Imbalanced Data Classification with the K-Closest Resemblance Classifier for Remote Sensing and Social Media Texts

by

Cheng Duan

Thesis submitted to the University of Ottawa
In partial fulfillment of the requirements
For the M.Sc. degree in
Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Data imbalance has been a challenge in many areas of automatic classification. Many popular approaches including over-sampling, under-sampling, and Synthetic Minority Over-sampling Technique (SMOTE) have been developed and tested in previous research. A big problem with these techniques is that they try to solve the problem by modifying the original data rather than truly overcome the imbalance and let the classifiers learn. For tasks in areas like remote sensing and depression detection, the imbalanced data challenge also exists. Researchers have made efforts to overcome the challenge by adopting methods at the data pre-processing step. However, in remote sensing and depression detection tasks, the main interest is still on applying different new classifiers such as deep learning which has powerful classification ability but still do not consider data imbalance as prime factor of lower classification performance.

In this thesis, we demonstrate the performance of K-CR in our evaluation experiments on a urban land cover classification dataset and on two depression detection datasets. The latter two datasets consist in social media texts (tweets), therefore we propose to adopt a feature selection technique Term Frequency - Category-Based Term Weights (TF-CBTW) and various word embedding techniques (Word2Vec, FastText, GloVe, and language model BERT). This feature selection method was not applied before in similar settings and we show that it helps to improve the efficiency and the results of the K-CR classifier.

Our three experiments show that K-CR can achieve comparable performance on the majority classes and better performance on minority classes when compared to other classifiers such as Random Forest, K-Nearest Neighbour, Support Vector Machines, Multi-layer Perception, Convolutional Neural Networks, and Long Short-Term Memory.

Acknowledgements

I would like to offer sincere gratitude to my thesis supervisors, Prof. Diana Inkpen from the University of Ottawa and Dr. Nabil Belacel from National Research Council Canada. This thesis would not success without their continuing assistance and support. I gratefully acknowledge the financial support provided by them through helping me find TA, RA and summer internship positions. I also thanks all committee members Dr. Yuhong Guo and Prof. Miodrag Bolic. Finally, I would like to thank the continuing support from my parents and my girlfriend.

Dedication

Dedicated to my beloved parents Xiuqing and Fenghua, my loved girlfriend Tianyi.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Machine learning algorithms have been among the most popular and widely-used techniques to handle classification tasks over the past years. One of the fundamental requirements for machine learning classifiers is data with appropriate quality. However, in practice, many datasets have an emerging challenge of imbalance. Imbalanced datasets affect supervised classification algorithms in various way, for example when assigning prior probabilities to each class (Chawla et al., 2004). In real-world applications, the challenge of imbalance in the data is quite common, including the areas of financial fraud detection, medical diagnosing, etc. More importantly, the minority class is often the crucial one that needs to be learned, and there could be a serious cost for incorrect classification Elkan (2001). For instance, detecting whether a user has a rare disease is a classification task with extreme imbalance in the data, because there is a limited volume of information about the rare diseases.

Generally, the typical solutions for addressing the imbalanced date challenge aims at modifying the dataset itself, to transform it into a balanced dataset. For instance, in under-sampling, the training data is modified by removing a certain number of instances from the classes with more data. This allows a classifiers to assign 'fair' prior probabilities to each class. However, this method could affect the generalization ability of the classifier when the original dataset is relatively small. With limited information (after the under-sampling) being used for learning, the classifier might not be able to generalize to future cases, even if it has good performance on the current dataset. Intuitively, instead of modifying the original dataset (by resampling), it is preferable to tackle the challenge from the classifier's perspective. More specifically, it would be better to have the classifier properly learn the data representation even with data imbalance. Even further, López et al. (2013) compared the existing approaches to solve the imbalanced dataset classification challenge. The authors concluded that future research is needed to overcome the imbalanced data challenge by focusing on detecting and measuring the most significant data properties, in order to be able to define good solutions. More specifically, they suggested to explore classification approaches that can overcome data imbalance in regard to six aspects: the

presence of small disjuncts, the lack of density or small sample size, the class overlapping, the noisy data, the correct management of borderline examples, and the dataset shift (We discussed these factors in detail in section 2.2).

Based on this motivated, we explore an algorithm named K-CR (K-Closest Resemblance Classifier) that was proposed by Belacel and Boulassel (2004). It was designed as a multi-criteria fuzzy classification procedure and it aims to tackle the challenge of imbalance in data at the algorithm design level. Based on the results from Belacel and Boulassel (2004), it can be concluded that K-CR outperforms other classifiers from the following perspectives:

- The K-CR classifier explores the intrinsic characteristics of the dataset. By intrinsic characteristics, it refers to how 'important' a feature is for a class. It assigns 'fair' prior probabilities to majority classes and to minority classes. More specifically, K-CR assigns n sets of weights to the n classes (one set per class). Those weights define the features' 'importance'. K-CR works well even on a multi-class imbalanced dataset by leveraging this feature weights. Moreover, Belacel and Boulassel (2004) mentioned that the K-CR classier can be improved by utilising feature selection techniques. Therefore, we will explore this possibility.

- It automatically provides weights for each feature of the dataset. Therefore, it is possible to provide detailed information about classification predictions.

With the above motivation and knowledge, we summarise our objectives in the next section.

1.2 Thesis Objectives

The first goal of this thesis is to implement the K-CR algorithm in Python, as it was previously implemented in C++ and Python is a more commonly-used programming language in Machine Learning. Python also has variety of useful libraries and packages that allow the implementation to be efficient. The second goal is to evaluate the K-CR algorithms in more applications. For this, we chose two tasks. The first task is urban land cover classification which is a common task in remote sensing research. We chose it with the motivation of evaluating the classifier on an imbalanced dataset from a major research field. The dataset we chose is the GEOBIA dataset (Johnson and Xie, 2013) which contains information extracted from images. Then, we chose the second task: depression detection from social media. We chose it for several reasons. K-CR was applied in medical applications (with imbalanced datasets) and achieved good performance. Another reason is that depression detection task becomes more and more important, as mental health issues have been growing during the past years. Recently, the problem became even more prevalent. For example, according to CMHA (2020), the number of calls related to mental health received daily in Nova Scotia is 2700% higher during COVID-19 pandemic than usual. For the depression detection tasks, we employed two social media datasets extracted from Twitter: Bell Lets Talk (Jamil et al., 2017) and CLPSych2015 shared task (Coppersmith et al., 2015) datasets. During our evaluation experiments, we have three sub-goals:

- As text data is involved in one of the tasks, one important step is mapping text to vectors of real numbers. Therefore, one of the sub-goals is to explore how different word embedding techniques contribute to classifiers' performance.
- Feature selection has been a proved effective solution for alleviating the data imbalance challenge. Thus, the second sub-goal is to explore an effective way to utilise a suitable feature selection technique with K-CR for text data.
- Finally, the third sub-goal is to explore how different parameters or factors affect K-CR's performance (such as the number of constructed prototypes).

1.3 Thesis Contributions

This thesis contributes from three aspects:

- We propose to adopt a feature selection technique named Term Frequency - Category-Based Term Weights (TF-CBTW) to further improve the performance of K-CR on imbalanced datasets. To our knowledge, this feature selection method was not applied before with state-of-the-art text classification methods.
- We evaluate our approach on urban land cover classification and depression detection tasks with imbalanced datasets. We show that our approach can achieve comparable performance on majority classes and better performance on minority classes than Random Forest, K-Nearest Neighbour, Support Vector Machines, Multi-layer Perception, Convolutional Neural Networks, and Long Short-Term Memory.
- For the first time, we implemented a Python version of K-Closest Resemblance (K-CR) classifier.

1.4 Thesis Organisation

This thesis has seven chapters in total:

- Chapter 2 discusses related work about the imbalanced data problems. Besides, we include knowledge of multi-criteria fuzzy classification methods since the K-CR classifier relates to them. Furthermore, urban land cover classification and depression detection are mentioned as they are the two tasks for exploring the imbalanced data problems.
- Chapter 3 presents background on several traditional classification methods, including K-Nearest Neighbour, Random Forest and Support Vector Machines. Then, a deep learning method, Convolutional Neural Networks (CNNs), is presented since we will use it to conduct experiments on the Urban Land Cover classification task.

Another deep learning method, Long Short-Term Memory (LSTM) is also presented since we will use it for the text classification tasks. In addition, since we will employ word embeddings for the text classifiers, we include background knowledge about word embedding techniques, including Word2Vec, FastText, GloVe and the state-of-the-art language model BERT. Finally, we introduce the Category-Based Term Weight feature selection method that we propose to add in order to improve our classifiers.

- Chapter 4 describes in detail the idea, the algorithm, and the applications of the K-Closest Resemblance classifier.

- Chapter 5 introduces the GEOBIA, Bell Let's Talk, CLPSych2015 datasets, the evaluation methods, our experimental setup, and our results. Part of the experiment results with the GEOBIA dataset were published in Belacel et al. (2020a).

- Finally, chapter 6 concludes the thesis and proposes several directions of future work.

Chapter 2

Related Work

In this chapter, we discuss related work about the imbalanced data problem. Besides, urban land cover classification and depression detection are discussed, as they are the two tasks for exploring the imbalanced data problems.

2.1 The Data Imbalance Challenge and Solutions

Classification tasks on imbalanced datasets have been long-standing challenges. We call a dataset imbalanced when one or more classes have a much larger number of instances compared to the other classes in the dataset. There are varies of approaches to quantify the ration that is considered imbalance. One of them is the imbalanced ratio (IR). According to Kotsiantis et al. (2006) and Lee et al. (2017), IR is a proportion samples in the number of majority class (negative class) to the number of minority class (positive class). Given a binary dataset with classes $C_{majority}$ and $C_{minority}$, IR is $(C_{majority}/C_{minority})$. Larger the IR is, more imbalance the dataset is. When IR is 1, the dataset is balanced.

Classifiers usually perform well for the dominating classes, while obtaining weak performance on the minority classes. Sometimes, such a classifier is mistakenly considered as a good one solely based on high accuracy scores (due maninly to the high accuracy on the dominating classes). However, in practice, the dominated (minority) classes are more important for domains such as business decisions or medical diagnosing. FernáNdez et al. (2013) showed that this challenge appears often in binary classification tasks, e.g., fraud detection, but also in multi-class classification tasks, like depression detection. As there are several dominated classes in multi-class datasets, the problem becomes more challenging to tackle.

Fernández et al. (2010) claimed that many classifiers assume the fed dataset is balanced. As a result, many classifiers are ineffectively-trained and do perform not well on predicting minority labels although a good overall score is achieved. There are several possible reasons for such failure:

- Classifiers are aiming at incorrect learning goal. In many cases, accuracy is used

to measure how well a classifier performed over a classification task. However, the accuracy scores can be easily manipulated by dominating classes.

- Since minority classes contain fewer number of instances, they may be considered as outliers by classifiers during the training process.

As data imbalance challenge continuously draws communities' attention, many methods have been developed. Basically, López et al. (2013) concluded them into 3 groups:

- Data re-sampling
- Cost-sensitive learning
- Ensemble techniques

Data Re-sampling

For data re-sampling, there are three popular groups of solutions:

- Under-sampling: this works by removing a certain amount of instances from the majority class. The goal is to keep the same amount of instances in the majority class as in the minority class. As a result, a balanced dataset is generated and can be used to train standard classifiers. Random under-sampling is usually the simplest method. But the drawback is also obvious. As instances are removed from the dataset, there is a great chance that instances with crucial information are the selected ones. Even if the learnt model achieves high performance during training, it may not generalise to an unseen dataset.
- Over-sampling: similar to under-sampling, this also works by modifying the original dataset. Instead of removing instances, it randomly copies the existing instances within the minority class. As a result, the minority class contains more instances, and the overall dataset becomes balanced. The main problem with this is overfitting, as there are repeated samples within the dataset.
- Hybrid-sampling: the methods integrate under-sampling and over-sampling.

In literature, Chawla et al. (2002) proposed SMOTE (Synthetic Minority Oversampling TEchnique) which has been one of the most widely accepted data re-sampling approaches to overcome the data imbalance challenge. It belongs to over-sampling group and achieves outstanding performance for its improvements on instances duplication. For instance in figure 2.1, there is an imbalanced two-dimensional dataset. There are two classes from which the blue ones belong to minority class. With SMOTE, the K-nearest-neighbors of each minority sample are first obtained. Then, as shown in figure 2.2, the algorithm synthesizes a new sample at a random location on the connected line between the focused sample and its nearest neighbours. The green dot in the figure represents the synthesised sample. Based on SMOTE, there are several extensions such as Borderline-SMOTE proposed by Han et al. (2005), Safe-level-smote developed by Bunkhumpornpat et al. (2009), etc.

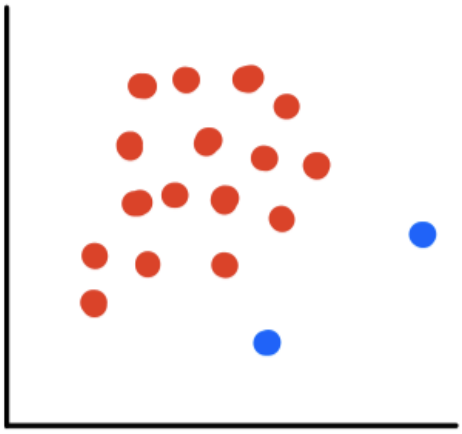


Figure 2.1: Example of an imbalanced dataset

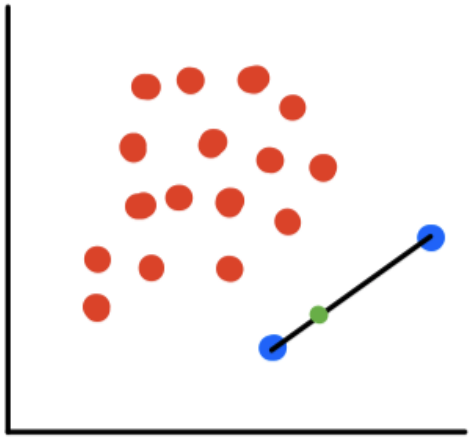


Figure 2.2: Example of SMOTE synthetic data point

Cost-Sensitive Learning

Data re-sampling methods mainly contribute by working on data level modifications. In contrast, Domingos (1999) points out that cost-sensitive learning integrates both data level and algorithm level efforts.

Domingos (1999) summarizes the main idea of cost-sensitive learning as taking the cost of each incorrect classification into account. Before classifiers' learning, a cost matrix is created, where the cost values within it are defined by experts of the tasks or from references. During learning, those cost values will be taken into account when minimising the classification loss. Due to the fact that the minority class is more important than majority class, a higher cost will be assigned for each incorrect classification of a minority class instance. Given a cost matrix, a classifier should learn to classify instances into the class with the lowest cost.

	Actual Negative	Actual Positive
Predicted Negative	0	88
Predicted Positive	5	0

Table 2.1: Example of a cost matrix for fraud detection

For example, assume table 2.1 represents a cost matrix for a fraud detection task. There is no cost when a correct prediction is made. However, there will be a \$88 cost if a positive sample is predicted as negative and a \$5 cost will be applied if a negative sample is predicted as positive. Based on these costs, a total cost matrix can be defined as follow:

$$C = C_{FN} * N_{FN} + C_{FP} * N_{FP} \tag{2.1}$$

where N represents the number of samples. A cost-sensitive learning algorithm aims to minimize the value of C.

The idea of adopting a cost matrix helps to overcome data imbalance. But there are two important disadvantages which stop users from adopting it. First of all, it is usually difficult to correctly define or acquire a good cost matrix. Taking medical classification tasks as an instance, doctors can provide insight information towards disease diagnosing. But they cannot quantify the cost of a incorrect classification. Ethical issues may also be involved if there lives can be lost as a result of incorrect classifications. Second, machine learning algorithms are rarely developed specifically for cost-sensitive learning. Unique modifications to each algorithm are required. The time for developing and testing them can be time consuming.

Ensemble Techniques

Polikar (2006) present an ensemble technique that employs multiple classifiers to learn from the same training data, independently. Then the predictions from each one of the classifiers are aggregated. This method aims to use crowd-classifiers to outperform any individual classifier in that crowd.

There are three major groups of ensemble techniques:

- BAGGing (Bootstrap AGGregating): It is proposed by Breiman (1996). BAGGing takes different learning algorithms in parallel and fit them independently, The training process is concurrent.

  Random Forest is one of the most popular BAGGing method. As shown in figure 2.3, a given dataset is firstly bootstrapped into multiple sub-samples. For each sub-sample, a decision tree is trained. Then, with a chosen aggregation algorithm, the final prediction is generated by aggregating the predictions from all the decision trees.

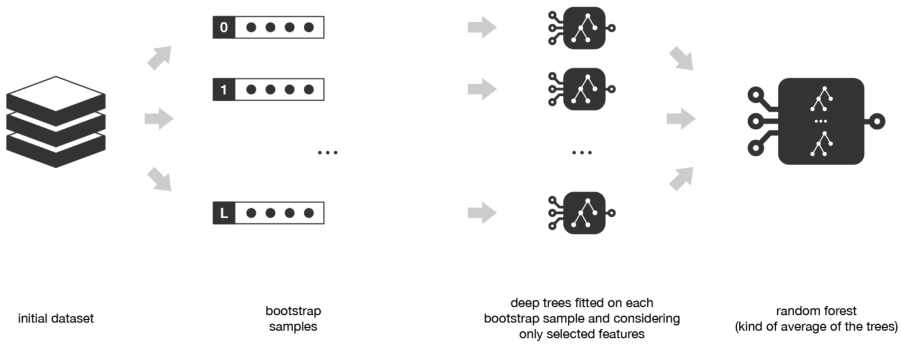


Figure 2.3: Example of BAGGing method adopted from Rocca (2019)

- Boosting: It is based on the question posed by Kearns and Valiant (1988, 1989). As shown in figure 2.4, Boosting takes different learning algorithms sequentially. Different with BAGGing that aims to reduce variance, boosting trains current learner with observations in the dataset which were badly handled by the previous learner. Some of popular boosting methods are adaptative boosting and gradient boosting.

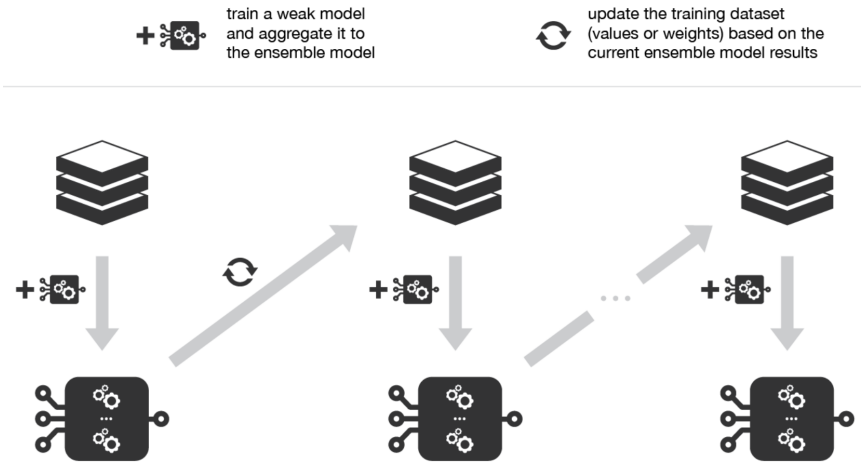


Figure 2.4: Example of Boosting method adopted from Rocca (2019)

- Stacking: Similar to BAGGing, stacking also takes different learners in parallel. As shown in figure 2.5, the initial dataset will firstly be split into two folds. L weak learners will be trained using the first fold. Then, the trained learners make predictions on the second fold. Finally, a meta-model is trained on the second fold based on the predictions obtained in previous step.

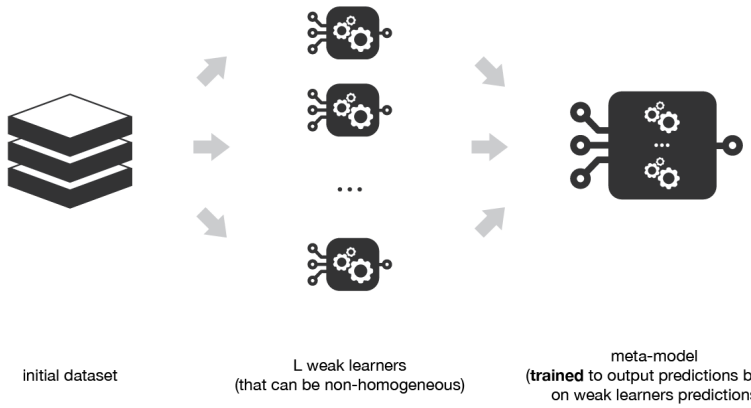


Figure 2.5: Example of Stacking method adopted from Rocca (2019)

López et al. (2013) conducted a series of experiments and summarized different particularities for the three groups above. The ensemble techniques show high accuracy, but their running time can be high. And more importantly, the trained methods are usually difficult to be comprehended by users. Cost-sensitive approaches perform as precise as ensemble methods. But the requirement of having a pre-defined cost-matrix limits its usage. At last, the data re-sampling approaches show robustness and good results. López et al. (2013) concluded that data re-sampling can be taken as the standard approach for solving data imbalance.

2.2 Multi-criteria Fuzzy Classification Methods

As we mentioned in 1.1, López et al. (2013) suggested to explore classification approaches that can overcome data imbalance in regard to six aspects. We further explored those six aspects:

- Small disjuncts. Holte et al. (1989) showed that high misclassification rates could be due to small disjuncts. It is difficult to eliminate the small disjuncts without affecting other disjuncts. Weiss (2004) discovered that the problem affects classification algorithms that are based on a divide-and-conquer strategy, such as decision tree and random forests.

- Lack of density. It refers to the situation where there is a lack of information for classification methods to learn and generalise. The problem becomes more challenging when an imbalanced dataset has a small sample size. Weiss and Provost (2003) conducted experiments with decision tree (C4.5), and they showed that the problem can be solved when there are enough samples, regardless of class distribution.

- Class overlapping. The problem refers to the situation where there are similar quantities of samples for each class in certain feature spaces. Denil and Trappenberg

(2010) focused on this topic and conducted experiments with SVM. Class overlapping caused consistent performance decreasing regardless of the size of training dataset. When class overlapping happened on an imbalanced dataset, the performance further droped.

- Noisy data. According to Weiss (2004), noisy data impacts minority classes more than classes in a normally distributed dataset. Seiffert et al. (2014) showed that Bayesian classifiers and SVMs perform better than rule-based or instance-based algorithm.

- Borderline samples. López et al. (2013) explains them as samples located at the border of the majority and the minority classes. Their experimental results showed that the classification accuracy of minority class instances largely improved when a methodology was used to highlight the borderline areas.

- Dataset shift. It is a problem when the training and the test set follow different data distributions Candela et al. (2009).

We learnt that there are two concepts which address the factors mentioned above. The first one is the fuzzy classification method, that is based on fuzzy logic. According to López et al. (2013), it suppresses the effect of outliers and noisy data by detecting and measuring the class-based importance of different training samples. The second concept is the multi-criteria classification. According to Roy (2013), it looks at the formulation for instances assignment. The assignment involves a step where the most significant data properties are examined. The examination follows profiles or prototypes where vectors of scores on particular criteria or attributes are provided. These profiles or prototypes are seen as independent representations of each class, and formed based on data attributes.

There are various multi-criteria classification methods such as the PROcédure d'Affectation Floue pour la problématique du Tri Nominal (PROAFTN) proposed by Belacel (2000) and Genetic Multicriteria Decision Analysis (gMCDA) proposed by Goletsis et al. (2004). Brasil Filho et al. (2009) took them as imortant multi-criteria classification methods and compared their performance. The authors concluded that PROAFTN has better overall results than gMCDA and the employment of prototypes has an important impact over the classifiers' performance. The K-Closest Resemblance (K-CR) classifier adopted in this thesis is an extension of the PROAFTN method, proposed by Belacel and Boulassel (2004). It inherited valuable parts of PROAFTN on overcoming data imbalance. But it is a prototype-based supervise learning method, rather than a fuzzy classification method like PROAFTN. We will introduce the K-CR classifier in section 4.

2.3 Land Cover Classification

Land Cover (LC) classification is one of the most important tasks in the remote sensing field (Maxwell et al., 2018). The main task of LC classification is to detect which class a land image belongs to or what kinds of land covers exist in the image. According to Pal and

Mather (2005), SVM was tested against the Urban Land Cover (ULC) classification problem and showed good results. Also in Ghimire et al. (2012) research, the ULC classification was tested against several classifiers including Random Forest (RF). The results show that ensemble methods (like Random Forests) perform better than non-ensemble methods (like Decision Tree). Furthermore, Maxwell et al. (2018) conducted a comprehensive review of applying multiple different classifiers over ULC classification task. They tested classifiers including Support Vector Machines, Decision Trees, Random Forest, Boosted Decision Tree, Artificial Neural Networks, and K-Nearest Neighbours. Among them, RF achieved the best performance over the two tested ULC datasets. In addition, Ma et al. (2019) did a meta-analysis of the deep learning usage in remote-sensing applications in 2019. Deep learning methods have been the main interest in ULC classification tasks since 2014.

Although there has been a large amount of research seek to improve ULC's classification accuracy, the main interest persists at adopting state-of-the-art classification algorithm such as the deep learning models. However, they do not guarantee the performance if there is data imbalanced. As mentioned above, Ghimire et al. (2012) used a Landsat-7 Enhanced Thematic Mapper-plus (ETM+) dataset. The authors analysed possible reasons for not obtaining good performance with well-defined evaluation criteria. One crucial factor, the imbalanced distribution of the classes, was not considered. But in the ETM+ dataset, there are 163 instances of 'urban' land, while only 22 of 'pasture' land. In Maxwell et al. (2018)'s case, the authors did find that data imbalance affects classification performance, but the only method they applied to tackle the challenge was random over-sampling. The latter has been shown above to not be the best method. From their experimental results, the best *kappa* was achieved when feature selection was applied, but without over-sampling. The *kappa* achieved with original dataset (without any feature selection or data balancing) was even higher than the one with over-sampling.

To summarise, there has been a lot of efforts on applying state-of-the-art classifiers to ULC classification task, and researchers gained some improvements by following this path. But the existing classifiers, including deep learning methods, are not naturally designed for imbalanced data classification. Researchers still address the problem by adopting one of the solutions mentioned in section 2.1. Those solutions will more or less modify and change the nature of the datasets. Therefore, it is valuable to explore a classifier which is designed to naturally overcome data imbalance.

2.4 Depression Detection

As Giger (2018) stated, there has been a rapid rising in using machine learning techniques to medical fields like risk assessment, detection and diagnosing. Besides, according to Canadian Mental Health Association (CMHA, 2020), mental health support has became one of the most required services during ongoing COVID-19 pandemic. Depression detection is one of the most important applications.

Depression conditions are usually challenging to identify. According to (CMHA, 2020), there are five characteristics for self mental health diagnosing. They are the ability to

enjoy life, resilience, balance, self-actualisation, and flexibility. These characteristics are delicate and subjective. Even if one manages to properly self-diagnose, there is no method being introduced on estimating categories (such as depression, PTSD) of mental health issues. As machine learning techniques developed, they attracted researchers' interests on developing prediction approaches with classification solutions.

In the modern society, many people share their thoughts and emotions through online platforms. According to Balani and De Choudhury (2015) and Andalibi et al. (2017), it has been proved that text from social media is 'valuable' in terms of providing insight knowledge about users' mental status. While in practice, there are many research using social media texts as a source of mental health issues detection. More specifically, machine learning techniques were employed to predict if the user has a mental health issue or not. The machine learning algorithms are typically trained on annotated social media texts.

In previous research, one of the most adopted resources for mental health issue detection is LIWC (Linguistic Inquiry Word Count) developped by Pennebaker et al. (2007). It is a textual analysis software which can be used to calculate the degree of each category of words being used by a user. Nichols (cited May 2020) mentions that it has been largely used in psychological studies, mental health diagnosing, etc. Jamison-Powell et al. (2012) used LIWC for detecting insomnia. Preoţiuc-Pietro et al. (2015) used it on depression detection. Apart from those, there is research conducted from other perspectives. Orabi et al. (2018) explored the performance of several deep neural networks including CNN and LSTM, on predicting mental health issues with a limited number of text data. Kumar et al. (2019) also developed an anxiouty-depression prediction model which is trained based on a anxiety-related lexicon.

It must be recognised that in previous work, the adoption of deep learning methods has achieved a certain level of success on depressing detection. However, similar to the situation in urban land cover classification task, applying state-of-the-art algorithms has been the main interest. There is not much focus on the data imbalance problem. The problem should be considered, as the rate of having mental health issues is relatively low (the minority classes are of high interest), and the number of available texts from those patient is even limited. It is expected to have imbalanced distributed datasets. However, there was not much efforts paid from this perspective. In 2015, Resnik et al. (2015) proposed a system based on SVM with TF-IDF to test against the dataset. But they did not mention any operations towards the data imbalance. Orabi et al. (2018) employed three different CNN models and Bidirectional LSTM to test against a CLPSych2015 shared task dataset (Coppersmith et al., 2015). There are 55.5% of control users (normal users), while only 21.5% for PTSD and 28% for depression. The authors noticed the data imbalance and performed a 5-fold cross-validation with stratified sampling. The method helps maintain same class distribution in each fold as the complete dataset. If the complete set is imbalanced, each fold will still be imbalanced. Therefore, the imbalance problem is left to the classifiers to overcome, while the tested ones are not designed to solve the problem. Jamil et al. (2017) collected a Bell Let's Talk dataset that contains about 40% data for the class of interest, named 'depressed'. This is a better situation than the one of the CLPSych 2015 dataset. But the dataset is still imbalance. The authors of the paper adopted undersampling and SMOTE to tackle data imbalance, which are not optimal solutions. In their

results, the best F1-score [1] was achieved with the undersampled dataset.

To summarise, research in depression detection field has similar trend as in LC classification. There has been a lot of effort on applying state-of-the-art classifiers, and researchers gained some extent of improvements by following this path. For the same reasons concluded at 2.3, there are necessities to explore a classifier which is designed to naturally overcome data imbalance.

2.5 Summary

In this chapter, we discussed the nature and solutions of the data imbalance challenge, the multi-criteria fuzzy classification methods and two practical use cases in land cover and depression detection classification tasks. We conclude that the most research conducted in both tasks mainly focused on adopting state-of-the-art classifiers, including some data re-sampling solutions. Those solutions do not necessarily contribute to the performance. The recommended research direction in the literature is to learn the data properties, which meets the K-CR classifier's objective. In the next chapter, we will introduce background knowledge on classification algorithms that we will use.

[1] F1-score is a statistic for measuring the performance of a classifier. It is composed of precision and recall.

Chapter 3

Background

This chapter outlines some of the most widely used classification algorithms, including K-Nearest Neighbour, Support Vector Machine, Random Forest, Convolutional Neural Network and Bidirectional Long Short-Term Memory. These classifiers are introduced as they are used in either remote sensing or depression detection in literature. Thus, they will also be compared with the K-CR classifier (which will be introduced in chapter 4) for their performance over imbalanced datasets in this thesis.

In addition, as an important part for the depression detection from texts, word embedding methods including Word2Vec, GloVe, FastText, BERT, and different manners of adopting them with classifiers are introduced. Finally, feature selection approaches are presented for their abilities to improve the performance of the classifiers over imbalanced datasets.

3.1 Traditional Machine Learning Algorithms

3.1.1 K-Nearest Neighbour

An instance-based learning algorithm makes predictions by comparing a new instance with the instances available in the training dataset rather than learning to generalise Daelemans et al. (2005). Because of this, instance-based algorithms require little time for training and spends more time on predicting.

K-Nearest Neighbour (KNN) is one of the instance-based or memory-based learning algorithms. The main concept of KNN is comparing the features of groups of instances. The instances which share similar features are neighbours and belong to same class label. The class label of a new unclassified instance can be determined by looking for its nearest neighbours. K indicates the number of neighbours to consider, and the class label with the most occurrences among those neighbours will be assigned to the new instance. The KNN algorithm follows 4 steps:

1. Choose an appropriate K value: The value of K largely impact the performance of KNN model. Different K indicates different number of neighbours to consider and it could results in different label assignment. It is important to find the best K which does not underfit or overfit the model. One solution is to do cross validation on the training set with different choices of K. Then to compute the average accuracy and F1-score of the folds for each K. At last, we can find the optimum K by comparing results of different K.

