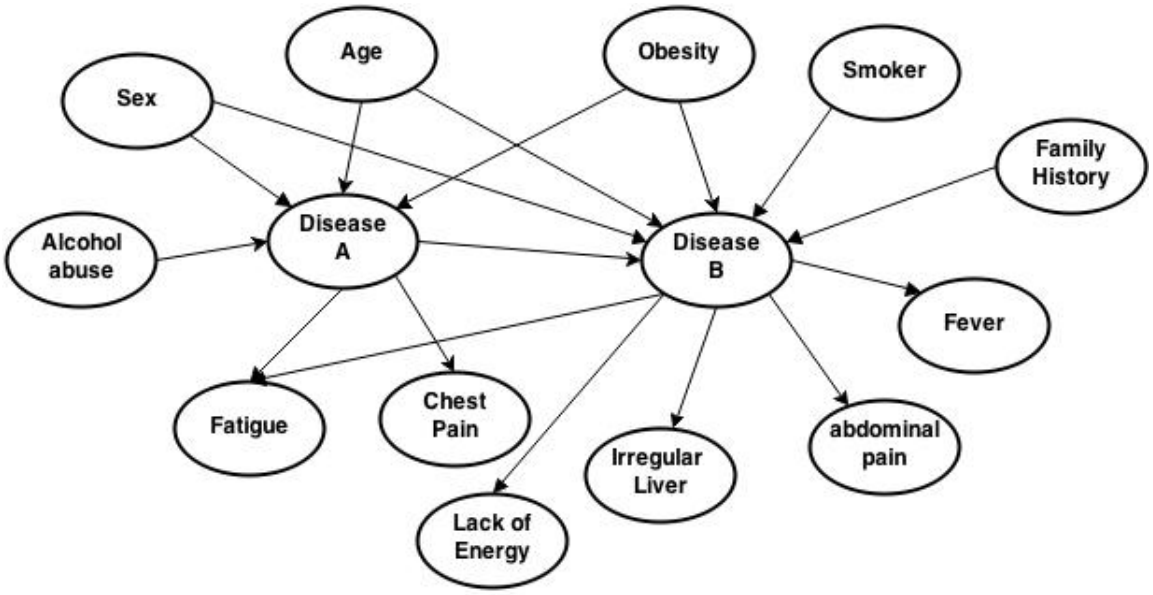


Figure 3.1: An example of a Bayesian network for two diseases.

Bayesian network is comprehensible, it might not be accurate. As well, classification time is an important factor in supervised learning, and when using ensemble of classifiers, increasing the number of members increases the classification time if the calculations are not done in parallel. In many online applications, the speed of determining class is an important practical consideration using the ensemble methods means we need to find the membership for each member of the ensemble, therefore the classification speed for the ensemble will be much lower than that of a single classifier. Another disadvantage of using ensemble of Bayesian networks is that a large amount of memory is required to store each network. This is crucial, particularly when the networks have very large numbers of variables and arcs.

There are two training steps for Bayesian network: training the structure of the network, conditional probability tables. In this chapter, we apply Inner Ensemble for training the structure of the Bayesian network. We believe our method achieves some of the performance improvement of the ensemble method, and maintains the comprehensibility, fast classification and smaller memory footprint of the original Bayesian network. However, we do not expect that applying Inner Ensemble on the Bayesian networks will give us accuracy similar to ensembles. This is because, first, some of the classification performance of ensembles is traded off against comprehensibility, classification performance and small memory usage. Secondly, ensembles improve the accuracy by working

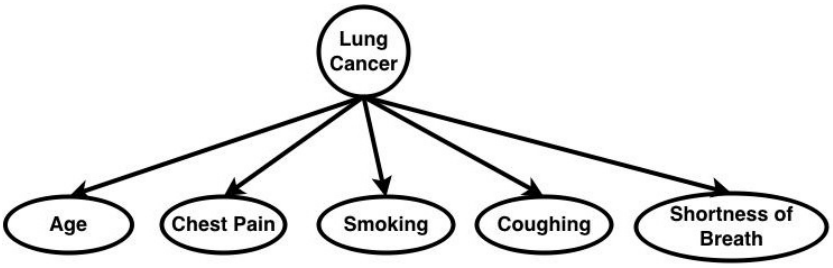


Figure 3.2: An example of Naive Bayes Network for Lung Cancer.

on the classification component of the Bayesian network, while we apply Inner Ensemble only to the structure of the Bayesian network, and the classification component is the same as the original network. However, we expect improved accuracy compared to the original network, due to better network structure.

3.2 Introduction to Bayesian networks

The Bayesian rule is defined as:

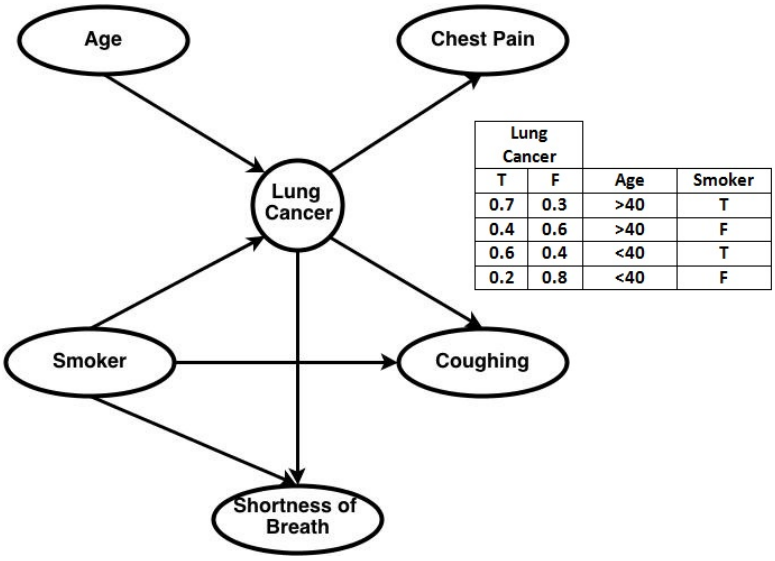
$$P(Y|X_1, X_2, ..., X_N) = \frac{P(X_1, X_2, ..., X_N|Y)P(Y)}{P(X_1, X_2, ..., X_N)} \tag{3.1}$$

For classification, Y represents the class, and $X_1, X_2, ..., X_N$ represent N features of the dataset. The main problem is how to calculate the likelihood, $P(X_1, X_2, ..., X_N|Y)$. In order to calculate the likelihood accurately we need many data instances. To make it easier, naive Bayes classifier makes a significant assumption that all of the attributes X_i are independent given the class value. Using this assumption the likelihood is calculated as:

$$P(X_1, X_2, ..., X_N|Y) = \prod_{i=1}^{N} P(X_i|Y) \tag{3.2}$$

However, this assumption may not be realistic. For example, assume that for lung cancer there are several related factors, such as smoking, age, chest pain and coughing. A sample naive Bayes network for this example is shown in Figure 3.2. According to the figure, among the factors there are some correlations that is not realistic to ignore; the correlation between smoking, coughing and chest pain for instance.

To solve this problem, unlike naive Bayes, which considers all the features to be independent with no correlation between them, Bayesian network assumes there are



Lung Cancer			
T	F	Age	Smoker
0.7	0.3	>40	T
0.4	0.6	>40	F
0.6	0.4	<40	T
0.2	0.8	<40	F

Figure 3.3: An example of a Bayesian network for Lung Cancer. The six ovals represent a set of conditional independence assumptions. Each node of the network has a probability distribution table (only the table for Lung Cancer is shown).

correlations between some features. More specifically, for Bayesian network each feature is a node in the network, and for each node there are two types of information available: the arc between two nodes of the network which shows the correlation between the variables, and the conditional probability table that is given for each variable (Friedman et al., 1997).

Figure 3.3 shows the Bayesian network version of the example in Figure 3.2, which is related to lung cancer. According to the figure, lung cancer depends on age, and if the person is a smoker. Also, lung cancer and smoking will cause coughing and shortness of breath, and lung cancer causes chest pain. In the figure, only the conditional probability table of one of the variables of lung cancer is shown. Generally, given the values of other remaining variables, the probability distribution of any subset of variables can be inferred using a Bayesian network.

3.2.1 Learning Bayesian network Structure

There are two different problems associated with Bayesian networks. One is how to learn the structure of the network, and the other is how to learn the entries in the conditional probability tables. Estimating the conditional probability table entries is

straightforward when all the variables are observable, but when some variables are not observable estimating the entries of these tables is similar to training the weights of the hidden layers of neural networks (Mitchell, 1997). In this work, we focus on learning the structure of the Bayesian network, which is not an easy task, and has been shown to be a NP-hard problem (Chickering, 1996). Consequently, the standard methods for learning the structure of Bayesian networks become heuristic searches, based on some score functions over some search space. In this subsection, we focus on the search methods for learning the structure, and in the next we focus on some popular score functions.

Our work is based on complete datasets, meaning we assume there are no hidden variables or missing data, we focus on these methods. One of the earliest algorithms for learning the structure of the Bayesian network is K2, which was introduced by Cooper and Herskovits (1992). K2 is a hill-climbing algorithm to find the best network among the space of all networks. It is based on the metrics to determine the better of two networks, and a given ordering of nodes to determine the sequence in which they are processed. The nodes of the network are the attributes of the dataset.

The algorithm starts with no parents for each node. The details of K2 is shown in Algorithm 10. π_i is the set of parents of the ith node. At each step (line 4), one parent is added to the node, and the score $g(i, \pi_i \cup \{z\})$, of the network structure is calculated (line 5). $Prec(x_i)$ denotes the nodes that precede node x_i in the ordered list. The parent that increases the score the most is added to the node's list of parents π_i. Adding parents to the node stops if the addition does not increase the score of the structure (lines 7-11). Upon completion, the algorithm produces a final structure. In this algorithm, g can be replaced by different score metrics. For example, we can use Bayesian metrics (Cooper and Herskovits, 1992) as a local score or cross-validation as a global score. In this work, we focus on leave-one-out cross validation (LOO) as a global metric, which is shown in Algorithm 11.

Another method to learn the structure of the Bayesian network is by using a local search over the network, which was introduced by Chickering et al. (1995). This algorithm searches among all possible networks by using local modifications to the network. The method starts with either a network that has no arcs, one with random arcs, or even a network that has been found using other algorithms. Then, for each pair of nodes in the network there are three possible modifications: adding an arc, removing an existing arc, or changing the direction of an arc if it does not cause any cycling in the network . At each iteration after each modification to the network a score function is calculated, and the modification that results in the highest score is applied to the network. To

Algorithm 10 K2 algorithm

Input: A set of n nodes, An ordering of the nodes, u maximum number of parents for each node, A database D containing N instances.
for i=1 to n **do**
  $\pi_i = \phi$, $P_{old} = g(i, \pi_i)$, OKTOProceed = true.
  while OKTOProceed **and** $|\pi_i| < u$ **do**
    z is the node in $Pred(x_i)$ - π_i that maximizes $g(i, \pi_i \cup \{z\})$
    $P_{new} = g(i, \pi_i \cup \{z\})$
    if $P_{new} > P_{old}$ **then**
      $P_{old} = P_{new}$, $\pi_i = \pi_i \cup \{z\}$
    else
      OKTOProceed = false
    end if
  end while
  write (Node: x_i, Parents of this node: π_i)
end for

Algorithm 11 LOO global score "g" for K2 algorithm.

$Acc = 0$.
$D = D_1, ...D_N$. {Instances}
for $i = 1$ to N **do**
  Estimate conditional probability for network using existing structure and D-D_i.
  $Acc = Acc + PredictAccuracy(D_i)$
end for
score is: $\frac{Acc}{N}$.

avoid being trapped in local maxima, iterated hill-climbing can be applied such that the algorithm is applied on a network, the network is randomly perturbed, and then the algorithm applied on the perturbed network.

Learning the Bayesian network structure for a massive dataset was introduced by Friedman et al. (1999b). To address the issue of a search through the possible networks, particularly for large datasets, being a time consuming task that requires searching and examining unreasonable network candidates. The speed of the algorithm can be significantly increased by limiting the search space and using limited candidate parents for each node. This algorithm is based on two steps. The first is the *Restrict* step, in which reasonable candidate parents for each node are selected using mutual information-based measure. The next is the *Maximize* step, in which the structure of the network is learned using the traditional hill-climbing method, considering the candidate parents in the restricted step, and also verifying the validity of the network. These two steps are repeated

until convergence is reached. The stop criteria for this algorithm can be score-based, which is when the score no longer changes, or at a specific number of iterations.

Andrew Moore (2003) introduced a new search method, called the Optimal Reinsertion algorithm, to learn the structure of the Bayesian network, with the main intent of learning the network faster. At each step of the algorithm, one node of the network is removed with all of the in and out arcs. Then the algorithm, which is based on a search called OR Search, is applied to find the optimal set of in and out arcs, considering some constraints. These newly found in-and-out arcs are considered to be the arcs of the removed node. The node, and the new in-and-out arcs, are then be added to the network again. The algorithm continues to remove and reinsert nodes until the final network does not change. The results show that this algorithm learns the network faster than the typical hill-climbing method, and in variety of datasets.

Teyssier and Koller (2005) introduced a simple and efficient way to learn the network, based on the ordering of the nodes. It focused on the fact that most methods to learn the network are very complicated and difficult to implement, or they simply do not work better than the traditional hill-climbing method. The work assumes that the best network for a given ordering of the nodes of the tree can be found efficiently, and that the optimal network can be identified using the optimal ordering. Thus, instead of searching in the space of the network, the search is applied to the space of different ordering of the nodes. To search the order space, the hill-climbing algorithm is used, with a swap operator to exchange the nodes of the given ordering, and a score function for returning the order of the node with the highest score. The search is applied for different random restarts.

Another method for learning a network is to use the evolutionary algorithm.Carvalho (2011) uses a special kind of genetic algorithm, called the cooperative co-evolutionary genetic algorithm, (CCGA) to learn a Bayesian network. For this kind of genetic algorithm the problem is divided into sub-problems, and each sub-problem is solved to generate the final problem. In the case of Bayesian network structure, it divides the problem into the ordering of the node, which is called permutation subpopulation, and the binary matrix, which is called binary subpopulation. For each subpopulation, standard GA is applied, and the final solution is obtained by combining the solutions of the two populations. Cyclic crossover and swap mutation are used for the permutation subpopulation, and two-point crossover and bit-flip mutation for the binary subpopulation. The results of this algorithm have been compare to the K2 algorithm on six different datasets, and the results show a better average network score for all the datasets.

There are many other methods for learning the structure of the BN, such as (Singh and Valtorta, 1995; Goldenberg and Moore, 2004). To keep our discussion concise we do not explain them here. For details of these methods refer to (Daly et al., 2011).

3.2.2 Ensembles of Bayesian networks

The focus of this chapter is applying Inner Ensembles for the structure of the Bayesian network, thus it is related to ensemble methods and network structure. In the previous sections we reviewed the methods for creating the network structure. To complete the discussion, in this section we include a literature review of the ensemble of the Bayesian network, and explain the differences compared to Inner Ensembles. Ensemble methods have been applied on Bayesian networks to improve performance, and some result in a single network and others in an ensemble of networks. Inner Ensembles is a way to improve the performance of the Bayesian network, with key advantages over the ensemble of Bayesian network: they maintain the comprehensibility of the network and, more importantly, they follow a principle that can apply to other algorithms as well.

Friedman et al. (1999a) estimated the confidence of the features of the Bayesian network using bootstrapping methods. The features could be: 1) the confidence of the existing edge between two nodes, 2) the confidence of the ordering relations between the edges (edge X before edge Y in ordering), or 3) the robustness of Markov Blanket[1] relations. In this work, two different bootstrapping methods are used: non-parametric, for which n BNs have been learned by sampling with replacements from the original dataset, and parametric, for which n BNs have been learned by n datasets sampled from another network B learned by the original dataset. These datasets are generated using sampling according to the distributions given by the Bayesian network. After each network has been learned, the confidence of each feature is calculated by averaging all the networks, then high confidence features are used as constraints to learn a better network.

To summarize, Friedman et al. (1999a) used ensembles to measure the confidence of some features of the network, and applies these confident measures to build another network. The method cannot be considered Inner Ensembles, because the ensemble is not used during the training, or for creating the network. Instead, it first creates BNs during the offline step, then finds the confidence of the features. More importantly, the method does not create a network to combine the networks in the ensemble of BNs into

[1]The Markov Blanket of a node in Bayesian network consists of its parents, children and its childrens parent.

a comprehensible network. A similar work by the same author estimates the probability features, such as if node X can be the parent of node Y, for a small dataset that uses sampling over the space of node orders, and then calculates the probability of a feature, given a node order (Friedman and Koller, 2003). Here, the aim of Inner Ensembles is not to build a better solution than this method, but to provide a principle to improve the performance and maintain the comprehensibility, with the advantage of being applicable to other algorithms.

Utz (2010) created an ensemble of Bayesian network inspired by the random forest technique. n components of the ensemble are trained by sampling the variables and data instances, and simple voting is applied to classify an instance, as with Bagging. Then, for aggregate views of the relationships between the variables for all the components of the network, the network is aggregated using a method called composite structure. This is a undirected graph with the variables as the vertex, and the weight of the edges proportionate to the number of times that particular edge appears in the components of the ensemble.

Comparing this method with Inner Ensembles, it is obvious that it is not the same. Though it aggregates all the networks into a single graph to gain a better understanding of the relationships between features, the graph is not a network because it is undirected and weighted. In addition, instead of using the final aggregated network to classify the instances, the classification is done by simple voting of each component of the ensemble method. This means that this method is an regular ensemble, and the accuracy is not calculated using the graph that shows the relationships among the features. With Inner Ensembles, the accuracy is calculated based on a comprehensible network, not on different networks. In addition, although this method can be applied to other algorithms for classification, the final models cannot be combined in the same way as other methods. Thus, another important advantage of Inner Ensembles is its applicability to other algorithms.

Other related work on Bayesian network ensembles includes that by Rosset and Segal (2002), in which the Boosting method is used for the density estimation, and, since Bayesian network is a graphical model for describing the joint probability distributions over the random variables, they used it as a weak classifier. A similar work by Jing et al. (2008) used Boosting on the Bayesian network to combine the discriminative data-weighting with generative training of the Bayesian network parameters. They interpreted their ensemble as a graphical model consisting of a collection of Bayesian networks. Garg et al. (2002) used Bayesian network as a framework to combine a number of simple

Algorithm 12 General guidelines for using Inner Ensembles

1: **Locate a decision maker inside the algorithm.**
2: **Find a measure based on which that decision maker works.**
3: **Indicate the input and output of the measure.**
4: **Apply the ensemble on the measure:**
5: Change the input of the measure in different ways. {It can be sampling of data or feature based modifications or other methods.}
6: Generate an output based on each input.
7: Combine the outputs
8: **Apply the output of the ensemble for decision making.**

classifiers, to show that classification can be improved by combining simple classifiers linearly. None of the above methods can be considered Inner Ensembles. The first and the second are regular ensemble methods, and the other used Bayesian network to combine other classifiers. Here, we now explain how to use Inner Ensembles to train the structure of the Bayesian network.

3.3 Applying Inner Ensembles to K2

There are two problems with Bayesian network: the first is learning the network, and the second is learning the probabilities. In this work, we focus on the first problem. An important decision for the K2 algorithm is deciding if a node can be the parent of another node. This decision is based on Leave One Out Cross-Validation (LOOCV).

The accuracy is calculated at each step in the K2 algorithm, using LOOCV before and after adding a parent to a node. Yet small increases in accuracy may be due to an outlier and not because two variables are dependent. Thus, we need to make a more robust decision about adding a parent to a node. To make following the general guideline easier, here we repeat it again in Algorithm 12. Following the general guideline, we apply Inner Ensemble to the K2 algorithm step by step.

First, we locate a decision maker. According to Algorithm 10, it decides if a node can be added as a parent. The next step is to find a scoring function for this decision maker, $g(i, \pi_i \cup \{z\})$. For this work, the scoring function for the K2 algorithm is the global scoring function, which works based on LOOCV. The global scoring function g(line 3 Algorithm 10) is calculated as shown in Algorithm11. Here, for LOOCV, the algorithm extracts one instance for validation, with the rest of the data forming the training set

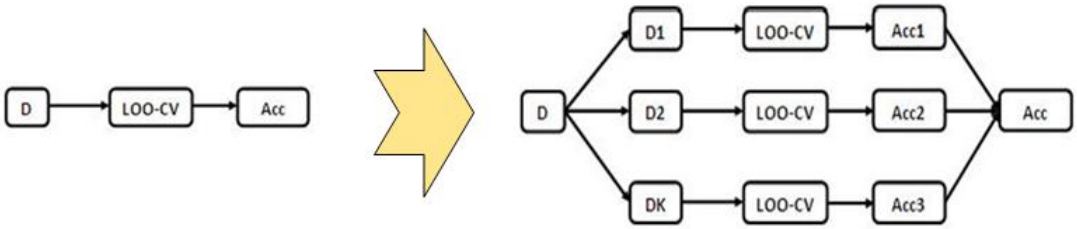


Figure 3.4: Applying Ensemble Method on LOOCV.

(line 4). At each iteration, the network is built using the training set and tested on the single instance. The final score of the network is the average of the scores across all splits of the data (lines 5,7). The next step, according to the general guideline (Algorithm 12), is to indicate the input and the output of the scoring function; the input is the training data and the output is the score. As mentioned, even a small increase in accuracy could result in adding a node as the parent of another node . Thus, for a more robust decision, instead of calculating the measure on one dataset, we calculate it on different samples of the dataset. Therefore, according to line 5 of the general guideline, we change the input of the measure by generating E samplings of the data, for which E is the ensemble size.

For each of the E samplings, the output of the LOOCV global score is calculated (line 6 algorithm 1), and the E outputs are combined by averaging (line 7), as in Figure 3.4 Using this procedure, the global score is redefined as shown in Algorithm 13, and we call this the $g_{Ensemble}$ score. Sampling of the data in algorithm 13 can be any type of sampling, with or without replacement, and it can also be a different size with respect to the original dataset. Depending on the type of sampling, we will have different ensemble methods. For example, sampling with replacement and a sample size of 100% gives us Bagging. Once the best structure has been selected, the algorithm proceeds in the conventional manner, calculating the conditional probability tables necessary for the complete Bayesian network. Our method has several interesting characteristics: it applies ensembles during the training step, the output is a single comprehensible network, and it uses Ensemble inside the algorithm to generate models. These indicate our method is an example of using Inner Ensembles.

Algorithm 13 Ensemble LOO global score "$g_{Ensemble}$" for K2 algorithm.

1: E {Ensemble Size}
2: $D = D_1, ...D_N$. {Instances}
3: $FinalScore = 0$.
4: **for** $k = 1$ to E **do**
5: $\quad D_S = Sample(D, Size)$ $\{D_S = D_{S1}, ...D_{SSize}\}$
6: $\quad FinalScore = FinalScore + LOOScore(D_S)$.
7: **end for**
8: score is: $\frac{FinalScore}{E}$

3.4 Experimental Results

In this section, we run several experiments with Inner Ensemble Bayesian Network (IEBN). The evaluation of IEBN is based on accuracy, quality of the network, size of the network and classification time. There are several measures to evaluate the quality of the network. Therefore, in the following sub-sections, we first briefly explain the experimental setup used for evaluating the performance of IEBN and measures that were used.

3.4.1 Experimental Setup

In this section, we explain how we run the experiments, and how we compare the performance of IEBN with the original network and different ensemble methods. Since we focus on the training of the structure of the Bayesian network, we do not expect to get as much accuracy improvement as the ensemble of the Bayesian network over the original network. This is because ensemble of the Bayesian network works on the classification, while IEKM works on the structure of the network. However, because we expect the quality of the network to be higher than the original network there will be improvements in accuracy over the original network, since the learned network represents the data better than the original network. We also expect that IEBN will improve the accuracy when ensemble methods improves it. This is because the metrics for learning the network is LOOCV, which is based on the accuracy. Then, if the ensemble method is able to improve the accuracy over a dataset, Inner Ensemble is expected to be able to improve the metric which is based on the accuracy, and as a result possibly improve the performance on the dataset. Here, first we compare IEKM with ensemble methods in

Table 3.1: Description of experimental datasets for Inner Ensemble K2.

Dataset	Num of Instances	Num of Features	Num of Classes
Breast-Cancer	286	10	2
Breast-w	699	10	2
Credit-a	690	16	2
Ecoli	336	8	8
Glass	214	10	7
Heart-c	303	14	2
Heart-statlog	270	14	2
Hepatitis	155	20	2
Iris	150	5	3
Labor	57	17	2
Liver	345	7	2
Lymph	148	19	4
Pima	768	9	2
Tic-tac-toe	958	10	2
Vote	435	17	2

terms of accuracy. More specifically we tested the following hypotheses:

- The accuracy of IEBN is generally higher than the original network, because with IEBN we expect to have a network that represents the data better than the one for the original network.
- Whenever ensemble methods improve accuracy, IEBN improves accuracy as well.