2. Calculate distances between the unclassified instance and the training instances: Distances are key criteria for KNN. There are three popular methods including:

   - Euclidean Distance: it is the L2 norm which takes the square root of the sum of squared distance between two data points as the distance.

     $$d_{Euclidean}(a, b) = \sqrt{\sum_j (a_j - b_j)^2} \tag{3.1}$$

     where a $= a_j, b_j$ represents the j_th attribute of instances a and b.
   - Manhattan Distance: it takes the sum of the absolute distance difference between two data points as the distance.

     $$d_{Manhattan}(a, b) = \sum_j |a_j - b_j| \tag{3.2}$$

     where a $= a_j, b_j$ represents the j_th attribute of the instances a and b.
   - 1-Cosine Distance: it takes the value of 1 minus the cosine distance between two vectors a and b.

     $$dist(a, b) = 1 - cos(\theta) \tag{3.3}$$

     where θ is the angle between two vectors.
   - Hamming Distance: it is different from the two above; it tells if two objects are in the same category or not.

3. Look for the K nearest neighbours of the unclassified instance: A optimised K value is computed in step 1. With the K and a distance option chosen in step 2, the algorithm can then be used to predict unclassified instance by looking for the K nearest neighbours.

4. Count and assign: AS shown in 3.1, K neighbours of the unclassified instance are chosen. Then, KNN counts the number of each class label in the neighbours. Assuming label B has the most occurrences when K=3, the instance will be classified to class B. This is also called voting.

KNN is a simple algorithm and it is easy to implement. The performance of KNN can be good when proper K and distance methods are selected, while it can also perform poorly when the dataset contains many outliers or noise. Furthermore, Larose and Larose (2014) point out that the prediction procedure with KNN is computationally expensive, since it does not have a process of learning the data distribution.

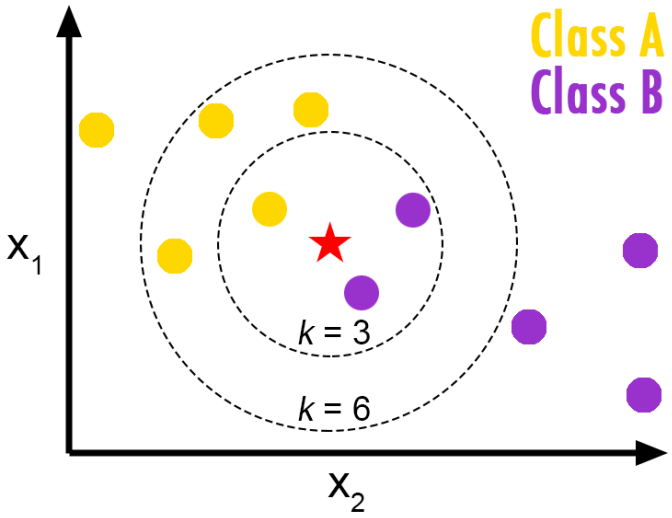


Figure 3.1: An example of using KNN to classify a new instance to class A or B

3.1.2 Support Vector Machines

Support Vector Machines (SVM) is another popular algorithm used by many researchers. Zhang et al. (2006) introduced that the objective of the SVM is distinctly classifing instances by calculating a n-dimensional hyperplane. The hyperplane separates the data points into two groups, where each group belongs to one class label. Figure 3.2 shows an example of using a two-dimensional hyperplane to separate data points. The left one represents the learning process of an SVM model. It maximizes the distances between the hyperplane and each side of data points. As Gandhi (cited Jan 2020) explained, to train an SVM model is to find the optimal hyperplane which maximizes the margin between the data points and hyperplane. The process of hyperplane optimisation follows the equation 3.4:

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} c(x, y, f(x)) \tag{3.4}$$

where w is a weighted vector and λ is an activation function. And $c(x, y, f(x))$ represents the hinge loss used by SVM which has two cases, as shown in equation 3.5. If the true value is same as the predicted value, the cost is 0. Otherwise, the cost will be the results of $1 - y * f(x)$. With the cost function 3.4 and the loss 3.5, SVM aims to achieve a minimum loss value by updating its weights through gradient descent.

$$c(x, y, f(x)) = \left\{ \begin{array}{ll} 0 & : if\ y * f(x) \geq 1 \\ 1 - y * f(x) & : otherwise \end{array} \right\} \tag{3.5}$$

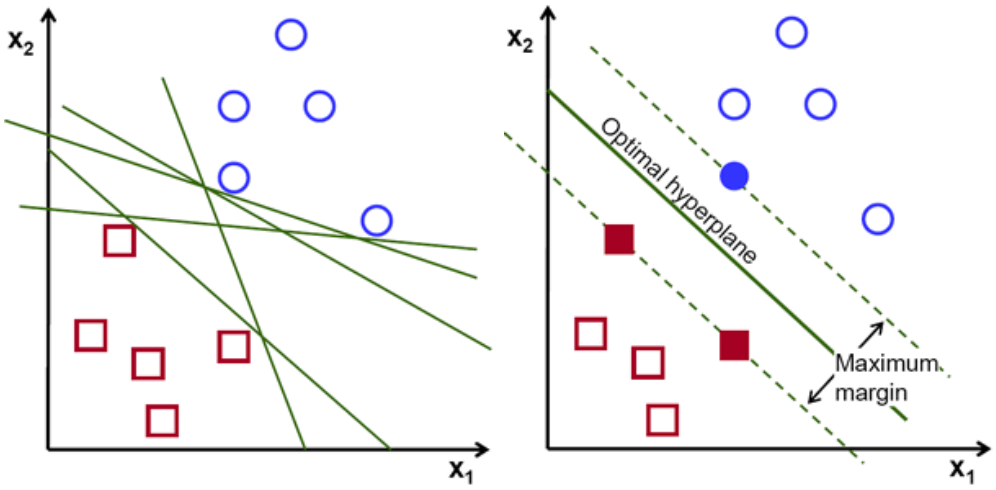


Figure 3.2: Gandhi (cited Jan 2020) provided an example of finding the hyperplane to separate blue circle and red square using SVM

From Duan et al. (2003)'s research, we learn that SVM has lower computational complexity comparing with KNN. The reasons is that SVM is not sensitive to the volume of dataset as KNN is. It is the number of selected support vectors (usually small) which affect SVM's computational complexity. Apart from that, SVM has an advantage of working on both low-dimensional and high-dimensional datasets. In other words, it works on either linear separable datasets or linear inseparable ones. It is possible to train and generalize a SVM model on high-dimensional datasets. In addition, SVM can choose from various kinds of kernel functions such as Linear, Gaussian,or Sigmoid. By design, SVM works on binary datasets. But there are methods such as "one-against-rest" strategy (Bishop, 2006) which can be adopted for multi-class classification.

3.1.3 Random Forest

Random Forest (RF) is another widely used classification algorithm. As mentioned by Kremic and Subasi (2016), RF is popular for its straightforward learning process and fast learning time. Liaw et al. (2002) explained that RF is as an ensemble of decision trees (as figure 3.3) which use random data selection for tree induction. When classifying with RF, each tree in the forest selects some data instances and outputs a classification decision. Then, all outputs are aggregated based on majority voting and in order to output a final decision.

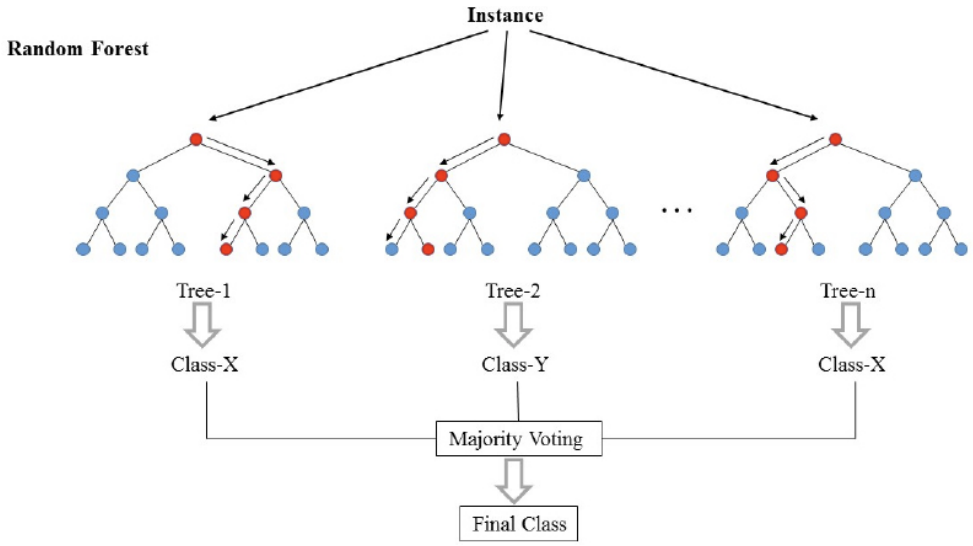


Figure 3.3: An example of Random Forest classifier Kumar

There are two popular types of decision trees, ID3 and CART. Both of them utilise two important terms. First, the impurity criterion is used to measure the quality of a split in the trees. The Gini Impurity is one of the measures when using RF on classification tasks. Singh and Gupta (2014) introduced it as the probability for a randomly chosen instance being incorrectly classified, where the instance is randomly labeled based on the data distribution. It is calculated as:

$$G = \sum_{i=1}^{C} p(i) * (1 - p(i)) \tag{3.6}$$

where C is the unique number of classes in the dataset and p_i stands for the probability of randomly choosing an item which has label i. When training with Gini Impurity, the goal is to maximize Gini gains at each split of the decision trees. Apart from Gini Impurity, Information Gain (IG) is another popular alternative. It is a measures based on Shannon (1948)'s Information Entropy. The entropy equation is:

$$E = -\sum_{i}^{C} p_i log_2 p_i \tag{3.7}$$

where p_i refers to the proportion of the dataset which have label i. Thereafter, as shown in equation 3.8, the IG is calculated by learning the entropy difference between the parent node and the nodes split from that parent node.

$$IG(T) = E(T) - E(T|a) \tag{3.8}$$

where $E(T)$ is the entropy of parent node T and $E(T|a)$ represents the sum of entropy of T's children. As stated by Singh and Gupta (2014), it is to remove more entropy at each split when training with IG.

Although RF is a simple algorithm based on a straightforward idea, it achieves good performance on both Computer Vision (CV) and Natural Language Processing (NLP) tasks. Kontonatsios et al. (2014) successfully used RF to automatically compile bilingual dictionaries. Kremic and Subasi (2016) achieved 97.17% accuracy for a face recognition task using RF. Potthast et al. (2016) also utilised RF on a 'clickbait' application and achieved 0.79 AUC [1] score. Therefore, it is still valuable to compare K-CR with RF's capabilities over data classification.

3.2 Artificial Neural Networks

Artificial Neural Networks (ANN) are important methods for classification tasks. 3.4 shows the simplest architecture of a neural network. There are various kinds of neural network used for different applications. In the following sections, background knowledge of Convolutional Neural Network and Bidirectional Long Short-Term Memory is presented due to their high performance in computer vision and/or natural language processing tasks.

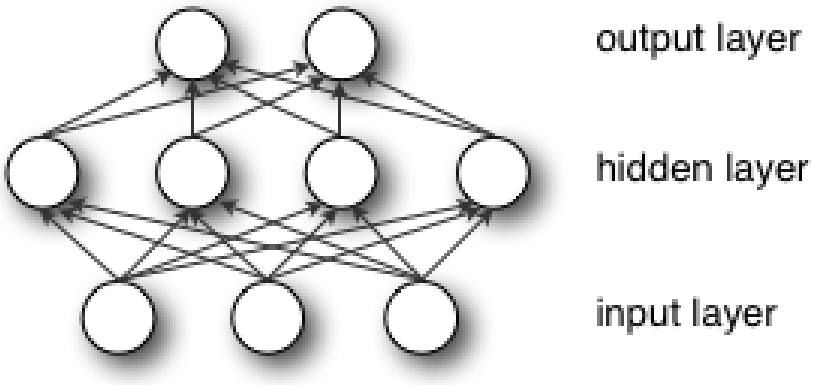


Figure 3.4: The structure of a MLP with single hidden layer

3.2.1 Convolutional Neural Networks

Tu (2018) describes Convolutional Neural Networks (CNNs) as feature detectors which are good at detecting indicative local patterns. As feature detectors, CNNs maintain certain freedom with regard to the position of the local patterns and can combine detected patterns to form fixed-size feature representations.

CNNs usually apply to computer vision tasks. However, from experiments of Yin et al. (2017), CNNs works comparable with classic NLP neural networks like RNN or

[1] Area Under the Curve (AUC) is a statistic between 0 and 1. If AUC is 0, it means that the classifier has no ability to distinguish between classes. In contrast, 1 indicates that the classifier has perfect ability to distinguish between classes.

LSTM [2] in NLP tasks including sentiment classification, relation classification, etc. Kim (word2vec2014) achieved good performance on sentiment analysis and topic categorization with CNNs. They used a simple CNN network and produced sentence embedding with word2vec [3]. Wang et al. (2015) added one extra layer on top of CNN to perform sentiment analysis. Gao et al. (2014) and Shen et al. (2014) from Microsoft explored using CNNs to learn semantically meaningful representations of sentences and performed information retrieval. These works indicate that CNNs can serve as feature detectors in NLP tasks. As stated by Tu (2018), it detects indicative local patterns in CV tasks, and indicative n-grams in the context of NLP tasks.

Convolution and pooling are two typical operations in CNNs. Assume there is a sentence with k words and a sliding window with size s. Then for this sentence, there are $s-k+1$ possible windows. As Tu (2018) mentioned, each of these windows is a kernel that captures local information and returns a scalar value. The values from all these windows result in a vector with size $s-k+1$, which is also named as the convolutional feature map. Since there are various (assume there are m) kinds of detectable n-gram patterns in the sentence, we will have m feature maps with size $s-k+1$. For each feature map, in order to keep the most informative parts, a pooling step is taken. Tu (2018) explained that the pooling help reduce the size of the feature maps by keeping relevant information and eliminating noise. The pooling step is usually set to produce fixed-size outputs. As a result, a convolution layer produces outputs with pre-set size. Two popular pooling techniques are max-pooling and average-pooling. The max-pooling looks for the maximum feature in each feature map, while average-pooling returns the average of feature values. To perform classification, CNNs are usually followed by a fully-connected layer which outputs the finally classification decision. Figure 3.5 shows a visual representation of applying CNNs on textual data, according to Zhang and Wallace (2015).

The pooling step in CNNs is expected to process sentences faster than classic text-targeted neural networks like LSTM and RNNs. Speed is an advantage of CNNs over some other neural networks. Apart from this benefit, Yin et al. (2017) pointed out that the size of hidden layers and batch size largely affect performance of CNNs. In the next section, LSTMs are introduced for their applicability to textual data.

3.2.2 Bidirectional Long Short-Term Memory Networks

Recurrent Neural Networks (RNNs) are classic neural networks that target textual data. As Sak et al. (2014) from Google mentioned, the cyclic connections make RNNs more powerful than feed-forward networks on modeling sequence data (such as text or speech data). RNNs feature a special looping mechanism as shown in figure 3.6. With the looping,

[2] Recurrent Neural Networks (RNNs) are classic neural networks that target textual data. As introduced by Sak et al. (2014), RNNs have cyclic connections which make them more powerful than feed-forward networks on modeling sequence data (such as text or speech data). Long-short Term Memory (LSTM) is a special version of RNN which aims to learn long-term dependencies. Details will be introduced in section 3.2.2

[3] Word2Vec is a word embedding model introduced by Mikolov et al. (2013a) and Mikolov et al. (2013b). The model aims to learn linguistic representations of words. The details will be introduced in section 3.3.1

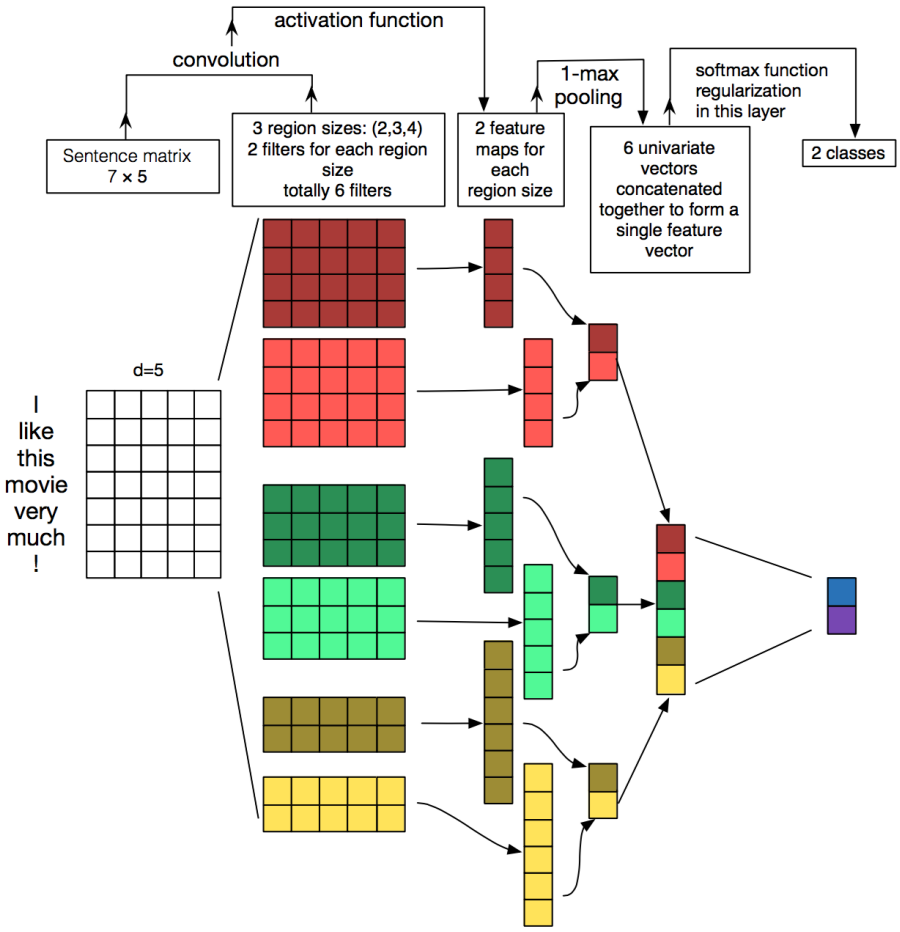


Figure 3.5: Visual representation of applying CNN on words matrix

the processed information from previous input step can be passed to the current step along with new inputs. Figure 3.7 shows an unrolled representation of RNNs. The looped RNNs can be seen as a chain of multiple copies of the recurrent unit. The information passes through from one unit to its successor. Each recurrent unit has a simple structure, such as a single *tanh* layer (figure 3.8).

In practice, RNNs may fail in cases where a large amount of prior knowledge is required for sequence modelling. It has been proved by Bengio et al. (1994) that RNN does not have the ability to learn information relation through gradient descent and large memory amounts are needed for training. They found that the gradient descent becomes progressively incompetent with increasing size of the temporal span over dependencies. LSTMs was developed to solve the issue.

Hochreiter and Schmidhuber (1997) introduced LSTMs in order to learn long-term dependencies. Similarly to RNNs, LSTMs also have a chain structure containing multiple copies of the same unit. For each unit in the LSTM, Colah (2015) presented four interaction layers, as shown in figure 3.9. The unit starts with a sigmoid layer named as "forget gate

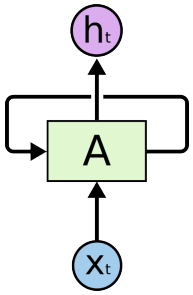


Figure 3.6: Recurrent Neural Networks. Image adopted from Colah (2015).

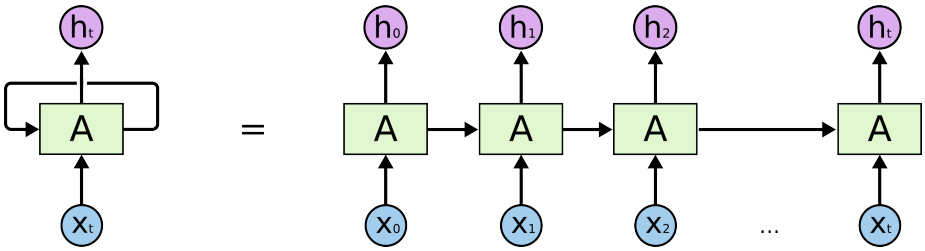


Figure 3.7: Unrolled Recurrent Neural Networks. Image adopted from Colah (2015).

layer". The layer follows equation 3.9, where h_{t-1} is the hidden state from previous LSTM cell and x_t stands for the new input. According to Colah (2015), the output varies between 0 (completely eliminate this information) and 1 (completely keep it).

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \tag{3.9}$$

The second layer is also a sigmoid layer named the "input gate layer" and is calculated as follows:

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{3.10}$$

This layer decides which values from the input will be updated. The third layer is a tanh layer. As explained by Colah (2015), this layer turns new candidate values C_t into a vector, and the vector is scaled by the level of expectation to update each state value. This layer follows the equation below:

$$C_t = \sigma(W_C * [h_{t-1}, x_t] + b_C) \tag{3.11}$$

Finally, the last layer combines a sigmoid operation and a tanh operation, as shown is equation 3.12. Colah (2015) concluded that this layer is the one that decides what information to output.

$$h_t = \sigma(W_O * [h_{t-1}, x_t] + b_O) * tanh(C_t) \tag{3.12}$$

Apart from the four layers, the top horizontal line in figure 3.9 refers to the cell state, where the information passes along cells.

Bidirectional LSTMs are a step further then traditional LSTMs. Instead of learning with one LSTM, bidirectional LSTMs learn with two LSTMs. One of them learns the sequence in its original order, while another one learns the sequence reversely. Graves and Schmidhuber (2005)'s research proved that bidirectional networks usually achieve better performance than unidirectional ones.

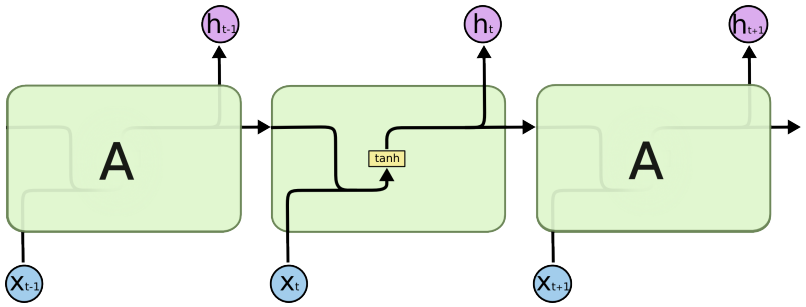


Figure 3.8: Structure of a recurrent unit in a standard RNN. Image adopted from Colah (2015).

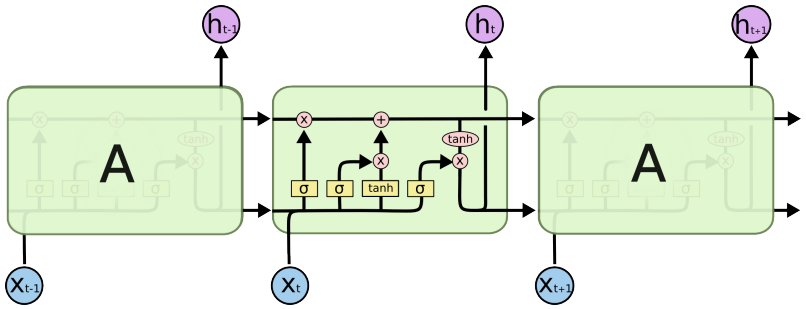


Figure 3.9: Visual representation of a LSTM network Colah (2015)

3.3 Word Embeddings

The depression detection tasks focus on detecting possible signs of depression from text-based sources. The classifiers cannot take the text directly as input. Therefore, word embedding vectors are used to represent the text and they are one of the important preliminary tasks.

Word embeddings are vector representations of vocabularies. There are many word embedding models such as word2vec (Mikolov et al., 2013a) (Mikolov et al., 2013b) which are able to convert text into word embeddings. In word embedding models, a word is projected into a low-dimensional vector, let's call it x_i. Thereafter, any learning algorithm can take the vectors x_i as input.

As an indispensable step, it needs to be considered that the quality of the word embeddings will affect the performance of the classifiers. Therefore, we searched for word embeddings with high quality regarding the following two aspects:

1. Adopting different word embedding approaches from literature. In this thesis, we first selected both Skip-gram and Continuous Bag-Of-Words from Word2Vec Mikolov et al. (2013a) and Mikolov et al. (2013b). We did this in order to be consistent with the ones used in Orabi et al. (2018). Their experimental results are referenced for performance comparison with our classifiers. To explore more possibilities, we also adopted the GloVe model from Pennington et al. (2014) and the FastText one from

?. Furthermore, the state-of-art language model BERT from Devlin et al. (2018) cannot be ignored.

2. Adopting feature selection techniques. The objective of adopting them is to distinguish samples from different classes by highlighting certain vocabulary words. In this case, the goal is to determine if any particular words can represent the characteristics of certain classes. We used the Term Frequency - Category-Based Term Weights (TF-CBTW) technique from Liu et al. (2007) to achieve this. With TF-CBTW, we expect to highlight (assigning higher weight or score) the words that project a sample to one of the classes and to weaken (assigning lower weight or score) the influence of the words that do not.

We will introduce the point 1 above in this section, and section 3.4 will present the details of point 2.

There is a challenge that needs to be noted: some classifiers tested in this thesis take matrices as input, while others, like K-CR, take vectors as input. This challenge and corresponding solutions are discussed more at the end of this section (3.3.5).

3.3.1 Word2Vec

Mikolov et al. (2013a) and Mikolov et al. (2013b) introduced the Word2Vec models. They aimed to train some small scale neural networks to learn linguistic representations of words. By taking a corpus of text as input, the model projects each unique word into a high-dimensional vector space. At the end, each unique word will be assigned a unique vector representation in a low dimensional space.

Word2Vec has two different architectures:

- Continuous Bag-Of-Words (CBOW): With this architecture, for a dataset with n unique words, each word within the vocabulary is converted into an n-dimensional vector. The conversion starts with one-hot encodings. Thus, for the i_{th} word in the vocabulary w, its vector is composed with 0s except 1 for index i. During the training, these vectors serve as input to the word2vec model, and the model learns words representations by using the knowledge of their surrounding words. A window size is set to define which words can be considered as "surrounding" words (context).

  Figures 3.10 and 3.11 represent the architecture of CBOW-based Word2Vec model. The model in figure 3.10 takes one single word x (a V dimensional vector) as knowledge, and tries to predict the vector representation of word y. In figure 3.11, the window size is increased. The model takes C words as knowledge. Instead of a vector, a $C * V$ matrix is used as input.

- Skip-gram: the Skip-gram model has a similar architecture to CBOW, but, as shown in figure 3.12, it has an opposite representation to the one in figure 3.11. Instead of taking the vectors of the surrounding words as input, it takes a vector of the target

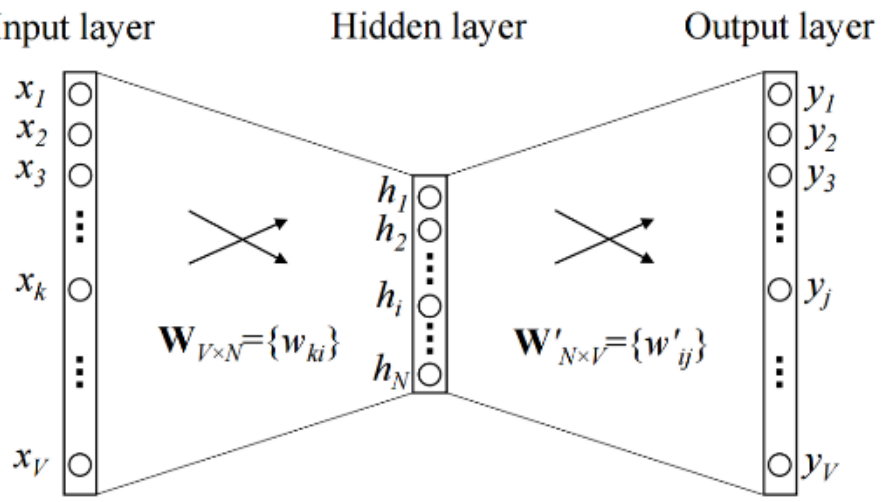


Figure 3.10: An example of CBOW-based Word2Vec model adopted from Rong (2014). The model takes a vector as input and outputs a low-dimensional vector at the end.

word as input. Then, the model outputs the prediction of the vector representations of the surrounding words. The model is trained with back-propagation to learn the best representation of the target words.

According to Mikolov et al. (2013a) and Mikolov et al. (2013b), CBOW-based Word2Vec models have better performance on large corpora and a faster learning process, while the model with the Skip-gram architecture performs better with smaller corpora with less frequent words.

3.3.2 GloVe

Pennington et al. (2014) proposed the popular GloVe (Global Vectors for Word Representations) model which an "unsupervised learning of word embeddings". GloVe is a count-based model, while Word2Vec model is a prediction-based model. According to the definition from Basnet (2017), GloVe learns the semantic similarity between words by counting their co-occurrence frequency.

As stated by Pennington (2014), GloVe takes matrices as input. The matrices include the frequency information of the words based on their appearance in context. Then, the matrices are factorized into low-dimensional matrices. Each row of the matrix is a vector representation of a text. During the training process, a customized weighted least-square objective is set to efficiently learn corpus co-occurrence statistics. Pennington et al. (2014) stated that GloVe is a unsupervised model for learning word embeddings, and it outperforms Word2Vec models in tasks like word analogy and word similarity. The author particularly pointed out that GloVe is designed to capture the vector differences of the meaning specified by the juxtaposition of two words. For instance, the representations of

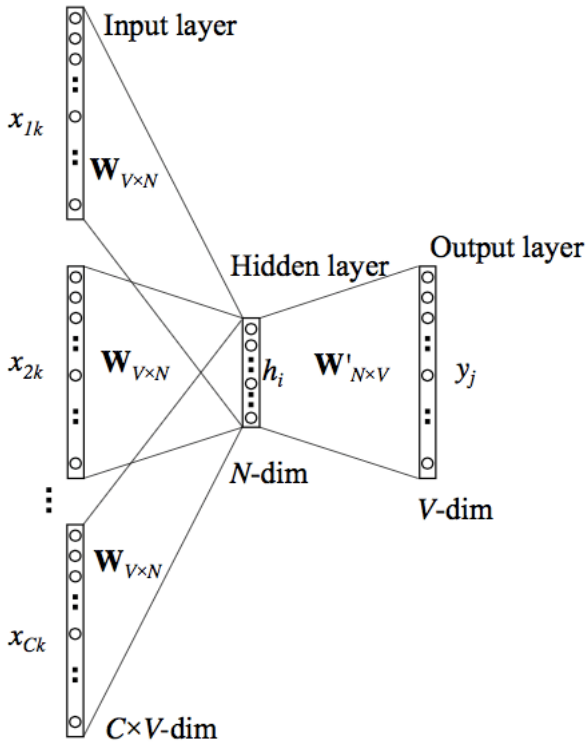


Figure 3.11: An example of CBOW-based Word2Vec model taken from Rong (2014). The model takes a matrix as input and outputs a vector at the end.

the words 'man' and 'woman' are shown in figure 3.13. GloVe seems to be able to capture the differences between the words.

According to Pennington et al. (2014), GloVe creates the embedding matrix by first calculating co-occurrence probabilities. For word w_1 and word w_2, N_{w_1,w_2} stands for the number of times the word w_2 appears in the context of the word w_1. Therefore, for any two words w_i, w_j, the co-occurrence probabilities are as follows:

$$P_{w_i,w_j} = N_{w_1,w_2}/N_{w_1} \tag{3.13}$$

Based on equation 3.13, for any given probe word w_m, the ratio of co-occurrence probabilities is:

$$R = \frac{P_{w_i,w_m}}{P_{w_j,w_m}} \tag{3.14}$$

where V_{w_i}, V_{w_j} and V_{w_m} refer to the vector representation of words w_i, w_j and w_m respectively. The ratio value provides relational information on which two words are more related. If R is 1, the probe word relates to both w_i and w_j. If R is larger than 1, the probe word relates to word w_i. Otherwise, it relates to word w_j. Thereafter, the vectors are produced with a soft constraint:

$$V_{w_i}^T * V_{w_m} + b_{w_i} + b_{w_m} = log(N_{w_i,w_k}) \tag{3.15}$$

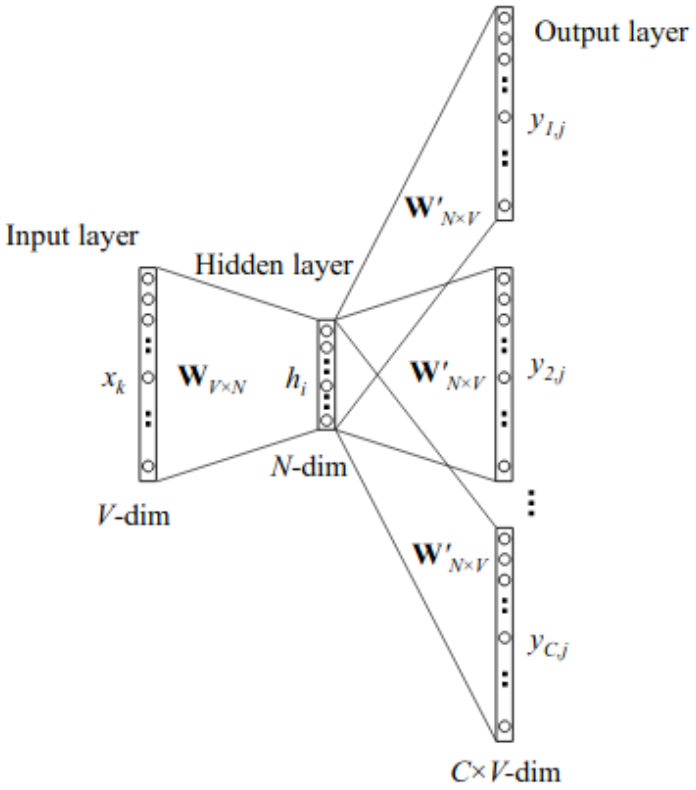


Figure 3.12: An example of skipgram-based Word2Vec model (Rong, 2014)

where b_{w_i} and b_{w_m} are scalar biases. Finally, the GloVe model is trained by minimizing the following cost function:

$$J = \sum_{i,j=1}^{n} f(N_{w_i,w_j})(V_{w_i}^T * V_{w_m} + b_{w_i} + b_{w_m} - log(N_{w_i,w_k}))^2 \tag{3.16}$$

where n is the size of vocabulary. The function f provides customized weights for each word pair. As shown in equation 3.17, x_max is a threshold set by the user. The weight is $(x/x_max)^\alpha$ if N_{w_i,w_k} is smaller than the threshold. Otherwise, the weight is set to 1.

$$f(x) = \begin{cases} (x/x_max)^\alpha & if\ x < x_max \\ 1 & otherwise \end{cases} \tag{3.17}$$

3.3.3 FastText

Bojanowski et al. (2016) from Facebook introduced the FastText approach to help learn word representations. The authors focused on one drawback that Word2Vec and GloVe have: they cannot produce vector representations for Out-of-vocabulary (OOV) words. GloVe's approach of assigning random vectors to unseen words does not count as handling

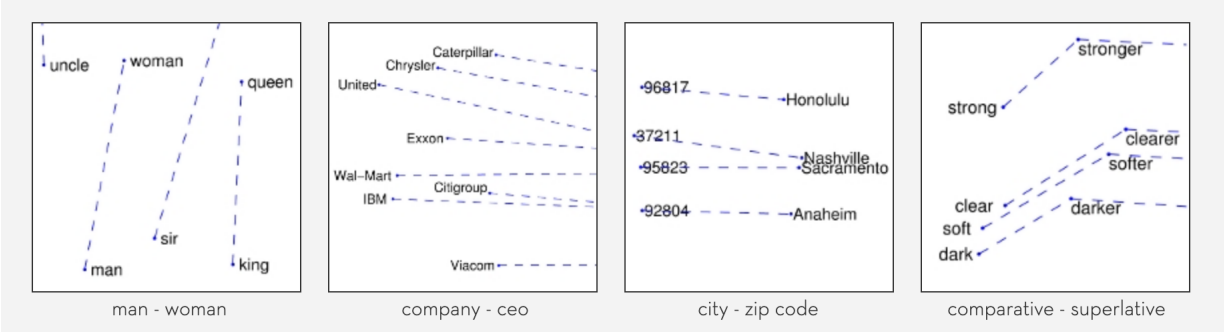


Figure 3.13: GloVe examples of words distinguishing and discrimination

OOV words. In contrast, as mentioned by Bojanowski et al. (2016) and Joulin et al. (2016), FastText is designed to solve the OOV issue by utilising sub-word information. This is especially useful for texts like tweets since they usually involve non-standard vocabularies. For tasks like depression detection, it is possibile that those unusual words contain crucial information for classification. It is not appropriate to ignore those words or to assign them random vectors.

As explained by Bojanowski et al. (2016), FastText solves the issue by taking $n-gram$ of the characters rather than each whole word as a unit. FastText learns embeddings for n successive characters from the words. The vocabulary is largely extended with these character-based embeddings. For instance, in figure 3.14, we assume the word "going" is OOV, and n is 3. With $3-grams$, the word "going" is divided into 5 different parts. The embedding of the word is formed as the sum of vectors of those parts. In addition, as stated by Bojanowski et al. (2016), the FastText model is trained with negative sampling. Sub-words obtained from n-gram are used as positive samples, and a group of randomly picked words from the corpus are used as negative samples. During the training process, a hierarchical softmax mechanism is adopted. As discussed in a earlier literature by Morin and Bengio (2005), this mechanism can dramatically reduce the computation complexity. According to Joulin et al. (2016): "FastText can be trained on more than one billion words in less than ten minutes using a standard multicore CPU, and classify half a million sentences among 312K classes in less than a minute.". This is another key reason for using FastText in our experiments.

3.3.4 BERT

BERT (Bidirectional Encoder Representations from Transformers) is a recent popular language modelling tool published by Google researchers Devlin et al. (2018). The development of BERT has been a milestone in the NLP area by presenting state-of-the-art performance in various NLP tasks including question-answering, natural language inference, etc.

BERT was developed on top of the transformer architecture (a recent developed attention model). More specifically, a bidirectionally trained transformer. The transformer was introduced by Vaswani et al. (2017) and includes a encoder and a decoder (as shown in

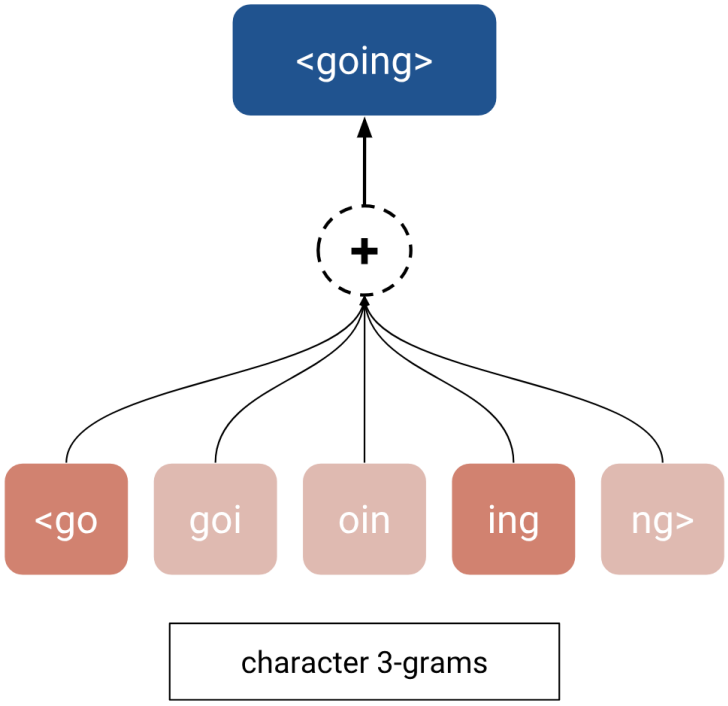


Figure 3.14: An example of producing embeddings for word "going" using FastText. Image adopted from Boukkouri (2018)

figure 3.15). Comparing to techniques such as RNN, BERT does not process a sequence input from either left to right or right to left (as shown in figure 3.16). According to Devlin et al. (2018), by processing the input sequence s bidirectionally, more in-depth information can be learnt by the model.

At a higher level, BERT constructs better language models by learning a bigger picture. It learns vector representation of a word with knowledge of its surrounding context, rather than focusing on predicting the next continuous word.

Overall, as a language model, it is able to produce word embeddings. Since it has been proved to perform well on various NLP tasks, it is valuable to include BERT in our experiments. In the next sub-sections, we briefly introduce BERT, we discuss about customized window sizes, and operations to convert sentences into vectors.

Sliding Windows

Sliding windows is not an existing concept in the original BERT structure but we need it to accommodate the datasets used in this thesis.

As shown in figure 3.17, BERT takes a sequence of tokens as input. As stated by Devlin et al. (2018), BERT is designed to accept sequences with maximum of 512 tokens.

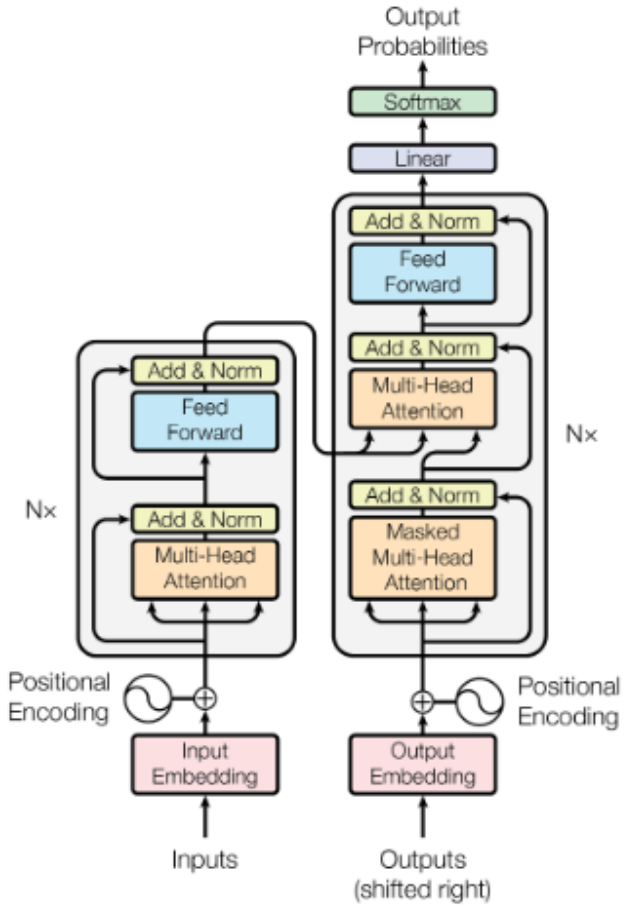


Figure 3.15: Architecture of the transformer Vaswani et al. (2017)

In practice, there are cases in which a sequence has more tokens than this limit. Taking the Bell Let's Talk dataset (details in section 5.3) used in this thesis as an example, it consists of tweets from different users. Since we focused on the user-level depression detection task, all the tweets of a user are combined into one long sentence (text). As a result, there are sequences as long as 10,000 tokens. This length exceeds BERT's limit. Therefore, an efficient approach is required to recursively feed the input sequences. Within this thesis, we adopted two sliding window strategies:

- Fixed size window: This is actually a commonly- adopted method in industry. A fixed size window stands for a fixed size selector. The long input sequence is divided into several pieces by the window, while certain level of overlapping is maintained from piece to piece. The overlapping is needed to minimize the possibilities that a token located at the end of piece A loses its required contextual information which is in the piece A+1. In this thesis, we set the size to 512, to maximize the volume of context that BERT can learn from. The overlapping size between every two window pieces is set to 128. In order words, to process the long sequences of each user, a 512-size window will slide from left to right until the end of each sequence. The

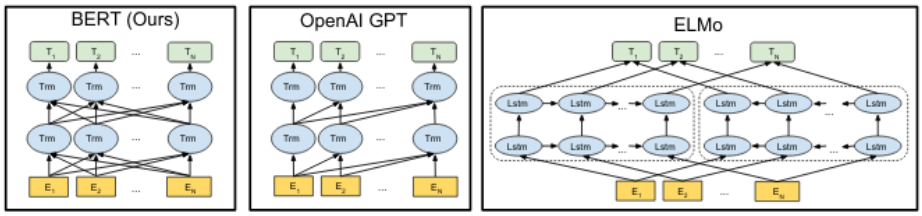


Figure 3.16: Differences of model architecture Devlin et al. (2018)

window slides right 384 tokens at each sliding step, to maintain the desired 128 tokens overlapping.

- Dynamic size window: Similar to the first strategy, a window is used to slide through the sequence, but instead of using a fixed-size window, a dynamic-sized window is used. The intuition is that the long sequence of each user is composed of the history of user's tweets. Theoretically, it is more reasonable to provide information from each individual tweet as contextual background to the language model. Therefore, the window size is dynamically set to the length of each individual Tweet. Only the tokens of each independent tweet will be fed into BERT at one time.

However, since Twitter has a limitation on the number of characters, it can be argued that a long context is divided into several tweets by a user. As a consequence, tweets are not independent of each other. It is unclear which kind of strategy can maintain the most of contextual information. Therefore, we planned experiments to explore how to maximize BERT's ability in case of long input sequences.

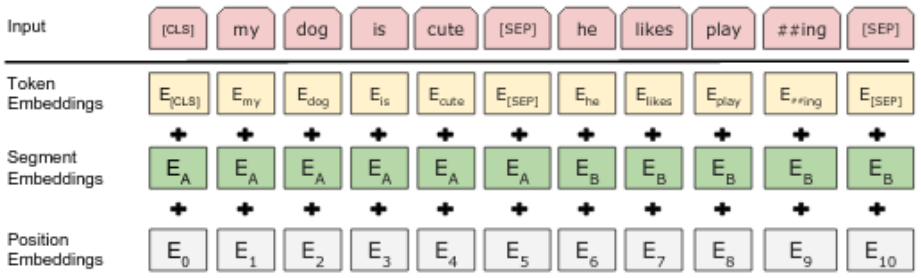


Figure 3.17: BERT input representation adopted from Devlin et al. (2018)

Embedding with BERT

Apart from the required pre-processing steps mentioned in last section, one important post-processing step is required as well. Due to the fact that BERT was developed as a language model rather than targeting any specific task, further steps (adding extra layers to perform a task, such as question answering) are required.

To produce word embeddings with BERT, one step is to select from different embedding strategies. As shown in figure 3.18, for each token, there are 12 layers which can be used to create contextualized word embeddings. As stated by Devlin et al. (2018), the decisions about which individual layer or which combination of layers should be used highly depends on the task. The authors tested a series of embedding strategies. The results are shown in figure 3.19. They were obtained by feeding word embeddings from different embedding strategies to a BiLSTM for a Named Entity Recognition (NER) task.

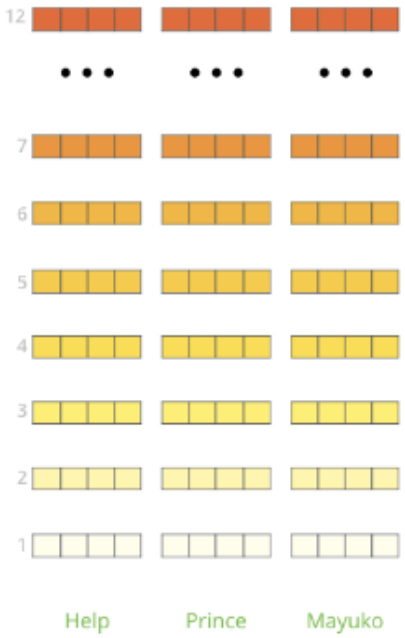


Figure 3.18: Encoder Layers of each token adopted from Alammar

Based on the results from (Devlin et al., 2018), we selected the two following embedding strategies in this thesis:

- Second-to-last hidden layers: the strategy achieves 0.956 F1-score. It takes information from the second to the last layers, and forms single vectors as embeddings. Firstly, we extracted layers from the second to the last. Then, the extracted layers are averaged along the y-axis. For instance, the result of taking the average of the values of dimension 0 from all layers is used as the value of the dimension 0 of the created vector. The created vector has a size of 768. In this thesis, this strategy is used to "directly" generate embeddings for sentences. "Directly" means that there is no middle step for generating word embeddings to convert them into sentence embeddings.

- Sum of the last four hidden layers: the strategy achieves 0.959 F1-score. We adopted this strategy to "indirectly" generate sentence embeddings. This strategy focuses on the last four hidden layers. We first sum them up into one vector along the y-axis.

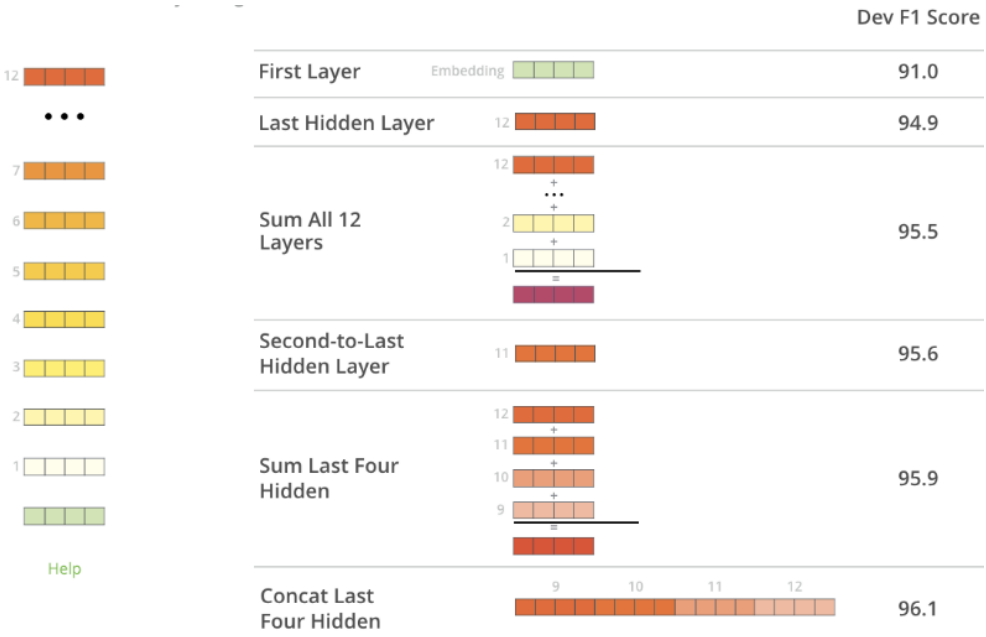


Figure 3.19: F1-score of different embedding strategy. Adopted from Devlin et al. (2018) and Alammar

> The vector represents one of the words in a sentence. A matrix is formed by stacking the vector representations of all words. Finally, we take the average along the y-axis of the matrix to produce the vector representation for the sentence.

Apart from the two strategy introduced above, there is another one with the highrst F1-score (0.961). The reason for not choosing it considers the length of the resulting vector. As shown in figure 3.19, the output vector is created by concatenating the last four hidden layers. The length of the vector is (4* 768) which is 4 times longer than the vectors created by other strategies. Because it is not much better than summing the last four hidden layers, we did not choose it, in order to reduce the processing time of the classifiers. Overall, it is not clear which one of the strategies listed above would provide more contextual information to the classifiers. This is an important factor to explore during the experiments.

3.3.5 Sentence to Vector

Except for BERT with "second-to-last hidden layers" strategy, other mentioned word embedding techniques concentrate only on word-level embeddings. It is necessary to introduce the way we will adopt word embeddings with the K-CR classifier, as well as the differences with the word embeddings used for other classifiers such as Deep Learning models.

First of all, it needs to be remarked that this is not about the word embeddings models themselves, but about the manner of employing the embedding vectors. For instance, Orabi et al. (2018) conducted experiments with CNN and BiLSTM in their paper. Both of those two classifiers can take matrices as input. Therefore, the authors can simply stack all vector representations of the words, and construct a matrix as the sentence embedding.

However, some classifiers, such as K-CR can only take vectors, instead of matrices, as input. Therefore, we adopted two simple approaches: addition and averaging to convert matrices into vectors:

- Addition: the matrix is transformed into a vector by summing along y-axis. For a $m * n$ matrix, m_s values of the 0_{th} dimension are summed up. The result of addition is assigned to the 0_{th} dimension of the output $1 * n$ vector.
- Averaging: based on the addition results, there is one further step to take average on each dimension. For the $1 * n$ vector output from the addition, the value of each dimension is divided by m as in n_i/m, where i is $(0...n)$.

Either of the approaches mentioned above comes with a simple and straightforward intuition. However, it is highly possible to lose certain contextual information. It is necessary to explore better approaches in future work.

3.4 Feature Selection with TF-CBTW

Feature selection is another important pre-processing step that can be applied. As mentioned before, one of the objectives of this thesis is to use K-CR to perform classification on imbalanced datasets without data-re-sampling or ensemble approaches. To accomplish the goal, apart from the efforts from the classifiers themselves, it is necessary to consider other dataset aspects as well.

As described in section 2.1, the data imbalance issue exists when one of classes dominates other classes within a dataset. Thereafter, the classifiers cannot sufficiently learn from minority classes. As a consequence, the classifiers have unsatisfactory performance. In this thesis, we expect the K-CR classifier to tackle the issue. But at in the same time, we also adopt the Term Frequency-Inverse Category Based Term Weight (TF-CBTW) to help increase the performance of K-CR. TF-CBTW helps with the following two aspects:

- Eliminate words with low importance when TF-CBTW will be used on the depression detection tasks. More specifically, the TF-CBTW can be used to form a scored list for the corpus. The words with high scores will be kept, while the ones with low score are eliminated.
- Highlighting terms: this is inspired by the fact that K-CR learns and constructs prototypes feature by feature. It is important to highlight the most informative features in the datasets. Then, K-CR can easily capture the highlighted information during prototype construction. In other words, with this application, the pre-processed words vectors can involve class-based dimension importance information. The detail of feature highlighting will be discussed in section 3.4.2.

In the following sub-sections, we will first introduce the Term Frequency-Inverse Document Frequency which is the foundation of the TF part of TF-CBTW. Then, we will present the background of TF-CBTW and its usage in this thesis.

3.4.1 Term Frequency-Inverse Document Frequency

There are two kinds of feature selection approaches: local feature selection and global feature selection. As explained by Liu et al. (2007), local feature selection refers to discovering distinguishable features for certain classes. Global feature selection means to find the best features across all classes. Since the objective is to learn class-based term scores, local feature selection approach is our preferred choice.

According to Mohod and Dhote (2014), Term Frequency-Inverse Document Frequency (TF-IDF) is a popular feature selection method. It focuses on both local and global feature selection. TF-IDF is composed by two parts:

- TF: which defines the local weight. It specifies the frequency of occurrence of a term within a document. In other words, it defines the weight for a term t within a document d.
- IDF: which defines the global weight. It specifies the global importance of a term t within a specific document d.

Based on the definition, it is obvious that the TF portion is working in a sense of defining the local importance of each term. In addition, we learnt from Leopold and Kindermann (2002) and Lan et al. (2005) that TF works well even on its own. Therefore, we can retain the TF portion. Then, we move to solve the question "how to transform IDF from defining global weight into a class-based importance indicator". In other words, we aim to replace IDF with another indicator so that instead of looking at a term in a global scale (for the whole corpus of texts), it will look at the terms in each class (the documents of each individual class).

3.4.2 Category-Based Term Weight

Liu et al. (2007) proposed the Category-Based Term Weight (CBTW). They developed CBTW with the idea of employing it as a term selection method to replace the IDF portion of TF-IDF.

Before we introduce CBTW, a few notations need to be specified:

- Notation A refers to the number of documents of class c which contain the term t
- Notation C refers to the number of documents of class c which do not contain the term t
- Notation B refers to the number of documents of classes other than c which contain the term t
- Notation D refers to the number of documents of classes other than c which do not contain the term t

	c	$\bar{c}$
t	A	B
$\bar{t}$	C	D

Table 3.1: Information used for generating CBTW

We adopted the table 3.1 from Liu et al. (2007) to visually represent these relationships. Based on the table, Liu et al. (2007) also introduced two important roles:

- A/B: one of the indicators which show whether a feature f is good for representing class c or not. The higher A/B is, the better the feature f is.
- A/C: another indicator which compares features f_1 and f_2. It aims to find the better feature to represent the class c.

There are eight variations of CBTW equations introduced using the terms above. For equations 3.18, 3.19, 3.20 and 3.21, Liu et al. (2007) takes the product of A/B and A/C to combine the probability information:

$$CBTW_1 = log(1 + \frac{A}{B} * \frac{A}{C}) \tag{3.18}$$

$$CBTW_2 = log(1 + \frac{A}{B} + \frac{A}{C}) \tag{3.19}$$

$$CBTW_3 = log(1 + \frac{A}{B}) * log(1 + \frac{A}{C}) \tag{3.20}$$

$$CBTW_4 = log[(1 + \frac{A}{B}) * (1 + \frac{A}{C})] \tag{3.21}$$

Each of them can be further extended to the equations 3.22, 3.23, 3.24 and 3.25:

$$CBTW_5 = log(1 + \frac{A+B}{B} * \frac{A+C}{C}) \tag{3.22}$$

$$CBTW_6 = log(1 + \frac{A+B}{B} + \frac{A+C}{C}) \tag{3.23}$$

$$CBTW_7 = log(1 + \frac{A+B}{B}) * log(1 + \frac{A+C}{C}) \tag{3.24}$$

$$CBTW_8 = log[(1 + \frac{A+B}{B}) * (1 + \frac{A+C}{C})] \tag{3.25}$$

Recalling the original TF-IDF, for a term t_i in document d_j, $TF(t_i, d_j)$ represents the TF value. And TF-IDF can be represented as:

$$TF - IDF(t_i, d_j) = TF(t_i, d_j) * IDF(t_i) \tag{3.26}$$

where

$$TF(t_i, d_j) = \begin{cases} 1 + log(n(t_i, d_j)) & if \quad n(t_i, d_j) > 0 \\ 0 & otherwise \end{cases} \tag{3.27}$$

With CBTW introduced, we could have several new score indices. For instance, with CBTW1, the score index is defined as 3.28:

$$\begin{aligned} score\_index(t_i) = & \quad TF(t_i, d_j) * CBTW_1(t_i) \\ = & \quad TF(t_i, d_j) * log(1 + \frac{A}{B} * \frac{A}{C}) \end{aligned} \tag{3.28}$$

Similarly with CBTW3, the score is defined as 3.29:

$$\begin{aligned} score\_index(t_i) = & \quad TF(t_i, d_j) * CBTW_3(t_i) \\ = & \quad TF(t_i, d_j) * log(1 + \frac{A}{B}) * log(1 + \frac{A}{C}) \end{aligned} \tag{3.29}$$

Liu et al. (2007) conducted classification experiments with two imbalanced datasets: MCV1 and Reuters-21578. They compared their proposed approach with existing feature selection methods including Correlation Coefficient, Chi-square, Information Gain, Odds Ratio and Relevance Frequency. Results show that the experiment with CBTW1 outperformed other approaches and achieved the highest F1-score. Therefore, we selected CBTW1 and formed the TF-CBTW feature selection method.

3.4.3 TF-CBTW Usage

The TF-CBTW methods is used to give scores to terms to help increase K-CR's performance. Recall from section 3.3.5 that sentences are converted from a matrix into a vector. The term scores S can be added to the matrix M in order to highlight certain terms (which is the main objective mentioned at the beginning of this chapter).

The score is used before the addition or avenging operations of sentence2vector. Each row of matrix M_i is a independent vector which represents one actual term in a tweet. In the same time, with TF-CBTW, a score S_t can be obtained for each word. Therefore, it is simple to produce the score of the vector, to include term score information into the vector. Thereafter, a new sentence matrix with class-based highlighted terms can be constructed by stacking newly obtained vectors. A pseudo-code of the entire process is shown in algorithm 1.

TF-CBTW with BERT

TF-CBTW usage with BERT is different than for other word embedding methods. For the other methods, the model produces a word vector directly, so TF-CBTW can be applied directly to the vector. However, BERT creates vectors by working around the layers within the hidden state. Because there are weights associated with each neuron of each layer, it does not make sense to apply TF-CBTW (as another weights) on any of those layers. Therefore, the TF-CBTW is applied in another way.

Algorithm 1 Apply TF-CBTW score to a sentence vector

$nC \leftarrow$ Number of classes
$nT \leftarrow$ Number of tweets
$nW \leftarrow$ Number of words
$words \leftarrow$ Set of unique words in entire data set
$score \leftarrow$ A list of three sets of score for each unique word of each class

for $c \leftarrow 1$, nC **do**
    for w *in* $words$ **do**
        $score[c][w] = TF - CBTW[w]$
    end for
end for

for $c \leftarrow 1$, nC **do**
    There are nT Tweets belongs to class m
    for $t \leftarrow 1$, nT **do**
        There are nW unique words belongs to tweet t
        for $w \leftarrow 1$, nW **do**
            Apply word embedding on each word to obtain the corresponding vector V_w
            $V_w^{new} = V_w^{old} * score[c][w]$
        end for
    end for
end for

Replace each row of original matrix with V_w^{new}
Continue with sentence to vector operations, either addition or averaging

Within the corpus, each term will be assigned a TF-CBTW term score. As a result, a ranked list of the words (terms) can be created based on the scores. Before a sequence is fed to BERT, terms that are not 'important' will be removed from the sequence according to the ranked list. The 'importance' is defined by setting a threshold on the ranked list. For instance, the top 1000 terms in the list are 'important' terms. Different threshold settings were tested during experiments and will be discussed in section 5.3.3

3.5 Summary

In this chapter, we discussed several popular machine learning classifiers. Models' architecture and performance in the literature are introduced. We concluded that traditional classifiers like RF, SVM and KNN are still showing compelling performance on modern ML tasks. Along with popular deep learning techniques like CNN and Bi-LSTM, we need to tested in our experiments in order to compare them to the K-CR classifier. In addition, we introduced a series of popular word embedding techniques and the BERT language model. Each of them has different strengths. Experiments are needed to discover which one contributes the most to improving the performance of K-CR for our tasks. Finally, we discussed a feature selection method, TF-CBTW. Experiments are also needed to learn to what extent it contributes to the performance of K-CR. In the next chapter, the K-CR classifier will be introduced in detail, including its background, algorithm, and applications.

Chapter 4

The K-Closest Resemblance Classifier

The K-Closest Resemblance Classifier (K-CR) was developed by Belacel and Boulassel (2004) as an extension of his PhD research (Belacel, 1999).

K-CR is a supervised learning algorithm. It is able to measure the resemblance between instances by calculating certain preference relations. It produces several sets of rules named as *prototypes* for each class. The sets of rules refer to sets of intervals corresponding to the features in the dataset (details about the rules or prototypes are included in section 4.1.1). An instance will be classified using these prototypes. Each inference will produce a score, and the entire process will produce a set of scores. Then, the instance will be assigned to the class of the prototype with the highest score. To summarize, K-CR measures the closeness between an instance and all available prototypes. Then the algorithm assigns an instance S to a class c when and only when S is close enough to at least one of the prototypes of class c. The K-CR classifier works by the following six steps:

1. Learning phase
2. Score matrix
3. Outranking relations
4. Score function
5. Choose the set of prototypes
6. Assignment decision

In the following sections, we will introduce each of the six steps mentioned above. In addition, existing applications which use the K-CR classifier are also mentioned.

4.1 Learning Phase

Similarly to other popular machine learning algorithms, K-CR requires a training phase to learn the targeted data representation so that it can perform classification on unseen data points later on. There are two main objectives during K-CR's learning phase:

- Producing prototypes which represent the data of each class
- Simulating the optimised 'importance' weights for each feature of each class

4.1.1 Producing Prototypes

The prototypes in K-CR have the role of filtering or grouping the input instances. Each prototype encodes information or standards for each feature of an instance. Those standards are used to judge whether an instance is qualified for the class which the prototype belongs to. In other words, the idea is to determine the representation of general value of feature 0 for class 0, 1, ... n, respectively. Then it repeatedly performs the same operations for all available features in the dataset. This operation is a discretization step. There are several discretization options such as linear regression, K-Means, etc. In this thesis, K-Means was chosen. In future work, it could be explore how different discretization approaches affect K-CR's performance.

Pseudo-code of using K-Means to discretize is shown in algorithm 2. As shown in figure 4.1, K-Means is applied class by class. For instances of each class, K-Means generates K sets of interval boundaries for each feature. As a result, a $n * k$ matrix is created where each index holds one interval. Then, the sets of intervals are used to construct prototypes in the next step.