We also compare the quality of the networks generated by IEBN with the ones generated by BN in terms of BDeu and MDL. In compare to the original network, we tested the following hypotheses:

- In the original network, some of the arcs may not represent the relation between two attributes with enough confidence. For those cases, we expect that the majority vote is not given by the ensemble members. Therefore, we generally expect that for IEBN the size of the network in terms of the number of arcs is smaller than the original network.

- The quality of the network is higher than the original network which means the IEBN, represents the relationships among the attributes better than the original network.

For this experiment, we used 15 datasets from the UCI repository (A. Asuncion, 2007); the characteristics of the datasets are shown in Table 3.1. For all the datasets we used stratified 10-fold cross-validation, except for Ecoli, for which we used 3-fold cross-validation to ensure we have all the classes in all of the folds. In addition, for the smaller datasets such as Labor and Iris we repeated the experiments 10 times, to reduce the variance. We compare our method with two famous ensemble methods, Bagging and Boosting, in terms of accuracy and classification time. We analyzed the performance of IEBN based on these different parameters:

- *Ensemble Size:* 10 different ensemble sizes 10-100.
- *Sample Size:* 9 different sampling sizes 10%-90%.
- *Resampling Mode: without* replacement.

In our experiments we used nested cross-validation on each dataset. Inner cross-validation (5-fold cross-validation) works on the training set to find the best parameters, and these parameters are then used to train a model using the training set. Finally, the learned model is used to classify the test set using outer cross-validation. This procedure is depicted in figure 3.5. The procedure involves running each dataset $10 \times 90 \times 5 = 4500$ times, of which 10 is the 10-fold outer cross-validation, 90 is the number of parameters (10 ensemble sizes 10-100 $\times$ 9 different sample sizes) and 5 is the 5-fold inner cross-validation.

The quality of the learned network is assessed using BDeu and MDL scoring functions, which are also shown in Figure 3.5. To measure the size of the network, we count the number of parents that are connected to each node of the network. If the dataset has N attributes, and A_i is the number of parents that are connected to the ith attribute, then the total number of arcs for the network is calculated by the following equation.

$$Arcs = \sum_{i=1}^{N} A_i \tag{3.3}$$

In addition, for all of the experiments we calculate the percentage of changes in the network size when we use the IEBN. This percentage is calculated as follows:

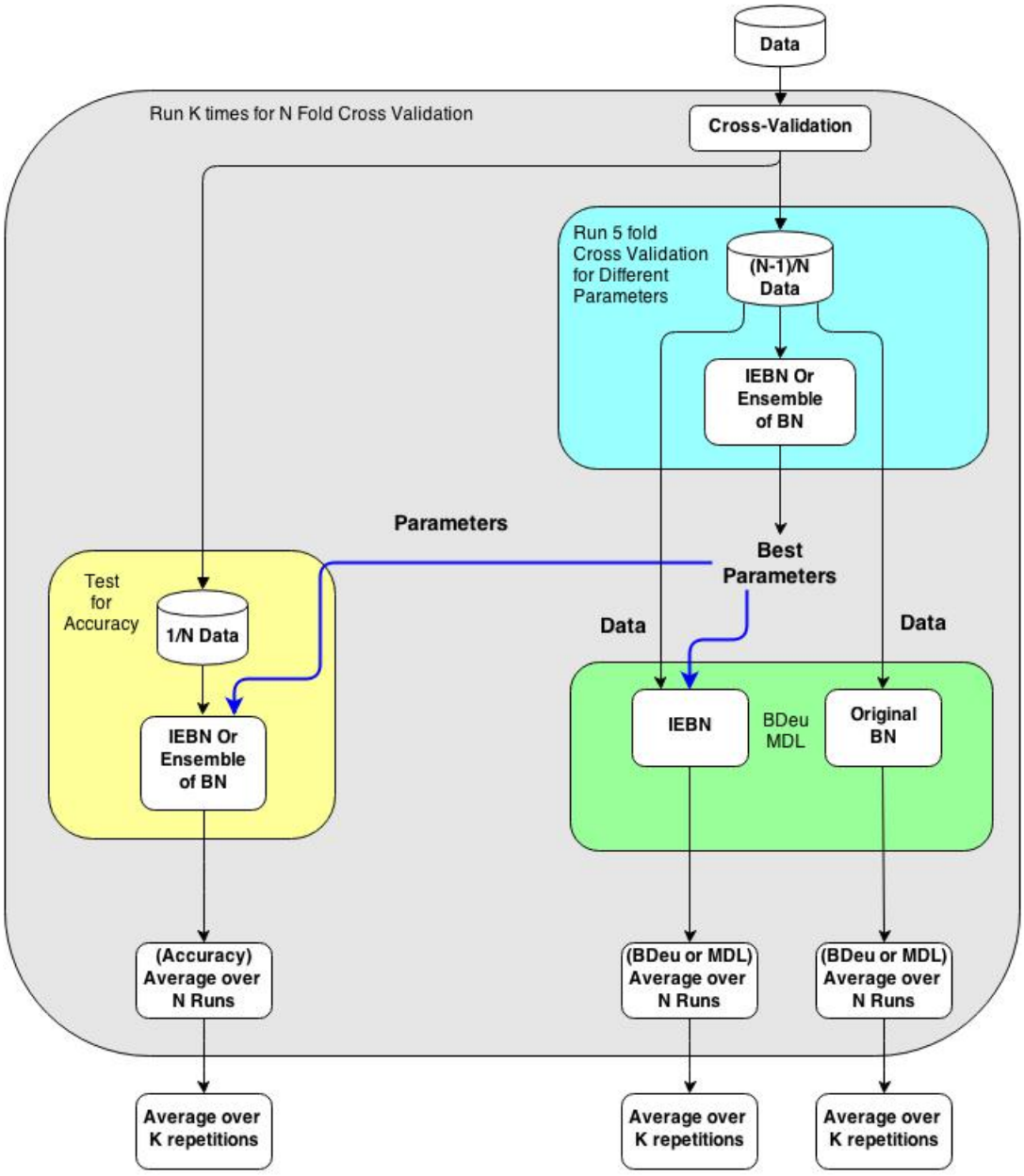


Figure 3.5: Procedure for finding the best parameters for Inner Ensemble Bayesian network (IEBN) and Bagging.

$$P = (\frac{Arcs_{BN} - Arcs_{IEBN}}{Arcs_{BN}}) \times 100 \tag{3.4}$$

We also compare the classification time of IEBN with those of Bagging and Boosting, and investigate the effect of ensemble size on the classification time of IEBN, Bagging and Boosting. In all the experiments, Bagging is represented by the BaGging Bayesian network (BGBN), and Boosting is represented by the BOosting Bayesian network (BOBN). Finally, for all the experiments, we used Friedmans test with $alpha = 0.05$, and for post-hoc tests we use Nemenyis test with $alpha = 0.05$. We then use the following procedure to evaluate our method.

- We compare the performance of our method with the original method and two ensemble methods Bagging and Boosting in terms of accuracy and classification time. This is done in sections 3.4.2, 3.4.3.
- In section 3.4.4 we compare the effect of ensemble size on classification time of IEBN with Bagging and Boosting.
- In section 3.4.5 we compare the quality of the original learned network with the one learned by IEBN in terms of BDeu, MDL and average network size.

3.4.2 Compare with Bagging

In this section, we compare the performance of IEBN with Bagging and the original BN, in terms of accuracy, network size and classification time. Table 3.2 presents the results of this experiment. The table shows the comparison of the accuracy and classification time, over the number of folds for each dataset. As we discussed in Chapter 1, the assumption in all our experiments is that Inner Ensembles works on datasets for which regular ensembles can improve the performance. Indeed, datasets whose performance cannot be improved by regular ensembles are not good choices for Inner Ensembles. Therefore, we use datasets that have even minimally improved performance due to ensemble methods, and remove those for which ensemble methods decreases the performance, or with performance that is not improved compared to the regular method. In the table, we remove the datasets for which Bagging and IEBN do not improve the performance; these are highlighted in red. The average performances are shown for all the datasets, excluding those that are highlighted.

Table 3.2: Comparison of IEBN with original Bayesian network and Bagging in Terms of Accuracy and Classification Time

	BN		**IEBN**		**BGBN**	
Dataset	**Acc**	**Time**	**Acc**	**Time**	**Acc**	**Time**
Breast-Cancer	69.21	0	70.57	0	70.99	73
Breast-w	96.71	4	97	2	97	105
Credit-a	**85.8**	**2**	**85.51**	**2**	**85.8**	**256**
Ecoli	81.55	3	82.44	3	86.31	115
Glass	71.03	3	69.63	3	73.36	152
Heart-C	79.85	1	80.86	3	82.84	208
Heart-S	79.63	1	81.11	3	82.59	55
Hepatitis	79.33	1	80.66	1	83.87	55
Iris	92.86	0.6	93.07	1.1	94.33	15.8
Labor	89.3	0.3	89.43	0.6	92.11	21
Liver	59.42	0	59.42	1	64.35	43
Lymph	83.04	5	83.9	4	85.81	115
Pima	75.65	2	76.05	3	75.91	129
Tic-tac	92.9	7	93.32	2	97.7	221
Vote	95.86	1	96.09	2	96.09	78
Average on All Datasets	82.14	2.06	82.60	2.05	84.6	109.45
After Removing Datasets						
Average	81.88	2.06	82.4	2.05	84.52	98.99
(Accuracy)						
Average Rank	2.89		1.96		1.14	
Q-Value	2.46				2.17	
(Time)						
Average Rank		2.64		2.36		1
Q-Value		0.76				3.59

As seen in the table, the results of Friedman's test and Nemenyis q-value comparison of each method with IEBN, for accuracy, number of arcs in the network and classification time, is also included. Because we used Friedmans test, we worked on the average of all of the datasets, not single dataset values. Thus, this test does not consider the significance of each dataset, and it is not based on the variances. Therefore, we did not include the variances for each dataset in Table 3.2. Also we run t-test for each single dataset and we did not get significant results on each datasets. However, overall on average on all of the datasets, our results is significant according to Friedmans test. According to the table, the rank of IEBN in terms of accuracy is 1.96, which is second to Bagging. Also, the critical value for Nemenyis test for three models is 2.34. Thus, we find that the q-value for comparing IEBN with BN is 2.46,somewhat higher than 2.34. This means that on average with all of the datasets, IEBN performs significantly better than BN. In addition, with respect to the classification time, the rank of IEBN is 2.36, which is also second to Bagging, which means it has the lowest classification time after the original Bayesian network. The q-value for the comparison of IEBN and BN is 0.76 which is less than 2.34, indicating that, in terms of classification time, IEBN and BN are not significantly different. But the q-value for the comparison of IEBN and Bagging is 3.59, which means that IEBN classifies the instances significantly faster than Bagging. Also from the table, except for the liver and glass datasets, whenever Bagging improves the performance, IEBN improves the performance as well. However, the improvement is smaller than Bagging, because Bagging works on the classification step, while IEBN works on the structure of the network, which is a step before classification. It is possible to apply Inner Ensembles to the probability tables of the Bayesian network. The simplest way to do this is to sample the dataset, then calculate the entries of each probability table for each sample. The final entries are determined by averaging all the entries calculated by the different samplings. However, this technique is somewhat based on speculation, and further investigation and testing is needed. To summarize, we conclude that IEBN works significantly better than BN in terms of accuracy but the accuracy improvement is not as large as the improvement with Bagging. In addition, in terms of classification time, IEBN works significantly faster than Bagging and comparably to the original BN, which is consistent with what we expect for IEBN. So, in this section we have shown how our method achieves some of the performance improvement of Bagging with smaller networks, while maintaining the comprehensibility, classification performance and fast classification time of the original Bayesian network.

Table 3.3: Comparison of IEBN with original Bayesian network and Boosting in Terms of Accuracy and Classification Time

	BN		IEBN		BOBN	
Dataset	**Acc**	**Time**	**Acc**	**Time**	**Acc**	**Time**
Breast-Cancer	69.21	0	70.57	0	69.23	5
Breast-w	96.71	4	97	2	96.57	37
Credit-a	85.8	2	85.51	2	86.52	23
Ecoli	81.55	3	82.44	3	81.85	20
Glass	**71.03**	**3**	**69.63**	**3**	**69.16**	**3**
Heart-C	79.85	1	80.86	3	79.87	13
Heart-S	79.63	1	81.11	3	81.11	2
Hepatitis	79.33	1	80.66	1	81.29	21
Iris	92.86	0.6	93.07	1.1	94.33	10.4
Labor	89.3	0.3	89.43	0.6	90.7	8
Liver	**59.42**	**0**	**59.42**	**1**	**57.68**	**0**
Lymph	83.04	5	83.9	4	84.46	87
Pima	75.65	2	76.05	3	76.43	9
Tic-tac	92.9	7	93.32	2	98.75	51
Vote	95.86	1	96.09	2	95.4	27
Average on All Datasets	82.14	2.06	82.60	2.05	82.89	21.09
After Removing Datasets						
Average	84.75	2.15	85.39	2.05	85.89	24.11
(Accuracy)						
Average Rank	2.77		1.65		1.58	
Q-Value	2.84				0.2	
(Time)						
Average Rank		2.62		2.31		1.08
Q-Value		0.78				3.14

3.4.3 Compare with Boosting

In this section, we compare the performance of IEBN with Boosting and original BN in terms of accuracy and classification time. Table 3.3 shows the results of this experiment. In the table, we compare the accuracy and classification time over the number of folds for each dataset. We removed the datasets for which Boosting and IEBN do not improve the performance (Glass and Liver); these are highlighted in red. In the table the average performances are shown for all the datasets, excluding those that are highlighted.

According to the table, the accuracy ranking of IEBN is 1.65, second to Boosting. Also, the critical value of the Nemenyis test for three models is 2.34, and the q-value for comparing IEBN with BN is 2.84,which is higher. This means that, on average, IEBN performs significantly better than BN on all of the datasets. As well, according to the table the q-value for comparing IEBN with Boosting is 0.2, which is less than the critical value. This means that IEBN and Boosting are not significantly different.

With respect to classification time, the rank of IEBN is 2.31,second to original BN, indicating it has the lowest average classification time after original BN. The q-value for comparison of IEBN and BN is 0.78, which is far less than 2.34, which means that, in terms of classification time, IEBN and BN are not significantly different. But the q-value for comparison of IEBN and Boosting is 3.14, which means that IEBN classifies the instances significantly faster than Boosting.

To summarize, our conclusion after comparing IEBN with BN and Boosting is similar to what we concluded for Bagging: IEBN works significantly better than BN in terms of accuracy with significantly smaller networks, but this accuracy improvement is not as large as the improvement with Boosting. However, the results are much closer to Boosting than Bagging. In addition, in terms of classification time IEBN works significantly faster than Boosting, and comparably to original BN.

3.4.4 Classification Time

In this section, we run experiments to investigate the effect of ensemble size on classification time. The output of Inner Ensemble is a single model, increasing the ensemble size does not affect the size of the model. For the ensemble methods of Boosting and Bagging, increasing ensemble size should increase the classification time. To test these two hypotheses we use different ensemble sizes of 10 to 100, and for each ensemble size we run Bagging, Boosting and IEBN (with a fixed sample size 70%) on each of the datasets, and calculate the classification time. We then average over all the datasets. Figure 3.6 shows the average time for different ensemble sizes for BGBN, BOBN and IEBN. From this figure we see that:

1. By increasing the ensemble size, the classification time for Bagging increases as we expected.

2. By increasing the ensemble size, the classification time for IEBN does not change much.

3. The classification time for Bagging is higher than Boosting, and by increasing the ensemble size the Boosting classification time increases to a point, after which it only changes a small amount. This is because, for Boosting, if the error of the classifier is larger than 50% it will be eliminated. Thus, here for Boosting, after a certain ensemble size the number of classifiers does not change much, and consequently the classification time increases only minimally.

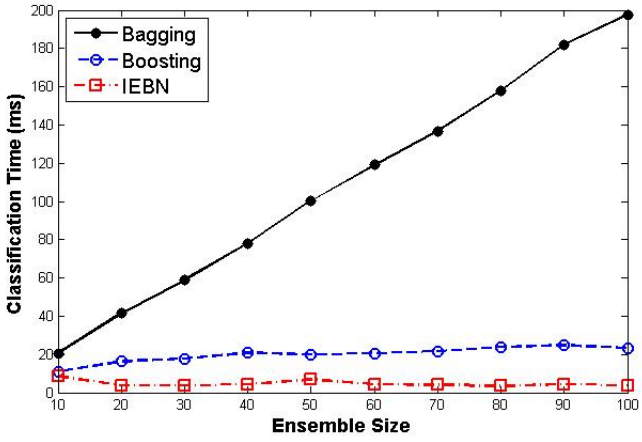


Figure 3.6: Classification time (mS) for Bagging and Boosting vs IEBN.

3.4.5 Network Quality

In this section, we compare the quality of the networks generated by IEBN with those generated by original BN. There are several measures to evaluate the quality of the network. Thus here first we explain the measures we used and then we use those measures to evaluate the quality of the networks.

Scoring Functions

The learning of the Bayesian network involves finding the network that best fits the data according to certain score functions. Generally, these score functions can be divided into two groups, local and global. The local metrics have the property that the score of the whole network can be decomposed as the sum or product of the score of each individual node. Global metrics do not have this property, and the whole network must be considered to determine the score. The global score metrics are based on cross-validation, a measure of how well a Bayesian network performs on a given dataset to predict its future performance (Bouckaert, 2004). Local functions are usually divided into two groups, Bayesian and Information-theoretic(de Campos, 2006).

To determine how well the network itself represents the training data, the quality of the network should be measured. In addition, to guide the heuristic search to learn the structure of the network, local measures determine the quality of the networks that have already been learned. If the network is represented by G and the data by D, the best network maximizes $p(G|D)$ or $p(G, D)$, because $p(D)$ is the same for all the possible

networks. Thus, as Bayesian measures all relate to estimating $p(G, D)$ and, consequently, $p(G|D)$, they can be interpreted as an estimate of the probability of network G given dataset D. The higher the Bayesian score, the higher the probability of network G given dataset D, which means that the network is a better fit to the data. Information theoretic measures are based on the compression, and provide an estimate of the length of the code induced by G to represent D. The smaller the length of the code, the higher the degree of fitness of G to D. Also, as it is easier to work in the logarithmic space, in practice the scoring functions use the value of $log(p(G, D))$, rather than $p(G, D)$.

In the literature, many researchers use Bayesian measures to compare their methods with others, and to measure the quality of the networks learned from the training set (Friedman et al., 1999b; Andrew Moore, 2003; Goldenberg and Moore, 2004; Teyssier and Koller, 2005; Carvalho, 2011). In this work, to measure the quality of the network, we used a well-known measure from the Bayesian group, BDeu, and one of the best known information theoretic measures, Minimum Description Length (MDL). For more information about other measures, refer to the work of (de Campos, 2006).

BDeu

To understand the Bayesian measures, we first need to explain the notation that is found in most of the literature (de Campos, 2006; Cooper and Herskovits, 1992).

Assume that Z is a set of n variables, each variable x_i in Z can take r_i possible values $v_{i1}, , v_{ir_i}$, and dataset D has m cases: that is, m instances. Each variable x_i in G has a set of parents, π_i. w_{ij} is the jth unique value of parent π_i, and there are q_i of these unique values for parent π_i. Then, N_{ijk} is defined as the number of instances of dataset D for which variable x_i has the value v_{ik}, and parent π_i has the value w_{ij}:

$$N_{ij} = \sum_{k=1}^{r_i} N_{ijk} \tag{3.5}$$

The DBeu measure originally introduced by Buntine (1991) and defined as:

$$g_{BDeu}(G : D) = log(p(G)) + \sum_{i=1}^{n} \left[\sum_{j=1}^{q_i} \left[log \left(\frac{\Gamma(\frac{\eta}{q_i})}{\Gamma(N_{ij} + \frac{\eta}{q_i})} \right) + \sum_{k=1}^{r_i} log \left(\frac{\Gamma(N_{ijk} + \frac{\eta}{r_i q_i})}{\Gamma(\frac{\eta}{r_i q_i})} \right) \right] \right] \tag{3.6}$$

In which η is a constant and for $log(p(G))$ is usually assumed uniform distribution and because it is a constant and can be removed.

Minimum Description Length (MDL)

The *Minimum Description Length* (MDL) scoring function (Lam and Bacchus, 1994; de Campos, 2006) selects the Bayesian network that requires the least length to represent the data. It consists of two parts: the model description length, and the description length of the data given the model. A good model minimizes the sum of the model description length and the description length of the data given the model.

The length of the description of the data given the model is the negative of the log-likelihood, which is the logarithm of the likelihood function of the data with respect to the network. Thus, a higher log-likelihood measure means a higher likelihood of the data given the network, and a smaller description length of the data given the network. The log-likelihood is defined as :

$$LL_D(G) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} log \left(\frac{N_{ijk}}{N_{ij}} \right) \tag{3.7}$$

The second term is the description length of the model itself. The description length of the network is proportional to the number called the network complexity.

$$C(G) = \sum_{i=1}^{n} (r_i - 1) q_i \tag{3.8}$$

And the proportionality factor is $\frac{1}{2} log(N)$ for which N is the number of instances in dataset. Then the description length of the network is:

$$\frac{1}{2} C(G) log(N) \tag{3.9}$$

Finally MDL is defined as the sum of the negative of the log-likelihood and description length of the network. By changing the signs for maximization problems, it is defined as:

$$g_{MDL}(G : D) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} log \left(\frac{N_{ijk}}{N_{ij}} \right) - \frac{1}{2} C(G) log(N) \tag{3.10}$$

In this equation, the first term should be maximized, because that means a higher likelihood of the data given the network and a smaller description length of the data given the network. And the second term should be minimized, which means a smaller description length of the network. Therefore, a larger MDL means the coding (Network) that requires the minimum length to represent the message (Data).

Table 3.4: Quality of the network learned by IEBN and BN in terms of BDeu, MDL and size of the network.

	Org	IEBN	Org	IEBN	Org	IEBN	
Dataset	**BDeu**		**MDL**		**Arcs**		**Pct Dec**
Breast-Cancer	-11199.74	-6230.17	-7378.87	-4738.58	14.4	12.2	15.28 %
Breast-w	-4352.61	-4260.97	-4274.28	-4202.75	13.5	11.9	11.85 %
Creadit-a	-28226.18	-21803.89	-19353.76	-15686.75	31.3	27.3	12.78 %
Ecoli	-1707.95	-1401.06	-1500.97	-1314.43	11.3	11	2.65 %
Glass	-6975.97	-11790.91	-4455.84	-6392.63	16.4	15.1	7.93 %
Heart-C	-1596.57	-1540.47	-1564.12	-1523.63	27.2	21.7	20.22 %
Heart-S	-4945.09	-4017.1	-4134.38	-3527.23	17.6	15.7	10.8 %
Hepatitis	-1805.42	-1694.48	-1691.78	-1608.04	36.3	33.8	6.89 %
Iris	-452.86	-436.9	-461.2	-447.17	5.28	4.73	10.42 %
Labor	-982.56	-773.26	-791.74	-675.85	22.34	21.37	4.34 %
Liver	-317.7	-317.7	-318.19	-318.19	0.5	0.5	0 %
Lymph	-11511.32	-5497.64	-7281.54	-4201.15	50.1	40.3	19.56 %
Pima	-3649.29	-3613.22	-3599.77	-3626.66	8.6	7.6	11.63 %
Tic-tac	-10243.58	-10220.49	-9713.46	-9698.23	23.4	23	1.71 %
Vote	-4028.39	-4024.98	-4034.78	-4032.41	13.1	12.3	6.11 %
Average	-6133.015	-5174.88	-4703.65	-4132.91	19.42	17.23	9.48 %
Average Rank	1.9	1.1	1.83	1.17	1.03	1.97	
Q-Value	3.1		2.58		3.61		

Measuring the Quality of the Networks

In this section, we compare the quality of the networks generated by IEBN with those generated by original BN. To do this, as in Figure 3.5, we calculate the quality of the network generated using the training set for both BN and IEBN for each training set of the cross-validation, and then average the results over different folds. We use BDeu and MDL measure to evaluate how the networks represent the datasets. In addition, we calculate the number of arcs for each network, and average this over different folds. The results are shown in Table 3.4. According to the table, the BDeu and MDL measures for IEBN are higher than those for BN on all the datasets, except for Glass and Liver. For the Glass dataset, IEBN could not find the network that represents the dataset better than BN, and for the Liver dataset, both methods generated the same network. Because BDeu measure is an estimate of how probable the network is, given the dataset, it shows how it represents the dataset. The results of Friedmans test are also reported in this table, and for BDeu the rank of the IEBN is 1.1, which is smaller than the rank of BN, which is 1.9. As well, the $q - value$ Nemenyis test is 3.1, which is higher than the 1.96 critical value for two models. This shows that, on average over all the datasets, IEBN generates networks that represent data significantly better than original BN.

According to Table 3.4, IEBN requires smaller length of the code to represents the

dataset. Also according to the Friedman test, the average rank of IEBN is 1.17 which is smaller than the one for BN that is 1.83. Also the $q - value$ Nemenyis test is 2.58 which is higher than the critical value for two models which is 1.96. This shows that average on all of the datasets, IEBN generates networks that need significantly smaller length of code than the one for original BN to represent data. This shows again that the networks generated by IEBN, are more fitted to the datasets than the ones generated by BN.