Algorithm 2 Apply the clustering method k-means to discretize the attributes' value

```
nC ← Number of classes
nA ← Number of attributes
nI ← Number of intervals (i.e., number of clusters)
for m ← 1, nC do
    for n ← 1, nA do
        Apply k-Means algorithm on each attribute X_n
        The generated k clusters represent the intervals' boundaries (i.e., {I^1_nm, I^2_nm})
    end for
end for
```

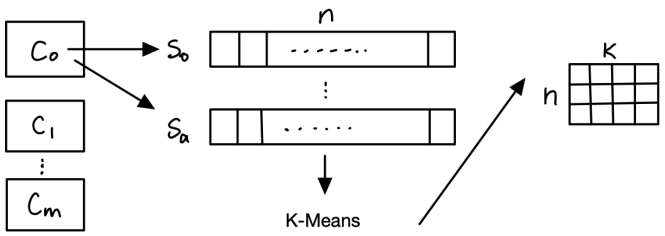


Figure 4.1: Diagram of performing discretization

In algorithm 3, we use pseudo-code to present the steps of creating prototypes based on the sets of intervals obtained above. One of the sub-tasks within this step is to filter 'useful' intervals and eliminate less useful ones. This step helps reduce the number of

prototypes created. The processing time of the K-CR will decrease with a lower number of prototypes. The sub-task starts by setting a threshold value β. In the meantime, a value p is calculated, which refers to the percentage of values of the attribute n that locate within the interval I_{nm} of the class m. If P is equal or higher than β, the interval I_{nm} is preserved. Otherwise, the interval will be eliminated.

Algorithm 3 Building the classification model for K-CR.

Determine a ***threshold*** β as reference for interval selection
$nC \leftarrow$ Number of classes
$i \leftarrow$ Prototype's index
$nA \leftarrow$ Number of attributes
$nI \leftarrow$ Number of intervals/(clusters) for each attribute
$I^x_{nm} \leftarrow$ Intervals $\{I^{1x}_{nm}, I^{2x}_{nm}\}$ for each attribute X_n in each class C^m by applying the algorithm 2
$p \leftarrow$ Percentage of values within the interval I^x_{nm} per class
Generate intervals according to the algorithm 2
$p \leftarrow 0$
for $m \leftarrow 1$, nC **do**
  $i \leftarrow 0$
  for $n \leftarrow 1$, nA **do**
    for $x \leftarrow 1$, nI **do**
      if p of $I^x_{nm} \geq \beta$ **then**
        Choose this interval to be part of the prototype P^m_i
        Go to the next attribute X_{n+1}
      else
        Discard this interval and find another one (*i.e.,* I^{x+1}_{nm})
      end if
    end for
  end for
  if $(b^m_i \neq \emptyset \ \forall g_{jh}\ )$ **then** $i \leftarrow i + 1$
  end if
  (Prototypes' composition):
  The selected branches from attribute X_1 to attribute X_n represent the induced prototypes for the class C^m
end for

Figure 4.2 shows an example of the process within the learning phase. Assuming there are three attributes for each sample and K is set to 4 for K-Means. We will obtain a $3 * 4$ matrix where each circle represents one interval. After interval elimination, we obtain the graph on the right hand side. The circles within the graph are connected from top to bottom. Each complete connection is considered as a prototype. In this example, there are 6 different prototypes constructed.

In theory, the prototypes constructed in this phase are the key for K-CR to better handle imbalanced datasets. The main reason is that the prototypes are class-based. In other words, the K-CR learns the data representation of each class independently. Instances of a class will not dominate or weaken the 'importance' of the instances of any other classes.

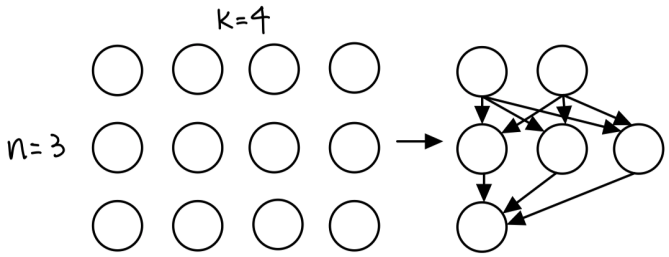


Figure 4.2: Example of learning phase

Therefore, the prototypes of each class are constructed fully based on its properties. They will not be influenced by the properties of other classes, with higher amounts of data points. It is obvious from the example above that some of the constructed prototypes are similar to each other. Some of them share same intervals for all attributes, except for only one difference. Similar prototypes may cause redundant computation and waste time. It is an important task in the future to develop a method for detecting and combining redundant prototypes.

4.1.2 Weight Simulation

Weight simulation is another vital aspect during K-CR's learning phase. During the optimisation process, intial weights are assigned to all existing features in the dataset and the weights are continually updated. The updating rules will be introduced in the subsequent paragraphs. It needs to be remarked that those simulated weights provide the "interpretation" ability of the K-CR classifier. It is possible to learn which feature has higher 'importance' for the task of classification by comparing the values of the weights. For example, in the Iris dataset (Fisher, 1936), there are 3 classes and 4 features for each instance. The weights learnt by using the K-CR classifier are shown in table 4.1. We can learn that the weight of the feature petal width (0.6750) is much higher than that of the other three features. Therefore, it is the most important feature to focus on when classifying flowers in the Iris dataset. Furthermore, we can take petal length as the secondary reference, since it was assigned the second highest weight (0.2446) during training.

It can be argued that the Artificial Neural Networks also evolve by learning certain weights. However, by design, the ANNs project the original features of inputs into a higher dimensional space. The projected features are not understandable by humans. For instance, if we employ a simple MLP (with a 128 neurons in the hidden layer) to process the Iris dataset, the 4 original features are first passed to the hidden layer. At this point, it is not possible to understand the actual meaning of each neuron. Although some weights are learnt during the training process, they are not usable for understanding the classification process. The interpretation ability is especially required for many tasks, such as medical diagnosing. The K-CR classifier is more suitable than ANNs when interpretation ability is required.

As shown in equation 4.1, all features are initialised with the same weights $w_1, w_2, ..., w_n$

Features	sepal length	sepal width	petal length	petal width
Learnt Weight	0.0526	0.0278	0.2446	0.6750

Table 4.1: Learnt weights for each feature of the Iris dataset using K-CR

at simulation step 0, where $w_1 + w_2 + ... + w_n = 1$. Then, K-CR performs the prototype construction process. After that, the prototypes and the weights are evaluated against a validation set. A classification accuracy is generated to reflect the performance of classification. This accuracy will be recorded as the best accuracy $accu_{best}$, and the weights are set as the best weights $wegihts_{best}$. Then the $wegihts_{best}$ will be simulated for I (preset by user) iterations. For each iteration, we randomly initialised a new set of weights by also following equation 4.1. Each weight is an integer between 1 and the number of features nA. Then the weights will be normalised as w_i/nA. For example, instances from the Iris dataset have 4 features. Each feature is randomly initialised with a weight among $[1, 2, 3, 4]$. After normalisation, the final weights will be $[3/4, 2/4, 1/4, 3/4]$. With the new set of weights $weights_{new}$, a new accuracy $accu_{new}$ is calculated through the validation process. If $accu_{new}$ is higher than $accu_{best}$, the value of $accu_{new}$ is assigned to $accu_{best}$ and $weights_{new}$ are assigned to $weights_{best}$. Otherwise, K-CR proceeds to the next iteration of the optimisation. The $weights_{best}$ obtained after all iterations are considered as the optimised weights, and they will used for testing.

$$\begin{cases} w^n = 1/nA & step = 0 \\ w^n = R_{\{1,2,3...nA\}}/nA & step > 0 \end{cases} \tag{4.1}$$

We also implemented an improved version of the weight optimisation approach. The original version learns one sets of weights for all the features regardless of the differences between classes. We think that one feature may have different "importance" in different classes. Thus, instead of learning one single set of weights for all classes, we designed our K-CR to learn nC sets of weights, where nC stands for the number of classes within the dataset. This design aims to learn weights which can independently reflect the properties of each class. This idea is based on the same intuition as the prototypes construction.

4.1.3 Handling Data Imbalance

By this stage, we have covered the core of handling data imbalance with K-CR. First is the class-based prototypes. As introduced above, K-CR constructs prototypes for each class based on samples from and only from that particular class. The process of construction is independent between different classes. The samples of minority classes will not be ignored or considered as outliers. And the prototypes of minority classes will not be influenced by the samples from majority classes. Secondly, K-CR benefits from simulating a set of m sets of weights. m indicates the number of classes exist in the dataset. Having independent sets of weights help the classifier to understand the difference between classes. Thereafter, the K-CR can perform better classification.

4.2 Score Matrix

Within the learning phase, we constructed prototypes and obtained optimised weights. These prototypes and weights are used to predict which class an instance belongs to. For any instances to be classified, they will be tested against each of the constructed prototypes. The objective is to locate the best prototype that fits the instance. Therefore, an approach is required to evaluate which prototypes fits best. Belacel (1999) named this approach as the score matrix in the K-CR classifier.

The components of the score matrix are determined based on the relationship between the unseen new sample S and the i_{th} prototype P_i^C of the class C to be evaluated. The relationship between them is defined using absolute distance. As shown in algorithm 4, the calculation is conducted at feature level. For the m_{th} feature S_m, it is compared with P_m^C which is the m_{th} interval of the prototype P of class C. The comparison is as follow:

$$\begin{aligned} if\ P_m^C[0] \leq S_m \leq P_m^C[1] \quad then \quad d_m(S_m, P_m^C) = 0 \\ if\ S_m > P_m^C[1] \quad then \quad d_m(S_m, P_m^C) = S_m - P_m^C[1] \\ if\ S_m < P_m^C[0] \quad then \quad d_m(S_m, P_m^C) = P_m^C[0] - S_m \end{aligned} \tag{4.2}$$

where $P_m^C[0]$ refers to the left boundary of the interval P_m^C, and $P_m^C[1]$ refers to the right one. S_m indicates the value of the m_{th} feature of the sample S. This rule can be concatenated as in equation 4.3.

$$d_m(S_m, P_m^C) = max\left\{0,\ S_m - P_m^C[1],\ P_m^C[0] - S_m\right\} \tag{4.3}$$

The equation 4.3 uses the same notation as equation 4.2. Table 4.2 presents an example of the score matrix.

Algorithm 4 Calculating the absolute distance

1: $nA \leftarrow$ Number of attributes
2: $S \leftarrow$ Unseen sample
3: $P \leftarrow$ Prototype to be evaluated
4: **for** $m \leftarrow 1$, nA **do**
5: Calculating the absolute distance according to rules 4.2 as well as equation 4.3 between S_m and interval P_m
6: **end for**

The score matrix is an important component for establishing a preference relational system. With the prototypes, the weights and the score matrix developed above, K-CR has acquired essential components to determine the relationships between prototypes.

4.3 Outranking Relations

With the understanding of the score matrix, next step is to learn how to compare a sample with the prototypes in order to decide which class the sample should be assigned to.

	S_0	S_1	.....	S_m	.....	S_n
P_0^C	$d_m(S_0, P_0^C)$	$d_m(S_1, P_0^C)$	.....	$d_m(S_m, P_0^C)$	.....	$d_m(S_n, P_0^C)$
P_1^C	$d_m(S_0, P_1^C)$	$d_m(S_1, P_1^C)$	.....	$d_m(S_m, P_1^C)$	.....	$d_m(S_n, P_1^C)$
.....	.....	.....	.....	.....	.....	.....
P_m^C	$d_m(S_0, P_m^C)$	$d_m(S_1, P_m^C)$	.....	$d_m(S_m, P_m^C)$	.....	$d_m(S_n, P_m^C)$
.....	.....	.....	.....	.....	.....	.....
P_n^C	$d_m(S_0, P_n^C)$	$d_m(S_1, P_n^C)$	.....	$d_m(S_m, P_n^C)$	.....	$d_m(S_n, P_n^C)$

Table 4.2: An example of score matrix for sample S and 0_{th} to n_{th} prototypes of class C

According to Belacel (1999), the outranking relation defines a relationship between prototype P_i^h and prototype P_j^l. P_i^h indicates the i_{th} available prototype of class h, while P_j^l stands for the j_{th} available prototype of class l. The statement of "prototype P_i^h outranks prototype P_j^l" stands if and only if the calculated absolute distance for each feature between the new sample S and the prototype P_i^h is higher or equal to the ones between the new sample S and the prototype P_j^l. To describe this relation in a sense of "distance", prototype P_i^h outranks prototype P_j^l because sample S is closer to prototype P_i^h than prototype P_j^l.

This outranking method was originally introduced by Roy (2013) as partial outranking indices. Each index has the ability to indicate whether the following assumption is true or false:

- If the distance between sample s and prototype p^1 is larger than the distance between sample s and prototype p^2, prototype p^1 outranks the prototype p^2.

The outranking index O_m^S for the m_{th} feature of the sample S is defined as follows:

$$O_m^S(P_i^h, P_j^l) = \left\{ \begin{array}{ll} 1 & : if \;\; d_{mh}^i(S, P_i^h) \;\leq\; d_{ml}^j(S, P_j^l) \\ 0 & : otherwise \end{array} \right\} \tag{4.4}$$

With equation 4.4, we can obtain the outranking indices of the features. Then, the comprehensive outranking index can be determined by aggregating the outranking indices of all the features. During the aggregation, the features' importance which were determined during the learning phase 4.1 are taken into account as well. The process of aggregation follows equation 4.5.

$$\begin{array}{c} O(P_i^h, P_j^l) = \sum_{m=1}^{n} (w_m^b est * O_m(P_i^h, P_j^l)) \\ h = 1, ..., k; \;\; l = 1, ..., k \\ i = 1, ..., L_h \; and \; t =, ..., L_l \end{array} \tag{4.5}$$

where w_m stands for the relative importance that was assigned to the m_{th} feature of the sample S.

4.4 Score Function

Based on equation 4.5, we can obtain a series of outranking relationships between each pair of two prototypes. K-CR is able to select the best possible prototype based on these relationship knowledge.

First, K-CR will create a outranking graph as shown in figure 4.3. It is an oriented graph Belacel (1999), in which the nodes represent the different prototypes Ps, where $Ps = (P_i^{C_m}$. The edges of the graph represent the outranking relation value between each pair of two prototypes.

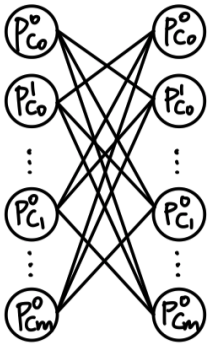


Figure 4.3: Outranking graph

Along with the graph, a score function is used. The function aims to provide scores as ranking criteria. Therefore, we created a ranked list for all the constructed prototypes. Based on this list, we can choose the most qualified prototype for the sample S. As introduced by Belacel and Boulassel (2004), there are several available choices of scoring functions:

- The leaving flow proposed by Brans and Vincke (1985):

$$s_1(b_i^h, P_s, O) = \sum_{P \epsilon P_s} O(b_i^h, P) \tag{4.6}$$

- The entering flow proposed by Brans and Vincke (1985):

$$s_2(b_i^h, P_s, O) = \sum_{P \epsilon P_s} (1 - O(P, b_i^h)) \tag{4.7}$$

- The net flow proposed by Brans and Vincke (1985):

$$s_3(b_i^h, P_s, O) = (1/n) \sum_{P \epsilon P_s} (O(b_i^h, P) - O(P, b_i^h)) \tag{4.8}$$

- The min leaving flow proposed by Bouyssou (1995):

$$s_4(b_i^h, P_s, O) = min_{P \epsilon P_s} O(b_i^h, P) \tag{4.9}$$

- The max entering flow proposed by Bouyssou (1995)Orlovsky (1978):

$$s_5(b_i^h, P_s, O) = 1 - max_{P \epsilon P_s} O(P, b_i^h) \tag{4.10}$$

- The Orlovski score proposed by Orlovsky (1978):

$$s_6(b_i^h, P_s, O) = min_{P \epsilon P_s} min(1 - O(P, b_i^h) + O(b_i^h, P); 1) \tag{4.11}$$

- The Min difference score proposed by Barrett et al. (1990):

$$s_7(b_i^h, P_s, O) = min_{P \epsilon P_s}(O(b_i^h, P) - O(P, b_i^h)) \tag{4.12}$$

Out of these score functions, we chose the one corresponding to equation 4.8. Brans and Vincke (1985) proposed it and Brans et al. (2005) further developped it as the PROMETHEE II Complete Ranking method. We chose it for its compatibility with the K-CR classifier. The PROMETHEE method requires two kinds of information in order to properly operate:

- Information between the criteria: Brans et al. (2005) explained it as the relative importance or weights for each criteria. In addition, the weights need to be non-negative and independent. In the K-CR classifier, one of the vital steps is learning the independent class-based weights for each feature. As introduced in section 4.1, the weights have values between 0 and 1. Therefore, they are non-negative. On one hand, K-CR is able to provide the required knowledge to the score function. On the other hand, the score function can utilise the most important learnt knowledge by K-CR.

- Information within each criterion: Brans et al. (2005) introduced this requirement as they encourage users to perform pairwise comparison with the score function. As mentioned in section 4.3 and at the start of this section, K-CR compares the prototypes in pairs. Therefore, K-CR meets the second requirement as well.

In summary, K-CR fits the requirements of the PROMETHEE method, and the comparison is made by utilising the important knowledge learnt by K-CR. It is suitable to adopt this score function. In the next section, we will explain the application of the adopted score function.

4.5 Prototype Choice

The main component of the PROMETHEE score function is the calculation of the outranking flows. Brans et al. (2005) proposed two directional flows: the positive outranking flow and the negative outranking flow. These flows are similar to water flowing directions. The positive flow stands for the flow from node a to all the other nodes, and the negative flow refers to the flow from others to node a.

Therefore, for n number of constructed prototypes, there are $(n-1)$ pairs of comparison for each prototype P. Figure 4.4 presents the positive flow and the negative flow.

- Positive outranking flow or leaving flow: it reveals how much the prototype P_i^h outranks other prototypes. It is calculated based on the outranking relations obtained in section 4.3. As shown in equation 4.13, the leaving flow of prototype P_i^h is the aggregation of the scores between P_i^h and other prototype:

$$\phi^+(P_i^h) = \sum_{P \in Ps} (S(P_i^h, P)) \tag{4.13}$$

  where higher the $\phi^+(P_i^h)$ is, better the prototype P_i^h is at outranking the others.

- Negative outranking flow or entering flow: it reveals how much other prototypes outrank the prototype P_i^h. Similarly to the leaving flow, the entering flow is calculated with the obtained scores. As shown in equation 4.13, the entering flow of prototype P_i^h is the aggregation of the scores between other prototypes and P_i^h. It needs to be remarked that $S(P_i^h, P)$ is not same as $S(P, P_i^h)$. The order of input prototypes affects the results. Therefore, "the scores between P_i^h and other prototypes" is different from "the scores between other prototypes and P_i^h". The entering flow is as follows:

$$\phi^-(P_i^h) = \sum_{P \in Ps} (S(P, P_i^h)) \tag{4.14}$$

  where the lower the $\phi^-(P_i^h)$ is, better the prototype P_i^h is at outranking the others.

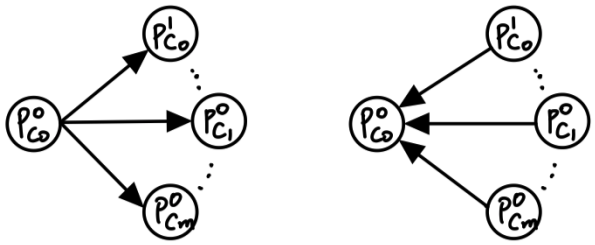


Figure 4.4: Positive flow on the left hand side and negative flow on the right hand side

Recall the score function 4.8 we chose. It is the net flow obtained by combining leaving flow and entering flow. According to Brans et al. (2005), the net flow is the balance between the positive flow and the negative flow. The higher the net flow is, the better the node a is. In the K-CR classifier, the values of the net flow are considered as the scores of the prototypes. The prototype with higher score fits better the sample S to be classified. A refined net flow equation is presented:

$$\phi(P_i^h) = \phi^+(P_i^h) - \phi^-(P_i^h) \tag{4.15}$$

Based on the obtained scores, it can be concluded that:

$$\forall P_i^h \in P_s \;\; and \;\; \forall P_l^t \in P_s : \;\; P_i^h \; is \; better \; than \; P_l^t \;\; iff \;\; \phi(P_i^h) > \phi(P_l^t) \tag{4.16}$$

Based on 4.16, we can form a ranked list of prototypes, as we mentioned earlier. With the list, it is possible to select the prototype with the highest score which fits the sample S best. As each prototype leads to one and only one of the classes, the sample S can be classified into the same class as the selected prototype.

4.6 K-CR Applications

In the literature, the K-CR classifier has been applied in several fields, such as medical application and recommender systems. In both applications, K-CR achieved good results.

4.6.1 Medical Application

Belacel and Boulassel (2004) applied the K-CR classifier to Astrocytic Tumours (ATs) diagnosis.

The method was tested on an experimental set of 250 cases of ATs. It is interesting that the dataset contained 3 classes, with 39 instances for class AST, 47 instances for class ANA, and 164 instances for class GBM. Therefore, it is an imbalanced data set. In addition, each instance has 26 features.

According to Belacel and Boulassel (2004), K-CR was compared to other classifiers, including Decision Tree, KNN, Logistic Regression and Multilayer Perceptron using 10-fold cross validation. K-CR achieved the best results with 66% accuracy. In addition, K-CR showed a better ability in terms of providing detailed information about the decision making. This ability is valuable to decision makers.

4.6.2 Recommender System

In 2020, Belacel et al. (2020b) applied the K-CR classifier to build an Amazon Products Recommender System.

K-CR was employed as a content-based filter to perform recommendation functionalities. In this application, two datasets were chosen for testing: Amazon Fine Food and Book Review. The Amazon Fine Food dataset includes 458,454 reviews, and the Book Review dataset has more than two times that number.

K-CR is compared with KNN. The inputs for both approaches were pre-processed in the same manner. For the Amazon Fine Food dataset, the results showed that about 71% of all recommendations were made better with K-CR. The number was 51% for the Book Review dataset.

4.6.3 Impact to This Thesis

Based on previous successful applications, several factors indirectly point to the possibilities for the applications and research conducted in this thesis.

First of all, the medical application reveals K-CR's capability of handling imbalanced datasets. It shows that it is worth employing K-CR in modern machine learning classification tasks. In addition, in the Amazon Recommender system, text data was used. It was proven that the K-CR classifier has the ability to handle text data. Therefore, it is valuable to explore the possibilities of employing K-CR on NLP classification tasks.

Overall, the successful applications has pushed the continued interests of applying K-CR on more imbalanced data sets in the thesis, such as the one from the first experiment in the next chapter. Furthermore, depression detection using social media data also attracted researchers' attention in recent years. First, it can be seen as a medical problem to some extent. Second, using social media data means that text data will be encountered. It is valuable to learn how K-CR performs when solving medical problems with imbalanced text-based datasets.

4.7 Contribution to the K-CR Methodology

The main contribution we made to the algorithm itself is combing K-CR and the TF-CBTW technique that we introduced in chapter 3.4. We expect to further improve the performance of K-CR over imbalanced datasets by highlighting the 'important' terms within each class. Thereafter, K-CR is able to construct better prototypes.

The diagram showed below is an example of utilising TF-CBTW with K-CR. First, as shown in figure 4.5, we applied TF-CBTW on original documents. As a result, a list of term scores are obtained. Then, we focus on each individual sentence. It starts by performing word embedding. Each sentence is converted into a word embedding matrix. In the next step, instead of directly converting the matrix into a sentence vector, we multiply each word embedding vector with its corresponding term score. At the end, the newly created word embedding matrix is converted into a sentence vector.

Figure 4.5: Processing the docs with TF-CBTW

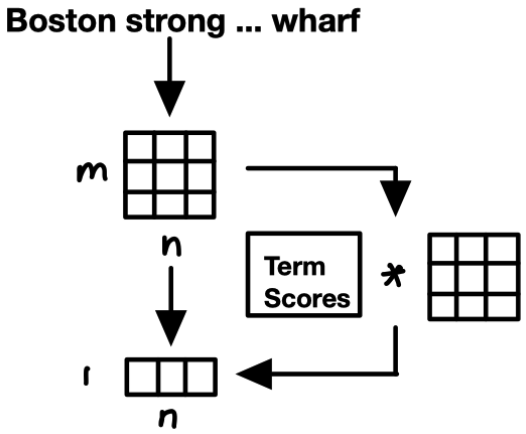


Figure 4.6: Adopting obtained term scores into word embeddings

4.8 Summary

In this chapter, we introduced the K-Closest Resemblance Classifier in detail. By reviewing the applications which adopted the K-CR classifier, we learnt about the possibilities to employ K-CR in different fields with imbalanced datasets. In the next chapter, we will introduce our experimental design and our results, and we will discuss them.

Chapter 5

Experiments and Results

For this thesis, we chose three datasets: GEOBIA, Bell Let's Talk, and the CLPSych2015 shared task dataset. Among them, the GEOBIA dataset is chosen for learning the Land Cover Classification tasks. The other two are chosen for learning the depression detection task. Since all these datasets have different data structure (such as different number of features and classes), we conducted three independent sets of experiments. Each of them will be introduced separately.

In this chapter, we will first introduce the evaluation approach adopted. Then, we will explain our datasets, experimental setup, results and discussions.

5.1 Evaluation Methods

In this thesis, we adopted the confusion matrix and the AUC-ROC analysis technique as evaluation approaches.

Confusion Matrix

The confusion matrix is a widely adopted evaluation matrix. It is a table which describes the performance of classifiers. As shown in figure 5.1, there are four important terms associate with the confusion matrix:

- TP (True Positive): it indicates the number of cases which the predicted positive class is indeed truly positive.
- TN (True Negative): it indicates the number of cases which the predicted negative class is indeed truly negative.
- FP (False Positive): FP represents the cases which the ground truth is negative while the predictions are positive.

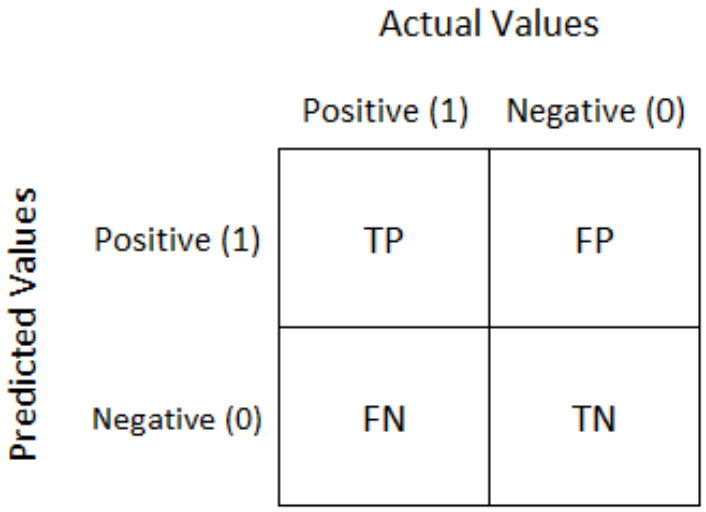


Figure 5.1: Standard format of a confusion matrix table

- FN (False Negative): Similarly, FN stands for those with positive ground truth while the predictions are negative.

The confusion matrix is designed for binary classification tasks. However, it has also been largely adopted in multi-class classification tasks with one-vs-all strategy.

The confusion matrix itself does not directly provide performance information of the classifiers. The statistics calculated based on the matrix will be used for evaluation. There are three popular statistics being used for evaluation:

- Precision: it is a rate which reflects how many predicted positive instances are truly positive.

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

- Recall: it is a rate which shows the classifier' ability of identifying positive instances.

$$recall = \frac{TP}{TP + FN} \tag{5.2}$$

- F-measure: it is another rate to show a combinations between precision and recall. F-measure is generally more used than Precision or Recall since the precision and recall can be easily affected by large True Negative rates.

$$F_1 - score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5.3}$$

In this thesis, we adopted the F1_measure for evaluation. More precisely, we adopted the weighted average F1_measure considering the data imbalance. With weighted values, the class distribution is taken into consideration. Then, we can obtain 'fair' performance

scores. We abandoned accuracy scores[1] because they may easily being affected by majority classes. There are cases in which the accuracy is high, while the predictions are correct only for the majority classes and incorrect for the minorities ones. It cannot effectively reflect how well the classifiers are performing across all classes in case of data imbalance. The module 'confusion_matrix' from the library 'scikit learn' is used to generate the weighted F1_scores.

ROC-AOC Analysis

We chose ROC-AOC Analysis technique as the second approach considering the data imbalance within the chosen datasets. AUC stands for the Area Under the Curve, while ROC is the Receiver Operating Characteristics. They work together to evaluate the performance of the classifiers in a sense of measuring how well they can distinguish between classes.

AUC is a statistic between 0 and 1. If AUC is 0, it means that the classifier has no ability to distinguish between classes. In contrast, 1 indicates that the classifier has perfect ability to distinguish between classes.

As to ROC, there are two important terms for plotting the ROC curve: True Positive Rate (TPR) and False Positive Rate (FPR). As shown in figure 5.2:

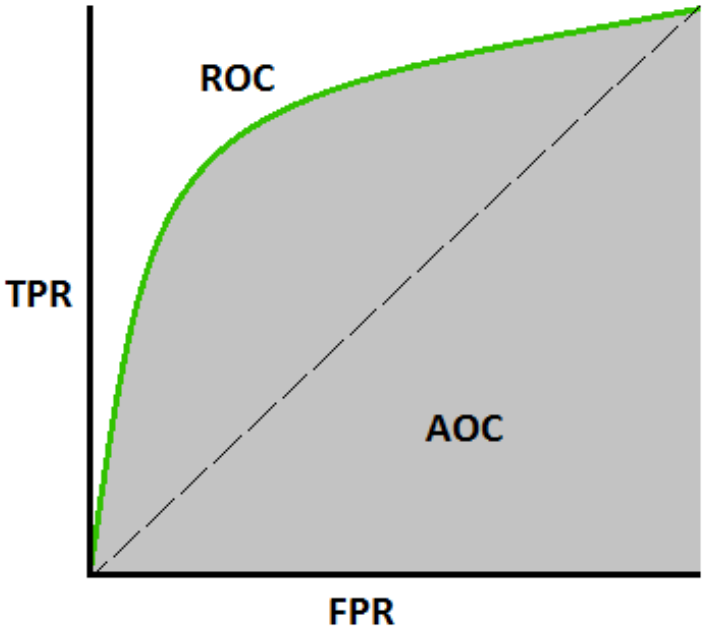


Figure 5.2: Example of an AUC-ROC curve

- TPR actually indicates the same information as recall. It uses the same equation as 5.2. In a ROC plot, TPR are placed along the Y-axis.

[1]Accuracy is defined as the number of correct predictions for all classes divided by the total number of examples to be classified.

- FPR follows the equation and takes the position on the X-axis.

$$FPR = \frac{FP}{TN + FP} \tag{5.4}$$

The ROC curve provides a visual representation of classifier' distinguishing ability. Taking figure 5.2 as an example, with a perfect classifier, the green curve would go straight up along Y-axis and straight right along the direction of X-axis. The higher the green curve is above the diagonal dotted line, the better the classifier is. Optionally, we can also judge the ability by comparing the proportion of the grey area. Higher proportion indicates better distinguishing ability. We adopted the library 'seaborn' to plot ROC curves.

5.2 GEOBIA Dataset

We evaluated our approach on the GEOBIA dataset first. Parts of the experiments were published in Belacel et al. (2020a). Johnson and Xie (2013) and Johnson (2013) created the dataset in 2013. As explained by Maxwell et al. (2018), it is composed of a set of high resolution urban land cover images which were obtained using geographic object-based image analysis (GEOBIA). Figure 5.3 presents an example of land covers.

Figure 5.3: An example of obtained urban land cover image (Johnson and Xie, 2013) (Johnson, 2013)

We obtained the dataset from the University of California, Irvine Machine Learning Repository created by Dua and Graff (2017). It needs to be remarked that only extracted

features are available in the dataset. In other words, the dataset is not composed of images. Instead, each instance in the dataset is a vector which contains extracted feature values. Therefore, we cannot apply CNNs on this dataset since CNNs takes matrix representation of images as input. However, for the purpose of comparison, we adopted the 'Conv1D' to do a 1-dimensional convolutional classification. The details will be discussed in section 5.2.2.

The dataset contains 675 instances in total. They are divided into training and test sets which contains 168 and 507 instances respectively. The instances are classified into nine types of urban land covers including Asphalt, Building, Car, Concrete, Grass, Pool, Shadow, Soil, and Tree. These land cover types are represented with numbers from 0 to 8 as shown in table 5.1. In addition, as mentioned by Zhiyuan et al., the dataset has 147 features which represent information of spectral properties, size, shape, and texture.

One of the reasons for choosing this dataset is the data imbalance. The table 5.1 summarised the data distribution in detail. If we look at the 'Total' column of the table, there are big differences in terms of the number of instances. For example, there are 116 instances for the class 'Concrete', while only 34 instances are available for the class 'Soil'. It is valuable to learn how the trained classifiers can generalise on the test set.

Class	Training	Test	Total
Asphalt / 0	14	45	59
Building / 1	25	97	122
Car / 2	15	21	36
Concrete / 3	23	93	116
Grass / 4	29	83	112
Pool / 5	15	14	29
Shadow / 6	16	45	61
Soil / 7	14	20	34
Tree / 8	17	89	106
Total	168	507	675

Table 5.1: Class distribution of the GEOBIA dataset

5.2.1 Experiments Setup

We designed two sets of experiments to compare the performance of the K-CR, KNN, RF, SVM, MLP and CNN:

- **K-fold cross validation on the training set of GEOBIA.** In this series of experiments, all participated classifiers used only the training set of the GEOBIA dataset which has 168 instances. The experiments are conducted based on K-fold cross validation, while in this case K was set to 10. With 10-fold cross validation,

the set are randomly portioned into 10 equal sized folds. 9 of the folds are used for training the classifiers, and the remaining 1 fold is used for testing the trained classifiers. The process will be conducted for 10 iterations, and each iteration will choose different combination of '9 folds'. We will take the average of results from 10 iterations for evaluation. For this set experiment, we used the module 'KFold' from library 'scikit learn' to perform k-fold cross validation.

- **Training with the training set and testing on the test set.** All classifiers are trained with the training set (168 instances). Then their performance are tested on the test set (507 instances).

5.2.2 Results and Discussion

10-Fold Cross Validation on the Training set

For all classifiers, we tested them against the dataset with different settings. The table 5.2 shows the best results of applying different classifiers on the training set of GEOBIA based on 10-fold cross validation. The first column lists all available classes within dataset same as table 5.1. Each row of the table except the last one documents the F1-score of each class from different classifiers, while the last row lists the weighted average F1-scores. These best results in the table were obtained when K-CR sets the number of intervals to 2, and 0.1 for beta. RF has the settings with 500 trees, SVM used the linear kernel, and KNN has the K equal to 3. The MLP contains 2 layers, and it is trained using the Adam optimizer with a learning rate of 0.0001 for 200 epochs.

	KCR	RF	SVM	KNN	MLP
Asphalt	0.89	0.84	0.61	0.22	0.52
Building	0.91	0.78	0.62	0.57	0.60
Car	0.90	0.87	0.87	0.87	0.61
Concrete	0.90	0.75	0.47	0.38	0.23
Grass	0.94	0.87	0.58	0.44	0.33
Pool	0.93	0.86	0.73	0.77	0.48
Shadow	0.88	0.84	0.64	0.32	0.29
Soil	0.88	0.67	0.40	0.09	0.21
Tree	0.94	0.87	0.57	0.41	0.38
Weighted Average F1-Measure	0.91	0.82	0.60	0.46	0.40

Table 5.2: Weighted Average F1-measure on the training set of GEOBIA

By looking at the last row of the table 5.2, K-CR has the weighted average score of 0.91 which is higher than that of all the other classifiers. It is 9% higher than RF which has the second highest weighted average F1-score. However, due to the fact that the GEOBIA is an imbalanced dataset, it is necessary to explore the performance for each individual class, especially for those with a low number of available samples. The K-CR classifier

outperforms the other classifiers on the minority classes. For class 'Soil', K-CR achieved 0.88 F1-score which is with 21% higher than RF's F1-score of 0.67. The class 'Car' which has the closest performance among the classifiers, K-CR still outperforms the second best with about 3%. We further focused on the AUC-ROC analysis of K-CR and RF (which has the second best average performance). Table 5.3 presents the AUC score of K-CR and RF. K-CR has higher average AUC than RF which means that K-CR has better distinguishing ability among classes than RF. If we focus on each individual class, K-CR outperforms the RF on all classes except the 'Asphalt' class. K-CR is 0.2% lower than RF at 0.93. However, 0.93 is not a low AUC score comparing with other class-based AUC scores of K-CR. This means that K-CR did a generally appropriate classification on 'Asphalt' class. While in RF case, the 0.95 is the highest AUC score among different classes. In other words, the best classification the RF can do is only at average level of what K-CR is capable of. Therefore, we concluded that K-CR outperforms RF on average and on minority classes. We further explored the ROC curves as shown in figures 5.4 and 5.5. The curves of K-CR are nearer to top-left corner than RF. In addition, the curves of K-CR are closely with each other even with data imbalance. Furthermore, we plotted the confusion matrices of K-CR and RF as shown in figures 5.6 and 5.7. It can be noticed from those figures that more instances falls on diagonal piles in K-CR than RF.

	K-CR	RF
Asphalt	0.93	0.95
Building	0.92	0.86
Car	0.96	0.93
Concrete	0.98	0.87
Grass	0.99	0.93
Pool	0.96	0.9
Shadow	0.93	0.9
Soil	0.89	0.81
Tree	0.94	0.91
Average AUC	0.91	0.895

Table 5.3: AUC score of K-CR and RF

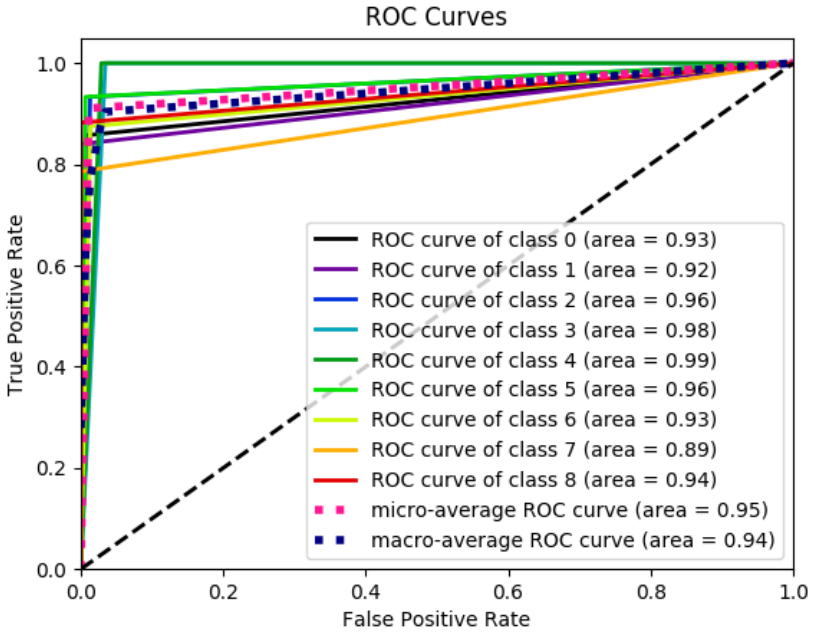


Figure 5.4: ROC curve of K-CR classifier on training set of GEOBIA

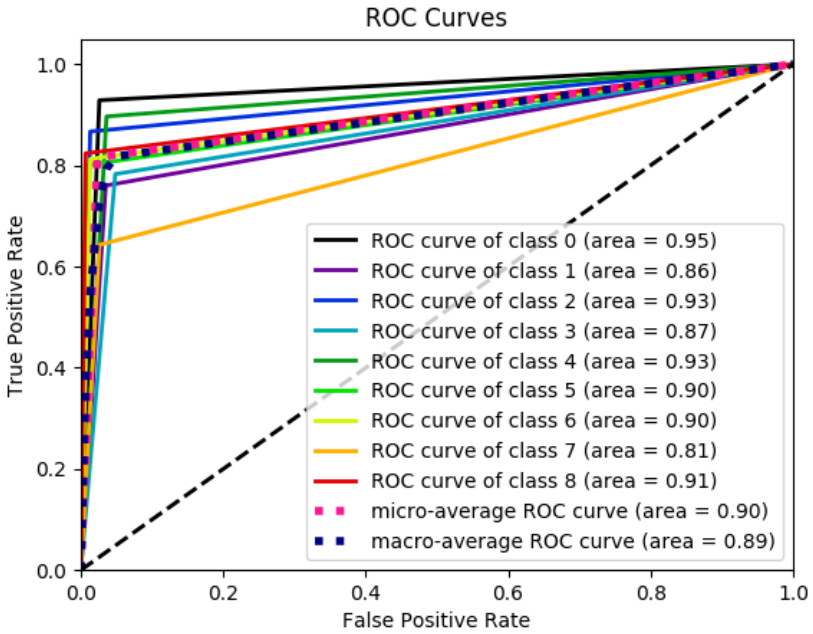


Figure 5.5: ROC curve of the RF classifier on the training set of GEOBIA

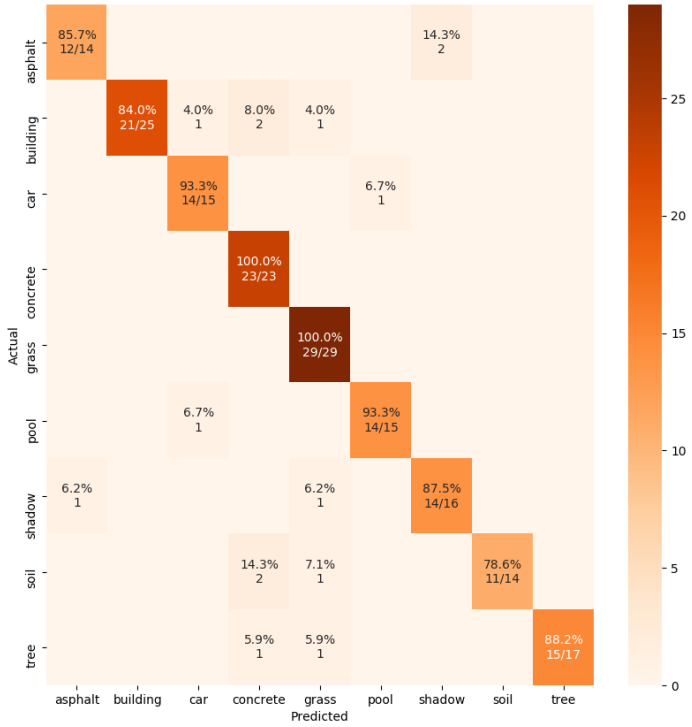


Figure 5.6: Confusion matrix of the K-CR classifier on the training set of GEOBIA

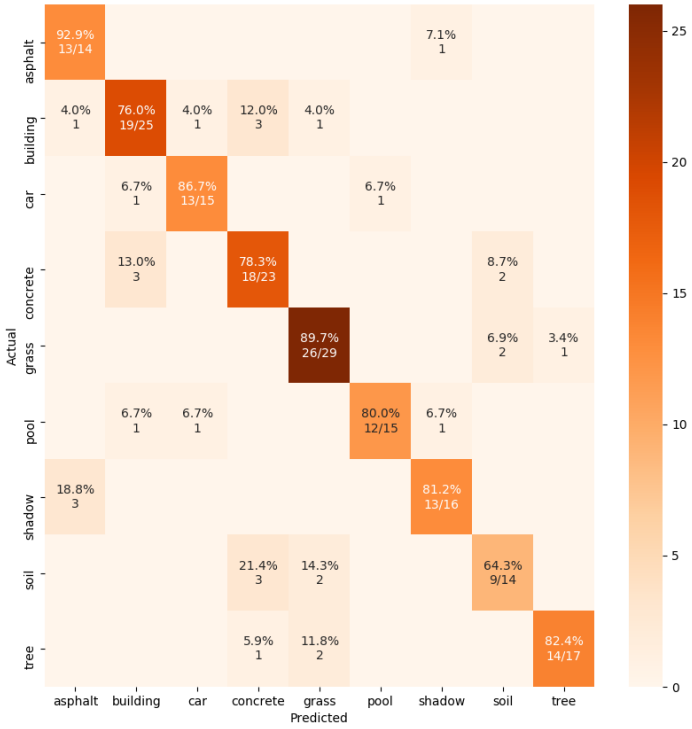


Figure 5.7: Confusion matrix of the RF classifier on the training set of GEOBIA

We further explored those misclassified instances with K-CR. In figure 5.6, there are 15

instances that were misclassified during cross validation. 9 of them are predicted to either the 'Concrete' or 'Glass' classes. Recall the class distribution in table 5.1, 'Concrete' and 'Glass' are two majority classes in the dataset. This reveals that although the K-CR learns the data distribution in imbalanced datasets and achieved much better performance, the data imbalance still affects the classifier to some extent. It is necessary to learn how K-CR can be further improved.

Train on Training Set, Test on the Testing Set

Table 5.4 represents the results of the second sets of experiments. Since there are only 168 instances used for training and 507 instances in test set, we expect a performance drop between the results in tables 5.2 and 5.4. Indeed, the results show that all classifiers performed worse in the second set of experiments. However, K-CR still has the best weighted average F1-score of 0.73. Consistent with the first set of experiments, RF also has the second best performance which is with 1% lower than K-CR. Recall the data distribution presented in table 5.1, 'Asphalt', 'Car', 'Pool' and 'Soil' are 4 minority classes in the dataset. RF outperformed K-CR on the classes 'Pool' and 'Soil' by 1%, respectively, but K-CR outperformed RF on the classes 'Asphalt' and 'Car' by 3%, respectively. Overall, K-CR still performs better than all other tested classifiers on this imbalanced dataset.

	K-CR	RF	SVM	KNN	MLP
Asphalt	0.80	0.77	0.65	0.25	0.50
Building	0.69	0.74	0.71	0.53	0.56
Car	0.72	0.69	0.68	0.52	0.19
Concrete	0.77	0.74	0.64	0.48	0.40
Grass	0.71	0.71	0.60	0.35	0.22
Pool	0.70	0.71	0.69	0.48	0.45
Shadow	0.68	0.71	0.49	0.20	0.44
Soil	0.51	0.52	0.23	0.13	0.17
Tree	0.78	0.73	0.50	0.42	0.31
Weighted Average F1-Measure	0.73	0.72	0.59	0.40	0.38

Table 5.4: Weighted Average F1-Measure on both training and testing sets of GEOBIA

We noticed that there is a performance dropping between the two sets of the experiments. One reason, as we have mentioned, is the lower number of training instances in the training set than is the test set (it is not clear why the train-test split is done this way). Since K-CR is a prototype-based classifier, it constructs prototypes based on information learnt solely from the training set. We suspected that the K-CR classifier could not acquire enough amount knowledge to construct prototypes which can effectively represent the data distribution. Therefore, the performance drops during generalisation. One directions of the future work is swapping the training set and test set, to find out how the classifier can generalise an a larger training set. Furthermore, this is the first time the K-CR classifier is being applied in remote sensing field. It is necessary to test its performance on other datasets in this field.

GEOBIA with CNN

Apart from the two sets of experiments introduced above, we ran an additional experiment with CNNs. As we mentioned, it is not fair to apply CNNs this dataset since only features already extracted from images are available. The original images are not included in the dataset. CNNs usually require a matrix input so that mechanisms such as convolutions filers can be applied on regions of the images.

However, we learnt that there are special versions of CNNs which target vector inputs. We wanted to try CNNs because they are known to obtain good performance on classification tasks. We implemented the special CNN with the 'Conv1D' and 'MaxPooling1D' layers adopted from TensorFlow. Then, we trained the network with the training set of the GEOBIA. We used the categorical cross entropy loss as the learning goal and employed different optimisers including Adam, SGD and RMSprop. The batch size was set to 8 as there is a small total number of instances in the dataset. The trained CNNs are tested against the test set. For each different network setup, the entire process is repeated 10 times. The average accuracy of 10 runs is taken as the final result. Among all the experiments, the best average accuracy obtained was 0.499 while in the last set of experiments, K-CR had an accuracy of 0.74 which is about 24% higher than the 1D-CNN.

5.2.3 Comparison to Related Work

It is necessary to note that Maxwell et al. (2018) also performed experiments over the same dataset. They systematically explored the performance of different classifiers including SVM, Decision Trees, RF, Boosted Decision Tree, Artificial Neural Networks and KNN on this dataset. In addition, the authors conducted the experiments using the same 10-fold cross validation setup as in this thesis. Therefore, besides comparing the results of our classifiers, we also compare the results with the ones from Maxwell et al. (2018).

The authors conducted experiments on the same dataset with or without feature selection or data re-sampling techniques. When there is no pre-processing on the dataset, their experiments showed that RF has the best performance with accuracy of 81.5%. After applying different data re-sampling techniques, the authors obtained their result with RF and over-sampling. The performance increased from 81.5% to 81.7%. As they did not mention other statistics than the accuracy, we created the table 5.5 to compare the performance of K-CR and RFs. Within the table, the second column shows the accuracy of RF conducted by us. The last two columns present the accuracy of RF obtained by Maxwell et al. (2018). First of all, both the column 2 and 3 present the accuracy obtained with same experiments setup (10-fold cross validation, no feature selection or data re-sampling). This meets the expectation that they have the same accuracy (0.815). It is a evidence that we correctly designed our experiments and utilised the dataset. It is reasonable to perform further comparison. Therefore, if we compare column 1 and 3, K-CR has almost 10% higher accuracy. It also has 9.3% higher accuracy compared to the accuracy in column 4. This is a big improvement since K-CR performs better with the original dataset than RF with the pre-processed dataset.

	K-CR	RF without preprocessing	RF-Maxwell without preprocessing	RF-Maxwell with over-sampling
Accuracy	0.91	0.8147	0.815	0.817

Table 5.5: Classification accuracies from (Maxwell et al., 2018)

5.3 Bell Let's Talk

We selected the Bell Lets Talk dataset as one of two datasets that we will use for for evaluating K-CR's classification ability on text data. As we mentioned before, we focused on the depression detection task with social media texts.

The Bell Lets Talk dataset was created by Jamil et al. (2017) from the NLP Lab of the University of Ottawa. The dataset is composed of collected real tweets from users who posted in the #BellLetsTalk 2015 campaign. This awareness campaign was created by Bell Canada to target topics about mental health issues. The tweets were first processed by applying topic modelling techniques such as LDA. Based on the obtained topics, labels were assigned accordingly. The obtained labels were then verified by human annotators.

Jamil et al. (2017) created two datasets based on the collected tweets:

- **60Users** which focuses on classifying tweets' labels.
- **160Users** which focuses on user-level classification.

In this thesis, we selected the **160Users** set to perform user-level classification. All instances are classified into two classes: 'depressed' and 'not depressed'. In addition, the dataset is split into 2 portions. There are 113 users (70%) for training and 47 users (30%) for testing as shown in table 5.6. In the training set, there are 64 users who belong to the 'not depressed' class and 49 users for the 'depressed' class. This is a slightly imbalanced dataset for the fact that about 60% users of this dataset belongs to one of the two classes.

	Training	Test	Total
Not Depressed Users / 0	64	27	91
Depressed Users / 1	49	20	69
Total Number of Users	113	47	160

Table 5.6: Training and Test split of 160Users dataset

5.3.1 Pre-processing

It is necessary to apply proper pre-processing of the tweets before any further operations. In this thesis, we followed the same pre-processing operations applied in Orabi et al. (2018). Information such as retweets, URL, mentions and non-alphanumeric characters were removed. In addition, we created a customized stopwords list to remove all the stopwords

except first, second, third person pronouns from all the tweets, in order to maintain possible mental condition information in the tweets by keeping the pronouns, as recommended by Pennebaker (2011).

Below is an example, a tweet from a user 'f8Da_Rvi'. The first sentence is the original tweet from the dataset, and the second sentence is one after being pre-processed:

- Boston Strong 6 mile run this morning. #bostonstrong @ The Landing at Long Wharf http://t.co/9Q0ocQYfLE'
- boston strong 6 mile run morning landing long wharf

Apart from the operations introduced above, another step was applied. Due to the fact that we focus on mental condition classification at user-level, K-CR and other classifiers expect user-level inputs. In other words, we should take all tweets of each user as one instance rather than taking each tweet as one instance. Therefore, for each user, we combined all his/her tweets into one single long sentence. For example, after the combination, the example sentence mentioned above for user 'f8Da_Rvi' is combined into the long sentence below:

- running boston childrens hospital make donation warrior dash virgins daddy fun jacks gift auen estate emma finally got her ears pierced proud my brave little girl another crazy saturday daddy jack auen estate happy 4th birthday jack love you love you love you forever ever ever cant believe 2nd grade auen estate love narragansett ri heading block island day vacation ipad still must narragansett ri totally diva mamas n divas great opportunity my little emma work dee week elite academy dance peaceful sleeper goin ride auen estate jacks new scooter thank you daddy auen estate new apparatus our home day auen estate **boston strong 6 mile run morning landing long wharf** we 21 kid free weekend doyles ...

As a result, for the 160 users in the dataset, we have 160 long sentences.

5.3.2 Experimental Setup

We designed the experiments considering several factors, including the choice of word embeddings, different BERT settings, and TF-CBTW included or not. Table 5.7 presents 13 sets of designed experiments.

Exp 1	Word2Vec-CBOW
Exp 2	Word2Vec-skipgram
Exp 3	FastText
Exp 4	GloVe
Exp 5	Word2Vec-CBOW + TF-CBTW
Exp 6	Word2Vec-skipgram + TF-CBTW
Exp 7	FastText + TF-CBTW
Exp 8	GloVe + TF-CBTW
Exp 9	BERT + 512 size windows + sentence based embedding
Exp 10	BERT + 512 size windows + sentence based embedding + TF-CBTW
Exp 11	BERT + 512 size windows + word based embedding
Exp 12	BERT + 512 size windows + word based embedding + TF-CBTW
Exp 13	BERT + dynamic size windows + sentence based embedding

Table 5.7: Experimental setup

In Experiments 1 and 2, we employed Word2Vec with two different architectures. Experiments 3 and 4 employed FastText and GloVe for word embeddings, respectively. Experiments 5 to 8 are same to the first four experiments, but add a TF-CBTW feature selection on top of the embeddings. Experiments 9 to 13 mainly focus on the language model BERT with different window settings and embedding settings.

In experiments 1, 2, 3, 5, 6 and 7, we trained our own Word2vec and FastText word embeddings models. For example, a Word2Vec-CBOW model is trained using the training set. Then, the trained model is used to create embeddings for the sentences from both the training and the test set. We adopted the Word2Vec and FastText packages from Gensim for embeddings model training. In experiments 4 and 8, we imported the pre-trained 'glove-twitter-200' package. As to experiments from 9 to 13, we adopted the pre-trained BERT model 'BERT-base-uncased'. The BERT model is imported from library 'pytorch-pretrained-bert'. As we mentioned in section 3.3.4, BERT does not produce word embeddings directly. Thus, we implemented necessary post-processing procedures with modules from the library 'huggingface'.

Finally, figure 5.8 presents a comprehensive experimental procedure for the Bell Lets Talk dataset. Both the training and the test dataset are first pre-processed by removing punctuation, stopwords, etc. When using word2vec and FastText, the training set is used to train the word embedding models. Otherwise, pre-trained GloVe and BERT are imported. Then, sentence embeddings are created for instances from both the training and the test set. Lastly, we conduct experiments and obtain results for different classifiers.

5.3.3 Results and Discussion

Effect of Different Word Embedding Techniques

Table 5.8 presents experimental results from experiments 1, 2, 3 and 4. We included four statistics: overall accuracy, weighted average F1-score, minority class precision, and minority class F1-score. In this case, the minority class refers to the 'depressed' class.

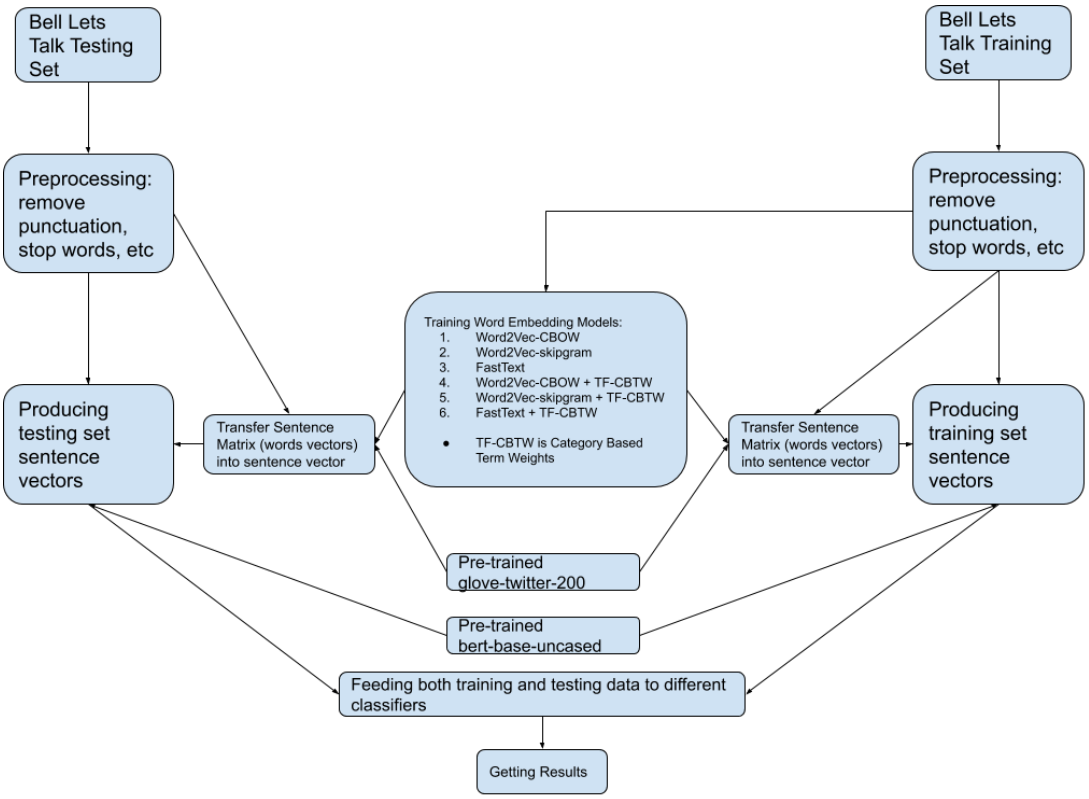


Figure 5.8: Process of experiments with the Bell Lets Talk dataset

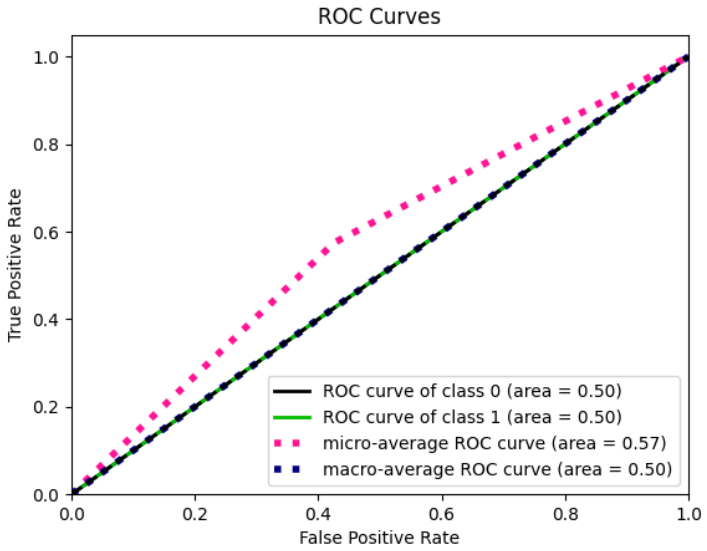


Figure 5.9: ROC curve of the SVM classifier on Exp 1

From the results, we first notice that SVM has 0 minority class precision and F1-scores on all 4 experiments. At the same time, SVM has 0.57 overall accuracy and 0.42 weighted

average F1-score on the 4 experiments. These are the best results obtained for SVM after tuning different hyper parameters, including regularization parameter C, kernel, coefficient and tolerance. The results reveal that SVM is completely biased towards the majority class. Taking experiment 1 as an example, the figure 5.9 presents the obtained ROC curve. The black solid line represent the ROC curve of class 0, and the green solid line for class 1. Both of them superimpose with the diagonal line. It means that SVM does not have the ability to distinguish between these two classes at all.

In addition, as one of the objective is to explore classifiers' performance on minority classes, we created the figure 5.10. It presents the minority class F1-scores of all tested classifiers. They are the scores shown in table 5.8. In the figure, each classifier has one set of 4 bars. For each set of bars, they present experiments 1 to 4 from left to right. From the figure, it is clear that K-CR has higher minority class F1-scores than other classifiers regardless of which word embedding technique is used. Apart from that, we noticed that K-CR is less sensitive to word embedding differences than other classifiers. With any of four tested word embedding techniques, K-CR has consistent minority class F1-scores, while the scores vary a lot for the other classifiers.

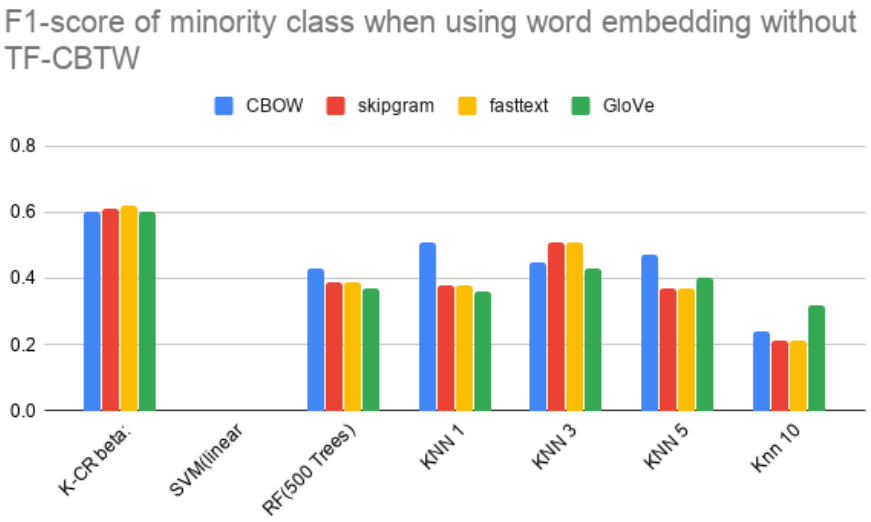


Figure 5.10: F1-score of minority class when using word embedding without TF-CBTW

Exp 1: Word2Vec-CBOW

	Overall Accuracy	Overall Weighted Average F1	Minority Precision	Minority F1
K-CR beta:0.15, interval:6, iterations: 10	0.43	0.25	0.43	0.6
SVM (linear kernel)	0.5745	0.42	0	0
RF (500 Trees)	0.5532	0.55	0.47	0.43
KNN 1	0.5957	0.59	0.53	0.51
KNN 3	0.6383	0.61	0.64	0.45
KNN 5	0.6596	0.63	0.7	0.47
KNN 10	0.5957	0.52	0.6	0.24

Exp 2: Word2Vec-Skipgram

	Overall Accuracy	Overall Weighted Average F1	Minority Precision	Minority F1
K-CR beta:0.15, interval:6, iterations: 10	0.5331	0.5	0.47	0.61
SVM (linear kernel)	0.5745	0.42	0	0
RF (500 Trees)	0.5319	0.52	0.44	0.39
KNN 1	0.5106	0.5	0.41	0.38
KNN 3	0.6383	0.63	0.6	0.51
KNN 5	0.5745	0.55	0.5	0.37
KNN 10	0.5319	0.47	0.38	0.21

Exp 3: FastText

	Overall Accuracy	Overall Weighted Average F1	Minority Precision	Minority F1
K-CR beta:0.15, interval:6, iterations: 10	0.55	0.53	0.49	0.62
SVM (linear kernel)	0.57	0.42	0	0
RF (500 Trees)	0.53	0.53	0.44	0.39
KNN 1	0.51	0.5	0.41	0.38
KNN 3	0.64	0.63	0.6	0.51
KNN 5	0.57	0.55	0.5	0.37
KNN 10	0.53	0.47	0.38	0.21

Exp 4: GloVe

	Overall Accuracy	Overall Weighted Average F1	Minority Precision	Minority F1
K-CR beta:0.15, interval:6, iterations: 10	0.57	0.57	0.5	0.6
SVM (linear kernel)	0.5745	0.42	0	0
RF (500 Trees)	0.5745	0.55	0.5	0.37
KNN 1	0.5532	53	0.46	0.36
KNN 3	0.5532	0.55	0.47	0.43
KNN 5	0.5532	0.54	0.47	0.4
KNN 10	0.6383	0.57	0.8	0.32

Table 5.8: Results from experiments 1,2,3,4

We further explored if our observations stand for the popular language model BERT in experiments 9 and 11. Table 5.9 presents the obtained results. With BERT embeddings, K-CR maintains its superior ability on minority class classification. After combining BERT

with the methods from figure 5.10, we present a new figure 5.11, from which we notice that all the classifiers benefit from employing word embeddings produced by BERT. Even SVM, which completely failed during the first 4 experiments, gains the ability to classify minority class instances. It becomes the second best classifier in terms of minority class F1-score.