In addition, we compare the size of the network in terms of the number of arcs. According to the table, in all of the datasets, except for liver dataset where both BN and IEBN generate the same network, IEBN generate smaller network. Freidmans test shows that the average rank in terms of network size for IEBN is 1.97, larger than the 1.03 rank forBN. Also the $q - value$ Nemenyis test is 3.61, higher than the 1.96critical value for two models. This shows that, on average over all the datasets, IEBN generates networks that are significantly smaller than those generated by BN. The last column of the table shows the percentage that the size of the network decreased using IEBN, compared to BN. This percentage is calculated using equation 3.4. The table shows that, on average over all the datasets, IEBN generates networks 9.45% smaller than BN.

To summarize, these experiments validated two hypotheses. First, IEBN generates networks that represent data significantly better than BN, and need significantly smaller codes to represent the data. Second, IEBN generates networks with a significantly smaller number of arcs. Thus, for all these reasons we can conclude that, since IEBN generally generates significantly simpler networks, understanding these networks will be easier, which means that IEBN results in more comprehensible networks.

3.4.6 Measuring the Comprehensibility

The comprehensibility of the Bayesian network is the degree to which it is understandable. An example of an application of the Bayesian network is medical diagnosis, where the comprehensibility of the network is more important than the accuracy of the results, as medical experts should know the different symptoms related to a particular disease. In such cases, doctors must be convinced of the correctness of the Bayesian network so they can trust the model, and this correctness can be assessed by its comprehensibility. Although regular measures, such as the number of arcs and the Minimum Description Length (MDL), might provide ideas about how the model is interpretable, they may not be enough to convince an expert user. For example, when an arc is eliminated from a network the network is smaller, but it is not clear whether that arc was the correct

one to remove. A similar but reverse situation is when we add an arc to the network. Therefore, we need another way to validate the comprehensibility of the network; there are several options, each with advantages and disadvantages. The first option is to design a measure based on the relationships of different attributes, which can extract the relationships between the features of the Bayesian network and measure the degree of conflict among the relations. The advantage of this method is that it can be very fast, while the disadvantage is that it only measures the degree of conflict among the relations; we also need to know if the relations are correct, which requires more information about the domain. Another way to measure comprehensibility is subjectively; that is, train a Bayesian network on a specific dataset, then ask experts in this particular dataset to rate the understandability of the network on a scale of 0 to 10. The advantage of this method is that it is very accurate, but it is also very expensive in terms of time, particularly for measuring the comprehensibility of the Bayesian network for multiple datasets. In addition, finding experts for each dataset could be difficult, and they would need considerable time to do the ratings. To best measure the comprehensibility we need a method that is both accurate and time efficient. However, if we can find benchmark datasets with all the correct relations among the features already extracted by an expert, we can simply train the Bayesian network on the benchmark datasets, then extract the relations and compare them to the ground truth relations which are already given by an expert. The advantages of this method are that it is accurate and time efficient, and it does not require experts to confirm the correctness of the relations among the attributes; although there is still the challenge of finding suitable datasets. To summarize, though measuring the comprehensibility of the Bayesian network is not in the scope of this thesis, it should be addressed in future research.

3.5 Summary

In this chapter, we argue that using Inner Ensembles can provide the comprehensibility of the Bayesian network, while keeping some of the advantages of the ensembles, such as improved performance over the original network. However, some of the accuracy is traded off for comprehensibility. We use Inner Ensembles to learn the structure of the Bayesian network, and call our method Inner Ensemble Bayesian network (IEBN). The results show that, on average overall the datasets, IEBN results in significantly smaller networks that represent dataset significantly better than original BN. In addition, the performance of the IEBN is compared to that of BN and two ensemble methods, Bagging and Boosting.

The results show that, on average over all the datasets, IEBN works significantly better than BN. And, generally when ensemble methods improve the accuracy, IEBN improves the accuracy as well. However, this improvement is smaller than with ensembles, because ensembles work directly in the classification step, while IEBN is applied on the structure of the network, which is the step before classification. Finally our results show that by increasing ensemble size, the classification time for IEBN does not change while the classification time for Bagging increases as we expected.

Chapter 4

Inner Ensemble K-Means Clustering

4.1 Introduction

In this chapter, we apply Inner Ensembles to unsupervised learning, using the popular K-Means algorithm clustering method (Macqueen, 1967; Wagstaff et al., 2001). K-means has useful properties, one of the best being that it characterizes each cluster in terms of a single point, called a *prototype*. Each prototype captures the distribution of a group of data points, based on their similarity or proximity to the prototypes. That a prototype is an effective way of representing a particular concept is central to certain theories in Cognitive Science (Lakoff, 1987). Some advantages of prototypes are (Tan et al., 2005):

1. **Summarization:**

   The main advantage of prototypes is that they provide a means to summarize the given data in a few representatives, thus leading to interpretability of the cluster structures. This summarization is very important, because it avoids having to memorize too many and/or irrelevant details, and significantly reduces the amount of memory used. Also, many techniques for analyzing the data have quadratic time and space complexity, and are not practical for very large datasets. Here, data summarizing plays the important role of allowing the algorithm to be applied only to cluster prototypes, rather than the entire dataset.

2. **Compression:**

   Another advantage is with respect to data compression, and is known as vector quantization. This is often applied on sound, image and video data, which has many similar data objects, loss of some information is acceptable, and reduction of

the data is essential. With this type of compression, each data object is represented by the index of the prototype associated with the cluster containing the data object.

3. **Efficiently Finding Nearest Neighbors:**

   It is possible to find the nearest neighbours more quickly using prototypes. Doing this usually requires computation of the distance between each pair of objects, which can be very time consuming. Using prototypes can significantly reduce the number of calculations. The distance between two prototypes provides an estimate of the distance between two clusters. If two clusters are far apart, the nearest neighbour of an object in one of the clusters cannot be in the other cluster, which means the nearest neighbour of an object must be in nearby clusters. Thus, it is only necessary to find the nearby clusters, by calculating the distances between prototypes, find the nearest neighbours on those clusters, and ignore the rest.

Using traditional ensemble of K-Means clustering means we lose the prototypes that are the cluster centers for K-Means clustering, because ensemble of K-Means creates a co-association matrix that is not based on the center of the clusters. Instead, the entry i, j of the co-association matrix is 1 if ith and jth data instances are in the same cluster. After this step, the final partitions are extracted as groups of data instances belonging to some cluster. Prototypes are very difficult or even impossible to be found using another step after cluster ensemble because there is no guarantee that the number of clusters remains the same before and after using cluster ensembles; thus we lose information here. In addition, each member of a cluster ensemble may have different prototypes. For example, for webpage clustering, each member of the ensemble can cluster web pages according to different criteria, such as links, or even page content. Therefore, each ensemble member has different prototypes, and after combining the clusterings these prototypes are eliminated and cannot be found. Using Inner Ensembles lets us keep the cluster prototypes, and maintain comprehensibility. As well, using regular ensemble methods for K-Means requires additional $O(N^2)$ memory for the co-association matrix; N being the number of data objects. This is a considerable amount of memory, particularly for very large datasets. Using Inner Ensembles, we do not need to store the co-association matrix, which is another advantage of Inner Ensemble clustering over regular ensemble clustering. In addition, ensemble of clustering is time consuming task. The process requires running the clustering algorithm K times, where K is the ensemble size, calculating the co-association matrix with a time complexity of $O(N^2)$ for N data objects, and running another algorithm on the co-association matrix to generate the final partitions. This

makes it very difficult to apply ensemble methods for online clustering. Using Inner Ensembles makes it possible to use ensemble methods for incremental clustering. For each incoming instance, it is only necessary to assign each instance to each cluster, K times . In future work, we will investigate how to apply Inner Ensembles to online clustering. In this thesis, to show that Inner Ensembles works for unsupervised learning, we mainly focus on offline K-Means clustering. Generally, using Inner Ensembles for clustering gives us the previously mentioned advantages over ensemble of clustering, while gaining the performance advantages of the ensemble method. In this chapter, we first explain K-Means clustering, then discuss the general guidelines for applying Inner Ensembles. We call the resulting algorithm Inner Ensemble K-Means (IEKM).

4.2 K-Means Clustering

The K-Means clustering algorithm takes the input parameter K, which is the number of cluster centers, and partitions N objects into K clusters, so that the clusters are separate from one another while the data objects inside each cluster are close to each other as possible. For K-Means, cluster similarity is measured with reference to the mean value of the objects in each cluster; this is known as the clusters *centroid* or *centre of gravity*. The closeness of the objects to the cluster centers is measured using different types of distance measurement, such as Euclidean distance. K-Means clustering starts with initialization of the cluster centers. It then assigns each data object to its closest cluster center, according to the Euclidean distance. Finally, each cluster center is updated by averaging all the data objects in each cluster. The algorithm stops when there are no more changes in cluster centers, or when it reaches the maximum number of iterations. Algorithm 14 details the steps in K-Means clustering (Duda and Hart, 1973).

4.2.1 Ensemble of K-Means

The cluster ensemble problem can be defined as issues with combining multiple partitionings of the data. Different clusterings partition the space in different ways, and cluster ensemble combines the results of these clusterings. This means that cluster ensemble works on the labels, and it does not use the original variables of the data for final partitioning. This is more difficult than classifier ensemble, because the problem of correspondence must be solved as well. Each ensemble member can be a K-Means clustering, with a different number of clusters and a different number of features. In this work, we

Algorithm 14 K-Means clustering algorithm.

1: Initialize K cluster centers $\mu_1, \mu_2, ..., \mu_K$
2: Data: $X = \{X_1, X_2, ..., X_N\}$
3: **repeat**
4: **for** All data instances X_i **do**
5: $m = argmin_K\{d(X_i, \mu_K)\}$ {Closest cluster center to each instance}
6: Assign instance X_i to cluster center μ_m
7: **end for**
8: Update cluster centers.
9: **until** (No assignment change or max iterations)
10: **Return** The clusters.

used six different clustering ensemble methods from two groups: co-association based, and hyper-graph based methods. With these groups, the partitions are first converted to a co-association matrix or hyper-graph representation, and then another algorithm uses the result to generate the final partitions.

Entry (i, j) of the co-association matrix is 1 if instance i and j are in the same cluster. First, for each ensemble member, that is, each K-Means clustering, the co-association matrix is calculated, and the final co-association matrix is generated by averaging all the matrices. One of the methods for cluster ensembles is based on hierarchical clustering that works on the co-association matrix as the similarity matrix. This method is called Hierarchical Agglomerative Clustering Consensus (HAC), and applies to different hierarchical clusterings in the final co-association matrix.

Another method of cluster ensemble is based on hyper-graph representation. The co-association graph is first converted to a hyper-graph representation, and different graph-partitioning algorithms partition it into different clusters. This group contains three different algorithms: the Cluster-based Similarity Partitioning Algorithm (CSPA), the Hyper-Graph Partitioning Algorithm (HGPA) and the Meta-Clustering Algorithm (MCLA). For the details of these algorithms, refer to the Related Work chapter of this thesis.

Algorithm 15 General guidelines for using Inner Ensembles

1: **Locate a decision maker inside the algorithm.**
2: **Finding a measure based on which that decision maker works.**
3: **Indicating the input and output of the measure.**
4: **Applying the ensemble on the measure:**
5: Changing the input of the measure in different ways. {It can be sampling of data or feature based modifications or other methods.}
6: Generating an output based on each input.
7: Combining the outputs
8: **Applying the output of the ensemble for decision making.**

4.3 Applying Inner Ensembles on K-Means Clustering

To make following the general guideline easier, here we repeat it again in Algorithm 15. The first step of the algorithm is to identify a decision maker inside the K-Means algorithm. The most important and most obvious decision makers assign each data object to its closest cluster center. The next step is to find a score function. The score function for closeness to the center of the cluster for K-Means clustering is the Euclidean distance. The input of the score function is cluster centers and the data object, and the output is the distance of the data object to each cluster center. Using Euclidean distance the closest cluster center to each data object is found. Thus we can consider the closest cluster center to each data instance as the output of the measure. The most important part of the algorithm is how to apply ensemble methods on the measure. In order to do this, we need to change the input of the Euclidean distance.

One way to generate several inputs is to sample the data in each cluster, and using these samples we find the closest cluster center. However, this method did not generate good results, because the sampling did not provide enough diversity to assign each instance to the closest cluster center. Instead, to produce more diversity, we argue that some parts of the space can be partitioned better using a subset of features, and other parts partitioned better using another subset. To illustrate, we use an example.

Figure 4.1 shows the scatter plot of the Iris dataset from the UCI repository. This dataset has four attributes, excluding the class attribute; thus it has six different pairs of attributes. In the figure, the classes of Iris for each pair of attributes are shown in different colors. The blue dots in the figure represent the class called Setosa, which it is

Algorithm 16 Inner Ensemble K-Means clustering algorithm.

```
1: Initialize K cluster centers μ_1, μ_2, ..., μ_K
2: Data: X = {X_1, X_2, ..., X_N}
3: repeat
4:    for All data instances X_i do
5:       m = {m_1, ..., m_K}.
6:       for  j= 1 to Ensemble size do
7:          S = Sample(FeatureSpace)
8:          I = argmin_K{d(X_Si, μ_K)} {Closest cluster center to each instance}
9:          m_I = m_I + 1
10:      end for
11:      L = argmax_k(m_k) {Voting}
12:      Assign instance X_i to cluster center μ_L
13:   end for
14:   Update cluster centers.
15: until (No assignment change or max iterations)
16: Return The clusters.
```

linearly separable from the other two classes if we use each of the six pairs of attributes. However, the other classes, shown by the black and red dots, are not linearly separable, and we can separate them differently using each pair of attributes. For example, if we only use the Sepal Length and Sepal Width attributes (Figure 4.1 top left plot), it is not possible to separate the black and red dots, because they conflict with each other. But, by using the Sepal Length and Petal Length attributes (Figure 4.1 top right plot), the black and red dots can be more easily separated. In fact, using each attribute pair, some part of the data can be linearly separable; this might not be the case if we use the other pairs of attributes, or even all of the attributes. More generally, by using different subsets of attributes, different parts of the data could be more separable than when we use all the attributes. Thus, based on this example, if we somehow use different attribute subsets instead of all the attributes, we could get better clustering results. In fact, using different attribute subsets gives the data object the opportunity to be assigned to the best cluster on one of the attribute subsets.

To generate different inputs for the Euclidean distance, we use different subsets of the features by sampling from the feature space. The sampling can be with or without replacement, and in different sizes. The next step of the general guideline is to generate

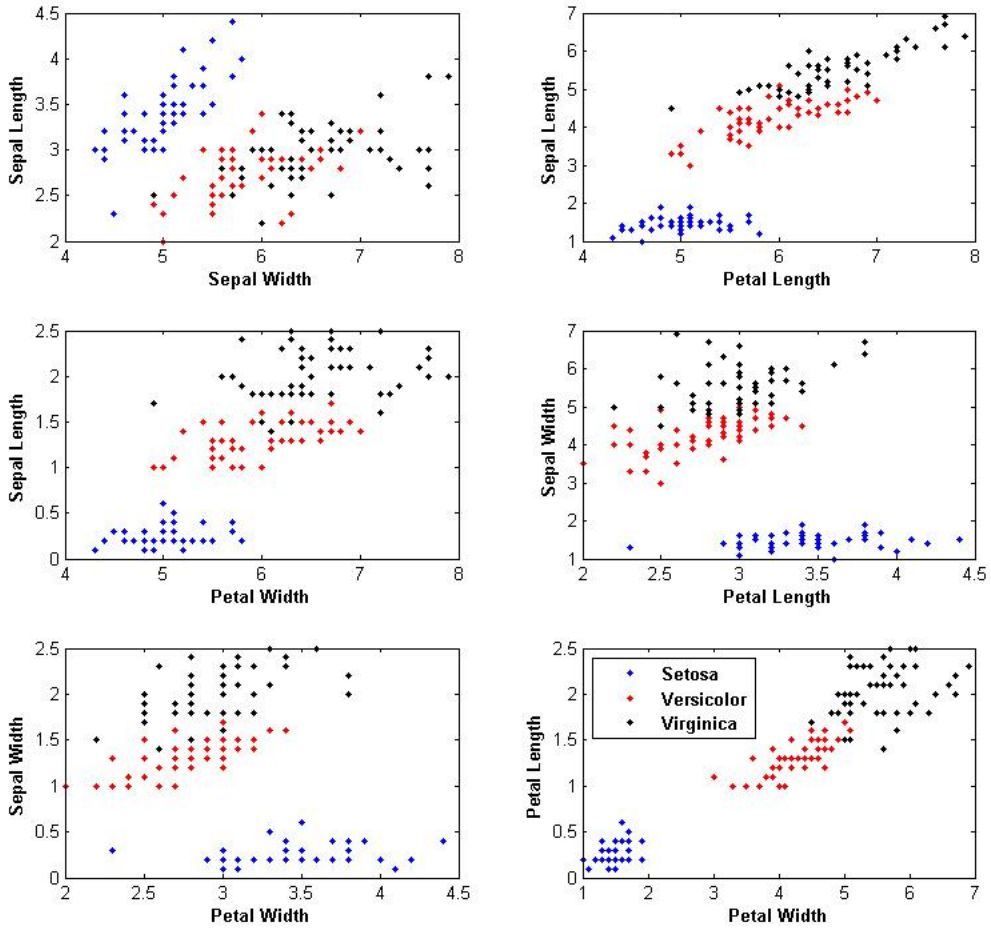


Figure 4.1: Scatter plot of Iris dataset.

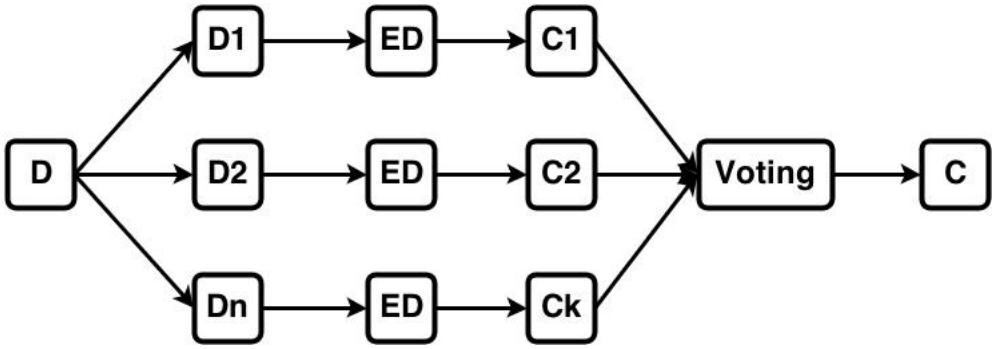


Figure 4.2: Applying Inner Ensemble on K-Means.

the output of each generated input, which is the best cluster center for each data object, according to the corresponding feature subset.

The final step is to combine the outputs. We do this using a majority vote, with the cluster center with the highest vote for different feature subsets selected as the best cluster center for the current data object. This is shown in Figure 4.2, where D shows the dataset and D_i is the data with the ith feature subset. Also, C_i and ED denote the ith cluster center and Euclidean distance respectively. To summarize, in each iteration of K-Means, instead of using all the attributes to calculate the distance between each data object and each cluster center we use random subsets of features, and for each subset we find the best cluster center. Finally, using voting, we find the best center for each data object. The diversity is created because different subsets of the features separate different parts of the data space. The resulting algorithm is called Inner Ensemble K-Means or (IEKM). Algorithm 16 shows Inner Ensemble K-Means clustering.

One important difference is that it is very difficult if it is not impossible to apply cluster ensembles to online clustering. The first step (generating different clusterings) and the second step (combining the results of the clusterings) of the cluster ensemble should be done incrementally, but even if this is possible it is very time consuming, since, for example, creating a co-association matrix has a time complexity of $O(n^2)$,in which n is the number of data samples. Thus, applying ensemble method for online clustering is very difficult, and to the best of our knowledge the closest work on this is (Prodip, 2007). This is not actually cluster ensemble for data stream, but instead a scalable framework that divides the dataset into different parts, uses regular clustering for each part, then uses cluster ensemble to combine the centroids of each part of the data. However, it is possible to apply Inner Ensemble to the online K-Means algorithm without greatly impacting the running time, as Inner Ensemble makes it possible to

use ensemble methods for online clustering. A major problem with cluster ensemble is solving the correspondence issue, which we do not need to do for IEKM since we keep the cluster centers.

In addition to the differences between IEKM and cluster ensemble there are also some similarities, one of which is we can create diversity in a similar way. For example, here we create diversity for IEKM by sampling the feature space, and we create clustering ensembles in the same manner. For both methods we can use different approaches, such as sampling the data instances. Another similarity is we expect that both methods are robust to noisy data; this will be tested in the experimental results section.

4.4 Experimental Results

In this section, we compare the performance of Inner Ensemble K-Means with that of K-Means and cluster ensembles. To do this, from all the possible performance measures we chose to work with two: Normalized Mutual Information (NMI) (Strehl et al., 2002) and Purity measure (Rendón et al., 2011). We selected these two measures because: First, NMI is usually used to calculate the performance of cluster ensembles, they belong to two different families of cluster validation indices, and to calculate the performance of cluster ensembles we need measures that are based only on the labels of the data and not on the values of the data features, because with cluster ensembles we only have access to the final labels of the data. Second, we chose the Purity measure as well because NMI is based on the true labels of the data, and we want another measure that does not rely on these, such as Purity.

1. **Normalized Mutual Information**

   Determines the shared information between clusters and uses the true labels of the clusters. It is defined as follows:

$$NMI(X,Y) = \frac{I(X,Y)}{\sqrt{H(X)H(Y)}} \tag{4.1}$$

   In this equation, $I(X,Y)$ is the mutual information between two random variables X, Y; $H(X)$ is the entropy of X; X is the result of the clustering; Y contains the true labels. More specifically if X and Y are two random variables described by cluster labeling $\lambda^{(a)}$,$\lambda^{(b)}$ with number of clusters $k^{(a)}$ and $k^{(b)}$ then Normalized Mutual Information is defined as:

$$NMI(\lambda^{(a)}, \lambda^{(b)}) = \frac{\sum_{h=1}^{k^{(a)}} \sum_{l=1}^{k^{(b)}} n_{h,l} log(\frac{n.n_{h,l}}{n_h^{(a)}.n_l^{(b)}})}{\sqrt{(\sum_{h=1}^{k^{(a)}} n_h^{(a)} log\frac{n_h^{(a)}}{n})(\sum_{l=1}^{k^{(b)}} n_l^{(b)} log\frac{n_l^{(b)}}{n})}} \tag{4.2}$$

for which $n_h^{(a)}$ and $n_l^{(b)}$ are the number of the instances in the first and the second cluster and $n_{h,l}$ denote the number of objects in cluster h and l.

2. **Cluster Purity**

   Measures the coherence of a cluster is defined as follows:

$$Purity = \sum_{j=1}^{m} \frac{n_j}{n} P_j \tag{4.3}$$

In this equation, n is the total number of instances, m is the number of clusters, and P_j is the fraction of the cluster that the largest class of objects is assigned to. For both measures, the larger the value the better the results.

4.4.1 Experimental Setup

In this section, we run several experiments for our new versions of K-Means clustering. In this section we test two different hypotheses.

- The performance of Inner Ensemble K-Means is superior to the original K-Means.
- Whenever the ensemble methods work, Inner Ensembles work too

Finally we run several experiments to test the robustness of Inner Ensembles on noisy data. Here to test the robustness of Inner Ensemble to noise, the experiments is run on the Inner Ensemble K-Means with the hypothesis: In presence of the feature noise, IEKM performs better than original K-Means and when ever cluster ensemble is robust to noise, IEKM is robust to noise too. For our experiments, we compare results on 15 datasets from UCI repository (A. Asuncion, 2007), each with the necessary numerical attributes used by such a clustering system. The characteristics of the data is shown in Table 4.1 [1].

[1]For the datasets with more than 500 instances, we sample the data without replacement to 500 instances.

Table 4.1: Description of experimental datasets for Inner Ensemble K-Means.

Dataset	Instances	Features	Classes
Ecoli	336	8	8
Glass	214	10	7
Heart-Stat	270	14	2
Iris	150	5	3
Letter	20000	17	26
Optdigits	5620	65	10
Pendigits	10992	17	10
Pima	768	9	2
Segment	2310	20	7
Sonar	208	61	2
Vehicle	846	19	4
Vowel	990	14	11
Waveform	5000	41	3
Wine	178	14	3
Yeast	1484	9	10

For these experiments, we compare the performance of Inner Ensemble K-Means with the two different groups of cluster ensemble methods we explained in Chapter 2. The first is a graph-based approach that includes three different methods CSPA,MCLA and HGPA (Strehl et al., 2002). The second is based on pair-wise similarity (Nguyen and Caruana, 2007), and this group first builds a similarity matrix, called a co-association matrix, then uses it for different forms of hierarchical clustering; the group is also called Hierarchical Agglomerative Clustering Consensus (HAC). There are three different methods for HAC based on hierarchical clustering: single, average and complete linkage. There are different ways to generate cluster ensembles, one of which is generating them based on different feature subsets (Strehl et al., 2002). Because Inner Ensemble K-Means is based on different subsets of features, for a fair comparison we use this method to generate cluster ensembles as well. The performance of Inner Ensemble K-Means is analyzed based on the following different parameters:

- *Ensemble Size:* 10 different ensemble sizes 10-100.
- *Sample Size:* 9 different sampling sizes 10%-90%.
- *Resampling Mode: with* or *without* replacement (However in this work we choose sampling without replacement).

To find the best parameters for cluster ensembles and Inner Ensemble K-Means, we use cross-validation. The general approach for the experiments on noisy and clean data is to first divide the data into N folds, then choose a fold as the test set to run K-Means, Inner Ensemble K-Means and cluster ensemble on . The rest of folds are train data to choose the best parameter for Inner Ensemble and cluster ensemble, and to find this parameter we run these methods for different parameters on the train data, and return the best parameters to run on the test fold. Then we use these best parameters to run the Inner Ensemble, cluster ensemble and original K-Means on the test data a certain number of times, each time with different initial cluster centers. In the next section, we explain how to add noise, and the technique to compare the performance of the algorithms on noisy data. For our experiments, we run each test data that is on each fold ten times, then average the results. In all experiments we use 5-fold cross-validation. This procedure is shown in figure 4.3. Also, for all experiments we use Friedmans test with $alpha = 0.05$, and for the post-hoc test we use Nemenyi's test with $alpha = 0.05$.

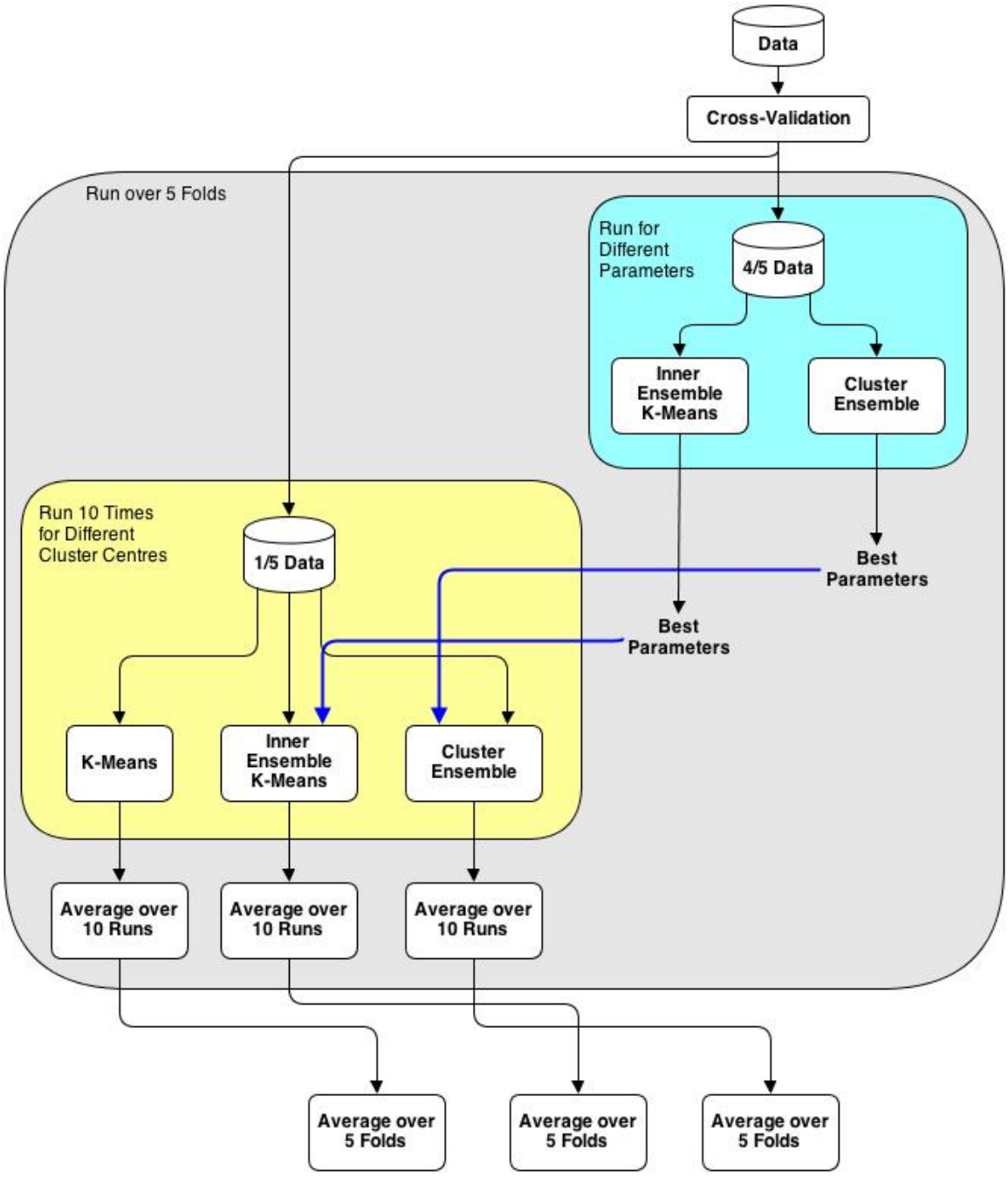


Figure 4.3: Procedure for comparing the performances of Inner Ensemble, Cluster Ensemble and K-Means.

Table 4.2: Comparison of K-Means, Inner Ensemble K-Means and Ensemble K-Means (G-Based, HAC) in terms of NMI for Sampling Without Replacement.

		No Rep	HAC			G-Based		
Dataset	**K-Means**	**IE-K-Means**	**S-Linkage**	**A-Linkage**	**C-Linkage**	**CSPA**	**HGPA**	**MCLA**
Heart-S	0.04	0.25	0.14	0.24	0.28	0.26	0.07	0.30
Iris	0.79	0.82	0.79	0.79	0.78	0.81	0.78	0.81
Letter	0.65	0.66	0.64	0.68	0.68	0.67	0.67	0.70
Optdig	0.68	0.73	0.68	0.74	0.73	0.71	0.69	0.71
Pendig	0.68	0.70	0.67	0.71	0.70	0.67	0.66	0.71
Pima	0.02	0.08	0.03	0.13	0.11	0.12	0.01	0.09
Segment	0.57	0.66	0.61	0.68	0.66	0.64	0.62	0.59
Vowel	0.33	0.37	0.32	0.40	0.37	0.39	0.35	0.48
Wine	0.45	0.74	0.55	0.70	0.66	0.68	0.56	0.68
Average	0.47	0.56	0.49	0.56	0.55	0.55	0.49	0.56
[W D L]		[9 0 0]	[4 2 3]	[8 1 0]	[8 0 1]	[8 0 1]	[6 0 3]	[9 0 0]

4.4.2 Compare with Cluster Ensembles

Table 4.2 shows the results of comparing Inner Ensemble with original K-Means and six different cluster ensemble methods from two different families, in terms of the NMI measure when we use sampling without replacement. Because we do not expect Inner Ensembles to work whenever ensemble methods do not improve the performance, we remove the datasets for which both ensembles and Inner ensembles did not improve the performance over the original K-Means. However in this chapter for clear presentation of the results, we remove the datasets for which a majority of ensemble systems didn't do better on both NMI and purity measures[2]. Instead we report the detailed experiment on all of the datasets for each cluster ensemble and for all of the datasets that each ensemble improves the performance in appendix A. The results in appendix A shows similar conclusion as we have in this chapter in terms of statistically significance. From the table, we can see that the only cluster ensembles that work comparably to Inner Ensemble on average on all of the datasets, are the average linkage and MCLA methods. On average, all other ensemble methods do not work as well as Inner Ensemble over all the datasets. Also from the table, we can see that the number of times a method improves the performance, does not improve the performance, and decreases the performance compared to the original K-Means. This is shown in the table by win, draw and loss [W,D,L].

The table shows that Inner Ensemble improves the performance of a larger number of datasets compared to any other ensemble method, and it does not decrease the per-

[2]Removed datasets are Ecoli, Glass, Sonar, Vehicle, Wavef and Yeast

Table 4.3: Friedmans test results for comparing K-Means and each cluster ensemble with IEKM for NMI measure.

	K-Means	IE-K-Means	S-Linkage	A-Linkage	C-Linkage	CSPA	HGPA	MCLA
F-Stat			13.56	12.39	10.67	10.89	11.56	13.56
Q-Value	3.30		3.06					
	2.71			0.59				
	2.82				0			
	3.06					0.47		
	3.30						2.36	
	3.06							0.24
Avg Rank	2.57	1.00	2.44					
	2.94	1.67		1.39				
	2.89	1.56			1.56			
	2.89	1.44				1.67		
	2.67	1.11					2.22	
	3.00	1.56						1.44

formance of any datasets. The only ensemble method that works comparably to Inner Ensemble is the MCLA method, which performs well on the same number of datasets. In Table 4.3, a statistical test has been performed to compare each Inner Ensemble with each ensemble method and K-Means. From the table, we can see that Friedmans statistic for all of the ensemble methods is larger than six, which is the critical value for three models. This means that we can reject the null hypothesis, as at least one of the IE-K-Mean, K-Means or cluster ensemble methods is significantly different from the others. The critical value for the Nemenyi test for three models is 2.34, and in the K-Means column in the table all the $q - values$ are larger than 2.34, which means that on average IEKM performs significantly better than K-Means over all the datasets. The table also shows that, for all the other cases the $q - value$ is less than 2.34, with the exception of Single-Linkage. This means that IEKM does not have significantly different performance compared to all other ensemble methods, except for Single-Linkage, for which it performs significantly better. The table also shows the average rank of the algorithms: in three out of six situations IEKM has the best rank, in one situation it has an equal rank to the ensemble method, and in two situations it ranks second after ensemble methods.

Table 4.4 shows a comparison of the results of Inner Ensemble with original K-Means and six different cluster ensemble methods from two different families, in terms of the Purity measure when we use sampling without replacement. From the table, we can see that the only cluster ensembles that work better than Inner Ensemble are MCLA and CSPA, and all other methods are either comparable or worse. Inner Ensemble works better than K-Means over all the datasets, and the only cluster ensemble that works

Table 4.4: Comparison of K-Means, Inner Ensemble K-Means and Ensemble K-Means (G-Based, HAC) in terms of Purity for Sampling Without Replacement.

		No Rep	**HAC**			**G-Based**		
Dataset	**K-Means**	**IE-K-Means**	**S-Linkage**	**A-Linkage**	**C-Linkage**	**CSPA**	**HGPA**	**MCLA**
Heart-S	0.62	0.78	0.68	0.73	0.73	0.77	0.60	0.79
Iris	0.87	0.90	0.86	0.87	0.87	0.92	0.90	0.90
Letter	0.45	0.46	0.45	0.48	0.48	0.46	0.46	0.44
Optdig	0.66	0.71	0.63	0.72	0.71	0.72	0.69	0.71
Pendig	0.65	0.70	0.60	0.68	0.69	0.68	0.67	0.69
Pima	0.66	0.68	0.67	0.72	0.71	0.69	0.66	0.70
Segment	0.61	0.63	0.59	0.69	0.68	0.71	0.66	0.63
Vowel	0.32	0.36	0.32	0.36	0.36	0.40	0.37	0.46
Wine	0.70	0.93	0.75	0.86	0.86	0.88	0.75	0.89
Average	0.62	0.68	0.62	0.68	0.68	0.69	0.64	0.69
[W D L]		[9 0 0]	[3 2 4]	[7 1 1]	[7 1 1]	[9 0 0]	[7 1 1]	[8 0 1]

better than K-Means overall the datasets is CSPA.

Table 4.5 shows the results of a statistical test performed to compare each Inner Ensemble with each ensemble method and K-Means. The null hypothesis here is that there is no method that is significantly different from others. From the table we can see that Friedmans statistic for all the ensemble methods is larger than six, which is the critical value for three models. This means that we can reject the null hypothesis which means that at least one of the methods, IE-K-Mean, K-Means or cluster ensemble, is significantly different from the others. The critical value for the Nemenyi test for three models is 2.34, and the K-Means column in the table shows that all the $q - values$ are larger than 2.34, which means that IEKM performs significantly better than K-Means on average over all the datasets. The table also shows that, with the exception of Single-Linkage, for all the other cases the $q-value$ is less than 2.34, which means that, except for Single-Linkage for which it performs significantly better compared to all other ensemble methods, IEKM performance is not significantly different. Also from the table we can see the average rank of the algorithms: in five of six situations IEKM has the best rank, and in one situation it has the second rank after ensemble methods.

4.4.3 Robustness to Noise

In this section, we investigate the effect of noise on Inner Ensembles. We first add noise to the data, then following the figure 4.3, we use the best parameters we found on 4/5 of the data for Inner Ensemble, and cluster ensembles on 1/4 of the data. We then run the clustering on noisy data 10 times, each time with different initial cluster centers. This

Table 4.5: Friedmans test results for comparing K-Means and each cluster ensemble with IEKM for purity measure.

	K-Means	**IE-K-Means**	**S-Linkage**	**A-Linkage**	**C-Linkage**	**CSPA**	**HGPA**	**MCLA**
F-Stat			13.56	12.06	12.17	13.72	10.50	10.72
Q-Value	3.06		3.30					
	3.06			0.11				
	3.18				0.35			
	2.94					0.47		
	3.18						1.06	
	2.94							0.23
Avg Rank	2.44	1.00	2.56					
	2.94	1.50		1.56				
	2.94	1.44			1.61			
	3.00	1.61				1.39		
	2.83	1.33					1.83	
	2.89	1.50						1.61

procedure is repeated for each of the folds, and the final results for each fold are averaged over different runs which is 10 and final result for all of the folds is averaged over all of the folds which is 5. We also use both NMI and Purity measures for this experiment. The main problem here is how to inject the noise into the dataset. There are different ways of adding noise in the literature that depend on the clustering algorithm, and the concept of noise that the researchers are attempting to explore. Many papers discuss how to build a clustering algorithm that can remove, filter or ignore noise during the clustering. These methods assume that the noisy instances are not generated by a noisy concept, but by different distribution. This kind of noise is sometimes called background noise, and generating this noise will add extra instances to the dataset that do not belong to any concept, and therefore should not be clustered at all. Background noise can be ignored, filtered or removed from the dataset, which is why these methods used this kind of noise to test their clustering.

Pelleg and Baras (2007) investigated the noise in semi-supervised clustering. The work is based on two constraints, must-link (ML) and cannot-link (CL), and the unlabeled data plus the constraints are used for clustering. The constraints are noisy in this method, and to add noise two instances are randomly selected, and their labels changed randomly. If two labels are similar an ML constraint is added, otherwise a CL constraint is added.

Another work by Fisher (1989) introduces noise-tolerant conceptual clustering. This work specifically explores the impact of noise on the prediction accuracy of the COBWEB clustering algorithm. In this type of clustering noise is injected into the attributes, because COBWEB works with attribute values. In this paper, noise is defined as incorrect

reporting of the attribute values, and to inject the noise the attribute values are randomly replaced with some probability.

In work by Li et al. (2007), the author mentioned that real world problems often contain noise. There are two important challenges in the cluster community; "the first is how to obtain correct clustering from noisy data and the second is how to obtain the correct clustering from noisy data and remove the noise". This work considered the second clustering problem, and the algorithm is designed such that it can group the noise points in another group of clusters. Thus, the algorithm works based on the type of noise that is added to the data as additional data, which is sometimes called background noise. Another algorithm that works for separation of noise from the data is the work by (Zaharie and Zamfirache, 2005).

The different concepts of noise used in experiments depends on the clustering algorithms. In this work, we randomly select different subsets of the feature space, and for each subset we calculate the distances of the instances to the cluster centers which use feature values. Therefore, it is logical that our method is robust to the noise that is injected into some features, because if one feature is noisy because of the selection of different feature subsets, that noisy feature could be selected only a small number of times, or maybe not be selected at all. Even if the noisy feature is selected for some feature subsets, because Inner Ensemble works by selecting N different feature subsets, for which N is the ensemble size, the other feature subsets that do not contain that particular noisy feature may compensate for the effect of that feature. Thus, our method works for this kind of noise, and for our experiments we chose to inject the noise to the attributes.

To introduce the noise to the datasets, we selected 50% of the most significant features, using the Information Gain measure, then added different levels of noise to the values of the selected attributes. More specifically, N% noise means that we randomly select N% of data instances, and then change the attribute values of those instances for 50% of the most significant attributes. If an attribute is numerical, its values are replaced by a randomly generated number drawn from the uniform distribution spanning the maximal and minimal values of that attribute (Abbasian et al., 2010).

For our experiments, we chose the datasets from the previous section for which both IE-K-Means and cluster ensemble improved the performance, because for noisy data we expect similar behaviour here. We also added three different noise levels, 20%,40% and 60%, and then followed a similar procedure to that in figure 4.3. More specifically, first we add noise, then we find the best parameters on 4/5 of the data and use these on 1/5 of

Table 4.6: Comparison of K-Means, Inner Ensemble K-Means and Ensemble K-Means in terms of NMI Average on Different Noise Levels.

Noise	Dataset	K-M	IE-K-M	S-Lnkg	A-Lnkg	C-Lnkg	CSPA	MCLA	HGPA
	Hear-S	0.04	0.09	0.05	0.07	0.07	0.14	0.1	0.03
Avg	**Iris**	0.39	0.48	0.42	0.47	0.49	0.47	0.48	0.41
on	**Letter**	0.59	0.61	0.57	0.62	0.62	0.68	0.64	0.61
20%,	**Optdig**	0.53	0.55	0.51	0.57	0.55	0.55	0.54	0.52
40%	**Pendig**	0.53	0.55	0.5	0.56	0.55	0.54	0.53	0.51
and	**Pima**	0.02	0.04	0.04	0.05	0.05	0.06	0.04	0.02
60%	**Segment**	0.28	0.33	0.3	0.4	0.39	0.39	0.32	0.31
Noise	**Vowel**	0.26	0.31	0.25	0.32	0.32	0.53	0.42	0.28
Level	**Wine**	0.2	0.35	0.22	0.3	0.29	0.42	0.27	0.2
	Average	0.32	0.37	0.32	0.37	0.37	0.42	0.37	0.32
	[W D L]		[9 0 0]	[5 0 4]	[9 0 0]	[9 0 0]	[9 0 0]	[8 1 0]	[4 2 3]

Table 4.7: Friedmans test results for comparing K-Means and each cluster ensemble with IEKM on noisy data (average on all noise levels) for NMI measure.

	K-Means	IE-K-Means	S-Linkage	A-Linkage	C-Linkage	CSPA	HGPA	MCLA
F-Stat			12.17	14.00	14.00	14.39	12.17	12.17
Q-Value	3.18		2.82					
	2.83			0.70				
	2.83				0.70			
	2.71					0.94		
	3.18						2.82	
	3.18							0.35
Avg Rank	2.56	1.06	2.39					
	3.00	1.67		1.33				
	3.00	1.67			1.33			
	3.00	1.72				1.28		
	2.56	1.06					2.39	
	2.94	1.44						1.61

the data, and repeat this ten times on 1/5 of the data. Finally, we average the results for five different folds. For all the experiments we used Friedmans test with $alpha = 0.05$, and for post-hoc testing we used Nemenyis test with $alpha = 0.05$. For every experiment in this section we report the average results on all the noise levels, and show the detailed results for each noise level in appendix A. The statistical test was also performed, similar to with the clean data.

Table 4.6 compares the differences between K-Means, Inner Ensemble K-Means and Ensemble K-Means (G-Based, HAC) in terms of the NMI average for different noise levels. The table shows that only CSPA is more robust than IEKM. More specifically, IEKM is more robust than HGPA and Single-Linkage, and its performance is similar to the rest of the cluster ensembles. Also, IEKM is more robust than the original K-Means

Table 4.8: Comparison of K-Means, Inner Ensemble K-Means and Ensemble K-Means in terms of purity Average on Different Noise Levels.

Noise	Dataset	K-M	IE-K-M	S-Lnkg	A-Lnkg	C-Lnkg	CSPA	MCLA	HGPA
	Hear-S	0.61	0.66	0.6	0.62	0.63	0.7	0.65	0.59
Avg	**Iris**	0.67	0.73	0.67	0.71	0.72	0.74	0.74	0.71
on	**Letter**	0.37	0.37	0.37	0.39	0.38	0.48	0.38	0.38
20%,	**Optdig**	0.52	0.54	0.49	0.56	0.54	0.56	0.54	0.52
40%	**Pendig**	0.52	0.54	0.48	0.56	0.55	0.55	0.54	0.52
and	**Pima**	0.66	0.67	0.66	0.67	0.67	0.67	0.67	0.66
60%	**Segment**	0.42	0.44	0.41	0.47	0.47	0.48	0.45	0.43
Noise	**Vowel**	0.27	0.3	0.27	0.31	0.31	0.48	0.37	0.3
Level	**Wine**	0.57	0.67	0.56	0.63	0.64	0.65	0.63	0.6
	Average	0.51	0.55	0.50	0.55	0.56	0.59	0.55	0.52
	[W D L]		[8 1 0]	[0 4 5]	[9 0 0]	[9 0 0]	[9 0 0]	[9 0 0]	[5 3 1]

on all of the datasets, similar to Average-Linkage, Complete-Linkage and CSPA.

As with clean data, we performed a statistical test to compare Inner Ensemble with each ensemble method and K-Means for NMI measure; this is shown in table 4.7. From the table we can see that Friedmans statistic for all the ensemble methods is larger than six, which is the critical value for three models. This means that we can reject the null hypothesis. The critical value for the Nemenyi test for three models is 2.34. According to the table, in the K-Means column all the $q-values$ are larger than 2.34, which means that IEKM performs significantly better than K-Means on average over all the datasets. The table also shows that, with the exception of Single-Linkage and HGPA, the $q-value$ for all the other cases is less than 2.34. This means that, except for Single-Linkage and HGPA, for which IEKM performs significantly better, in all the other cases its average performance is similar to ensemble methods, over all noise levels. The table also shows the average rank of the algorithms. In three out of six situations IEKM has the best rank, and in three situations it ranks second after ensemble methods.

Table 4.8 compares the differences between K-Means, Inner Ensemble K-Means and Ensemble K-Means (G-Based, HAC), in terms of average purity at different noise levels. From this, in terms of the performance IEKM is superior to Single-Linkage and HGPA, it is equal to Average-Linkage and MCLA, and it is less robust than Complete-Linkage and CSPA. Also, IEKM works better than K-Means on eight of nine datasets, and comparably on one of nine. However, Average-Linkage, Complete-Linkage, CSPA and MCLA work better than original K-Means on all nine of nine datasets.

As with clean data, we performed statistical tests to compare Inner Ensemble with each ensemble method and K-Means, for the purity measure; this is shown in table 4.9.

Table 4.9: Friedmans test results for comparing K-Means and each cluster ensemble with IEKM on noisy data (average on all noise levels) for purity measure.

	K-Means	**IE-K-Means**	**S-Linkage**	**A-Linkage**	**C-Linkage**	**CSPA**	**HGPA**	**MCLA**
F-Stat			12.06	12.39	12.17	14.39	12.39	9.56
Q-Value	2.24		3.41					
	2.71			0.58				
	2.83				0.35			
	2.24					1.53		
	3.06						1.88	
	2.71							0.58
Avg Rank	2.17	1.11	2.72					
	2.94	1.67		1.39				
	2.94	1.61			1.44			
	2.94	1.89				1.17		
	2.67	1.22					2.11	
	2.94	1.67						1.39

In the table the Friedmans statistic, the average rank of Friedmans test for all of the datasets, and the Nemenyis $q - values$ from the comparison of each method with Inner Ensemble, are shown under each method.

From the table, we can see that Friedmans statistic for all the ensemble methods is larger than six, which is the critical value for three models. This means that we can reject the null hypothesis. The critical value for the Nemenyi test for three models is 2.34. The table shows that, except for the comparison of IEKM with Single-Linkage and CSPA, the $q - values$ are larger than 2.34, which means that for those cases IEKM performs significantly better than K-Means on average over all the datasets. When we compare IEKM with Single-Linkage, although IEKM performs better than both K-Means and Single-Linkage, the statistical results show that it is not significantly different than K-Means, but significantly better than Single-Linkage. This is because we expect the performance of cluster ensemble to be better than IEKM and original K-Means, but here Single-Linkage performs even worse than original K-Means, and for average ranking IEKM is closer to the original K-Means. This is why the statistical test shows that IEKM does not improve the performance significantly. The table also shows the average rank of the algorithms. In two of six situations IEKM has the best rank, and in four it ranks second after ensemble methods. Overall, in all cases the IEKM performance is not significantly different than that of cluster ensemble methods, but significantly better than original K-Means.

In conclusion, for both NMI and purity IEKMs performance in all the cluster ensemble methods is not significantly different than cluster ensembles, and it usually (six of six

for the NMI measure, and four of 6 for the purity measure) performs significantly better than the original K-Means. In addition, except for one of the cluster ensembles (CSPA), compared to the other cluster ensembles its performance is similar or better for noisy domains. This is consistent with our hypothesis, and with our expectation that IEKM maintains the advantages of cluster ensemble over the original K-Means to some extent; in this case with respect to performance and robustness to noise.