	Exp 9: Bert (512 token windows) +sentence			
	Overall Accuracy	Overall Weighted Average F1	Minority Precision	Minority F1
K-CR beta:0.15, interval:6, iterations: 10	0.7245	0.72	0.67	0.68
SVM (linear kernel)	0.7021	0.7	0.65	0.65
RF (500 Trees)	0.6806	0.67	0.69	0.55
KNN 1	0.5106	0.5	0.41	0.38
KNN 3	0.617	0.59	0.58	0.44
KNN 5	0.5745	0.56	0.5	0.41
KNN 10	0.7234	0.71	0.77	0.61

	Exp 11: Bert (512 token windows) +word			
	Overall Accuracy	Overall Weighted Average F1	Minority Precision	Minority F1
K-CR beta:0.15, interval:6, iterations: 10	0.7	0.7	0.64	0.67
SVM (linear kernel)	0.6809	0.68	0.62	0.63
RF (500 Trees)	0.7234	0.71	0.77	0.61
KNN 1	0.5319	0.53	0.44	0.42
KNN 3	0.617	0.6	0.57	0.47
KNN 5	0.617	0.61	0.56	0.5
KNN 10	0.7021	0.68	0.75	0.56

Table 5.9: Results of experiments 9 and 11

We also conducted data analysis to understand what is the difference between the contributions of the sentence vectors created by BERT and other word embedding techniques. Taking vectors created by word2vec-CBOW and BERT as examples, dimension reduction was first applied on sentence vectors with PCA and tSNE. The results are plotted in figures 5.12a, 5.12b, 5.12c and 5.12d. However, we were not able to notice clear distribution differences between BERT and word2vec-CBOW. In either case, the instances of both classes mixed together. Since SVM shows dramatical improvements in experiments 9 and 11 and taking the fact that SVM may use a hyper-plane as a separation surface, instances might be clearly separated in the 3 dimensional space. Figure 5.13 presents the data distribution of BERT in 3D space. Unfortunately, instances of two classes are still not visually separable. We conclude that this is a drawback of the dimension reduction techniques. We suspect that PCA and tSNE cannot produce the most representative reduced dimensions from the 769-dimensional vectors created by BERT.

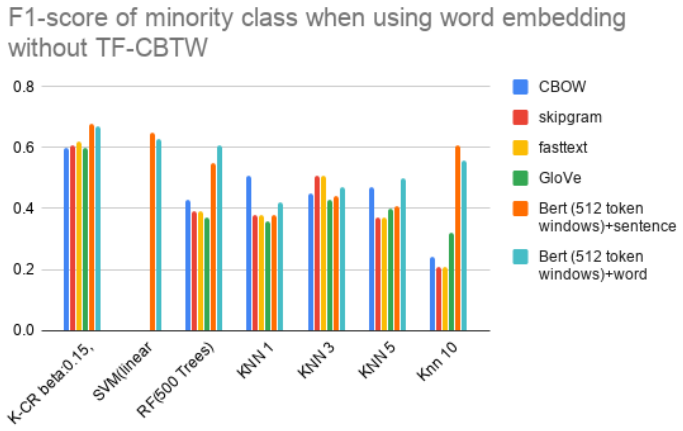
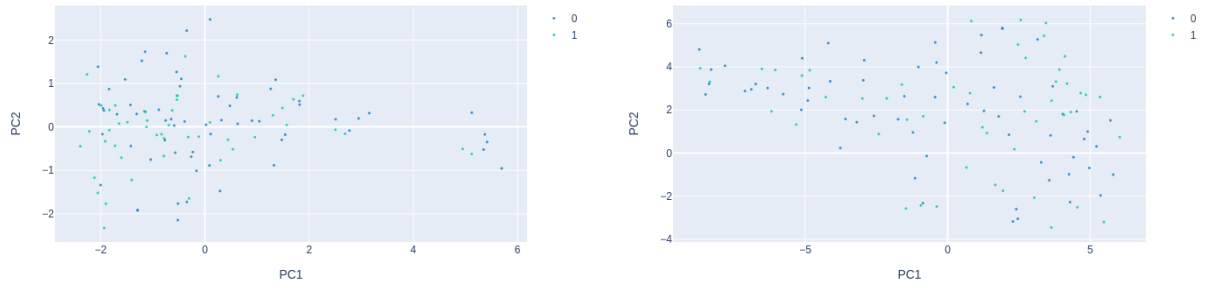


Figure 5.11: F1-score of minority class when using word embedding without TF-CBTW

(a) Data distribution of BERT embedding with PCA

(b) Data distribution of BERT embedding with tSNE



(c) Data distribution of word2vec-CBOW embedding with PCA

(d) Data distribution of word2vec-CBOW embedding with tSNE

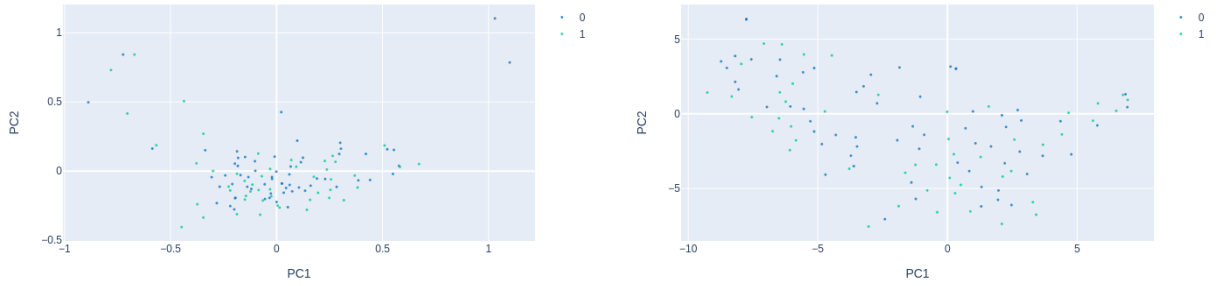


Figure 5.12: Data distribution of sentence vectors

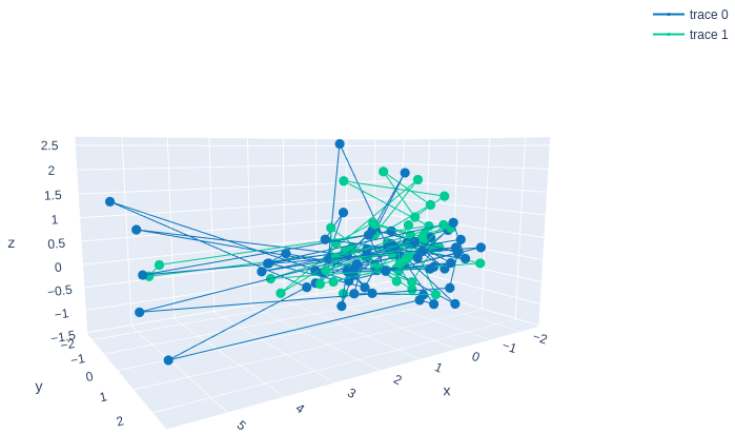


Figure 5.13: 3D visualization of sentence vectors created by BERT with PCA

To summarise, K-CR has the best classification performance with minority classes. With its design, K-CR learns data distributions effectively regardless of which word embedding technique is used.

Effect of Applying TF-CBTW

In experiments 5, 6, 7 and 8, we adopted TF-CBTW. Since TF-CBTW can highlight 'important' words for each class, we expect to obtain better classification performance on all classifiers. Table 5.10 presents the results of experiments 5-8. We notice that all classifiers have a dramatic performance improvement regardless of which word embedding technique is chosen. In terms of either weighted average F1-score or minority class F1-score, they have almost the same scores. To visually present the improvement, we created figures 5.14 and 5.15 based on experiments 1 and 5. We can clearly see the improvement in accuracy and weighted average F1-score in Figures 5.14 and 5.15, respectively. We conclude that TF-CBTW not only improves performance of minority class classification, but also improves the overall accuracy. In other words, although the dataset is imbalanced, the added TF-CBTW factor provides extra information to classifiers so that the classifiers can properly learn from the data.

Exp 5: Word2Vec-CBOW+TF-CBTW

	Overall Accuracy	Overall Weighted Average F1	Minority Precision	Minority F1
K-CR beta:0.15, interval:6, iterations: 10	0.9627	0.96	1	0.95
SVM	0.9787	0.98	1	0.97
RF (500 Trees)	1	1	1	1
KNN 1	1	1	1	1
KNN 3	0.978723	0.98	1	0.97
KNN 5	0.9787	0.98	1	0.97
KNN 10	0.9787	0.98	1	0.97

Exp 6: Word2Vec-Skipgram+TF-CBTW

	Overall Accuracy	Overall Weighted Average F1	Minority Precision	Minority F1
K-CR beta:0.15, interval:6, iterations: 10	0.9627	0.96	1	0.95
SVM	0.9787	0.98	1	0.97
RF (500 Trees)	1	1	1	1
KNN 1	0.9787	0.98	1	0.97
KNN 3	0.9787	0.98	1	0.97
KNN 5	0.9574	0.96	1	0.95
KNN 10	0.9574	0.96	1	0.95

Exp 7: FastText+TF-CBTW

	Overall Accuracy	Overall Weighted Average F1	Minority Precision	Minority F1
K-CR beta:0.15, interval:6, iterations: 10	0.96	0.96	1	0.95
SVM	0.9574	0.96	1	0.95
RF (500 Trees)	0.9149	0.93	0.83	0.91
KNN 1	0.9574	0.96	1	0.95
KNN 3	0.9574	0.96	1	0.95
KNN 5	0.9574	0.96	1	0.95
KNN 10	0.9574	0.96	1	0.95

Exp 8: GloVe+TF-CBTW

	Overall Accuracy	Overall Weighted Average F1	Minority Precision	Minority F1
K-CR beta:0.15, interval:6, iterations: 10	0.96	0.96	1	0.95
SVM	0.9787	0.98	1	0.97
RF (500 Trees)	1	1	1	1
KNN 1	0.9787	0.98	1	0.97
KNN 3	0.9787	0.98	1	0.97
KNN 5	0.9787	0.98	1	0.97
KNN 10	0.9574	0.96	1	0.95

Table 5.10: Results from experiments 5, 6, 7, 8

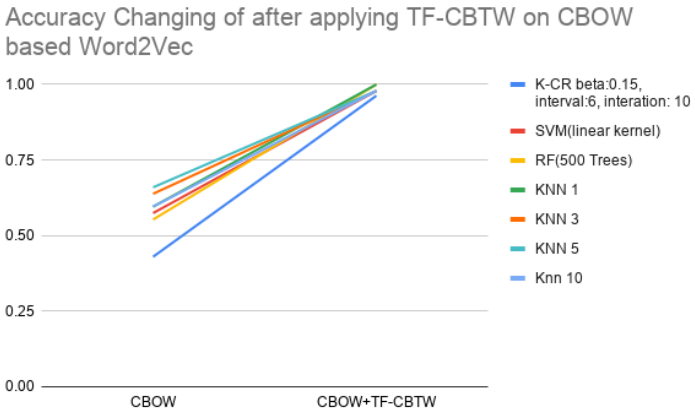


Figure 5.14: Accuracy change after applying TF-CBTW on CBOW based Word2Vec

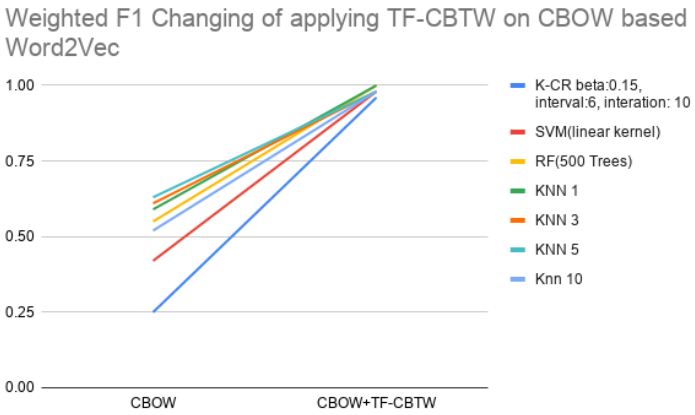


Figure 5.15: Weighted F1-score change after applying TF-CBTW on CBOW based Word2Vec

Apart from the word embeddings mentioned above, we aimed to employed the TF-CBTW with BERT as well. However, due to the nature that BERT, it does not producing word embeddings directly, the TF-CBTW weights cannot be adopted like other word embedding models. We applied TF-CBTW to BERT as a word filter. Before a sentence is processed, it is filtered by the TF-CBTW ranking list and only the top 100,000 (or other threshold set by user) words will be maintained. The filtered sentence is then fed into BERT. Comparing the minority class F1-scores of experiment 10 in table 5.11 and the minority class F1-scores of experiment 9 in table 5.9, performance drops occur for all the tested classifiers after applying TF-CBTW. This is expected, since BERT works better by taking advantage of its ability to correlate contextual information. The application of the TF-CBTW breaks the contexts apart.

	Exp 10: Bert (512 token windows)+ TF-CBTW+sentence			
	Overall Accuracy	Overall Weighted Average F1	Minority Precision	Minority F1
K-CR beta:0.15, interval:6, iterations: 10	0.64	0.64	0.57	0.6
SVM (linear kernel)	0.6383	0.62	0.62	0.48
RF(500 Trees)	0.5745	0.57	0.5	0.5
KNN 1	0.5532	0.55	0.47	0.46
KNN 3	0.5957	0.58	0.53	0.46
KNN 5	0.7021	0.7	0.67	0.63
KNN 10	0.7021	0.69	0.71	0.59

Table 5.11: Results of experiment 10

Effect of Different Window Size for BERT

In experiments 9 and 13, another comparison is made to explore BERT's performance with different window size settings. The results are shown in table 5.12. When BERT has fixed size windows, classifiers achieve better performance. With the K-CR classifier, it is 17% higher with fixed size windows BERT than dynamic size windows in terms of minority F1-score. In section 3.3.4, we discussed the reasons why we expect that BERT with dynamic windows would provide better embeddings. However, the results are the opposite to what we expected. One possible reason is that the tweets of a user are not independent, as we thought. Factors such as emotion are consistent through the history of the tweets. In other words, a depressed user will display negative emotions in all his/her tweets. Therefore, with fixed size window, BERT can learn more comprehensive contextual information and produce better embeddings.

	Exp 9: Bert (512 token windows)+ sentence			
	Overall Accuracy	Overall Weighted Average F1	Minority Precision	Minority F1
K-CR beta:0.15, interval:6, iterations: 10	0.7245	0.72	0.67	0.68
SVM (linear kernel)	0.7021	0.7	0.65	0.65
RF (500 Trees)	0.6806	0.67	0.69	0.55
KNN 1	0.5106	0.5	0.41	0.38
KNN 3	0.617	0.59	0.58	0.44
KNN 5	0.5745	0.56	0.5	0.41
KNN 10	0.7234	0.71	0.77	0.61
	Exp 13: Bert (Dynamic windows)+ sentence			
	Overall Accuracy	Overall Weighted Average F1	Minority Precision	Minority F1
K-CR beta:0.15, interval:6, iterations: 10	0.55	0.56	0.48	0.51
SVM (linear kernel)	0.62	0.61	0.56	0.53
RF(500 Trees)	0.55	0.54	0.47	0.4
KNN 1	0.45	0.44	0.33	0.32
KNN 3	0.49	0.48	0.38	0.33
KNN 5	0.47	0.45	0.33	0.29
KNN 10	0.49	0.44	0.3	0.2

Table 5.12: Results of experiments 9 and 13

Effect of Different Post-processing for BERT

In experiments 9 and 11, we further explored the effect of applying different post-processing procedures for BERT. The results are presented in table 5.9. In experiment 9, BERT produced sentence vectors by summing second to last hidden layers and taking average. While in experiment 11, instead of producing sentence vectors, BERT produced vectors for each word within the sentence by summing together the last four layers. Then another step is taken to sum all words vectors of the sentence together into one single sentence vector. In table 5.9, K-CR in experiment 9 is 1% higher than the one in experiment 11 in terms of minority F1-score. SVM also has 2% better performance. However, RF and KNN work better with the embeddings produced in experiment 11. The results are consist with Devlin et al. (2018) and Alammar's summary: "there is no explicit guidelines on which post-processing application is the best". The choice will based on use cases, classifiers, and datasets.

Processing Time

For a classifier, processing time is also a key factor of performance. In the section where we introduced the algorithm of K-CR, we mentioned that there are a few parts to be explored and improved in the future in order to optimise the processing time. For now, we expect to have longer processing time with K-CR than for other classifiers. Overall, among all experiments conducted over the Bell Lets Talk Dataset, K-CR has the longest processing time on both training and predicting.

In table 5.13, we include processing time of all classifiers in exp 3 and exp 7. In exp 3, K-CR has obvious longer training time than other classifiers. For RF with 500 trees, which is considered a time consuming classifier, it is still about 30 times faster than K-CR. In addition, K-CR spent 0.38106 seconds in total to make predictions. The difference to other classifiers is relative small. However, taking into account the fact that there are only 47 samples in the testing set, the difference in prediction time is expected to increase as the number of samples increase. Furthermore, KNN has longer prediction time than training time since KNN does not really have a training process. Exp 7 has same experimental setting except adopting the TF-CBTW technique. The observations in exp 3 stand in exp 7. The adoption of TF-CBTW does not make much difference to the processing time of K-CR. In table 5.14, we also include another pair of experiments. Exp 4 and exp 8 also have the same experimental settings except that exp 8 adopted TF-CBTW. The observations still stand for these two experiments.

	Exp 3		Exp 7	
	Training	Predicting	Training	Predicting
K-CR beta:0.15, interval:6, iterations: 10	14.510596	0.381060	12.664747	0.409361
SVM (linear kernel)	0.14636	0.001299	0.172903	0.000207
RF (500 Trees)	0.482540	0.026686	0.391976	0.024849
KNN 1	0.000477	0.003083	0.000430	0.002441
KNN 3	0.000396	0.003087	0.000352	0.002446
KNN 5	0.000391	0.003060	0.000347	0.002450
KNN 10	0.000403	0.003148	0.000358	0.002487

Table 5.13: Processing time in seconds for of exp 3 and exp 7.

	Exp 4		Exp 8	
	Training	Predicting	Training	Predicting
K-CR beta:0.15, interval:6, iterations: 10	9.303222	0.339910	8.540981	0.4093610.370944
SVM (linear kernel)	0.12783	0.000882	0.118793	0.000236
RF (500 Trees)	0.442598	0.026735	0.393122	0.025562
KNN 1	0.000419	0.00245	0.000393	0.002143
KNN 3	0.000342	0.002495	0.000315	0.002124
KNN 5	0.000345	0.002505	0.000310	0.002144
KNN 10	0.000341	0.002521	0.000316	0.002202

Table 5.14: Processing time in seconds for exp 4 and exp 8.

In summary, the K-CR does have longer processing time. It did not obviously affect the performance of K-CR with Bell Lets Talk datasets is due to small number of samples. It will become an issue when dealing with larger datasets.

5.3.4 Comparison to Related Work

As mentioned in related work, Jamil et al. (2017) also conducted thorough experiments on the Bell Let's Talk dataset. We compare our results with theirs.

In their paper, they performed three sets of experiments. The first one trained a linear SVM on the original dataset. The second one trained the same SVM with the dataset balanced using SMOTE. The last one trained SVM with the dataset balanced by undersampling. For each set of the experiments, they trained the model on the training set and test it on the testing set. They obtained the best F1-score (0.7317) and accuracy (0.766) with a linear SVM and undersampled dataset. We achieved better F1-score (0.96) and accuracy (0.9627) with K-CR and TF-CBTW applied.

Furthermore, we conducted experiments with same settings using another prototype-based classifier - LVQ (Learning Vector Quantization). LVQ is an artificial neural network

	F1-Score	Accuracy
Jamil et al. (2017)	0.7317	0.766
K-CR+TF-CBTW	0.96	0.9627
GRLVQ	0.42	0.57
GRLVQ + TF-CBTW	0.84	0.87

Table 5.15: Experiment results of using GLVQ or GRLVQ

algorithm. In experiments conducted by Sousa et al. (2019), they compared the performance of different types of LVQ or LVQ extensions including LVQ1, LVQ2.1, LVQ3, GLVQ and GRLVQ. The results showed that GRLVQ (Generalized Relevance Learning Vector Quantization) had the best performance. Therefore, we selected GRLVQ to compare to. As shown in table 5.15, in the first round of experiments, TF-CBTW was not applied. GRLVQ shows worse performance than Jamil et al. (2017) and K-CR. However, after the TF-CBTW was applied in the second round of experiments, the performance of GRLVQ dramatically increased. It achieved 11% higher F-score and 11% higher accuracy. But it still perform worse than K-CR with TF-CBTW. The results prove that K-CR has better performance than the state-of-the-art prototype-based classifier LVQ, and the application of TF-CBTW can largely help the performance of classifiers.

5.4 CLPSych2015 Shared Task

We also tested a second text-based dataset: CLPSych2015. We aim to further explore if the discoveries we had with the Bell Lets Talk dataset carry over.

Coppersmith et al. (2015) released the Computational Linguistics and Clinical Psychology 2015 (CLPSych2015) shared task. They collected wweets from three types of users:

- Diagnosis of depression users
- Post traumatic stress disorder (PTSD) users
- Matched control users from communities which are demographically-matched

The shared task provided both training and test sets. However, since we did not participate in the shared task, we cannot use the test set. The labels for test set are not available. Therefore, we considered the training set only in this thesis. As shown in table 5.16, there are 1,145 users in the training set. Among them, 327 are users diagnosed with depression and 246 are diagnosed with PTSD. The rest of 572 users are age and gender-matched control users. According to Coppersmith et al. (2015), they applied several mechanisms to address ethical issues. They applied a white-list approach to protect the identity of all users. The approach included operations such as anonymized screen names and URLs by applying salted hashing. They also removed geo-location information and device information. Numerical labels are assigned to users with different mental conditions: 0 for control users, 1 for depression, and 2 for PTSD.

	Control	Depressed	PTSD
Number of Users	572	327	246
Number of Tweets	1,250,606	742,793	544,815

Table 5.16: CLPSych 2015 shared task training set statistics

5.4.1 Experiment Setup

Apart from external factors which will affect the performance of the K-CR classifier, we aim to explore any possible internal factors which may affect the performance. Therefore, we adopted the the CLPSych2015 dataset to conduct more experiments.

There are two factors to be considered. The first one is the number of prototypes constructed. The number of effective prototypes directly affect the classification ability of K-CR. The second one is the threshold set to the ranked list. The ranked list refers to the score list introduced in section 3.4.3. We designed 9 experiments, as shown in table 5.17. First, experiment 1 is designed to be compared to experiment 4. This is to further confirm the power of TF-CBTW which has been proven in the experiments with the Bell Lets Talk dataset. Apart from experiment 1, all other experiments have TF-CBTW applied by default. For experiments 2, 3 and experiments 4, 5, the objective is to explore how do different threshold affect K-CR's performance. We tested two threshold values: 1,000 and 10,000. In experiments 6, 7, 8 and 9, the main objective is to explore the performance of K-CR with different numbers of prototypes. In experiment 6 and 8, K-CR is set to create only one prototype for each class. While in experiments 7 and 9, it is set to create multiple prototypes. Experiments 1-9 mentioned above are trained and tested using 5-fold cross validation. In addition to experiments 1-9, we designed experiment 10 to train K-CR on the CLPSych2015 dataset and test it on the Bell Lets Talk dataset. The objective is to explore how well is K-CR generalizing between two independently-created mental health datasets.

Exp 1	FastText+1000
Exp 2	GloVe+TF-CBTW+1000
Exp 3	GloVe+TF-CBTW+10000
Exp 4	FastText+TF-CBTW+1000
Exp 5	FastText+TF-CBTW+10000
Exp 6	GloVe+TF-CBTW+1000+single prototype
Exp 7	GloVe+TF-CBTW+1000+multi prototypes
Exp 8	FastText+TF-CBTW+1000+single prototype
Exp 9	FastText+TF-CBTW+1000+multi prototype
Exp 10	CLPSych2015+Bell Lets Talk

Table 5.17: Experimental setup

Finally, figure 5.16 presents a similar experimental procedure with the CLPSych2015 dataset. The CLPSych2015 shared task dataset is firstly pre-processed by removing punc-

tuation, stop words, etc. Then the dataset is used to train the word2vec and FastText models. For GloVe and BERT, pre-trained models are imported. With embeddings models, we can create word embeddings for the dataset. At last, we conduct experiments and obtain their results by feeding instances into the K-CR classifier.

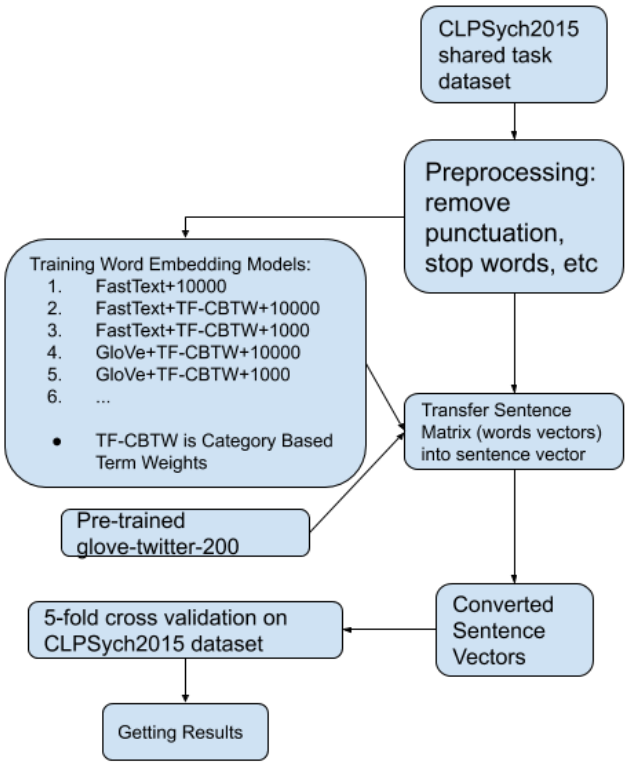


Figure 5.16: Workflow with CLPSych2015 dataset

5.4.2 Results and Discussion

Effect of Applying TF-CBTW

In table 5.18, we present the experiment results of experiment 1 and experiment 4. In both experiments, we employed the same FastText word embedding model. Besides, both of the thresholds are set to 1000.

It is clear that there is a dramatic improvement after applying TF-CBTW. The weighted average F1-score increased from 0.47 to 0.92. For the 'depressed' and 'PTSD' (two minority classes in the dataset) classes, the improvements are 53% and 78%, respectively, in terms of F1-score. In figure 5.17, it is easier to observe the improvements.

To summarise, the results are consistent with the results obtained on the Bell Let's Talk dataset. The TF-CBTW feature selection can effectively increase the performance of the K-CR classifier.

	K-CR with beta = 0.15, interval = 6	
	Exp 1: FastText+1000	**Exp 4: FastText+TF-CBTW+1000**
Overall Accuracy	0.52	0.92
Overall Weighted Average F1	0.47	0.92
'Depressed' class precision	0.41	0.83
'Depressed' class F1	0.35	0.88
'PTSD' class precision	0.58	0.99
'PTSD' class F1	0.18	0.95

Table 5.18: Results of experiment 1 and 4

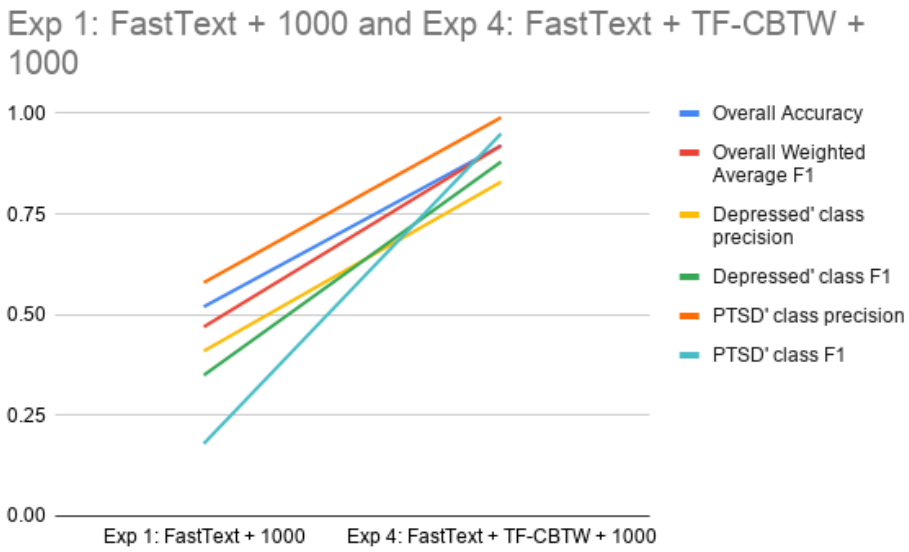


Figure 5.17: Results of experiments 1 and 4

Effect of Different Threshold

In experiments 2, 3, 4 and 5, the objective is to explore how will different thresholds affect the performance of the K-CR classifier. We expect to have better performance with smaller threshold, since smaller threshold means more 'not important' words are removed from the sentences. In theory, the classifier can better learn the representation of each class. In order to control the variables, there are two sets of comparisons: between experiment 2 and experiment 3, and between experiment 4 and experiment 5.

In table 5.19, we present the results of the first one. We applied TF-CBTW and employed the GloVe for word embeddings in both experiments. In experiment 2, the

threshold is set to 1,000, while in experiment 3 it is set to 10,000. From the table, we can see that the results of experiment 2 are better than those of experiment 3. The F1-score of the 'Depressed' class in experiment 2 is 7% higher than that in experiment 3. The F1-score of the 'PTSD' class in experiment 2 is 1% higher that in experiment 3. In other words, with 1,000 as threshold, the K-CR has better classification performance on both minority classes in the dataset. Similarly, we created table 5.20 to present the results of experiments 4 and 5, with the same setup except that we employed FastText instead of GloVe for word embeddings. From the table, we observe that the results of experiment 4 are better than those of experiment 5. We have the same observation that the threshold 1,000 is better than threshold 10,000. The F1-score of the 'Depressed' class in experiment 4 is 2% higher than that in experiment 3, while the F1-score of the 'PTSD' class is the same.

The observed results meet our expectation. We further explored why smaller threshold values can lead to better performance on the K-CR classifier. We figured that more noisy words are filtered with smaller threshold, and the remaining words provide better and more accurate knowledge for K-CR to create prototypes. However, this is highly depends on the choice of word embedding models. We employed GloVe and FastText, which do not learn word embeddings from contextual information. Although the context is broken by removing 'noisy' words from sentences, FastText and GloVe are not seriously affected, while this will not work with the BERT language model due to its nature of correlating contextual information. As discussed in section 5.3.3, the generated word embeddings have worse quality after removing 'noisy' words from sentences with BERT. The results of experiment 9 and experiment 11 conducted with the Bell Lets Talk dataset proved that K-CR has worse performance with ranked list applied to BERT.

	K-CR with beta = 0.15, interval = 6	
	Exp 2: GloVe+TF-CBTW+1000	**Exp 3: GloVe+TF-CBTW+10000**
Overall Accuracy	0.94	0.89
Overall Weighted Average F1	0.92	0.89
'Depressed' class precision	0.95	0.9
'Depressed' class F1	0.9	0.83
'PTSD' class precision	1	1
'PTSD' class F1	0.94	0.93

Table 5.19: Results of experiments 2 and 3

	K-CR with beta = 0.15, interval = 6	
	Exp 4: FastText+TF-CBTW+1000	**Exp 5: FastText+TF-CBTW+10000**
Overall Accuracy	0.92	0.91
Overall Weighted Average F1	0.92	0.91
'Depressed' class precision	0.83	0.83
'Depressed' class F1	0.88	0.86
'PTSD' class precision	0.99	0.98
'PTSD' class F1	0.95	0.95

Table 5.20: Results of experiments 4 and 5

Effect of Different Number of Created Prototypes

In experiments 6, 7, 8 and 9, we aim to learn how does the number of constructed prototypes affect the performance of K-CR. We expect to have better performance when there are multiple prototypes created. Similarly, to control the variables, there are two pairs of experiments: experiments 6 and 7, and experiments 8 and 9.

In table 5.21, we present the results of experiments 6 and 7. We applied TF-CBTW, 1,000 as threshold and employed GloVe for word embeddings in both experiments. The difference is that the K-CR creates only one 'best' prototype for each class in experiment 6, while in experiment 7, K-CR creates multiple prototypes by following the algorithm of prototype construction. Therefore, there are 3 prototypes in total in experiment 6. After checking the workflow, we saw that K-CR created an average of 211.2 prototypes during the 5-fold cross validation. From the table, the results of experiment 7 are better than those of experiment 6. The F1-score of the 'Depressed' class in experiment 7 is 27% higher than that in experiment 6. The F1-score of the 'PTSD' class in experiment 7 is 4% higher than in experiment 6. In other words, with multiple created prototypes, K-CR has better classification performance on both minority classes in the dataset. The difference is clearer in the ROC curves 5.18a and 5.18b. The curves in 5.18b are more centralised and closer to the top-left corner. Similarly, we created the table 5.22 to present the results of experiment 8 and 9, with the same setup except that we employed FastText instead of GloVe for word embeddings. The same as in experiment 6, experiment 8 also created 3 prototypes, and K-CR created an average of 262.8 prototypes in experiment 9. From the table, we observed that the results of experiment 9 are better than those of experiment 8. We have the same observation as on the previous pair of experiments, that K-CR with multiple prototypes is better than with single prototypes. The F1-score of the 'Depressed' class in experiment 9 is 20% higher than that in experiment 8. The F1-score of the 'PTSD' class in experiment 9 is 7% higher than that in experiment 8.

These results meet our expectation, because prototypes are key mechanisms in the K-CR algorithm. The classifier solely relies on the quality of the prototypes to do accurate classification. The data distribution in text is complex. More prototypes means more

variants of a class are considered. Therefore, it is necessary to set the K-CR to learn multiple prototypes.

	K-CR with beta = 0.15, interval = 6	
	Exp 6: GloVe+TF-CBTW+1000 +single prototype	**Exp 7: GloVe+TF-CBTW+1000 +multiple prototypes**
Overall Accuracy	0.66	0.94
Overall Weighted Average F1	0.66	0.92
'Depressed' class precision	0.46	0.95
'Depressed' class F1	0.63	0.9
'PTSD' class precision	1	1
'PTSD' class F1	0.9	0.94

Table 5.21: Results of experiments 6 and 7

	K-CR with beta = 0.15, interval = 6	
	Exp 8: FastText+TF-CBTW+1000 +single prototype	**Exp 9: FastText+TF-CBTW+1000 +multiple prototypes**
Overall Accuracy	0.73	0.92
Overall Weighted Average F1	0.74	0.92
'Depressed' class precision	0.51	0.83
'Depressed' class F1	0.68	0.88
'PTSD' class precision	1	0.99
'PTSD' class F1	0.88	0.95

Table 5.22: Results of experiments 8 and 9

Training on CLPSych2015 and Testing on Bell Lets Talk

One last experiment was done by training the K-CR classifier on the CLPSych2015 dataset and testing on the Bell Lets Talk dataset. During the implementation, there was a problem that there are different numbers of classes in these two datasets. In CLPSych2015 dataset, there are 3 classes 'Control', 'Depressed' and 'PTSD', while in Bell Lets Talk dataset, there are only two classes including 'Not Depressed' (which is same as 'Control') and 'Depressed'. Therefore, we maintained the 'Control' (or 'Not Depressed') class in both dataset. Then we removed the 'PTSD' instances from the dataset.

Table 5.23 presents the results of the experiment. For this experiment, we employed the best setting learnt through the previous experiments: K-CR is set to create multiple

(a) ROC curve of experiment 6

(b) ROC curve of experiment 7

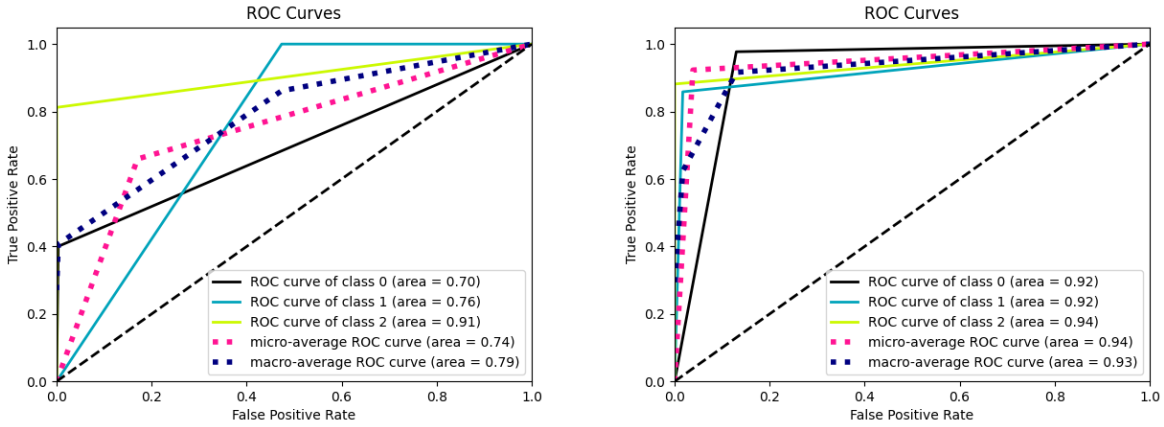


Figure 5.18: ROC curves of the K-CR classifier in experiments 6 and 7

prototypes, applying TF-CBTW, FastText as word embedding mechanism, and threshold as 1,000. From the table, we achieved 0.95 weighted average F1-score and 0.95 on the minority 'Depressed' class. The ROC curve 5.19 proves that the trained classifier on the CLPSych2015 dataset can be generalised to a unseen dataset and performs well and without bias on the majority class. Furthermore, the table 5.24 presents two more sets of results for comparison. The left one did not apply the TF-CBTW. The right one used single prototypes. Comparing the results from 5.24 and 5.23, the performance of generalising to unseen data dramatically dropped. This further proves the benefits of applying TF-CBTW and of creating multiple prototypes for the K-CR classifier.

	Exp 10: K-CR with beta = 0.15, interval = 6
	FastText+multi-prototypes+1000+TF-CBTW
Overall Accuracy	0.95
Overall Weighted Average F1	0.95
Depressed' class precision	0.9
Depressed' class F1	0.95

Table 5.23: Results of testing on the Bell Lets Talk dataset by using the K-CR classifier trained on the CLPSych2015 dataset

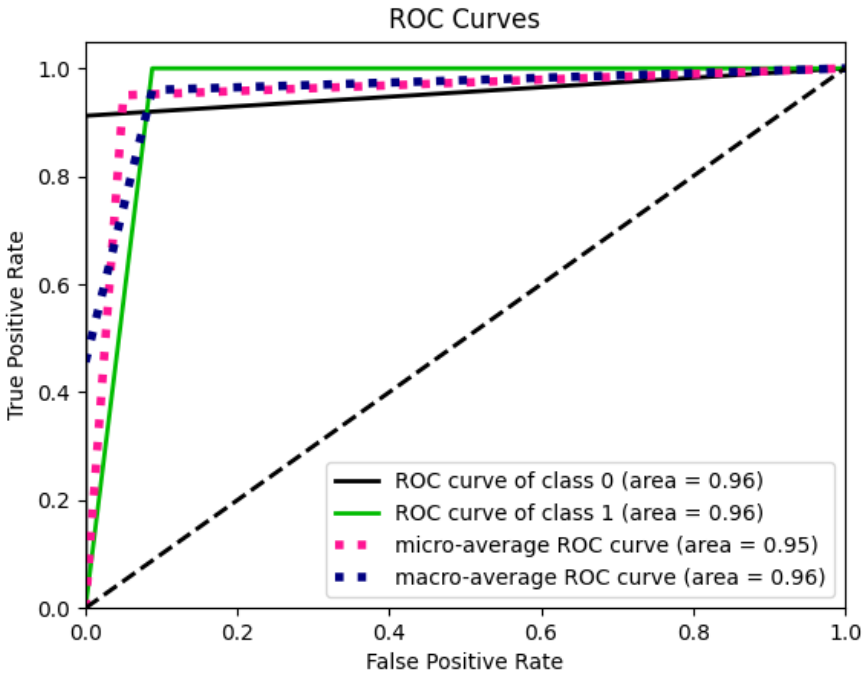


Figure 5.19: ROC curve of testing on the Bell Lets Talk dataset by using the K-CR classifier trained on the CLPSych2015 dataset

	K-CR with beta = 0.15, interval = 6	
	FastText +multi-prototypes +1000	FastText +single-prototype +1000
Overall Accuracy	0.57	0.43
Overall Weighted Average F1	0.52	0.26
Depressed' class precision	0.5	0.43
Depressed' class F1	0.29	0.6

Table 5.24: Results of testing on the Bell Lets Talk dataset by using the K-CR classifier trained on the CLPSych2015 dataset

5.4.3 Comparison to Related Work

Orabi et al. (2018) also conducted experiments against the CLPSych2015 dataset. They focused on obtaining better classification performance by proposing a novel multi-task deep learning approach and by optimizing the word embeddings.

Their first experiment was conducted on the training set with 5-fold cross validation. They tested the dataset against several classifiers including SVM, several versions of CNNs and BiLSTM. They achieved the best F1-score of 0.86967 and accuracy of 0.87957 with CNNWithMax and optimized word embeddings. In our experiments, we obtained the best

results in exp 7, by employing the K-CR classifier with GloVe word embeddings, with TF-CBTW applied, with the threshold set to 1,000 and K-CR set to produce multiple prototypes. We obtained F1-score 0.92 which is about 6% higher than that reported by Orabi et al. (2018). We also have about 6% improvement in terms of accuracy.

In their second experiment, they trained the classifiers on the training set and tested them on the testing set. We conducted the same experiments as introduced before. From their results, they obtained the best F1-score (0.82252) and accuracy (0.83117) with MultiChannelCNN and optimized word embeddings. In our experiments, we employed K-CR with FastText word embeddings. The threshold is set to 1,000. K-CR is set to create multiple prototypes and the TF-CBTW is applied. We obtained an F1-score of 0.95 which is about 12% better and an accuracy of 0.95 which is about 11% better.

We also conducted further experiments to compare to other prototype based classifiers, as we did on the Bell Lets Talk dataset. We applied GRLVQ with same settings, on the CLPSych2015 dataset. As learned in the previous experiments, GRLVQ performs better with TF-CBTW applied. Therefore, we only tested GRLVQ with TF-CBTW applied. As shown in table 5.25, GRLVQ has the lowest performance among all three classifiers. Therefore we can say taht K-CR performs better than the state-of-the-art classifiers on this task.

	F1-score	Accuracy	AUC	Precision	Recall
MultiChannelCNN + Optimised word embedding	0.82252	0.83317	0.923	81.626	84.439
K-CR+FastText+ multi-prototypes+ 1000+TF-CBTW	0.95	0.95	0.956	0.96	0.95
GRLVQ +TF-CBTW	0.775	0.81	0.86	0.73	0.82

Table 5.25: Binary classification task by training the model on the CLPsych2015 dataset and testing on the Bell Lets Talk dataset

5.5 Summary

In this chapter, we tested the K-CR classifier against 3 datasets: GEOBIA, Bell Lets Talk, and CLPSych2015. We explored external and internal factors which may affect the performance of K-CR. To conclude, K-CR shows better ability on imbalanced dataset classification. This ability is consistent through the three datasets we used. In next chapter, we will conclude what we learnt for our experiments, summarize our contributions, and discuss directions of future work.

Chapter 6

Conclusion and Future Work

6.1 Conclusions

In this thesis, the K-Closest Resemblance classifier was introduced. We first implemented the algorithm from scratch in Python. Comparing with previous literature in the remote sensing or depression detection fields, we try to improve classification performance from a data imbalance perspective. Instead of applying widely used data re-sampling (or SMOTE) to change the original dataset, K-CR tackles the challenge by considering the data imbalance problem at the algorithm designing level. To prove our expectations, we conducted experiments with three different datasets:

- With GEOBIA, we compared K-CR with RF, KNN, SVM, and CNN. We also imported experiment results on the same dataset from Maxwell et al. (2018). K-CR outperformed all the other classifiers. K-CR showed a superior capability on minority class classification.

- With Bell Lets Talk, we explored the effect of different word embedding techniques, applying TF-CBTW, setting different window size for BERT and employing different post-processing procedure for BERT. We learnt that K-CR is not sensitive to different word embedding techniques (word2vec, FastText and GloVe). Furthermore, the TF-CBTW feature selection can dramatically increase the performance of K-CR. Also, BERT with fixed-size window is better than dynamic size window under certain conditions. Finally, the post-processing procedure chosen for generating word embeddings is highly depending on the application.

- With CLPSych2015, we further explored if the above observations are consistent and explored the internal factors that can affect the performance of K-CR. We confirmed the effect of the TF-CBTW on this new dataset. In addition, we learnt that under certain conditions, smaller threshold values for the ranked list results leads to better performance for the K-CR classifier. Furthermore, we concluded that creating multiple-prototypes is better than creating only one prototype for each class. Finally, with one additional experiment where we trained on the CLPSych2015 dataset and

tested on the Bell Let's Talk dataset, we showed K-CR's generalisation ability for classifying unseen data.

6.2 Summary of Contributions

In this thesis, we contributed by implementing a Python version of the K-CR classifier. We also proposed to adopt a feature selection technique: TF-CBTW, to further improve the performance of K-CR on imbalanced datasets. Furthermore, we applied the K-CR classifier in urban land cover classification and depression detection tasks. We showed that our approach can achieve comparable performance on majority classes and better performance on minority classes than Random Forest, K-Nearest Neighbour, Support Vector Machines, Multi-layer Perception, Convolutional Neural Networks, and Long Short-Term Memory, especially for minority classes.

6.3 Future Work

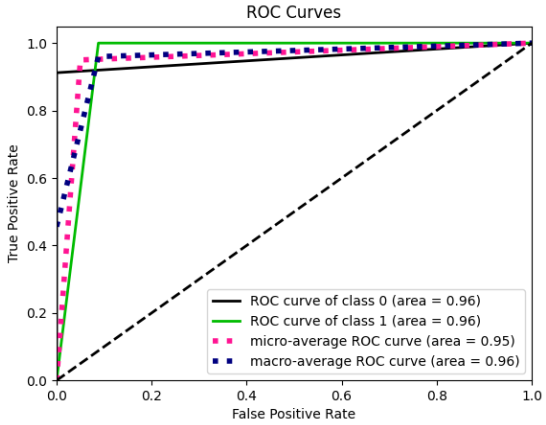
In future research, an important task is to conduct a thorough comparison between K-CR and deep learning techniques. The comparison can be made from the perspective of training time, prediction time, ability to handle imbalanced datasets, and classification performance. In addition, the current implementation of K-CR leads to long training times because of possible redundant prototypes being created. An effective way to eliminate unnecessary prototypes is another important task to explore in future work. Another direction is to explore other choices of discretization approaches during the learning phase of K-CR. It is valuable to explore if any other choices can further increase the performance of K-CR. Finally, due to the undeniable power of deep learning techniques, it is an interesting task to partly combine neural network architectures with the K-CR algorithm in order to generated explainable predictions (by designing ways to visualize the closest prototypes).

APPENDICES

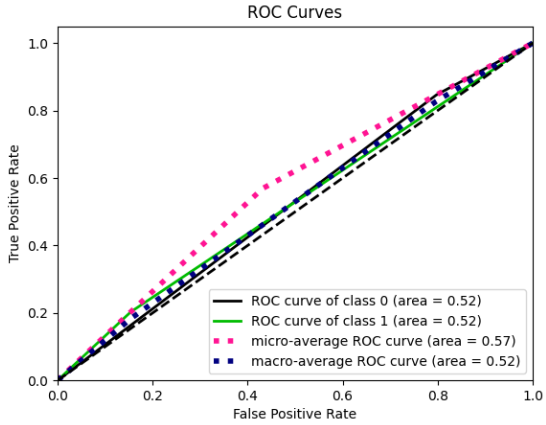
Appendix A

More Experimental Results on CLPSych2015 and Bell Lets Talk Datasets

(a) Multi-prototype K-CR with FastText embedding, 1000 selected top words from ranked list and TF-CBTW applied



(b) Multi-prototype K-CR with FastText embedding, 1000 selected top words from ranked list without applying TF-CBTW



(c) Single-prototype K-CR with FastText embedding, 1000 selected top words from ranked list without applying TF-CBTW

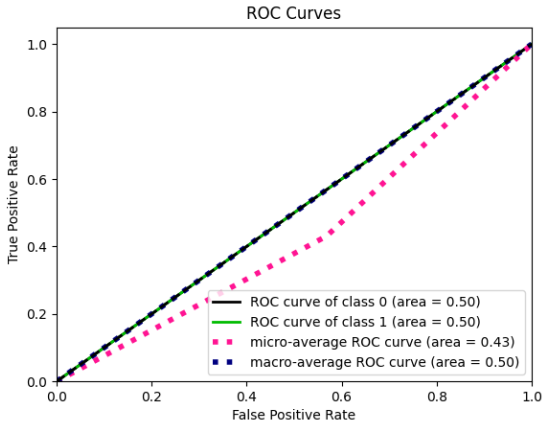
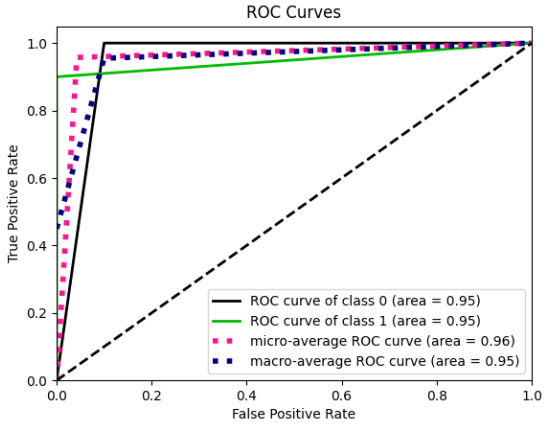
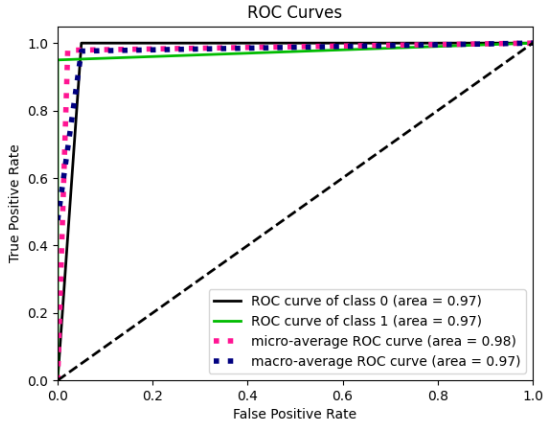


Figure A.1: ROC curves of training K-CR on CLPSych2015 dataset and testing on Bell Lets Talk dataset.

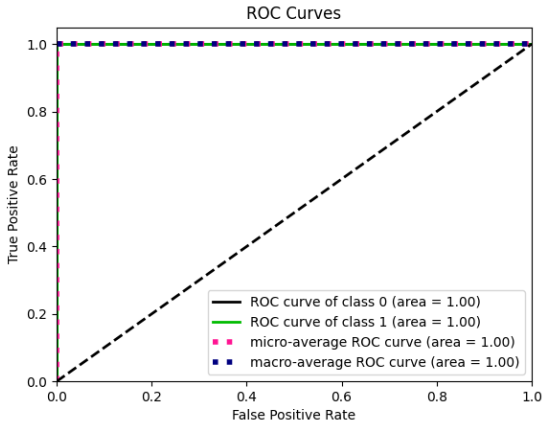
(a) K-CR



(b) KNN



(c) RF



(d) SVM

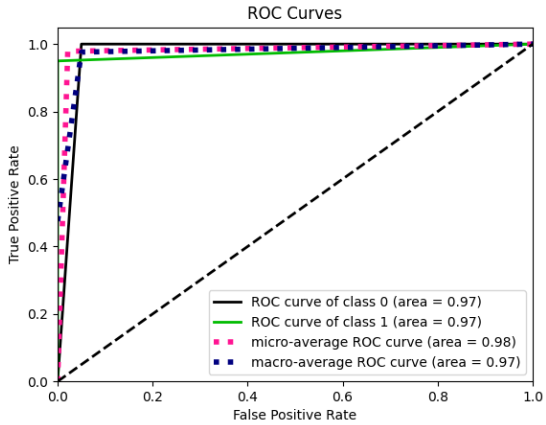
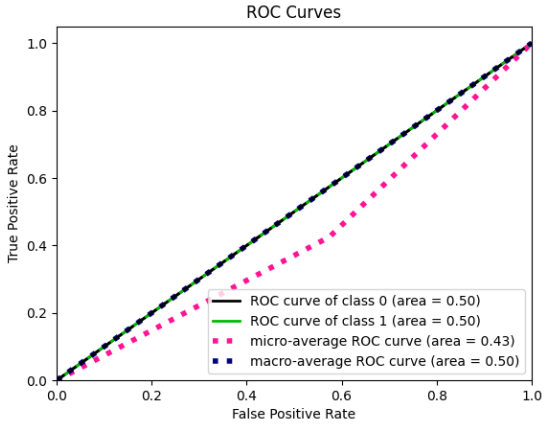
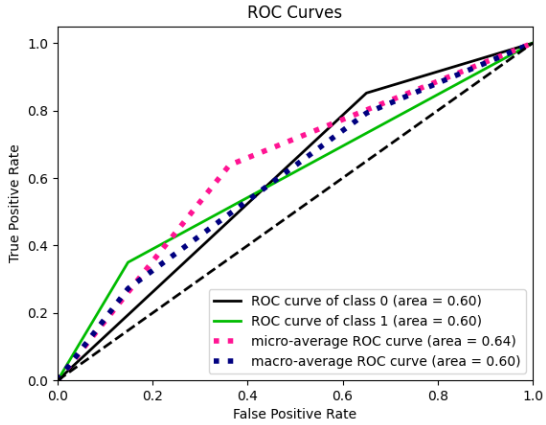


Figure A.2: ROC curves of 10 cross validation on Bell Lets Talk Dataset. Word2Vec-CBOW embedding used. TF-CBTW applied.

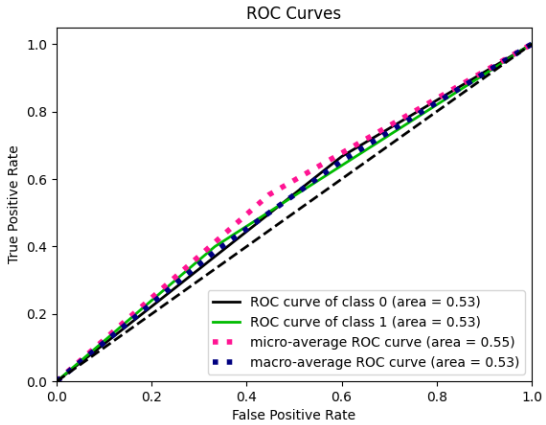
(a) K-CR



(b) KNN



(c) RF



(d) SVM

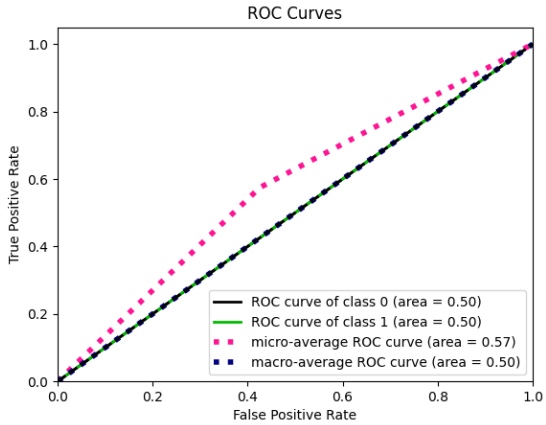
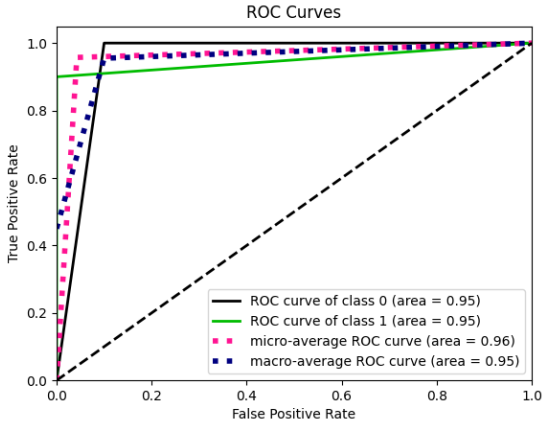
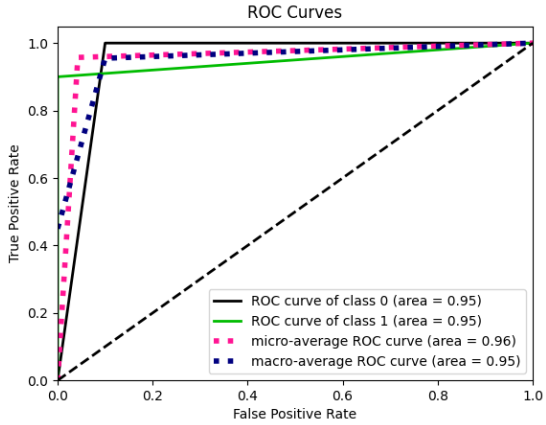


Figure A.3: ROC curves of 10 cross validation on Bell Lets Talk Dataset. Word2Vec-CBOW embedding used. TF-CBTW not applied.

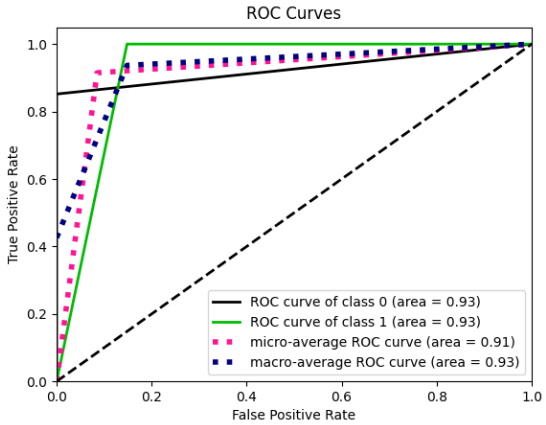
(a) K-CR



(b) KNN



(c) RF



(d) SVM

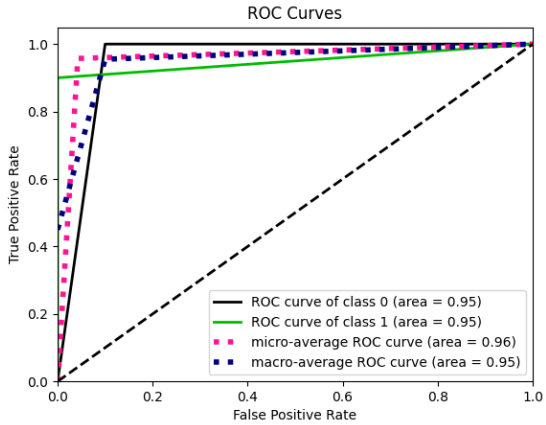
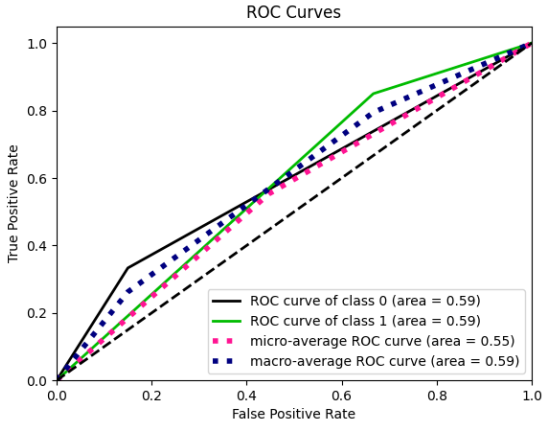
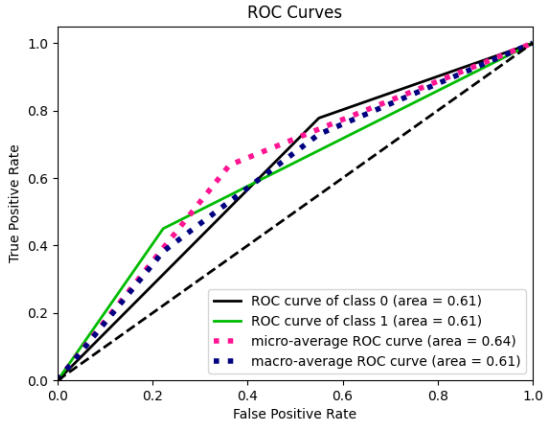


Figure A.4: ROC curves of 10 cross validation on Bell Lets Talk Dataset. FastText embedding used. TF-CBTW applied.

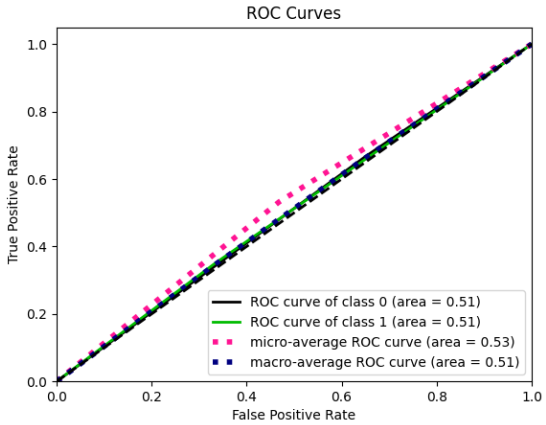
(a) K-CR



(b) KNN



(c) RF



(d) SVM

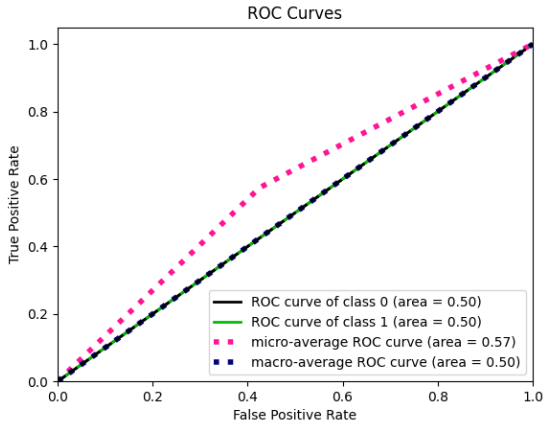
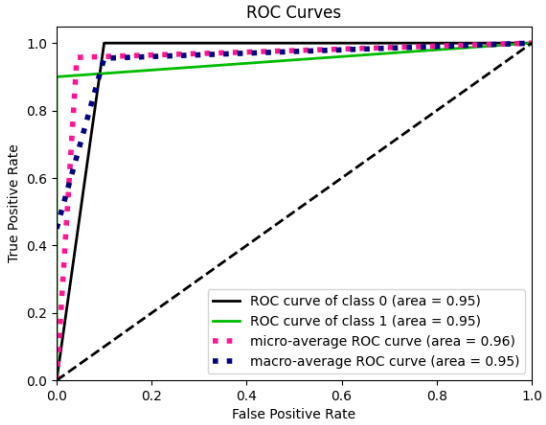
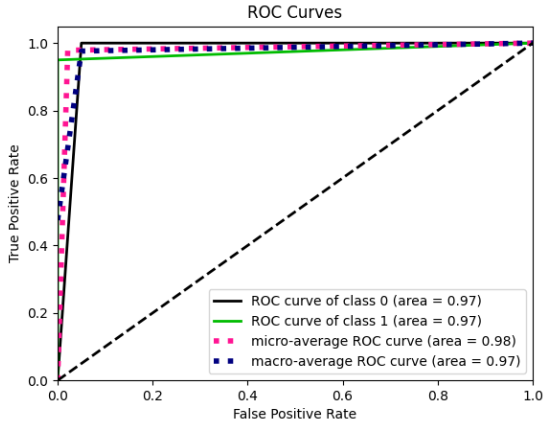


Figure A.5: ROC curves of 10 cross validation on Bell Lets Talk Dataset. FastText embedding used. TF-CBTW not applied.