4.4.4 Memory Usage

For space complexity, for K-Means clustering we have an array of $O(n)$ that contains the label of each of the clusters for which n is the number of instances. We also require $O(k)$ additional memory cluster centers. And while running the Inner Ensemble K-Means we need to vote for each data instance. Thus, for each data instance we need k memory in order to see how many times that data instance is assigned to each cluster. Thus, the total memory we need is $O(k \times n)$. For all the cluster ensemble methods, we need the same amount of memory as for K-Means for each cluster ensemble member. As well, for HAC methods the similarity matrix must be calculated, which requires an additional $O(n^2)$ of memory. Finally, for HAC methods, another clustering is performed on the similarity matrix, which requires another $O(n)$ of memory for the final labels. Thus, the total memory needed for HAC methods is $O(n) + O(n^2) + O(n) = O(n^2)$ (Duda and Hart, 1973).

Also, for the G-Based method, memory is needed for the hyper-graph representation. According to Chapter 2, if we have k clusters and n data instances, and L different clusterings for which L is the ensemble size, then the number of rows in the matrix for the hyper-graph is n, and the number of columns is $L \times k$; thus, the memory needed is $O(nLk)$. Finally, the total memory for G-Based methods is $O(n)$ for each ensemble member, plus $O(nLk)$ for the consensus function, and $O(n)$ for the final labels; this is equal to $O(nLk)$. As seen, the memory complexity of Inner Ensemble K-Means is lower than both the G-Based and HAC methods. This is particularly evident with online clustering, since each instance will be assigned to the best cluster available, and this is done in every iteration. Thus, for online K-Means the amount of memory is $O(1)$, because a single instance is assigned to a cluster center every time. Now, if we have the online version of Inner Ensemble K-Means, in order to assign a data instance to the cluster centers for each iteration we need $O(k)$ memory, to see which cluster center is best for the inbound instance. With the cluster ensemble method we cannot cluster

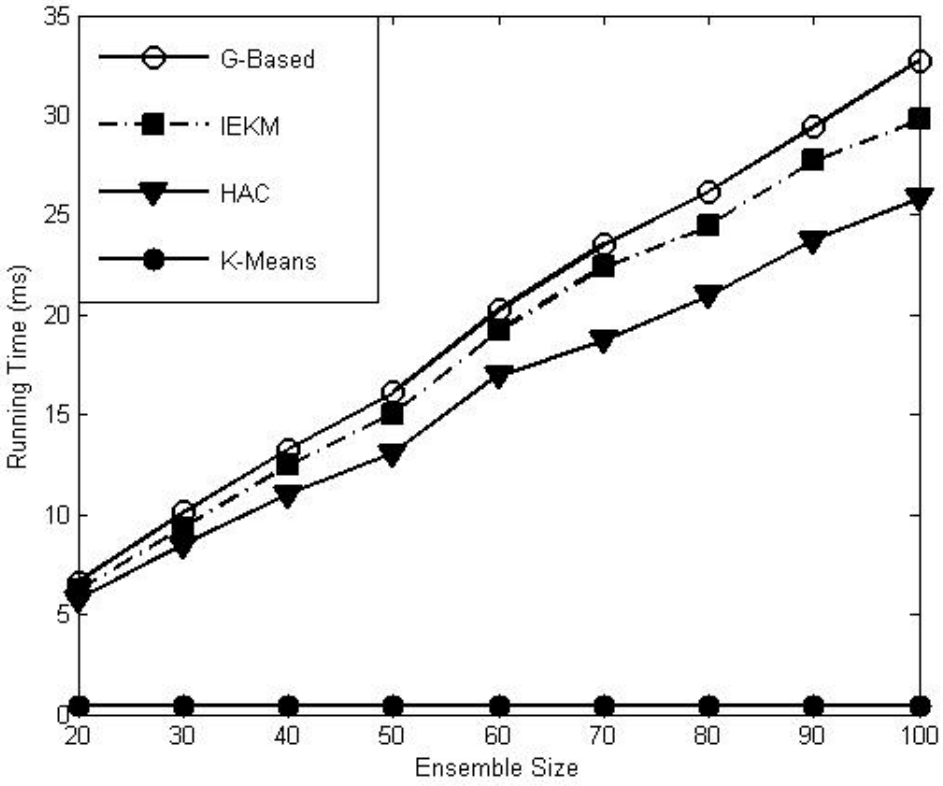


Figure 4.4: Compariosn of Running Time for K-Means, Inner K-Means, HAC and G-Based (mS).

instances in an online manner, because different ensemble members must be online, and the similarity matrix and hyper-graph representation must be updated online. This is not practical, because either their size grows as the number of instances grows in an online manner, or, even if we can keep m most recent instances, we still need $O(m^2)$ and $O(mLk)$ memory. And if it is possible, a HAC or G-Based method still needs to work on the similarity matrix or hyper-graph representation, which is done in offline manner. For all these reasons, applying cluster ensembles in an online manner is very difficult if not impossible, and it requires a similar amount of memory. Therefore, for online applications the memory complexity of Inner Ensemble is significantly less than cluster ensemble. In addition, Inner Ensemble K-Means can be implemented for online clustering, which is another advantage over cluster ensemble.

4.4.5 Running Time

In this section, we compare the time complexity of IEKM with K-Means and cluster ensembles. We ran the original K-Means, Inner Ensemble K-Means, HAC and G-Based methods on every dataset ten times, and for nine different ensemble sizes from 20 to 100. For each run we calculate the running time, and averaged these over all the datasets for comparison . We also averaged over Single, Average and Complete Linkage for HAC and MCLA, CSPA and HGPA for the G-Based method. Figure 4.4 compares the running times of K-Means, Inner Ensemble K-Means, HAC and G-Based. The figure shows that original K-Means is faster than all the other methods as expected. Also as expected, Inner Ensemble K-Means is faster than the G-Based method, but not faster than HAC.

The speed advantage of Inner Ensemble K-Means can be seen in online clustering. As mentioned in the previous sub-section, with online clustering each instance will be assigned to the best cluster as it is available, and this is done in every iteration. Thus, with online K-Means, for each incoming data instance we have k different comparisons for each cluster center, that is $O(k)$, and for Inner K-Means we have k comparisons r times, for which r is ensemble size. Therefore, the complexity is $O(rk)$. With the cluster ensemble method, we cannot cluster instances in an online manner, which means that for each incoming data instance we have to keep M most recent instances, and cluster these using G-Based or HAC; all this will not be done in a reasonable time. Thus, one advantage of Inner K-Means over cluster ensemble is that it can be applied for online clustering.

4.5 Summary

In this chapter, we argued that using cluster ensembles has many advantages, such as robustness to noise and better performance. And we noted that it has problems as well, one of which is that by using ensemble of K-Means we lose the prototypes which provide many benefits, including data summarization, data compression and efficient finding of nearest neighbours. Other disadvantages are it requires large amounts of memory and time, which makes it very hard if not impossible to use cluster ensembles for online clustering. To address these issues, in this chapter we apply Inner Ensemble to K-Means clustering, and call the resulting algorithm Inner Ensemble K-Means (IEKM). By using IEKM we retain the prototypes, and use significantly less memory than cluster ensemble methods. The experiments showed that generally, on clean data averaged over all the

datasetsIEKM, 1)performs significantly better than original K-Means, and 2) performs comparably to cluster ensemble. In addition, experiments on noisy data showed that IEKM generally 1) performs significantly better than original K-Means, and 2) except for one cluster ensemble (CSPA), IEKM performs comparably to cluster ensemble. We also compared the running time of IEKM to cluster ensembles and original K-Means. For offline clustering, we found that IEKM does not perform faster than cluster ensemble. However, we argued that for online clustering, the performance of the cluster ensemble will be the same in terms of speed, because it needs to calculate different clusterings for each incoming data, and combine the partitions to get the final clustering. However, with IEKM each incoming data is assigned to the best cluster. To do this, IEKM needs only r comparisons for each K cluster centers, which is $r \times K$ comparisons, significantly less than its performance with offline data, for which it needs this degree of comparison for each single data, and each iteration. Therefore, for online clustering the complexity of IEKM decreases significantly, while for cluster ensemble it remains the same, which means we can argue that it performs significantly faster than cluster ensembles. Finally, in terms of memory, we argue that IEKM has less complexity than cluster ensembles.

Chapter 5

Applying Inner Ensembles to other Algorithms

5.1 Introduction

So far in this work we have proposed a framework for applying Inner Ensembles to different algorithms, and explored this idea on two algorithms from supervised and unsupervised learning. One way to extend this idea is to apply Inner Ensembles on more algorithms from different categories, and we suggested a possible way to extend it to eight more algorithms. However, what we are proposing here is only speculation, and potential ways to use Inner Ensembles on different algorithms; we do not have experimental results to confirm the methods. But it does demonstrate how we can use Inner Ensembles on other algorithms. Table 5.1 summarizes this idea, including the name of the algorithm, the decision maker inside the algorithm and a possible way to apply ensembles on the decision maker. We attempted to cover a range of algorithms; 1) For supervised learning, we considered neural network, naive Bayes, K-Nearest Neighbor and. 2) For unsupervised learning: Hierarchical clustering, conceptual clustering (COBWEB) and one of the best known online clustering methods, CluStream. 3) Another Learning Paradigm: Association Rules learning; A-Priori. 4) In addition to showing how broadly this idea is applicable, we introduced it for two other algorithms besides supervised and unsupervised learning. These algorithms are a pair of methods to prune the decision tree. This idea also can be extended in other ways, such as using different ensemble methods like Boosting rather than methods similar to Bagging. To make following the general guideline easier, in this chapter, we repeat it again in algorithm 17.

Algorithm 17 General guidelines for using Inner Ensembles

1: **Locate a decision maker inside the algorithm.**
2: **Find a measure based on which that decision maker works.**
3: **Indicate the input and output of the measure.**
4: **Apply the ensemble on the measure:**
5: Change the input of the measure in different ways. {It can be sampling of data or feature based modifications or other methods.}
6: Generate an output based on each input.
7: Combine the outputs
8: **Apply the output of the ensemble for decision making.**

5.2 Neural Network

Here, we show how to apply Inner Ensembles to the artificial neural network. McCulloch and Pitts attempted to form an abstract mathematical model of a neuron (Figure 5.1). In the figure, the neuron has one input layer, a summation unit and a threshold unit. The combination of the summation and threshold units is called a node (Gose et al., 1996). In the diagram of the neural network, the nodes are indicated by an open circle. The output of a neuron is 1, if:

$$\sum_{i=1}^{M} w_i x_i > T. \tag{5.1}$$

for which x_i is ith input, and w_i is its corresponding weight. If the artificial neurons are trainable classifiers they are called perceptrons. The neural network has different layers: the first layer is the input layer and the last is the output layer. It can have only input and output layers, or it could have hidden layers of nodes between the input and output layers. Many researchers continue to develop different types of multi-layer networks, some of which replaced the threshold function with another S-shape curve function, called the sigmoidal function. One of the simplest and most general ways to train the neural network is *back propagation* algorithm, which is based on two main steps. The first is the feed-forward step, in which the output of the nodes is calculated starting at layer 1 and working to the output layer K. The second is the back propagation step, which updates the weights of the network, and begins at layer K and works backward to layer 1. The back propagation is shown in algorithm 18 (Gose et al., 1996).

According to the general guidelines to apply Inner Ensembles, we must first locate a

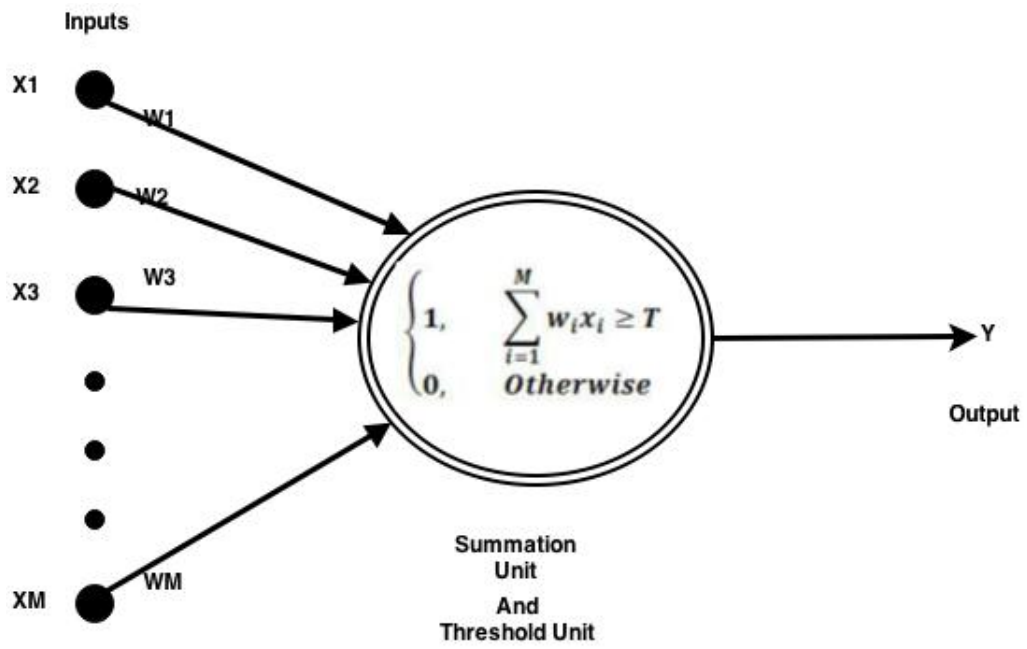


Figure 5.1: The Mode of The Neuron (Gose et al., 1996) According to the Mathematical Model by (McCulloch and Pitts).

decision maker inside the algorithm; each node is a decision maker which decides about the weights of the network during the learning step (line 7 of algorithm 18). The output of each node is calculated using:

$$x_j^{(k+1)} = R(\sum_{i=1}^{M_k} w_{ij}^{(k+1)} x_i^{(k)}) \tag{5.2}$$

For which x_j^k is the output of node j in layer k; M_k is the number of nodes in layer k; w_{ij}^k is the weight of the connection from node i in layer $k-1$ to node j in layer k and R is a sigmoid function:

$$R(s) = \frac{1}{1+e^{-s}}. \tag{5.3}$$

This is the measure that the performance of the decision maker is based on. The input of each node is the output of the nodes of the previous layer, each of which has a weight. We then need to change the input of the decision maker, and since it does not work based on a group of data instances, we cannot sample the data to change the input. Instead, the decision maker decides based on different subsets of the input, instead of all the inputs. It does this by using different methods of sampling the input, which can

Algorithm 18 Back Propagation Algorithm.

Initialize the weights w_{ij} to small random values.
repeat
  for n= 1 to N **do**
    First layer of the network $(x_1^{(0)}, ..., x_{M_0}^{(0)})$ is set to features of sample n.
    Feed-Forward

    $x_j^{(k+1)} = R(\sum_{i=1}^{M_k} w_{ij}^{(k+1)} x_i^{(k)})$ for $0 \leq k \leq K-1$, $1 \leq j \leq M_{k+1}$.
    Back-Propagation

    $\delta_j^{(k)} = x_j^{(k)}(1 - x_j^{(k)})(x_j^{(k)} - d_j)$. for $1 \leq j \leq M_k$
    $\delta_j^k = x_j^k(1 - x_j^k) \sum_{j=1}^{M_{k+1}} \delta_j^{k+1} w_{ij}^{k+1}$ for $K-1 \leq k \leq 1$, $1 \leq i \leq M_k$.
    $w_{ij}^{(k)} = w_{ij}^{(k)} - c\delta_j^{(k)} x_i^{(k-1)}$ for all i, j, k.
  end for
until $w_{ij}^{(k)}$ do not change significantly

be with or without replacement with different sample sizes. This is similar to what we had for Inner K-Means clustering. The outputs are then calculated and combined using averaging. So, instead of equation 5.2, we will have:

$$x_j^{(k+1)} = \frac{1}{E} \sum_{e=1}^{E} x_{je}^{(k+1)} \tag{5.4}$$

For which E is ensemble size and $x_{je}^{(k+1)}$ is the output of the node x in layer $(k+1)$ when its input is the eth subset of nodes in layer (k). Then after applying Inner Ensembles we have new Inner Ensemble Back Propagation methods which is shown in algorithm 19.

5.3 Naive Bayes

One of the most practical Bayesian learning methods is the *Naive Bayes* classifier. This classifier is applied on the domains in which each data instance is represented by a conjunction of attribute values. The Bayesian approach to classifying an instance is to find the most probable class, given the different attribute values. Assuming that the class can have different sets of values V, the Bayesian method classifies a data instance as:

Algorithm 19 Inner Ensemble Back Propagation Algorithm.

Initialize the weights w_{ij} to small random values.
repeat
  for n=1 to N **do**
    First layer of the network $(x_1^{(0)}, ..., x_{M_0}^{(0)})$ is set to features of sample n.
    Feed-Forward

    For $k = 0, ..., K-1$, compute the following for nodes $j = 1, ..., M_{k+1}$.
    $x_j^{(k+1)} = 0$
    for e=1 to E **do**
      Sample the nodes of the layer k: S_e. $\{S_e$ contains a subset of nodes in layer $k\}$
      for each S_e compute: $x_j^{(k+1)} = x_j^{(k+1)} + R(\sum_{i=1}^{M_{ke}} w_{ij}^{(k+1)} x_{ei}^{(k)})$
    end for
    $x_j^{(k+1)} = \frac{x_j^{(k+1)}}{E}$
    Back-Propagation

    $\delta_j^{(k)} = x_j^{(k)}(1 - x_j^{(k)})(x_j^{(k)} - d_j)$. for $1 \leq j \leq M_k$
    $\delta_j^k = x_j^k(1 - x_j^k) \sum_{j=1}^{M_{k+1}} \delta_j^{k+1} w_{ij}^{k+1}$ for $K - 1 \leq k \leq 1$, $1 \leq i \leq M_k$.
    $w_{ij}^{(k)} = w_{ij}^{(k)} - c\delta_j^{(k)} x_i^{(k-1)}$ for all i, j, k.
  end for
until $w_{ij}^{(k)}$ do not change significantly

$$v = \underset{v_i \in V}{argmax} P(v_i | a_1, a_2, ..., a_n) \tag{5.5}$$

For which a_is are the attributes. By applying the Bayesian rule we will have:

$$v = \underset{v_i \in V}{argmax} P(a_1, a_2, ..., a_n | v_i) P(v_i) \tag{5.6}$$

Calculating $P(v_i)$ can simply be done by counting each v_is. However, calculating $P(a_1, a_2, ..., a_n | v_i)$ is very difficult, and requires a huge training dataset. Naive Bayes is based on the assumption that all the attributes of the dataset are independent. Using this conditionally independence assumption, we have:

$$P(a_1, a_2, ..., a_n|v_i) = \prod_j P(a_j|v_i) \tag{5.7}$$

And the Naive Bayes classifier is defined as:

$$v = \underset{v_i \in V}{argmax} P(v_i) \prod_j P(a_j|v_i) \tag{5.8}$$

Thus, Naive Bayes is a learning method that estimates $P(v_i)$ and $P(a_i|v_i)$ during the learning step, based on the training dataset. Then, during the testing step, the unknown instance is classified using equation 5.8 and estimated terms (Mitchell, 1997).

Discriminative classifiers learn the posterior probability for classification directly, while generative classifiers estimate the joint probability distribution, and then use the Bayesian rule for classification (Kaizhu Huang, 2005). One of the generative models is the naive Bayes method, which uses the frequency estimate of $P(a_i|v_i)$ by simply counting the data which maximize the likelihood. Several researchers have investigated the discriminative training of the generative models for Naive Bayes and Bayesian networks (Kaizhu Huang, 2005; Su et al., 2008). More specifically, they tried to select data instances on the boundary, and then update the $P(a_i|v_i)$ using these instances. Therefore, to use Inner Ensembles on Naive Bayes we try to find the examples on the boundary, using Boosting. The first step is to find a decision maker, and here with Naive Bayes we can consider each $P(a_i|v_i)$ to be a decision maker. The input of the decision maker is a group of data, and the output is the estimate of $P(a_i|v_i)$. For each decision maker we change the input using Boosting, and during each iteration we focus on the hard part of the data, according to the a_i attribute. Finally, the output of the decision maker is calculated for each sampling of the data, and the final output is calculated using weighted average. Algorithm 20 shows this procedure.

According to the algorithm, in line 6 we have different iterations which is our ensemble sizes , and in line 7, S_k is the weighted sampling of the dataset. In line 8 $P_k(a_i|v_j)$ is estimated according to S_k, and in line 9 $P(v_j)$ is estimated according to the original data, because the decision maker we would like to estimate is $P(a_i|v_j)$. In line 10 the data is classified using just one attribute, a_i, and the weights of the data instances will change according to the misclassified instances. In line 15 the sampling S_k is added to the ensemble D. Then, in the combining part of the algorithm, $P_k(a_i|v_j)$ is estimated for each of the ensemble members S_k, and a final $P(a_i|v_j)$ is estimated using the weighted average. This algorithm is applied on every attribute.

Algorithm 20 Inner Naive Bayes to Calculate $P(a_i|v_1), P(a_i|v_2), ..., P(a_i|v_c)$.

TRAINING:
Z is the original dataset.
$w^1 = [w_1^1, ..., w_N^1], w_j^1 \in [0,1], \sum_{j=1}^{N} w_j^1 = 1, w_j^1 = \frac{1}{N}$.
$D = \emptyset$,D: A set of samples.
L is the ensemble size.
for k=1 to L **do**
  $S_k = Sample(Z, w^k)$
  Calculate $P_k(a_i|v_j)$ according to S_k for $j = 1, ..., c$.
  Calculate $P(v_j)$ according to the original data Z for $j = 1, ..., c$.
  Classify S_k using $v = \underset{v_j \in V}{argmax} P(v_j) P_k(a_i|v_j)$
  $\epsilon_k = \sum_{j=1}^{N} w_j^k l_k^j$, $l_k^j = 1$ if z_j is missclasified otherwise $l_k^j = 0$
  if $\epsilon_k = 0$ or $\epsilon_k \geq 0.5$ **then**
    Ignore S_k
  else
    $\beta = \frac{\epsilon_k}{1-\epsilon_k}$ where $\epsilon_k \in [0, 0.5]$, $w_j^{k+1} = \frac{w_j^k \beta_k^{1-l_k^j}}{\sum_{i=1}^{N} w_i^k \beta_k^{1-l_k^i}}$, $j = 1, ..., N$.
    $D = D \cup S_k$.
  end if
end for
return D and $\beta_1, ..., \beta_L$.
COMBINING:
For each S_k in D,$P_k(a_i|v_j)$s are calculated.
$P(a_i|v_j) = \sum_{k=1}^{L} Ln(\frac{1}{\beta_k}) P_k(a_i|v_j)$ for$j = 1, .., c$.

5.4 K-Nearest Neighbor

Single Nearest Neighbour simply classifies an unknown data sample to the same class as its closest data instance in the training data. The closeness of the data is calculated by different distance measures, such as *Euclidean, Manhattan, Maximum* distance. A more general version of nearest neighbour is called *K-Nearest Neighbour* which, instead of classifying an unknown data based on the closest training data instance, it finds the K closest data instances in the training set, and classifies the unknown data to the same class as most of the K closest data instances this procedure is shown in algorithm 21 (Gose et al., 1996).

Algorithm 21 K-Neearest Neighbour Algorithm

1: **for** Each instance X **do**
2: Calculate the distance to all of training instances Y using $d_e(X,Y) = \sqrt{\sum_{i=1}^{n}(X_i - Y_i)^2}$ {Different measures can be used.}
3: The class of instance X, C_X is calculated as the label which is the most frequent among the K nearest training samples
4: **end for**

There are different ways to apply Inner Ensembles for KNN. We can simply sample the dataset, and find K nearest neighbours. However, this method has two drawbacks. In the case of nearest neighbour, it gives us an ensemble of nearest neighbours; in fact, in this case, Inner Ensemble is exactly equal to the usual ensemble methods, and so we have nothing new. The other problem is the *curse of dimensionality*, that is, when the dimensionality increases the error rate increases rapidly. Some researchers believe that the problem with distance measures is that they give similar weights to all of the features. The literature shows there have been efforts to solve this problem by assigning different weights to different features (Kohavi et al., 1997). However, to apply Inner Ensembles here it makes more sense to try to change the input by changing the feature space. More specifically, similar to what we did for K-Means clustering, we sample the feature space. Following the general guidelines, we first locate a decision maker for KNN that is deciding which is the closest training data to the unknown data. The decision maker works with different measures, and here we assume Euclidean distance is the measure:

$$d_e(X,Y) = \sqrt{\sum_{i=1}^{n}(X_i - Y_i)^2} \tag{5.9}$$

for which X and Y are two data points, and n is the total number of features. The next step is to determine the input and output of the decision maker. The input is data points, and the output is the distance between the data points. To change the input we sample the feature space, which can be with or without replacement, with different sample sizes. Then the distance between two points is calculated for each of the feature subsets, and the final distances are combined. To combine the distances we can either vote, or we can average all the distances. For example, using each subset of features we can find the K closest neighbours, and then vote to find the final K closest neighbour; or we can just average the distances and, according to the average, find the final K closest neighbour. If we use averaging, then the distance between two data points is calculated

Algorithm 22 Inner Ensemble K-Nearest Neighbour Algorithm

for Each instance X **do**
  Calculate the distance to all of training instances Y using
  $d_e(X,Y) = 0$
  for j=1 to N **do**
    Sample the feature space: F_j.
    $d_e(X,Y) = d_e(X,Y) + \sum_{j=1}^{N} \sqrt{\sum_{i=1}^{n_j}(X_{F_j i} - Y_{F_j i})^2}$
  end for
  $d_e(X,Y) = \frac{1}{N} d_e(X,Y)$
  The class of instance X, C_X is calculated as the label which is the most frequent among the K nearest training samples
end for

as:

$$d_e(X,Y) = \frac{1}{N} \sum_{j=1}^{N} \sqrt{\sum_{i=1}^{n_j}(X_{F_j i} - Y_{F_j i})^2} \tag{5.10}$$

For which N is the ensemble size, F_j is the jth feature subset that is created by sampling of feature space and n_j is the number of features in jth feature subset. The result is Inner Ensemble K-Nearest Neighbour which is shown in algorithm 22.

5.5 Hierarchical Clustering

In Chapter 2, we explained hierarchical clustering in detail; how it works on different measures, and has significant time and space complexity. Using Inner Ensembles for hierarchical clustering has two advantages over ensemble methods. First, it uses less space. For all cluster ensemble methods we need the same amount of memory as hierarchical clustering, and for the HAC combining method we need $n \times n$ additional memory, for which n is the number of data instances. In addition, for G-Based methods we need extra memory for hyper-graph representation. However, with Inner Ensembles we need only the same amount of memory as one hierarchical method.

There is more motivation to use online clustering methods. Although hierarchical clustering cannot be used for online methods, researchers have tried to create online versions of hierarchical clustering (El-Sonbaty and Ismail, 1998; Ribert et al., 1999).

Algorithm 23 General Agglomerative Clustering.

Input: $X_i, i = 1, 2, .., n$: Data instances
Begin with n clusters each containing on single instance.
for m=1 to n-1 **do**
  Find the most similar clusters C_k and C_j using $D(C_i, C_j) = \min_{a \in C_i, b \in C_j} d(a, b)$.
  Merge C_k and C_j.
end for

Algorithm 24 General Inner Ensemble Agglomerative Clustering.

Input: $X_i, i = 1, 2, .., n$: Data instances
Begin with n clusters each containing on single instance.
for m=1 to n-1 **do**
  Find the most similar clusters C_k and C_j using:
  $D(C_k, C_j) = 0$
  for m=1 to N **do**
    Sample the feature space {The result is X^m which is the data using only feature set m}
    $D(C_k, C_j) = D(C_k, C_j) + \min_{a \in C_k^m, b \in C_j^m} d(a, b)$.
  end for
  $D(C_k, C_j) = \frac{1}{N} D(C_k, C_j)$
  Merge C_k and C_j.
end for

As noted in the previous chapter, it is extremely difficult to use ensemble methods for online clustering, because the co-association matrix must be updated for every incoming data instance in each step, and all the ensemble members must work online. More importantly, the combining method, G-Based or HAC, must also be online. Thus, for online hierarchical clustering Inner Ensembles can use some of the advantages of ensemble methods, in cases that ensemble methods cannot be used. The general agglomerative hierarchical clustering is shown in algorithm 23 (Gose et al., 1996).

Using Inner Ensembles for hierarchical clustering is quite straightforward. Hierarchical clustering works by combining similar clusters at each level, based on a similarity measure. Thus, the decision maker will find the most similar clusters. This decision maker works with a different type of similarity measure. We focus on the minimum distance similarity measure, which results in a single linkage, and the rest of measures

can be treated in the same way. The minimum distance measure is:

$$D(C_i, C_j) = \min_{a \in C_i, b \in C_j} d(a, b). \tag{5.11}$$

The input for this measure is the members of the two clusters C_i and C_j, and the output is the distance between the two clusters. The input can be donein different ways, such as using different sampling of the clusters members, or sampling of the feature space. Then, for each k sampling we calculate the distance $D_k(C_i, C_j)$, and the final distance between two clusters is calculated by averaging (equation 5.12). The result is Inner Ensemble agglomerative hierarchical clustering which is shown in algorithm 24.

$$D(C_i, C_j) = \frac{1}{N} \sum_{i=1}^{N} D_i(C_i, C_j). \tag{5.12}$$

5.6 COBWEB

Conceptual clustering accepts a set of objects, and produces a classification scheme over the observations. Instead of pre-classifying objects, it is based on an evolution function that can discover the classes. COBWEB is a system for hierarchical conceptual clustering that incrementally creates a classification tree in which each node represents a concept. This classification tree can be used to predict missing attributes, or a class of new incoming objects (Fisher, 1987). It applies hill-climbing searching through the space of all of the hierarchical classifications to find the best classification method, and it uses four different operators, which allows COBWEB to search the space bi-directionally. The four operators are:

1. Placing an object in an existing class
2. Creating a new class
3. Merging
4. Splitting

For each incoming data instance, the best operator is selected and performed. The selection of the best operator is based on a measure that is called *Category Utility* which is defined as:

Algorithm 25 COBWEB algorithm.

```
1: FUNCTION COBWEB(Object,Root)
2: Update counts of the Root
3: if Root is a leaf then
4:    Return the expanded leaf to accomodate the new object.
5: else
6:    Find two children of the Root that best host the object: best_1, best_2.
7:    Perform one of the following which yields the best Category Utility (CU)
8:    a) newCategory(Object) .
9:    b) Merge(best_1, best_2), COBWEB(Object, Merge(best_1, best_2)) .
10:   c) Split(best_1), COBWEB(Object, Root).
11:   d) COBWEB(Object, best_1).
12: end if
```

$$CU = \frac{\sum_{k=1}^{n} P(C_k)[\sum_i \sum_j P(A_i = V_{ij}|C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2]}{n} \tag{5.13}$$

For which n is the number of categories, A_i is the ith attribute, V_{ij} is the jth value for ith attribute and C_k is the kth class. Algorithm 25 summarizes the COBWEB (Fisher, 1987).

There are different reasons to use Inner Ensembles for COBWEB clustering. The first and most important reason is to address the comprehensibility problem. COBWEB produces a classification tree, but using ensemble of COBWEB causes a loss of comprehensibility of the model. Using ensemble of COBWEB is also highly problematic because of two important characteristics of the method: the classification tree, and the fact that it incrementally allocates each input to each class. Therefore, using ensemble of COBWEB means we cannot have either the tree or the incremental ensemble method for clustering. Thus, if we want to take advantage of using ensemble of COBWEB, Inner Ensemble is an option. One problem of COBWEB is that it is very sensitive to the ordering of the instances and, as Fisher (1987) states,

"*Some orderings may require more than one instance of the same object to converge.*"

This suggests that we could use different sampling with replacement during the running of the algorithm, in order to have more than one instance of the same object, and decrease the sensitivity to the ordering of the data. It also suggests that during the running of the algorithm we may use different sampling with replacement to have more than

one instance of the same object to decrease the sensitivity to the ordering of the data. To use Inner Ensemble, the first step is to find the decision maker. Here, the decision maker will choose the best operator by placing the object in an existing class , creating a new class, merging or splitting. The decision making is based on the category utility in equation 5.13. The input of this measure is a set of data instances that has already been seen, and the output is an estimate of the qualityof the partition. The input can be changed in several ways, one of which is to use sampling of the data that has been seen so far, which can be with or without replacement. Assume we have L different samples of the data, for which L is the ensemble size. The category utility, $CU_1, CU_2, ..., CU,_L$ is calculated for each sample of data. There are two ways to make the final decision according to Inner Ensemble. The first is to simply average over different CU_ks to calculate the final CU, that is:

$$CU = \frac{1}{L}\sum_{i=1}^{L} CU_i. \tag{5.14}$$

The other way for the final decision is that for each category utility CU_k, make the best decision which is the best operator to do. Then using voting, the best operator to do is the one that is selected the most for different CU_ks.

$$Operator = argmax\{O_1, O_2, O_3, O_4\}. \tag{5.15}$$

For which O_k is the number of time that kth operator is selected as the best operator.

5.7 CluStream

In this section, we discuss how to use Inner Ensembles as an evolving stream data algorithm. We chose one of the best known clustering algorithms to cluster the evolving data streams (Aggarwal et al., 2003); the method is called *CluStream* framework. We thought of using Inner Ensembles with this algorithm when we were trying to use cluster ensemble for evolving stream data. Cluster ensemble is basically an offline method, so we could not apply it for stream data; even if we used a time window, the procedure for online clustering would be very slow. This method is generally based on a two-step procedure: the first step is online *Micro-Clusters* maintenance, and the second is called *Macro-Cluster* creation. The statistical information about the data locality is represented in terms of micro- clusters. The micro-clusters generated by the algorithm

Algorithm 26 CluStream Algorithm (Micro-Cluster Maintenance).

```
1: Off-line process starts with q initial set of micro-clusters.
2: while There is incomming data x do
3:    Find for x the closest micro-cluster c according to dist(C_j, X) {j = 1, ..., K and K
      is the number of micro-clusters}
4:    if x is closer to c than a threshold T then
5:       Update c to absorb x
6:    else
7:       Create a new micro-cluster for x
8:       if Any micro-cluster is considered as outlier then
9:          Delete the micro-cluster
10:      else
11:         Merge two micro-clusters
12:      end if
13:   end if
14: end while
```

serve as intermediate representations that can be maintained in an efficient way for online data. Then the representations are used by an offline component to create higher level clusters, known as macro-clusters, which can be understood by the user. In this section we focus on the first step which is shown in algorithm 26 (Aggarwal et al., 2003) [1].

The first step of micro-cluster maintenance is to create the initial q micro-clusters, using an offline process at the beginning of the data stream. Once these initial micro-clusters have been established, the online procedure is used to update them; this is where we can apply Inner Ensembles. Whenever a new data point arrives micro-clusters need to be updated, and each data point either joins an existing micro-cluster or creates a new micro-cluster of its own. At first, each data point joins to the closest micro-cluster. Here, as with K-Means, the decision maker must find the best micro-cluster to host the new incoming data. This decision maker is based on the distance between the center of the micro-cluster and the incoming data.

$$dist(C_j, X). \tag{5.16}$$

This value can be computed relatively easily. To use Inner Ensembles, we change the

[1] for more details of the algorithm refer to (Aggarwal et al., 2003)

Algorithm 27 CluStream Algorithm (Micro-Cluster Maintenance).

Off-line process starts with q initial set of micro-clusters.
while There is incomming data x **do**
  for i=1 to N **do**
    Sample the feature space S_i
    Find for x_{S_i} the closest micro-cluster c_i according to $dist(C_{jS_i}, X_{S_i})$ $\{j = 1, ..., K$ and K is the number of micro-clusters$\}$
  end for
  $c = max_{i=1toN} c_i$
  if x is closer to c than a threshold T **then**
    Update c to absorb x
  else
    Create a new micro-cluster for x
    if Any micro-cluster is considered as outlier **then**
      Delete the micro-cluster
    else
      Merge two micro-clusters
    end if
  end if
end while

input of the measure. Since we would like to calculate the distance quickly, similar to K-Means, we sample the feature vector. Again the sampling can be of different sizes, and with or without replacement. For each sampling S_i we find the closest micro-cluster, and the final closest micro-cluster is found by voting. The result is Inner Ensemble CluStream which is shown in algorithm 27

5.8 Pruning

In this section, to demonstrate how broadly this method can be applied, we show how to use Inner Ensembles on an algorithm that is not supervised or un-supervised learning. We choose two different pruning methods for the decision tree: *Cost Complexity Pruning* (Breiman et al., 1984) and *Reduced Error Pruning* (Quinlan, 1987).

Cost Complexity Pruning This pruning method has two steps. In the first step a sequence of trees $T_0, T_1, ..., T_k$, for which each T_{i+1} is the pruned version of the previous tree T_i in the sequence, is created by replacing one or more of its sub-trees with suitable leaves. The sub-trees that result in the lowest increase in the error rate per pruned leaf, according to the following measure, will be pruned.

$$\alpha = \frac{\varepsilon(pruned(T,t),S) - \varepsilon(T,S)}{|leaves(T)| - |leaves(pruned(T,t))|}. \tag{5.17}$$

for which S is a data instance, $\varepsilon(T,S)$ is the error of the tree T on sample S, $leaves(T)$ is the number of leaves in T, and $prunde(T,t)$ is the tree that is pruned by replacing node t with the appropriate leaf. This measure is calculated for all the samples S and the node or nodes with the minimum values of this measure will be pruned. Here to apply Inner Ensembles, we focus on the first step which is shown in algorithm 28(Breiman et al., 1984).

In the second step of the algorithm, the generalization error of each of the pruned trees, $T_0, T_1, ..., T_k$, is calculated, and the best pruned tree is selected. As can be seen, the output of pruning is a decision tree, not accuracy, which means we cannot use the usual ensemble method for pruning. However, to take advantage of using the ensemble method for pruning we can use Inner Ensemble, because it can be applied during the pruning step. Thus, the first step is to find a decision maker, which here is the decision about locating the best nodes to prune. This decision maker is based on equation 5.17. The input of the equation is a pruning set, and its output is a measure indicating the quality of the node. Then, to change the input of the measure we use L different samplings of the pruning set, which can be with or without replacement or with different sizes, for which L is the ensemble size, and for each sampling we calculate equation 5.17,α_i, for $i = 1, 2, ..., L$. The final measure is calculated by averaging. The result is Inner Ensemble Cost Complexity Pruning which is shown in algorithm 29.

$$\alpha = \frac{1}{N}\sum_{i=1}^{L}\alpha_i. \tag{5.18}$$

Reduced Error Pruning The Reduced Error Pruning method is one of the simplest methods of pruning a decision tree. There is no need to create a series of decision tress, $T_0, T_1, , T_k$. Instead, the method works by traversing the tree from the bottom up, and visiting each internal node. It checks each internal node to determine whether replacing the node with the most frequent class decreases the accuracy of the tree. If it does

Algorithm 28 The First Step of Cost Complexity Pruning

1: $T = \{T_0\}$ {T_0 is the original tree}
2: $i = 0$
3: **repeat**
4: Find the non-leaf subtree(s) of T_i with the minimum value of
5: $\alpha = \frac{\varepsilon(pruned(T_i,t),S)-\varepsilon(T_i,S)}{|leaves(T_i)|-|leaves(pruned(T_i,t))|}$.
6: Replace the subtree(s) by their respective best leaves in T_i and the result is T_{i+1}
7: $T = T \cup T_{i+1}$
8: $i = i + 1$
9: **until** T_{i+1} is a single leaf
10: **return** T

not reduce the accuracy, the node is pruned and replaced by the most frequent class. This procedure is repeated until further pruning decreases the accuracy (algorithm 30 (Quinlan, 1987)).

Using Inner Ensemble is similar to cost complexity pruning, as the decision maker decides if the current node should be pruned, under the assumption that there is separate data for pruning. This decision maker is based on the difference between accuracies, before and after the pruning of the node on pruning dataset.

$$Diff = \varepsilon(T, S) - \varepsilon(pruned(T, t), S). \tag{5.19}$$

for which S is the pruning set, and $pruned(T, t)$ is the pruned tree from which internal node t is replaced by the most frequent class. If $Diff \geq 0$,the node is pruned. The input for this measure is the pruned set, and the output is the difference between accuracies before and after pruning. Thus, we can change the input of the measure by sampling the pruning set. The output can be combined in different ways, one of which is to calculate the $Diff_k$ for each sampled prune set S_k, then average them over $Diff_k$s. The other way is to calculate $Diff_k$ for different sampled prune sets, and vote from the times that the accuracy does not decrease. If the accuracy for most of the S_ks is not decreased, the node is pruned. The result is Inner Ensemble Reduced Error Pruning which is shown in algorithm 31.

Algorithm 29 The First Step of Inner Ensemble Cost Complexity Pruning

$T = \{T_0\}$ $\{T_0$ is the original tree$\}$
$i = 0$
repeat
  Find the non-leaf subtree(s) of T_i with the minimum value of
  $\alpha = 0$
  for j=1 to L **do**
    $S_j = Sample(S)$
    $\alpha = \alpha + \frac{\varepsilon(pruned(T_i,t),S_j) - \varepsilon(T_i,S_j)}{|leaves(T_i)| - |leaves(pruned(T_i,t))|}$
  end for
  $\alpha = \frac{1}{L}\alpha$
  Replace the subtree(s) by their respective best leaves in T_i and the result is T_{i+1}
  $T = T \cup T_{i+1}$
  $i = i + 1$
until T_{i+1} is a single leaf
return T

5.9 A-Priori

In general, association rule mining is a two-step method. In the first step, the frequent item sets with frequencies at least equal to a predefined threshold, *minimum* support, are found . The next step is to use these frequent item sets to generate the association rules. Apriori algorithm is a method for mining frequent item sets for Boolean association rule mining. It is based on an iterative method, in which the $k + 1$-item set is found using the k-item set. First, a set of frequent 1-item sets are found by counting each item in the database, and selecting those with counts larger than minimum support; this set is

Algorithm 30 Reduced Error Pruning

repeat
  Visit each internal node from bottom up
  Calculate $Diff = \varepsilon(T, S) - \varepsilon(pruned(T, t), S)$
  if $(Diff \leq 0)$ **then**
    Replace the node by its best possible node.
  end if
until furthur pruning decreases the accuracy

Algorithm 31 Inner Ensemble Reduced Error Pruning

1: **repeat**
2: Visit each internal node from bottom up and for each node calculate the $Diff$:
3: $Diff = 0$
4: **for** i=1 to L **do**
5: $S_i = Sample(S)$ {L is the ensemble size}
6: $Diff = Diff + \varepsilon(T, S_i) - \varepsilon(pruned(T, t), S_i)$
7: **end for**
8: $Diff = \frac{1}{L} Diff$
9: **if** $(Diff \leq 0)$ **then**
10: Replace the node by its best possible node.
11: **end if**
12: **until** furthur pruning decreases the accuracy

called L_1. Next, L_1 is used to find L_2, which is the set of frequent 2-item sets. The procedure continues until no more frequent item sets can be found (Han et al., 2006).

According to this algorithm, Step 1 (line 6 of Algorithm 32 (Han et al., 2006)) finds the frequent 1-item sets, and then generates the candidates using item sets of the previous level (line 9).Then lines 10 to13 scan the database and count each of the candidates. Finally, all the candidates that satisfy minimum support (line 14) form the set of frequent item sets (line 17). For more details about the algorithm refer to (Han et al., 2006).

Many versions of the apriori algorithm have been proposed to improve the efficiency of the original algorithm(Han et al., 2006). One way to do this is through *sampling*. Using sampling, accuracy is traded off against efficiency. The basic idea of sampling is to pick a random sample S of the dataset D, and find the frequent item sets in S instead of D. In this way, all the data can be placed in memory and everything can be done in the main memory. Because the size of S is much smaller than D, the lower minimum support parameter is used for S. Then local frequent item sets of S are found (L^S), and the rest of the dataset D is used to find the actual counts for item sets in L^S. In this way, only one scan of the entire dataset D is needed to find the actual count of the item sets. However, as the item sets L^S might not be included in all of the global item sets the scan may not be accurate, and a second pass on the original dataset D to find missing frequent item sets could be required.

Using Inner Ensembles, we expect to increase the accuracy of the sampling method by finding more accurate local frequent item sets. In a sampling method, the support

Algorithm 32 Apriori Algorithm.

Input
D: A database of transactions.
$min - sup$: The minimum support count threshold.
Output
L,frequent item sets in D.
L_1=find frequent 1-item sets(D).
$k = 2$
while $L_{k-1} \neq \emptyset$ **do**
  C_k=generates the candidates using item sets of the previous level L_{k-1}
  for each transaction $t \in D$ **do**
    C_t is the subsets of t that are candidates
    each candidate $c \in C_t$ $count[c] = count[c] + 1$.
  end for
  $L_k = \{c \in C_k | count[c] \geq min - sup\}$
  $k = k + 1$
end while
Return $L = \bigcup_k L_k$

is smaller than the actual value, so the local frequent item sets might not actually be frequent. However, we can use Inner Ensemble for sampling to decide about the local frequent item sets. This decision maker is based on the count of the item sets. The input of the measure is a sample of the dataset, and the output is the count for each item set. Therefore, as explained, using Inner Ensemble as our sampling method requires using a sample S of a dataset, rather than the whole dataset D. Instead of only using S to count the number of item sets for S, we used different samples $S_1, S_2, ..., S_L$ from the dataset. Then the number of item sets in S, $count_k$, is calculated using each sample S_k, and the final number is determined using averaging over all $count_k$s. This procedure is more efficient than the original a priori, because we do not use the entire dataset to find the frequent item sets. However, it is also less efficient than the sampling method, because instead of using sample S for counting, we use different samples of the original dataset. In A priori, we traded off accuracy for efficiency, but here we try to trade off the accuracy less than with the sampling method. This procedure is shown in algorithm 33.

Table 5.1: Using Inner Ensembles on Different Algorithms.

Algorithm	**Decision Maker**	**Measure**	**Applying Inner Ensemble**
Neural Network	Nodes of the Network	$y = R(\sum_{i=1}^{M} w_i x_i)$	Different Subsets of the Input of the Node.
Naive Bayes	Each Feature a_i	$P(a_i\|v)$	Boosting to Calculate $P(a_i\|v)$
K-Nearest Neighbour	Closest Training Data to the Unknown Data	$d_e(X,Y) = \sqrt{\sum_{i=1}^{n}(X_i - Y_i)^2}$	Sampling of the Feature Space
Cost Complexity Pruning	Finding the best Node to Prune	$\alpha = \frac{\varepsilon(pruned(T,t),S) - \varepsilon(T,S)}{\|leaves(T)\| - \|leaves(pruned(T,t))\|}$.	Different Sampling of Pruning Set.
Reduced Error Pruning	Whether Prune The Current Node	$Diff = \varepsilon(T,S) - \varepsilon(pruned(T,t),S)$	Different Sampling of Pruning Set.
Hierarchical Clustering	Find the Most Similar Clusters to Combine	$D(C_i, C_j)$	Different Sampling of Clusters Members Or Different Sampling of the Feature Space
COBWEB	Choose The Best Operator	Category Utility Equation 5.13	Different Sampling of the Data
A-Priori	Local Frequent Item Sets	Local Count of the Item Sets	Different Samples of the Database
CluStream	Best Micro-Cluster That Hosts The Incoming Data	$dist(M_i, X)$	Different Sampling of the Feature Space

Algorithm 33 Inner Ensemble Sampling Apriori Algorithm.

Input
S: A sample of a database D.
$min - sup$: The minimum support count threshold for S.
Output
L,frequent item sets in S.
L_1=find frequent 1-item sets(S).
$k = 2$
while $L_{k-1} \neq \emptyset$ **do**
  C_k=generates the candidates using item sets of the previous level L_{k-1}
  for i=1 to L **do**
    $S_i = sample(D)$.
    for each transaction $t \in S_i$ **do**
      C_t is the subsets of t that are candidates
      for each candidate $c \in C_t$ $count_i[c] = count_i[c] + 1$.
    end for
  end for
  for each candidate $c \in C_t$ $count[c] = \frac{1}{L} \sum_{i=1}^{L} count_i[c]$
  $L_k = \{c \in C_k | count[c] \geq min - sup\}$
  $k = k + 1$
end while
Return $L = \bigcup_k L_k$

5.10 Summary

Applying Inner Ensembles to different algorithms requires some knowledge about the algorithm. In this chapter, to show how broadly Inner Ensembles is applicable, and give the reader an idea of how to apply Inner Ensembles in different algorithms, we suggest possible ways of applying this framework. Here, we tried to choose algorithms that are relatively different. From the supervised learning category we chose Neural Network, K-Nearest Neighbors and Naive Bayes, and from un-supervised learning we chose COBWEB, a conceptual clustering, hierarchical clustering, and an online clustering algorithm called CluStream. In addition, we selected two that are not in the supervised or unsupervised learning categories: an algorithm for pruning the decision tree, and A-priori, an association rule mining algorithm. We also included a table showing the

name of the algorithm, the decision maker inside the algorithm, and the possible ways of using ensembles on the decision maker. The idea is that by reading the table, the reader can apply Inner Ensemble in different similar algorithms. For example, we show different neural networks, such as RBF, and different online clustering algorithms. Thus, we expect that by using table 5.1 the reader can apply Inner Ensembles to RBF or other online clustering.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The main objective of this thesis is to explore the concept of 'wisdom of the crowd', and show that this idea can be applied more broadly than it has been previously. Our objective in the work is to expand how the machine learning community can make use of ensemble methods. We contend that one novel way of extending ensemble learning is to apply ensemble methods to the decisions used to generate the models, rather than combining the output of the algorithm, as is the case with regular ensemble methods. More specifically, ensemble methods can be applied on the decision makers inside the algorithm during the training of the algorithm, and thereby achieve more robust decisions. We call this unique approach Inner Ensembles. We designed a general framework to apply Inner Ensembles on different types of algorithm with the assumption of having minimal knowledge about the algorithms. The approach has numerous advantages over regular ensemble methods, including comprehensibility, simplicity, fast classification and a small memory foot print, and it maintains some of the existing benefits of regular ensembles, such as performance and stability. However, some absolute performance is traded off to gain the advantages over regular ensembles.

To show how broadly this idea is applicable, here we use the general guidelines to apply the approach to two different categories of machine learning algorithms, supervised learning and un-supervised learning. For supervised learning we applied it to Bayesian network, which involved two steps: learning the structure of the network and learning the conditional probability tables. We focused on learning the structure of the network. One of the characteristics of the Bayesian network is comprehensibility, which is essentially

eliminated when using regular ensemble methods. Inner Ensemble can be applied to maintain the comprehensibility while keeping the advantages of the ensemble methods, which provide more robust decisions for the structure of the network and, ultimately, a better network structure compared to the original. We run several experiments to compare the quality of networks learned by IEBN with those learned by original BN, and the results show that IEBN generates networks that are significantly smaller, and which represent the dataset significantly better than BN. We also run another set of experiments to compare the performance of IEBN with original BN and two different ensemble methods, Bagging and Boosting. The results show that IEBN still performs significantly better than original BN, but not as well as Bagging or Boosting, and that its performance is closer to Boosting than Bagging. In addition, whenever ensemble methods improve the performance, IEBN improves the performance as well. This result shows that, as expected, comprehensibility, simplicity and faster classification time are traded off for absolute performance, as compared to ensemble methods.

For un-supervised learning, we use a popular clustering method known as K-Means clustering. One of the advantages of K-Means is prototypes, which are representative points for each cluster, and offer several advantages, including summarization, compression and efficient locating of nearest neighbours. They also enhance human understanding of the structure of a particular problem or domain, in medical diagnosis for example. Prototypes do not exist when using ensemble of K-Means. We use Inner Ensembles to find the closest cluster center to each data instance for each iteration of the algorithm; we call the resulting method Inner Ensemble K-Means (IEKM). Experimental results show that IEKM performs better than original K-means, and comparably to two families of cluster ensemble methods. On noisy data it is more robust than the original method, but not as robust as ensemble methods in general.

In this work, we also suggest a possible way Inner Ensembles can be extended to apply to a range of eight different algorithms . For classification algorithms we studied neural network, K-Nearest neighbor and naive Bayes. For clustering we chose hierarchical clustering, conceptual clustering (COBWEB) and, to show how we can use Inner Ensembles on online algorithms, a well-known online clustering called CluStream. Finally, to show that the idea can apply to algorithms other than classification and clustering, we introduced it for two algorithms that are not supervised or unsupervised learning: pruning of the decision tree, and A-priori, an association mining rule.

To summarize, this thesis introduces a new way of using ensemble methods, and a new framework that we believe has the potential of allowing it to be applied on different

types of algorithms. Experimental results show that Inner Ensemble has benefits over regular ensemble methods, such as comprehensibility, fast classification, small memory footprint and simplicity, while maintaining other regular ensembles advantages, such as robustness and improved performance. However, these advantages have been traded off for the accuracy of regular ensembles, which means that the absolute performance of Inner Ensembles is between that of the original method and regular ensemble methods.

6.2 Limitations and Future Work

Inner Ensembles do have some limitations, the main one being that it does not produce the same performance as traditional ensemble methods. Although the results show that the performance of Inner Ensembles is comparable, and sometimes better, than ensemble methods for K-Means, it does not perform as well as Boosting and Bagging for Bayesian network. We believe this is because we focused on the structure of the Bayesian network, and Bagging and Boosting work with the classification aspect. Further improvement of the method by applying Inner Ensembles on the classification could terminate this performance disparity. However, this might not be completely achieved, due to the tradeoff of performance for other advantages, such as comprehensibility, simplicity and faster classification time. Another limitation of this work is that we use ensemble methods to make decisions during the training step, which makes the training of the algorithm very slow. Further research is required to decrease the time of the training step. Finally, with Inner Ensembles we use different ensemble and sample sizes inside the model. It would be useful if the algorithm could automatically select the best parameters at each step of the training step. This would eliminate the problems associated with finding one set of best parameters for each dataset during the experiments.

This work can be extended in different directions. For example, here we focus on two algorithms, but the framework can be applied to different methods as well. Also, we could use different kinds of ensembles on all algorithms. Another potential direction is to investigate the effect of diversity on the performance of Inner Ensembles. And finally, more research could be done on the effect of Inner Ensembles on bias and variance.

6.2.1 Using Different Ensemble Methods

Inside the algorithms we used ensemble methods similar to Bagging. Another direction could be using different ensemble methods, such as Boosting or DECORATE , inside the

algorithms, or even different sampling, weighted sampling for example. Inner Ensembles could also be used to select different metrics within a given machine learning algorithm, and to estimate the measure inside the algorithm. There are also different ways to combine methods other than voting and averaging, such as weighted majority vote or naive Bayes combination, that warrant further investigation.

6.2.2 Effect of Diversity

If we have a perfect classifier, there is no need to use ensemble methods. However, if the classifier makes errors, we need to find another classifier that does not make the errors. Thus, diversity is an important factor affecting the performance of the ensemble method. We should aim for classifiers that are as accurate and diverse as possible. There are studies on ensemble diversity in the literature, and several measures have been defined for it (Kuncheva, 2004). For Inner Ensembles, we want decision makers that are as accurate and diverse as possible. Therefore, the effect of diversity on the performance of Inner Ensembles requires more investigation, as it will lead to the creation of more efficient Inner Ensembles. There are several ways to ensure the diversity. One is for each ensemble member to select a feature set that is as different as possible from all other ensemble members sets, according to a certain measure. Another approach is to use weighted sampling, similar to that with Boosting. There are other possible techniques to ensure diversity, such as generating artificial data to increase the diversity of the ensembles, similar to the DECORATE algorithm (Melville and Mooney, 2003a).

6.2.3 Bias and Variance Decomposition

There have also been several studies about the effect of ensemble methods on bias and variance, particularly Bagging and Boosting. Overall, it has been assumed that Bagging generally reduces variance without affecting bias, and that Boosting mostly reduces bias in early iterations, and mainly reduces variance in later iterations (Kuncheva, 2004). For future work, we could investigate the effect of Inner Ensembles, different types of sampling and different ensemble methods, on bias and variance, to determine which is most affected by Inner Ensembles. This analysis could help identify the type of problems Inner Ensembles can address most effectively.

Appendix A

Additional Results for Chater 4

Table A.1: Comparison of K-means, Inner Ensemble K-means and Single Linkage in terms of Purity for Sampling Without Replacement.

Dataset	K-Means	IE-K-Means	S-Linkage
Ecoli	0.8	0.8	0.79
Glass	0.6	0.6	0.59
Heart-S	0.62	0.78	0.68
Iris	0.87	0.9	0.86
Letter	0.45	0.46	0.45
Optdig	0.66	0.71	0.63
Pendig	0.65	0.7	0.6
Pima	0.66	0.68	0.67
Segment	0.61	0.63	0.59
Sonar	0.58	0.57	0.56
Vehicle	0.47	0.47	0.45
Vowel	0.32	0.36	0.32
Wavef	0.6	0.59	0.58
Wine	0.7	0.93	0.75
Yeast	0.52	0.52	0.49
Average	0.61	0.65	0.60
After Removing Datasets			
Average	0.62	0.68	0.62
Average Rank	2.44	1.00	2.56
Q-Value	3.06		3.30

Table A.2: Comparison of K-means, Inner Ensemble K-means and Average Linkage in terms of Purity for Sampling Without Replacement.

Dataset	K-Means	IE-K-Means	A-Linkage
Ecoli	0.8	0.8	0.79
Glass	0.6	0.6	0.59
Heart-S	0.62	0.78	0.73
Iris	0.87	0.9	0.87
Letter	0.45	0.46	0.48
Optdig	0.66	0.71	0.72
Pendig	0.65	0.7	0.68
Pima	0.66	0.68	0.72
Segment	0.61	0.63	0.69
Sonar	0.58	0.57	0.57
Vehicle	0.47	0.47	0.47
Vowel	0.32	0.36	0.36
Wavef	0.6	0.59	0.61
Wine	0.7	0.93	0.86
Yeast	0.52	0.52	0.53
Average	0.61	0.65	0.64
After Removing Datasets			
Average	0.61	0.66	0.66
Average Rank	2.82	1.73	1.45
Q-Value	2.56		0.64

Table A.3: Comparison of K-means, Inner Ensemble K-means and Complete Linkage in terms of Purity for Sampling Without Replacement.

Dataset	K-Means	IE-K-Means	C-Linkage
Ecoli	0.8	0.8	0.79
Glass	0.6	0.6	0.61
Heart-S	0.62	0.78	0.73
Iris	0.87	0.9	0.87
Letter	0.45	0.46	0.48
Optdig	0.66	0.71	0.71
Pendig	0.65	0.7	0.69
Pima	0.66	0.68	0.71
Segment	0.61	0.63	0.68
Sonar	0.58	0.57	0.55
Vehicle	0.47	0.47	0.47
Vowel	0.32	0.36	0.36
Wavef	0.6	0.59	0.6
Wine	0.7	0.93	0.86
Yeast	0.52	0.52	0.51
Average	0.61	0.65	0.64
After Removing Datasets			
Average	0.61	0.67	0.67
Average Rank	2.90	1.55	1.55
Q-Value	3.02		0.00

Table A.4: Comparison of K-means, Inner Ensemble K-means and CSPA in terms of Purity for Sampling Without Replacement.

Dataset	K-Means	IE-K-Means	CSPA
Ecoli	0.8	0.8	0.75
Glass	0.6	0.6	0.62
Heart-S	0.62	0.78	0.77
Iris	0.87	0.9	0.92
Letter	0.45	0.46	0.46
Optdig	0.66	0.71	0.72
Pendig	0.65	0.7	0.68
Pima	0.66	0.68	0.69
Segment	0.61	0.63	0.71
Sonar	0.58	0.57	0.59
Vehicle	0.47	0.47	0.45
Vowel	0.32	0.36	0.4
Wavef	0.6	0.59	0.58
Wine	0.7	0.93	0.88
Yeast	0.52	0.52	0.5
Average	0.61	0.65	0.65
After Removing Datasets			
Average	0.61	0.67	0.68
Average Rank	2.86	1.82	1.32
Q-Value	2.45		1.17

Table A.5: Comparison of K-means, Inner Ensemble K-means and HGPA in terms of Purity for Sampling Without Replacement.

Dataset	K-Means	IE-K-Means	HGPA
Ecoli	0.8	0.8	0.75
Glass	0.6	0.6	0.62
Heart-S	0.62	0.78	0.6
Iris	0.87	0.9	0.9
Letter	0.45	0.46	0.46
Optdig	0.66	0.71	0.69
Pendig	0.65	0.7	0.67
Pima	0.66	0.68	0.66
Segment	0.61	0.63	0.66
Sonar	0.58	0.57	0.59
Vehicle	0.47	0.47	0.45
Vowel	0.32	0.36	0.37
Wavef	0.6	0.59	0.58
Wine	0.7	0.93	0.75
Yeast	0.52	0.52	0.48
Average	0.61	0.65	0.62
After Removing Datasets			
Average	0.61	0.67	0.63
Average Rank	2.73	1.59	1.68
Q-Value	2.67		0.21

Table A.6: Comparison of K-means, Inner Ensemble K-means and MCLA in terms of Purity for Sampling Without Replacement.

Dataset	K-Means	IE-K-Means	MCLA
Ecoli	0.8	0.8	0.78
Glass	0.6	0.6	0.6
Heart-S	0.62	0.78	0.79
Iris	0.87	0.9	0.9
Letter	0.45	0.46	0.44
Optdig	0.66	0.71	0.71
Pendig	0.65	0.7	0.69
Pima	0.66	0.68	0.7
Segment	0.61	0.63	0.63
Sonar	0.58	0.57	0.57
Vehicle	0.47	0.47	0.48
Vowel	0.32	0.36	0.46
Wavef	0.6	0.59	0.58
Wine	0.7	0.93	0.89
Yeast	0.52	0.52	0.51
Average	0.61	0.65	0.65
After Removing Datasets			
Average	0.61	0.66	0.67
Average Rank	2.85	1.60	1.55
Q-Value	2.80		0.11

Table A.7: Comparison of K-means, Inner Ensemble K-means and Single Linkage in terms of NMI for Sampling Without Replacement.

Dataset	K-Means	IE-K-Means	S-Linkage
Ecoli	0.64	0.64	0.65
Glass	0.45	0.47	0.45
Heart-S	0.04	0.25	0.14
Iris	0.79	0.82	0.79
Letter	0.65	0.66	0.64
Optdig	0.68	0.73	0.68
Pendig	0.68	0.7	0.67
Pima	0.02	0.08	0.03
Segment	0.57	0.66	0.61
Sonar	0.02	0.02	0.04
Vehicle	0.21	0.21	0.2
Vowel	0.33	0.37	0.32
Wavef	0.37	0.37	0.33
Wine	0.45	0.74	0.55
Yeast	0.35	0.36	0.33
Average	0.42	0.47	0.43
After Removing Datasets			
Average	0.44	0.50	0.45
Average Rank	2.50	1.23	2.27
Q-Value	3.24		2.64

Table A.8: Comparison of K-means, Inner Ensemble K-means and Average Linkage in terms of NMI for Sampling Without Replacement.

Dataset	K-Means	IE-K-Means	A-Linkage
Ecoli	0.64	0.64	0.64
Glass	0.45	0.47	0.45
Heart-S	0.04	0.25	0.24
Iris	0.79	0.82	0.79
Letter	0.65	0.66	0.68
Optdig	0.68	0.73	0.74
Pendig	0.68	0.7	0.71
Pima	0.02	0.08	0.13
Segment	0.57	0.66	0.68
Sonar	0.02	0.02	0.02
Vehicle	0.21	0.21	0.22
Vowel	0.33	0.37	0.4
Wavef	0.37	0.37	0.38
Wine	0.45	0.74	0.7
Yeast	0.35	0.36	0.36
Average	0.42	0.47	0.48
After Removing Datasets			
Average	0.43	0.49	0.50
Average Rank	2.84	1.73	1.42
Q-Value	2.84		0.78

Table A.9: Comparison of K-means, Inner Ensemble K-means and Complete Linkage in terms of NMI for Sampling Without Replacement.

Dataset	K-Means	IE-K-Means	C-Linkage
Ecoli	0.64	0.64	0.63
Glass	0.45	0.47	0.44
Heart-S	0.04	0.25	0.28
Iris	0.79	0.82	0.78
Letter	0.65	0.66	0.68
Optdig	0.68	0.73	0.73
Pendig	0.68	0.7	0.7
Pima	0.02	0.08	0.11
Segment	0.57	0.66	0.66
Sonar	0.02	0.02	0.01
Vehicle	0.21	0.21	0.21
Vowel	0.33	0.37	0.37
Wavef	0.37	0.37	0.37
Wine	0.45	0.74	0.66
Yeast	0.35	0.36	0.35
Average	0.42	0.47	0.47
After Removing Datasets			
Average	0.46	0.53	0.52
Average Rank	2.77	1.45	1.77
Q-Value	3.09		0.75

Table A.10: Comparison of K-means, Inner Ensemble K-means and CSPA in terms of NMI for Sampling Without Replacement.

Dataset	K-Means	IE-K-Means	CSPA
Ecoli	0.64	0.64	0.55
Glass	0.45	0.47	0.41
Heart-S	0.04	0.25	0.26
Iris	0.79	0.82	0.81
Letter	0.65	0.66	0.67
Optdig	0.68	0.73	0.71
Pendig	0.68	0.7	0.67
Pima	0.02	0.08	0.12
Segment	0.57	0.66	0.64
Sonar	0.02	0.02	0.04
Vehicle	0.21	0.21	0.17
Vowel	0.33	0.37	0.39
Wavef	0.37	0.37	0.33
Wine	0.45	0.74	0.68
Yeast	0.35	0.36	0.32
Average	0.42	0.47	0.45
After Removing Datasets			
Average	0.42	0.49	0.48
Average Rank	2.71	1.46	1.83
Q-Value	3.06		0.92

Table A.11: Comparison of K-means, Inner Ensemble K-means and HGPA in terms of NMI for Sampling Without Replacement.

Dataset	K-Means	IE-K-Means	HGPA
Ecoli	0.64	0.64	0.56
Glass	0.45	0.47	0.43
Heart-S	0.04	0.25	0.07
Iris	0.79	0.82	0.78
Letter	0.65	0.66	0.67
Optdig	0.68	0.73	0.69
Pendig	0.68	0.7	0.66
Pima	0.02	0.08	0.01
Segment	0.57	0.66	0.62
Sonar	0.02	0.02	0.03
Vehicle	0.21	0.21	0.16
Vowel	0.33	0.37	0.35
Wavef	0.37	0.37	0.24
Wine	0.45	0.74	0.56
Yeast	0.35	0.36	0.3
Average	0.42	0.47	0.41
After Removing Datasets			
Average	0.42	0.49	0.43
Average Rank	2.54	1.21	2.25
Q-Value	3.26		2.55

Table A.12: Comparison of K-means, Inner Ensemble K-means and MCLA in terms of NMI for Sampling Without Replacement.

Dataset	K-Means	IE-K-Means	MCLA
Ecoli	0.64	0.64	0.61
Glass	0.45	0.47	0.41
Heart-S	0.04	0.25	0.3
Iris	0.79	0.82	0.81
Letter	0.65	0.66	0.7
Optdig	0.68	0.73	0.71
Pendig	0.68	0.7	0.71
Pima	0.02	0.08	0.09
Segment	0.57	0.66	0.59
Sonar	0.02	0.02	0.02
Vehicle	0.21	0.21	0.21
Vowel	0.33	0.37	0.48
Wavef	0.37	0.37	0.36
Wine	0.45	0.74	0.68
Yeast	0.35	0.36	0.35
Average	0.42	0.47	0.47
After Removing Datasets			
Average	0.46	0.53	0.53
Average Rank	2.86	1.45	1.68
Q-Value	3.30		0.53

Table A.13: Comparison of K-means, Inner Ensemble K-means and Ensemble K-means (G-Based, HAC) in terms of NMI for Different Noise Levels.

Noise	Dataset	K-M	IE-K-M	S-Lnkg	A-Lnkg	C-Lnkg	CSPA	MCLA	HGPA
20%	**Hear-S**	0.04	0.17	0.06	0.1	0.11	0.16	0.15	0.03
	Iris	0.55	0.61	0.55	0.62	0.63	0.61	0.6	0.57
	Letter	0.61	0.63	0.6	0.64	0.64	0.68	0.66	0.63
	Optdig	0.59	0.61	0.58	0.65	0.62	0.62	0.6	0.58
	Pendig	0.6	0.61	0.57	0.62	0.61	0.61	0.61	0.59
	Pima	0.02	0.06	0.02	0.08	0.07	0.08	0.07	0.02
	Segment	0.39	0.45	0.41	0.51	0.51	0.44	0.42	0.42
	Vowel	0.28	0.34	0.27	0.34	0.33	0.53	0.42	0.29
	Wine	0.31	0.5	0.33	0.56	0.5	0.5	0.44	0.3
	Average	0.38	0.44	0.38	0.46	0.45	0.47	0.44	0.38
40%	**Hear-S**	0.04	0.05	0.04	0.05	0.05	0.07	0.09	0.03
	Iris	0.38	0.51	0.39	0.46	0.45	0.46	0.46	0.4
	Letter	0.59	0.61	0.56	0.62	0.62	0.68	0.63	0.61
	Optdig	0.51	0.53	0.49	0.55	0.53	0.52	0.52	0.5
	Pendig	0.53	0.55	0.5	0.56	0.55	0.54	0.53	0.51
	Pima	0.02	0.04	0.05	0.05	0.06	0.07	0.03	0.02
	Segment	0.27	0.32	0.29	0.4	0.38	0.35	0.33	0.31
	Vowel	0.26	0.3	0.24	0.32	0.32	0.53	0.41	0.28
	Wine	0.2	0.34	0.21	0.24	0.26	0.23	0.22	0.18
	Average	0.31	0.36	0.31	0.36	0.36	0.38	0.36	0.32
60%	**Hear-S**	0.04	0.06	0.04	0.05	0.06	0.18	0.05	0.03
	Iris	0.23	0.32	0.31	0.32	0.38	0.34	0.38	0.26
	Letter	0.57	0.58	0.55	0.6	0.6	0.68	0.64	0.59
	Optdig	0.48	0.5	0.46	0.5	0.49	0.52	0.49	0.47
	Pendig	0.46	0.48	0.44	0.51	0.49	0.48	0.45	0.44
	Pima	0.02	0.02	0.05	0.03	0.03	0.04	0.02	0.02
	Segment	0.19	0.22	0.19	0.28	0.27	0.38	0.22	0.21
	Vowel	0.25	0.28	0.23	0.31	0.31	0.53	0.42	0.27
	Wine	0.09	0.21	0.12	0.1	0.11	0.53	0.14	0.12
	Average	0.26	0.30	0.27	0.30	0.30	0.41	0.31	0.27
Avg	**Hear-S**	0.04	0.09	0.05	0.07	0.07	0.14	0.1	0.03
	Iris	0.39	0.48	0.42	0.47	0.49	0.47	0.48	0.41
	Letter	0.59	0.61	0.57	0.62	0.62	0.68	0.64	0.61
	Optdig	0.53	0.55	0.51	0.57	0.55	0.55	0.54	0.52
	Pendig	0.53	0.55	0.5	0.56	0.55	0.54	0.53	0.51
	Pima	0.02	0.04	0.04	0.05	0.05	0.06	0.04	0.02
	Segment	0.28	0.33	0.3	0.4	0.39	0.39	0.32	0.31
	Vowel	0.26	0.31	0.25	0.32	0.32	0.53	0.42	0.28
	Wine	0.2	0.35	0.22	0.3	0.29	0.42	0.27	0.2
	Average	0.32	0.37	0.32	0.37	0.37	0.42	0.37	0.32

Table A.14: Friedman's test results for comparing K-Means and each cluster ensemble with IEKM for NMI measure with 20% noise.

	K-Means	IE-K-Means	S-Linkage	A-Linkage	C-Linkage	CSPA	HGPA	MCLA
Q-Value	3.06		3.3					
	2.47			1.41				
	2.83				0.71			
	2.94					0.47		
	3.06						2.95	
	3.42							0.47
Avg Rank	2.44	1.00	2.57					
	3.00	1.83		1.17				
	3.00	1.67			1.33			
	3.00	1.61				1.39		
	2.50	1.06					2.44	
	3.00	1.39						1.61

Table A.15: Friedman's test results for comparing K-Means and each cluster ensemble with IEKM for NMI measure with 40% noise.

	K-Means	IE-K-Means	S-Linkage	A-Linkage	C-Linkage	CSPA	HGPA	MCLA
Q-Value	2.95		2.71					
	2.71			0.94				
	2.94				0.47			
	3.06					0.24		
	3.06						2.95	
	3.18							0.35
Avg Rank	2.50	1.11	2.39					
	3.00	1.72		1.28				
	3.00	1.61			1.39			
	3.00	1.56				1.44		
	2.50	1.06					2.44	
	2.94	1.44						1.61

Table A.16: Friedman's test results for comparing K-Means and each cluster ensemble with IEKM for NMI measure with 60% noise.

	K-Means	IE-K-Means	S-Linkage	A-Linkage	C-Linkage	CSPA	HGPA	MCLA
Q-Value	2.59		2.71					
	2.59			0.82				
	2.47				1.06			
	2.00					2.00		
	2.82						2.12	
	2.71							0.47
Avg Rank	2.39	1.67	2.44					
	2.94	1.72		1.33				
	2.94	1.77			1.28			
	2.94	2.00				1.06		
	2.56	1.22					2.22	
	2.78	1.50						1.72

Table A.17: Friedman's test results for comparing K-Means and each cluster ensemble with IEKM on noisy data (average on all noise levels) for NMI measure.

	K-Means	**IE-K-Means**	**S-Linkage**	**A-Linkage**	**C-Linkage**	**CSPA**	**HGPA**	**MCLA**
Q-Value	3.18		2.82					
	2.83			0.70				
	2.83				0.70			
	2.71					0.94		
	3.18						2.82	
	3.18							0.35
Avg Rank	2.56	1.06	2.39					
	3.00	1.67		1.33				
	3.00	1.67			1.33			
	3.00	1.72				1.28		
	2.56	1.06					2.39	
	2.94	1.44						1.61

Table A.18: Comparison of K-means, Inner Ensemble K-means and Ensemble K-means (G-Based, HAC) in terms of Purity for Different Noise Levels.

Noise	Dataset	K-M	IE-K-M	S-Lnkg	A-Lnkg	C-Lnkg	CSPA	MCLA	HGPA
20%	**Hear-S**	0.62	0.74	0.61	0.66	0.68	0.71	0.7	0.59
	Iris	0.77	0.8	0.77	0.81	0.82	0.83	0.83	0.8
	Letter	0.4	0.41	0.4	0.42	0.42	0.48	0.41	0.41
	Optdig	0.58	0.61	0.55	0.65	0.63	0.64	0.61	0.58
	Pendig	0.58	0.59	0.54	0.63	0.61	0.62	0.61	0.6
	Pima	0.66	0.68	0.67	0.69	0.69	0.67	0.68	0.66
	Segment	0.51	0.55	0.5	0.58	0.57	0.56	0.53	0.51
	Vowel	0.29	0.33	0.29	0.33	0.32	0.48	0.4	0.32
	Wine	0.64	0.78	0.64	0.78	0.75	0.77	0.75	0.68
	Average	0.56	0.61	0.55	0.62	0.61	0.64	0.61	0.57
40%	**Hear-S**	0.61	0.62	0.6	0.6	0.62	0.64	0.65	0.59
	Iris	0.68	0.76	0.66	0.69	0.7	0.73	0.72	0.72
	Letter	0.37	0.37	0.37	0.39	0.38	0.48	0.38	0.37
	Optdig	0.5	0.52	0.48	0.54	0.5	0.52	0.52	0.5
	Pendig	0.52	0.55	0.48	0.56	0.56	0.55	0.53	0.52
	Pima	0.66	0.66	0.66	0.66	0.66	0.67	0.66	0.66
	Segment	0.4	0.42	0.39	0.47	0.45	0.45	0.45	0.42
	Vowel	0.27	0.3	0.27	0.3	0.31	0.48	0.36	0.29
	Wine	0.58	0.68	0.56	0.61	0.66	0.61	0.61	0.58
	Average	0.51	0.54	0.50	0.54	0.54	0.57	0.54	0.52
60%	**Hear-S**	0.6	0.62	0.59	0.6	0.6	0.74	0.61	0.59
	Iris	0.57	0.63	0.58	0.63	0.64	0.67	0.68	0.6
	Letter	0.33	0.34	0.33	0.35	0.35	0.48	0.35	0.35
	Optdig	0.48	0.49	0.44	0.49	0.5	0.52	0.49	0.47
	Pendig	0.45	0.48	0.41	0.48	0.48	0.49	0.48	0.44
	Pima	0.66	0.66	0.66	0.66	0.67	0.66	0.66	0.66
	Segment	0.34	0.35	0.33	0.37	0.39	0.44	0.37	0.35
	Vowel	0.26	0.28	0.25	0.3	0.3	0.48	0.36	0.28
	Wine	0.49	0.56	0.49	0.5	0.51	0.58	0.52	0.54
	Average	0.46	0.49	0.45	0.49	0.49	0.56	0.50	0.48
Avg	**Hear-S**	0.61	0.66	0.6	0.62	0.63	0.7	0.65	0.59
	Iris	0.67	0.73	0.67	0.71	0.72	0.74	0.74	0.71
	Letter	0.37	0.37	0.37	0.39	0.38	0.48	0.38	0.38
	Optdig	0.52	0.54	0.49	0.56	0.54	0.56	0.54	0.52
	Pendig	0.52	0.54	0.48	0.56	0.55	0.55	0.54	0.52
	Pima	0.66	0.67	0.66	0.67	0.67	0.67	0.67	0.66
	Segment	0.42	0.44	0.41	0.47	0.47	0.48	0.45	0.43
	Vowel	0.27	0.3	0.27	0.31	0.31	0.48	0.37	0.3
	Wine	0.57	0.67	0.56	0.63	0.64	0.65	0.63	0.6
	Average	0.51	0.55	0.50	0.55	0.56	0.59	0.55	0.52

Table A.19: Friedman's test results for comparing K-Means and each cluster ensemble with IEKM for purity measure with 20% noise.

	K-Means	IE-K-Means	S-Linkage	A-Linkage	C-Linkage	CSPA	HGPA	MCLA
Q-Value	2.82		3.53					
	2.59			1.17				
	2.83				0.71			
	2.83					0.71		
	3.18						1.77	
	3.18							0.00
Avg Rank	2.33	1.00	2.67					
	3.00	1.77		1.22				
	3.00	1.67			1.33			
	3.00	1.67				1.33		
	2.72	1.22					2.05	
	3.00	1.50						1.50

Table A.20: Friedman's test results for comparing K-Means and each cluster ensemble with IEKM for purity measure with 40% noise.

	K-Means	IE-K-Means	S-Linkage	A-Linkage	C-Linkage	CSPA	HGPA	MCLA
Q-Value	1.77		3.18					
	2.24			0.11				
	2.36				0.24			
	2.36					0.94		
	2.59						2.00	
	2.47							0.35
Avg Rank	2.05	1.22	2.72					
	2.72	1.67		1.61				
	2.78	1.67			1.56			
	2.89	1.78				1.33		
	2.50	1.28					2.22	
	2.83	1.68						1.50

Table A.21: Friedman's test results for comparing K-Means and each cluster ensemble with IEKM for purity measure with 60% noise.

	K-Means	IE-K-Means	S-Linkage	A-Linkage	C-Linkage	CSPA	HGPA	MCLA
Q-Value	2.35		3.29					
	2.59			0.11				
	2.36				0.94			
	1.89					1.89		
	2.59						1.64	
	2.59							0.47
Avg Rank	2.22	1.11	2.66					
	2.83	1.61		1.56				
	2.89	1.77			1.33			
	2.89	2.00				1.11		
	2.56	1.33					2.11	
	2.89	1.68						1.44

Table A.22: Friedman's test results for comparing K-Means and each cluster ensemble with IEKM on noisy data (average on all noise levels) for purity measure.

	K-Means	**IE-K-Means**	**S-Linkage**	**A-Linkage**	**C-Linkage**	**CSPA**	**HGPA**	**MCLA**
Q-Value	2.24		3.41					
	2.71			0.58				
	2.83				0.35			
	2.24					1.53		
	3.06						1.88	
	2.71							0.58
Avg Rank	2.17	1.11	2.72					
	2.94	1.67		1.39				
	2.94	1.61			1.44			
	2.94	1.89				1.17		
	2.67	1.22					2.11	
	2.94	1.67						1.39

Bibliography

D. N. A. Asuncion. UCI Machine Learning Repository, 2007. URL `http://archive.ics.uci.edu/ml`.

H. Abbasian, C. Drummond, N. Japkowicz, and S. Matwin. Robustness of Classifiers to Changing Environments. In *Proceedings of the 23rd Canadian conference on Advances in Artificial Intelligence*, pages 232–243, 2010.

S. Abney, R. E. Schapire, and Y. Singer. Boosting applied to tagging and pp attachment. In *In Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 38–45, 1999.

C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A Framework for Clustering Evolving Data Streams. In *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 81–92, 2003.

E. Alexopoulos, G. D. Dounias, K. Vemmos, and E. Alexopoulos. Medical Diagnosis of Stroke using Inductive Machine Learning. In *Workshop on Machine Learning in Medical Applications, Advance Course in Artificial Intelligence*, pages 20–23, 1999.

W.-K. W. Andrew Moore. Optimal Reinsertion: A New Search Operator for Accelerated and More Accurate Bayesian Network Structure Learning. In *Proceedings of the 20th International Conference on Machine Learning*, pages 552–559, 2003.

E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Mach. Learn.*, 36(1-2):105–139, July 1999.

H. Blockeel and L. De Raedt. Top-Down Induction ofFirst-Order Logical Decision Trees. *Artificial Intelligence.*, 101:285–297, 1998.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's Razor. *Information Processing Letters*, 24:377–380, 1987.

R. Bouckaert. *Bayesian Network Classifiers in Weka*. Working paper series. Department of Computer Science, University of Waikato, 2004.

L. Breiman. Bagging predictors. In *Machine Learning*, pages 123–140, 1996a.

L. Breiman. Bias, Variance, and Arcing Classifiers. Technical report, Statistics Department, University of California at Berkeley, 1996b.

L. Breiman. Bias, Variance, and Arcing Classifiers. Technical report, 1996c.

L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. 1984.

C. E. Brodley and M. A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

W. L. Buntine. Theory Refinement of Bayesian Networks. In *Uncertainty in Artificial Intelligence*, pages 52–60, 1991.

A. Carvalho. A Cooperative Coevolutionary Genetic Algorithm for Learning Bayesian Network Structures. In *Proceedings of the Thirteenth Genetic and Evolutionary Computation Conference*, pages 1131–1138, 2011.

D. M. Chickering. Learning Bayesian Networks is Np-Complete. In *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130, 1996.

D. M. Chickering, D. Geiger, and D. Heckerman. Learning Bayesian Networks: Search Methods and Experimental Results. In *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, pages 112–128, 1995.

G. F. Cooper and E. Herskovits. A Bayesian Method for the Induction of Probabilistic Networks from Data. *Machine Learning*, 9:309–347, 1992.

M. W. Craven and J. W. Shavlik. Extracting Comprehensible Concept Representations from Trained Neural Networks. In *Working Notes on the International Joint Conference on AI Workshop on Comprehensibility in Machine Learning*, pages 61–75, 1995.

R. Daly, Q. Shen, and J. S. Aitken. Learning Bayesian Networks: Approaches and Issues. *The Knowledge Engineering Review*, 26:99–157, 2011.

L. M. de Campos. A Scoring Function for Learning Bayesian Networks Based on Mutual Information and Conditional Independence Tests. *Journal of Machine Learning Research*, 7:2149–2187, 2006.

J. Desmedt. *Computer-Aided Electromyography and Expert Systems*. Clinical Neurophysiology Updates. Elsevier, Amsterdam, 1989.

T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Mach. Learn.*, 40(2): 139–157, Aug. 2000a.

T. G. Dietterich. Ensemble Methods in Machine Learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, pages 1–15, 2000b.

P. Domingos. Why does bagging work? a bayesian account and its implications. In *In Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 155–158, 1997.

P. Domingos. Occam's Two Razors: The Sharp and the Blunt. In *Knowledge Discovery and Data Mining*, pages 37–43, 1998a.

P. Domingos. Knowledge Discovery Via Multiple Models. *Intelligent Data Analysis*, 2: 187–202, 1998b.

P. Domingos. A Unified Bias-Variance Decomposition and its Applications. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 231–238, 2000.

C. Drummond, S. Matwin, and C. Gaffield. Inferring and Revising Theories with Confidence: Analyzing Bilingualism in the 1901 Canadian Census. *Applied Artificial Intelligence*, 20(1):1–33, 2006.

R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Willey & Sons, New York, 1973.

K. Dwyer and R. Holte. Decision Tree Instability and Active Learning. In *Proceedings of the 8th European Conference on Machine Learning*, pages 128–139, 2007.

B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, New York, 1993.

Y. El-Sonbaty and M. A. Ismail. On-Line Hierarchical Clustering. *Pattern Recognition Letter.*, 19:1285–1291, 1998.

C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. From Ensemble Methods to Comprehensible Models. In *Proceedings of the 5th International Conference on Discovery Science*, pages 165–177, 2002.

D. H. Fisher. Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning.*, 2:139–172, 1987.

D. H. Fisher. Noise-Tolerant Conceptual Clustering. In *Proceedings of the 11th international joint conference on Artificial intelligence*, pages 825–830, 1989.

Y. Freund. Boosting a weak learning algorithm by majority. *Inf. Comput.*, 121(2): 256–285, Sept. 1995.

Y. Freund and R. E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37, London, 1995.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, Aug. 1997.

Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, Dec. 2003.

N. Friedman and D. Koller. Being Bayesian About Network Structure. A Bayesian Approach to Structure Discovery in Bayesian Networks. *Machine Learning*, 50:95–125, 2003.

N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian Network Classifiers. *Machine Learning*, 29:131–163, 1997.

N. Friedman, M. Goldszmidt, and A. J. Wyner. Data Analysis with Bayesian Networks: A Bootstrap Approach. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence*, pages 196–205, 1999a.

N. Friedman, I. Nachman, and D. Peér. Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm. In *Proceedings of the Fifteenth conference on Uncertainty in Artificial Intelligence*, pages 206–215, 1999b.

S. I. Gallant. *Neural Network Learning and Expert Systems*. The MIT Press, 1993.

F. Galton. Vox Populi. *Nature*, 75:450–451, 1907.

D. Gamberger, N. Lavrac, and S. Dzroski. Noise detection and elimination in data preprocessing: Experiments in medical domains. *Applied Artificial Intelligence*, 14(2): 205–223, 2000.

A. Garg, V. Pavlovic, and T. Huang. Bayesian Networks as Ensemble of Classifiers. In *Proceedings of the Sixteenth International Conference on Pattern Recognition*, pages 779–784, 2002.

W. Geamsakul, T. Matsuda, T. Yoshida, H. Motoda, and T. Washio. Performance Evaluation of Decision Tree Graph-Based Induction. In *Discovery Science*, pages 128–140, 2003.

P. Geurts and L. Wehenkel. Investigation and Reduction of Discretization Variance in Decision Tree Induction. In *Proceedings of the 11th European Conference on Machine Learning*, pages 162–170, 2000.

R. Ghaemi, N. Sulaiman, H. Ibrahim, and N. Mustapha. A Survey: Clustering Ensembles Techniques. In *World academy of science, Engineering and Technology*, pages 644–653, 2009.

A. Goldenberg and A. Moore. Tractable Learning of Large Bayes Net Structures from Sparse Data. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 44–, 2004.

E. Gose, R. Johnsonbaugh, and S. Jost. *Pattern Recognition and Image Analysis*. Prentice Hall PTR, Upper Saddle River, NJ, 1996.

I. Gurrutxaga, A. Ansuategi, O. Arbelaitz, J. M. Prez, J. Muguerza, and J. I. Martn. Comparison of two strategies, ctc and cmm, to combine m classifiers in a single comprehensible one. In P. Perner, editor, *Industrial Conference on Data Mining - Posters*, IBaI Report, pages 6–18, 2006a.

I. Gurrutxaga, O. Arbelaitz, J. M. Prez, J. I. Martn, and J. Muguerza. The effect of the used resampling technique and number of samples in consolidated tree's construction algorithm, 2006b.

I. Gurrutxaga, J. M. Prez, O. Arbelaitz, J. Muguerza, J. I. Martín, and A. Ansuategi. Ctc: An alternative to extract explanation from bagging. chapter Current Topics in Artificial Intelligence, pages 90–99. Springer-Verlag, Berlin, Heidelberg, 2007.

J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, CA, 2006.

D. E. Heckerman and B. N. Nathwani. Towards Normative Expert Systems: Part ii, Probability-Based Representations for Efficient Knowledge Acquisition and Inference Methods of Information in Medicine. In *Methods of Information in Medicine*, pages 106–116, 1992.

R. J. Hickey. Noise modelling and evaluating learning from examples. *Artif. Intell.*, 82: 157–179, 1996.

L. Hunter and T. E. Klein. Finding Relevant Biomolecular Features. In *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology, Bethesda, MD*, pages 190–197, 1993.

N. Japkowicz and M. Shah. *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, New York, NY, USA, 2011.

Y. Jing, V. Pavlovic, and J. M. Rehg. Boosted Bayesian Network Classifiers. *Machine Learning*, 73:155–184, 2008.

I. K. M. R. L. Kaizhu Huang, Zhangbing Zhou. Improving Naive Bayesian Classifier by Discriminative Training. In *International Conference on Neural Information Processing*, pages 49–54, 2005.

G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing.*, 20(1):359–392, 1998.

G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel Hypergraph Partitioning: Application in VLSI Domain. In *Proceedings of the 34th annual Design Automation Conference*, pages 526–529, 1997.

M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *J. ACM*, 41(1):67–95, Jan. 1994.

R. Kohavi and C. Kunz. Option Decision Trees with Majority Votes. In *International Conference in Machine Learning*, pages 161–169, 1997.

R. Kohavi and D. H. Wolpert. Bias Plus Variance Decomposition for Zero-One Loss Functions. In *Proceedings of The Thirteenth International Conference on Machine Learning*, pages 275–283, 1996.

R. Kohavi, P. Langley, and Y. Yun. The Utility of Feature Weighting in Nearest-Neighbor Algorithms. In *Proceedings of the Ninth European Conference on Machine Learning*, 1997.

L. I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004.

G. Lakoff. *Women, Fire and Dangerous Things*. University of Chicago Press, Chicago, 1987.

W. Lam and F. Bacchus. Learning Bayesian Belief Networks: An Approach Based on the MDL Principle. *Computational Intelligence*, 10:269–294, 1994.

Z. Li, J. Liu, S. Chen, and X. Tang. Noise Robust Spectral Clustering. In *IEEE 11th International Conference on Computer Vision*, pages 1–8, 2007.

F. Liu, F. Tian, and Q. Zhu. Bayesian Network Structure Ensemble Learning. In *Proceedings of the seventh International Conference in Advanced Data Mining and Applications*, pages 454–465, 2007.

R. Maclin and D. Opitz. An empirical evaluation of bagging and boosting. In *Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence*, AAAI'97/IAAI'97, pages 546–551. AAAI Press, 1997.

J. B. Macqueen. Some Methods of Classification and Analysis of Multivariate Observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

W. S. McCulloch and W. Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biology*, pages 115–133.

P. Melville and R. J. Mooney. Constructing Diverse Classifier Ensembles Using Artificial Training Examples. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 505–510, 2003a.

P. Melville and R. J. Mooney. Constructing diverse classifier ensembles using artificial training examples. In *Proceedings of the 18th international joint conference on Artificial intelligence*, IJCAI'03, pages 505–510, 2003b.

P. Melville, N. Shah, L. Mihalkova, and R. J. Mooney. Experiments on ensembles with missing and noisy data. In *In: Proceedings of the Workshop on Multi Classifier Systems*, pages 293–302. Springer Verlag, 2004.

R. Michalski. A Theory and Methodology of Inductive Learning. *Artificial Intelligence*, 20:111–161, 1983.

T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, 1997.

S. Morishita and J. Sese. Transversing Itemset Lattices with Statistical Metric Pruning. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 226–236, New York, NY, 2000.

K. Murthy. *On Growing Better Decision Trees from Data*. PhD thesis, Johns Hopkins University, Maryland, 1997.

N. Nguyen and R. Caruana. Consensus Clusterings. In *Proceedings of the 7th International Conference on Data Mining*, pages 607–612, 2007.

D. Pelleg and D. Baras. K-Means with Large and Noisy Constraint Sets. In *Proceedings of 18th European Conference in Machine Learning*, pages 674–682, 2007.

O. Pourret, B. Marcot, and P. Naim. *Bayesian Networks: A Practical Guide to Applications*. Statistics in Practice. Wiley, Chichester, 2008.

J. M. Prez, J. Muguerza, O. Arbelaitz, and I. Gurrutxaga. A New Algorithm to Build Consolidated Trees: Study of the Error Rate and Steadiness. In *Proceedings of the International Intelligent Information Processing and Web Mining Conference*, Advances in Soft Computing, pages 79–88, 2004.

J. M. Prez, J. Muguerza, O. Arbelaitz, I. Gurrutxaga, and J. I. Martín. Consolidated trees: classifiers with stable explanation. a model to achieve the desired stability in

explanation. In *Proceedings of the Third international conference on Advances in Pattern Recognition - Volume Part I*, ICAPR'05, pages 99–107, Berlin, Heidelberg, 2005. Springer-Verlag.

J. M. Prez, J. Muguerza, O. Arbelaitz, I. Gurrutxaga, and J. I. Martn. Consolidated trees: An analysis of structural convergence. In G. J. Williams and S. J. Simoff, editors, *Selected Papers from AusDM*, volume 3755 of *Lecture Notes in Computer Science*, pages 39–52. Springer, 2006.

J. M. Prez, J. Muguerza, O. Arbelaitz, I. Gurrutxaga, and J. I. Martín. Combining multiple class distribution modified subsamples in a single tree. *Pattern Recogn. Lett.*, 28(4):414–422, Mar. 2007.

P. H. Prodip. *Scalable Frameworks and Algorithms for Cluster Ensembles and Clustering Data Streams*. PhD thesis, University of South Florida, Florida, 2007.

J. R. Quinlan. Simplifying Decision Trees. *International Journal of Man-Machine Studies*, 27:221–234, 1987.

J. R. Quinlan. Bagging, boosting, and c4.5. In *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 725–730. AAAI Press, 1996.

E. Rendón, I. M. Abundez, C. Gutierrez, S. D. Zagal, A. Arizmendi, E. M. Quiroz, and H. E. Arzate. A Comparison of Internal and External Cluster Validation Indexes. In *Proceedings of the American Conference on Applied Mathematics and the 5th International Conference on Computer Engineering and Applications*, pages 158–163, 2011.

A. Ribert, A. Ennaji, and Y. Lecourtier. An Incremental Hierarchical Clustering. In *Proceedings of Vision Interface Conference*, pages 586–591, 1999.

L. Rokach. *Pattern Classification Using Ensemble Methods*. Series in Machine Perception and Artificial Intelligence. World Scientific, 2010.

S. Rosset and E. Segal. Boosting Density Estimation. In *Proceedings of the Sixteenth International Conference on Neural Information Processing Systems*, pages 641–648, 2002.

R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227–227, June 1990.

R. E. Schapire and Y. Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5): 1651–1686, 1998a.

R. E. Schapire, Y. Singer, and A. Singhal. Boosting and rocchio applied to text filtering. In *Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval*, pages 215–223, Melbourne, AU, 1998b. ACM Press, New York, US.

H. Schwenk and Y. Bengio. Training methods for adaptive boosting of neural networks. In *Proceedings of the 1997 conference on Advances in neural information processing systems 10*, NIPS '97, pages 647–653, Cambridge, MA, USA, 1998. MIT Press.

M. Singh and M. Valtorta. Construction of Bayesian Network Structures From Data: A Brief Survey and An Efficient Algorithm. *International Journal of Approximate Reasoning,*, 12:259–265, 1995.

P. Sollich and A. Krogh. Learning with ensembles: How overfitting can be useful. *Advances in Neural Information Processing Systems*, 8:190–196, 1996.

A. Strehl, J. Ghosh, and C. Cardie. Cluster Ensembles - A Knowledge Reuse Framework for Combining Multiple Partitions. *Journal of Machine Learning Research*, 3:583–617, 2002.

J. Su, H. Zhang, C. X. Ling, and S. Matwin. Discriminative Parameter Learning for Bayesian Networks. In *Proceedings of the 25th international conference on Machine learning*, pages 1016–1023, 2008.

J. Surowiecki. *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations*. Doubleday Books, 2004.

P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, 2005.

M. Telgarsky and A. Vattani. Hartigan's Method:K-means Clustering without Voronoi. *Journal of Machine Learning Research*, 9:820–827, 2010.

M. Teyssier and D. Koller. Ordering-Based Search: A Simple and Effective Algorithm for Learning Bayesian Networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 584–590, 2005.

R. Tibshirani. Bias, Variance, and Prediction Error for Classification Rules. Technical report, 1996.

C. Utz. Learning Ensembles of Bayesian Network Structures Using Random Forest Techniques. Master's thesis, School of Computer Science, University of Oklahoma, 2010.

G. Valentini and F. Masulli. Ensembles of learning machines. In *Neural Nets WIRN Vietri-02, Series Lecture Notes in Computer Sciences*, pages 3–19. Springer-Verlag, 2002.

L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, Nov. 1984.

A. Van Assche and H. Blockeel. Seeing the Forest through the Trees: Learning a Comprehensible Model from a First Order Ensemble. In *Proceedings of the 17th International Conference on Inductive Logic Programming*, pages 269–279, 2008.

V. N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

J. Vomlel. Two Applications of Bayesian Networks. In *Proceedings of Conference Znalosti*, pages 73–82, 2002.

K. Wagstaff, C. Cardie, S. Rogers, and S. Schrödl. Constrained K-Means Clustering with Background Knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 577–584, 2001.

N. Williams, S. Zander, and G. Armitage. A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification. *Special Interest Group on Data Communication*, 36:5–16, 2006.

W. H. Wolberg, W. N. Street, and O. L. Mangasarian. Machine Learning Techniques to Diagnose Breast Cancer from Image-Processed Nuclear Features of Fine Needle Aspirates. *Cancer Letters*, 77:163–171, 1994.

D. Zaharie and Zamfirache. Dealing with Noise in Ant-Based Clustering. In *Congress on Evolutionary Computation*, pages 2395–2401, 2005.

X. Zhu and X. Wu. Class noise vs. attribute noise: a quantitative study of their impacts. *Artif. Intell. Rev.*, 22(3):177–210, Nov. 2004.

X. Zhu, X. Wu, and Q. Chen. Eliminating class noise in large datasets. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.

X. Zhu, X. Wu, and Y. Yang. Error detection and impact-sensitive instance ranking in noisy datasets. In *Proceedings of the 19th national conference on Artifical intelligence*, pages 378–383. AAAI Press, 2004.

A. Zimmermann. Ensemble-Trees: Leveraging Ensemble Power Inside Decision Trees. In *Proceedings of the 11th International Conference on Discovery Science*, pages 76–87, 2008.