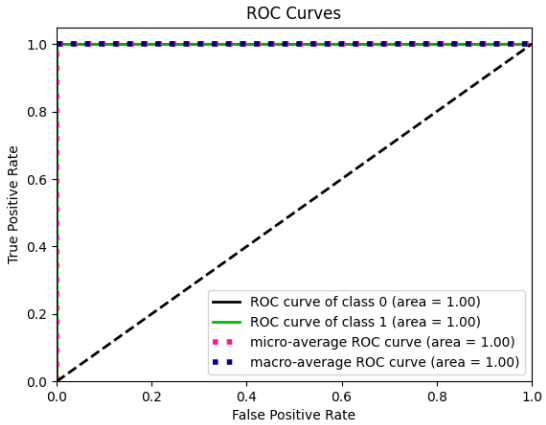
(a) K-CR



(b) KNN



(c) RF



(d) SVM

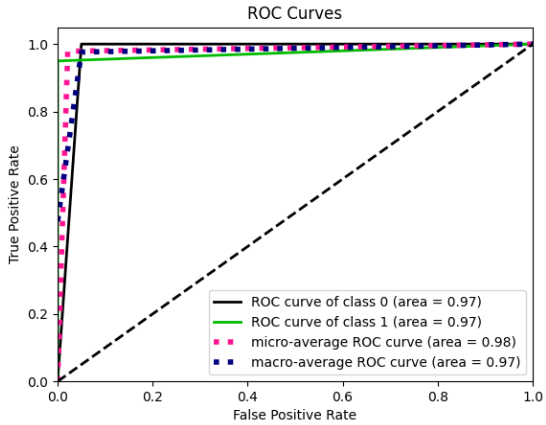
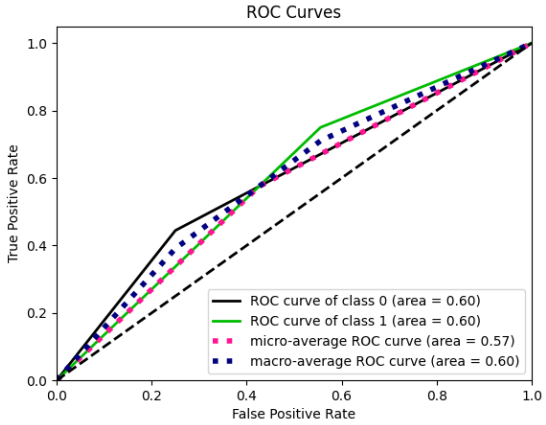
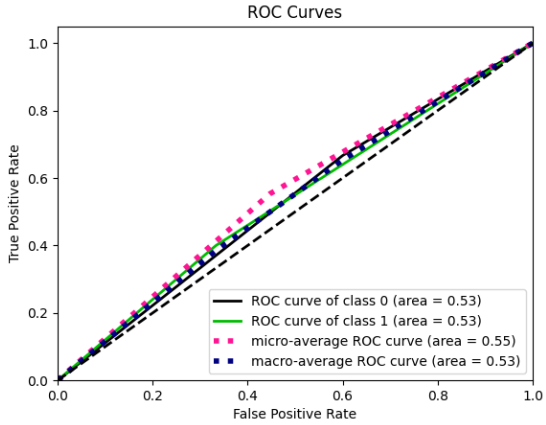


Figure A.6: ROC curves of 10 cross validation on Bell Lets Talk Dataset. GloVe embedding used. TF-CBTW applied.

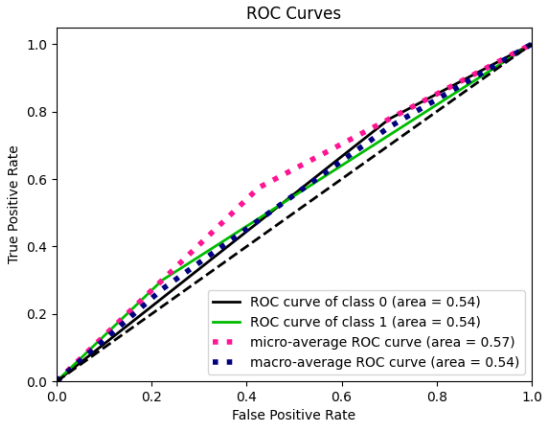
(a) K-CR



(b) KNN



(c) RF



(d) SVM

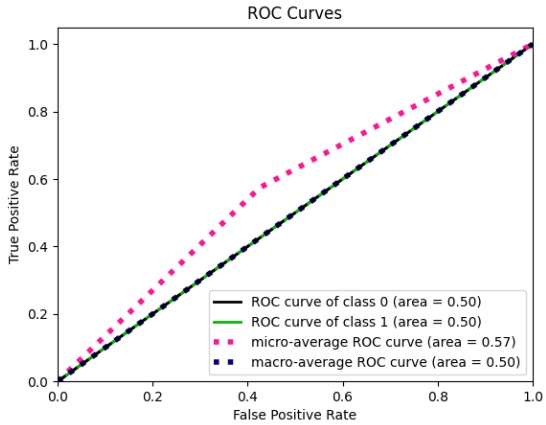
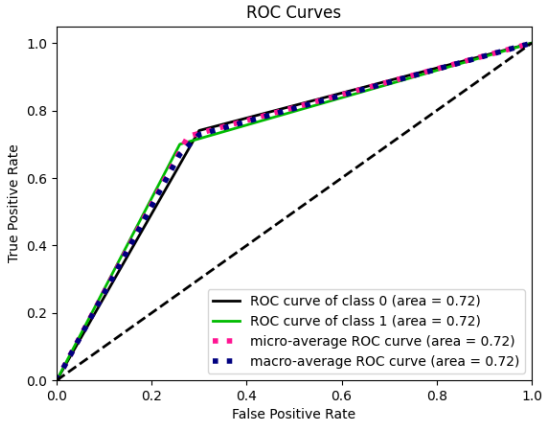
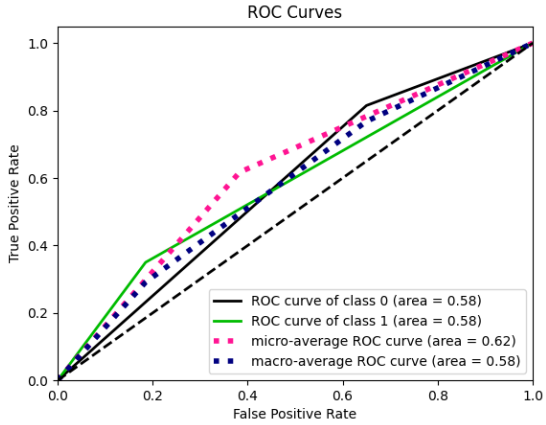


Figure A.7: ROC curves of 10 cross validation on Bell Lets Talk Dataset. GloVe embedding used. TF-CBTW not applied.

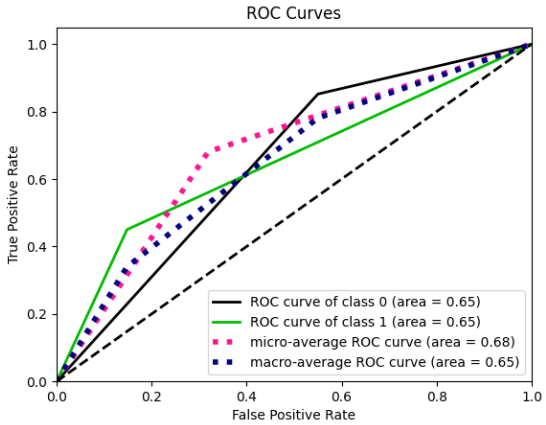
(a) K-CR



(b) KNN



(c) RF



(d) SVM

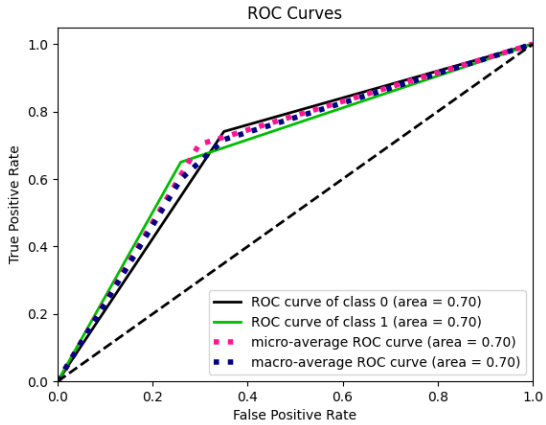
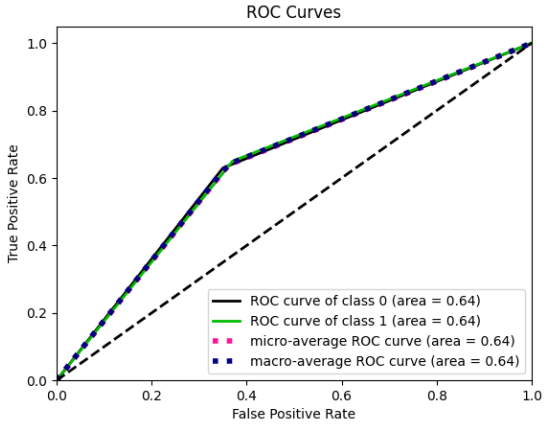
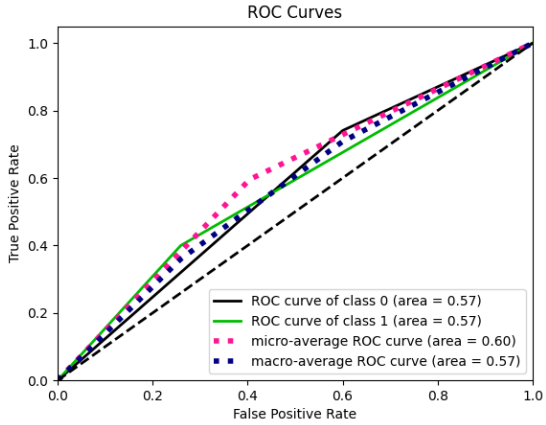


Figure A.8: ROC curves of 10 cross validation on Bell Lets Talk Dataset. BERT embedding used by generating sentence vector directly with fixed window size. TF-CBTW not applied.

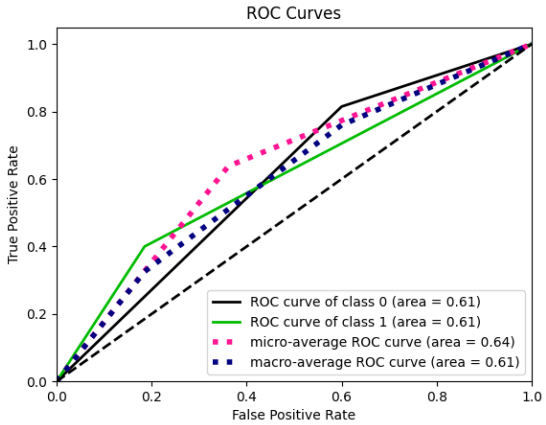
(a) K-CR



(b) KNN



(c) RF



(d) SVM

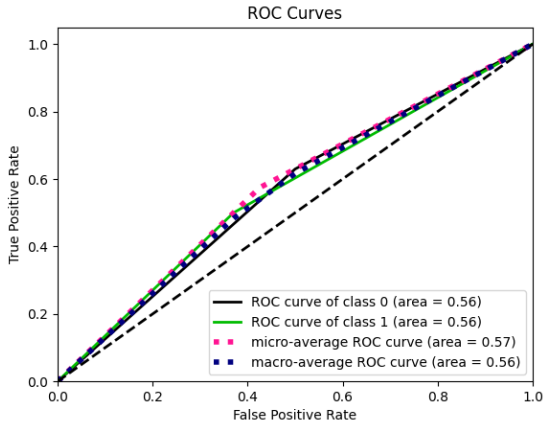
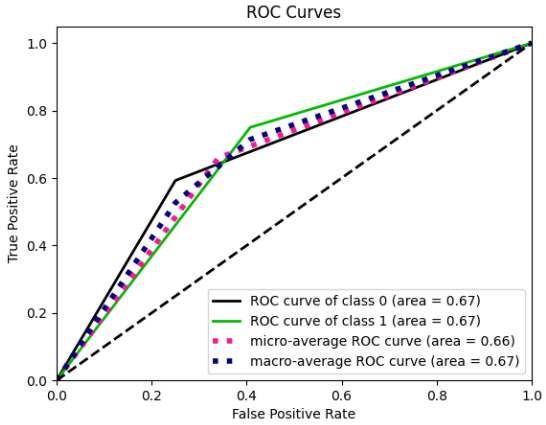
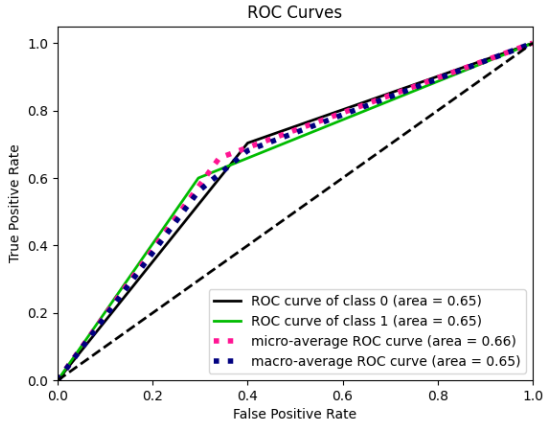


Figure A.9: ROC curves of 10 cross validation on Bell Lets Talk Dataset. BERT embedding used by generating sentence vector directly with fixed window size. TF-CBTW applied.

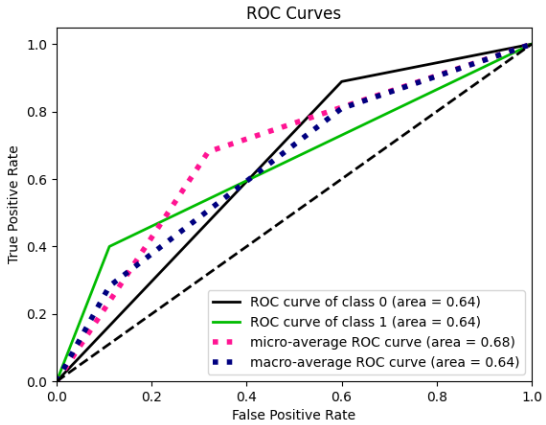
(a) K-CR



(b) KNN



(c) RF



(d) SVM

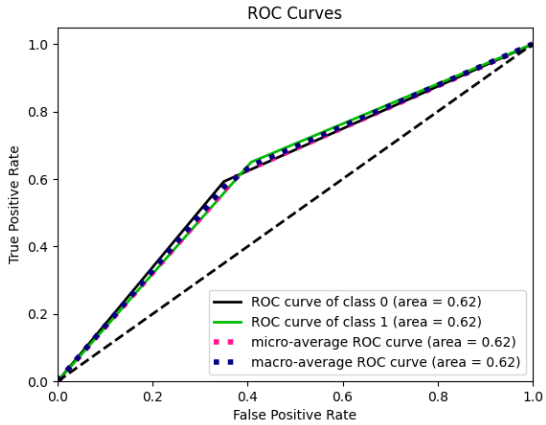
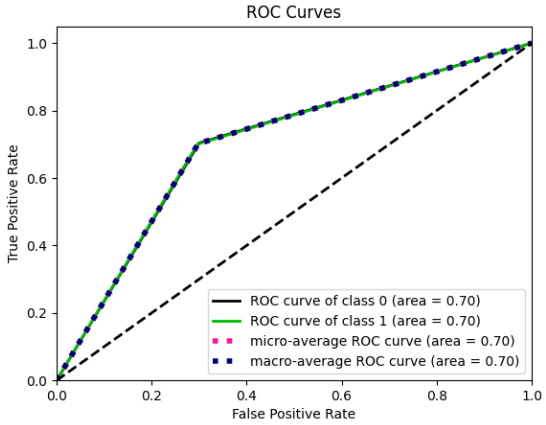
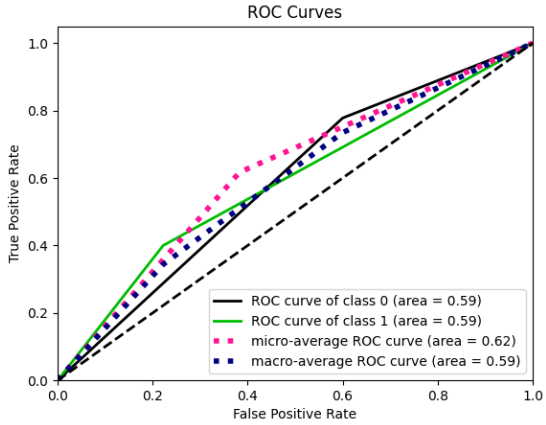


Figure A.10: ROC curves of 10 cross validation on Bell Lets Talk Dataset. BERT embedding used by generating word vector with fixed window size. TF-CBTW applied.

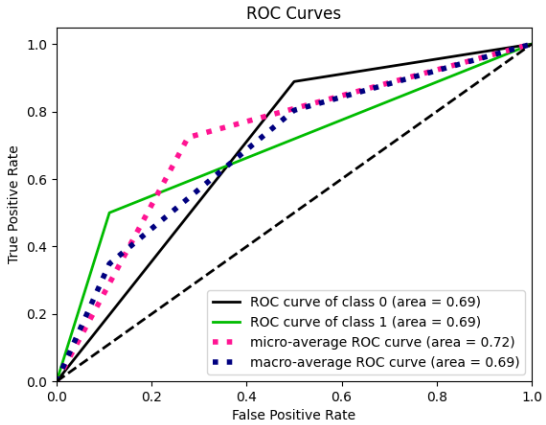
(a) K-CR



(b) KNN



(c) RF



(d) SVM

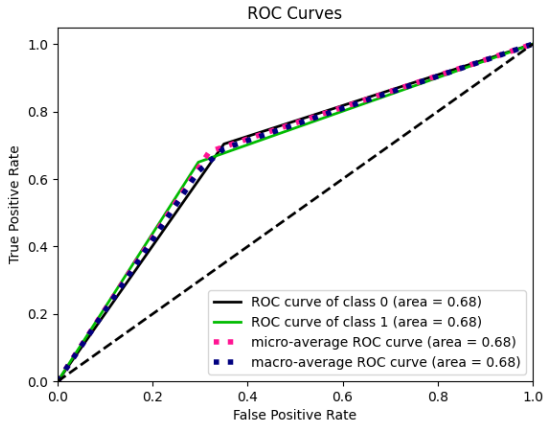
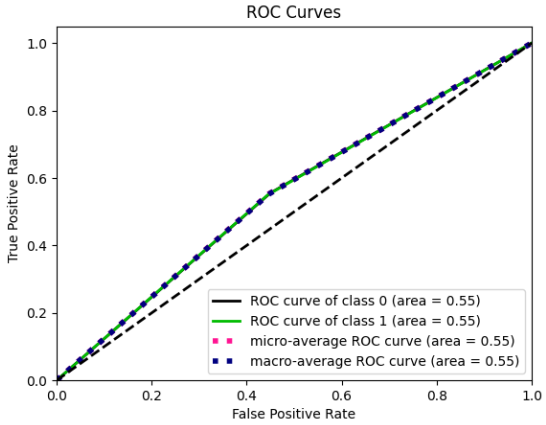
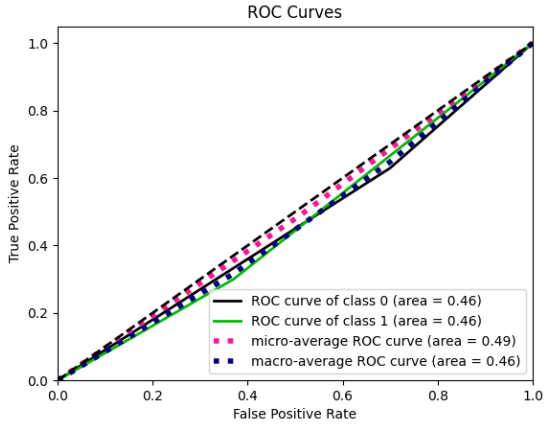


Figure A.11: ROC curves of 10 cross validation on Bell Lets Talk Dataset. BERT embedding used by generating word vector with fixed window size. TF-CBTW not applied.

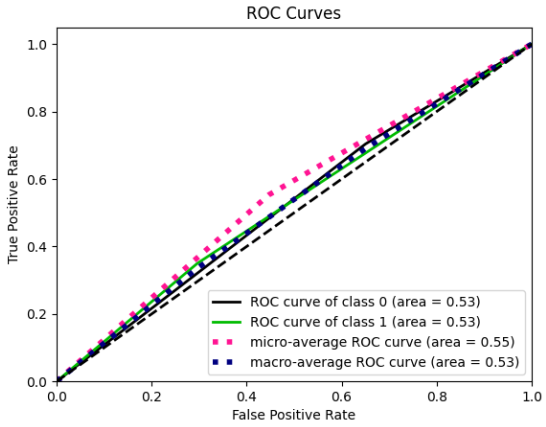
(a) K-CR



(b) KNN



(c) RF



(d) SVM

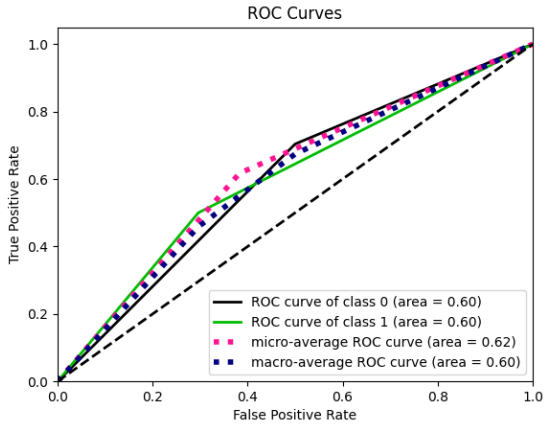


Figure A.12: ROC curves of 10 cross validation on Bell Lets Talk Dataset. BERT embedding used by generating sentence vector directly with dynamic window size. TF-CBTW not applied.

References

Jay Alammar. The illustrated bert, elmo, and co. (how nlp cracked transfer learning). http://jalammar.github.io/illustrated-bert/.

Nazanin Andalibi, Pinar Ozturk, and Andrea Forte. Sensitive self-disclosures, responses, and social support on instagram: the case of# depression. In *Proceedings of the 2017 ACM conference on computer supported cooperative work and social computing*, pages 1485–1500, 2017.

Sairam Balani and Munmun De Choudhury. Detecting and characterizing mental health related self-disclosure in social media. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, pages 1373–1378, 2015.

C Richard Barrett, Prasanta K Pattanaik, and Maurice Salles. On choosing rationally when preferences are fuzzy. *Fuzzy sets and systems*, 34(2):197–212, 1990.

Bikal Basnet. Nlp : Count based vs prediction models for word semantics. https://deepdatascience.wordpress.com/2017/04/08/nlp-count-based-vs-prediction-models-for-word-semantics/, 2017.

N. Belacel. *Méthodes de classification multicritère: Méthodologie et application à l'aide au diagnostic médical.* PhD thesis, Univ. Libre de Bruxelles, Belgium, Brussels, 1999.

Nabil Belacel. Multicriteria assignment method proaftn: Methodology and medical application. *European Journal of Operational Research*, 125(1):175–183, 2000.

Nabil Belacel and Mohamed Rachid Boulassel. Multicriteria fuzzy classification procedure procftn: methodology and medical application. *Fuzzy Sets and Systems*, 141(2):203–217, 2004.

Nabil Belacel, Cheng Duan, and Diana Inkpen. The k-closest resemblance classifier for remote sensing data. In *Canadian Conference on Artificial Intelligence*, pages 49–54. Springer, 2020a.

Nabil Belacel, Guangze Wei, and Yassine Bouslimani. The k closest resemblance classifier for amazon products recommender system. 02 2020b. doi: 10.5220/0009155108730880.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2016.

Hicham EL Boukkouri. Arithmetic properties of word embeddings. =https://medium.com/data-from-the-trenches/arithmetic-properties-of-word-embeddings-e918e3fda2ac, 2018.

Denis Bouyssou. A note on the 'min in favor'choice procedure for fuzzy preference relations. In *Advances in multicriteria analysis*, pages 9–16. Springer, 1995.

Jean-Pierre Brans and Ph Vincke. Note—a preference ranking organisation method: (the promethee method for multiple criteria decision-making). *Management science*, 31(6): 647–656, 1985.

Jean-Pierre Brans, Bertrand Mareschal, José Figueira, Salvatore Greco, and Matthias Ehrogott. *Promethee Methods*, pages 163–186. 01 2005. doi: 10.1007/0-387-23081-5_5.

A. T. Brasil Filho, P. R. Pinheiro, A. L. V. Coelho, and N. C. Costa. Comparison of two prototype-based multicriteria classification methods. In *2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making(MCDM)*, pages 133–140, 2009.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Chumphol Bunkhumpornpat, Krung Sinapiromsaran, and Chidchanok Lursinsap. Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 475–482. Springer, 2009.

J Quinonero Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. Dataset shift in machine learning. *The MIT Press*, 1:5, 2009.

Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Special issue on learning from imbalanced data sets. *ACM SIGKDD explorations newsletter*, 6(1):1–6, 2004.

CMHA. Canada must act now to prevent echo pandemic of mental illness due to covid-19. https://cmha.ca/news/canada-must-act-now-to-prevent-echo-pandemic-of-mental-illness-due-to-covid-19, 2020.

Christopher Colah. Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

Glen Coppersmith, Mark Dredze, Craig Harman, Kristy Hollingshead, and Margaret Mitchell. Clpsych 2015 shared task: Depression and ptsd on twitter. In *Proceedings of the 2nd Workshop on Computational Linguistics and Clinical Psychology: From Linguistic Signal to Clinical Reality*, pages 31–39, 2015.

Walter Daelemans, Antal Van den Bosch, et al. *Memory-based language processing*. Cambridge University Press, 2005.

Misha Denil and Thomas Trappenberg. Overlap versus imbalance. In *Canadian conference on artificial intelligence*, pages 220–231. Springer, 2010.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

P Domingos. A general method for making classifiers cost-sensitive. *Artificial Inelligence Group, Instituto Superior Técnico, Lisboa*, pages 1049–001, 1999.

Dheeru Dua and Casey Graff. UCI machine learning repository. https://archive.ics.uci.edu/ml/datasets/Urban+Land+Cover, 2017.

Kaibo Duan, S Sathiya Keerthi, and Aun Neow Poo. Evaluation of simple performance measures for tuning svm hyperparameters. *Neurocomputing*, 51:41–59, 2003.

Charles Elkan. The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence*, volume 17, pages 973–978. Lawrence Erlbaum Associates Ltd, 2001.

Alberto Fernández, Salvador García, Julián Luengo, Ester Bernadó-Mansilla, and Francisco Herrera. Genetics-based machine learning for rule induction: state of the art, taxonomy, and comparative study. *IEEE Transactions on Evolutionary Computation*, 14(6):913–941, 2010.

Alberto FernáNdez, Victoria LóPez, Mikel Galar, MaríA José Del Jesus, and Francisco Herrera. Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches. *Knowledge-based systems*, 42:97–110, 2013.

Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

Rohith Gandhi. Support vector machine - introduction to machine learning algorithms. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47, cited Jan 2020.

Jianfeng Gao, Patrick Pantel, Michael Gamon, Xiaodong He, and Li Deng. Modeling interestingness with deep neural networks. Technical Report MSR-TR-2014-56, October 2014. URL https://www.microsoft.com/en-us/research/publication/modeling-interestingness-with-deep-neural-networks/.

Bardan Ghimire, John Rogan, Víctor Rodríguez Galiano, Prajjwal Panday, and Neeti Neeti. An evaluation of bagging, boosting, and random forests for land-cover classification in cape cod, massachusetts, usa. *GIScience & Remote Sensing*, 49(5):623–643, 2012.

Maryellen L Giger. Machine learning in medical imaging. *Journal of the American College of Radiology*, 15(3):512–520, 2018.

Yorgos Goletsis, Costas Papaloukas, Dimitrios I Fotiadis, Aristidis Likas, and Lampros K Michalis. Automated ischemic beat classification using genetic algorithms and multicriteria decision analysis. *IEEE transactions on Biomedical Engineering*, 51(10):1717–1725, 2004.

Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610, 2005.

Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In *International conference on intelligent computing*, pages 878–887. Springer, 2005.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Robert C Holte, Liane Acker, Bruce W Porter, et al. Concept learning and the problem of small disjuncts. In *IJCAI*, volume 89, pages 813–818. Citeseer, 1989.

Zunaira Jamil, Diana Inkpen, Prasadith Buddhitha, and Kenton White. Monitoring tweets for depression to detect at-risk users. In *Proceedings of the Fourth Workshop on Computational Linguistics and Clinical Psychology—From Linguistic Signal to Clinical Reality*, pages 32–40, 2017.

Sue Jamison-Powell, Conor Linehan, Laura Daley, Andrew Garbett, and Shaun Lawson. " i can't get no sleep" discussing# insomnia on twitter. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1501–1510, 2012.

Brian Johnson and Zhixiao Xie. Classifying a high resolution image of an urban area using super-object information. *ISPRS journal of photogrammetry and remote sensing*, 83: 40–49, 2013.

Brian A Johnson. High-resolution urban land-cover classification using a competitive multiscale object-based approach. *Remote Sensing Letters*, 4(2):131–140, 2013.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, word2vec2014. URL http://arxiv.org/abs/1408.5882.

Georgios Kontonatsios, Yannis Korkontzelos, Jun'ichi Tsujii, and Sophia Ananiadou. Using a random forest classifier to compile bilingual dictionaries of technical terms from comparable corpora. In *14th Conference of the European Chapter of the Association for Computational Linguistics*, 2014.

Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al. Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, 30(1):25–36, 2006.

Emir Kremic and Abdulhamit Subasi. Performance of random forest and svm in face recognition. *Int. Arab J. Inf. Technol.*, 13(2):287–293, 2016.

Akshi Kumar, Aditi Sharma, and Anshika Arora. Anxious depression prediction in real-time social data. *Available at SSRN 3383359*, 2019.

Niraj Kumar. Random forest algorithm, an interactive discussion. https://www.linkedin.com/pulse/random-forest-algorithm-interactive-discussion-niraj-kumar/.

Man Lan, Chew-Lim Tan, Hwee-Boon Low, and Sam-Yuan Sung. A comprehensive comparative study on term weighting schemes for text categorization with support vector machines. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1032–1033, 2005.

Daniel T Larose and Chantal D Larose. *Discovering knowledge in data: an introduction to data mining*, volume 4. John Wiley & Sons, 2014.

Wonji Lee, Chi-Hyuck Jun, and Jong-Seok Lee. Instance categorization by support vector machines to adjust weights in adaboost for imbalanced data classification. *Information Sciences*, 381:92–103, 2017.

Edda Leopold and Jörg Kindermann. Text categorization with support vector machines. how to represent texts in input space? *Machine Learning*, 46(1-3):423–444, 2002.

Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

Ying Liu, Han Tong Loh, Youcef-Toumi Kamal, and Shu Beng Tor. Handling of imbalanced data in text classification: Category-based term weights. In *Natural language processing and text mining*, pages 171–192. Springer, 2007.

Victoria López, Alberto Fernández, Salvador García, Vasile Palade, and Francisco Herrera. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information sciences*, 250:113–141, 2013.

Lei Ma, Yu Liu, Xueliang Zhang, Yuanxin Ye, Gaofei Yin, and Brian Alan Johnson. Deep learning in remote sensing applications: A meta-analysis and review. *ISPRS journal of photogrammetry and remote sensing*, 152:166–177, 2019.

Aaron E Maxwell, Timothy A Warner, and Fang Fang. Implementation of machine-learning classification in remote sensing: An applied review. *International Journal of Remote Sensing*, 39(9):2784–2817, 2018.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013b.

SW Mohod and CA Dhote. Feature selection technique for text document classification: An alternative approach. *International Journal on Recent and Innovation Trends in Computing and Communication*, 2(9):2914–2917, 2014.

Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer, 2005.

Ryan Nichols. Linguistic inquiry and word count. https://hecc.ubc.ca/quantitative-textual-analysis/qta-practice/linguistic-inquiry-and-word-count/, cited May 2020.

Ahmed Husseini Orabi, Prasadith Buddhitha, Mahmoud Husseini Orabi, and Diana Inkpen. Deep learning for depression detection of twitter users. In *Proceedings of the Fifth Workshop on Computational Linguistics and Clinical Psychology: From Keyboard to Clinic*, pages 88–97, 2018.

SA Orlovsky. Decision-making with a fuzzy preference relation. *Fuzzy sets and systems*, 1 (3):155–167, 1978.

Mahesh Pal and PM Mather. Support vector machines for classification in remote sensing. *International journal of remote sensing*, 26(5):1007–1011, 2005.

James W Pennebaker. The secret life of pronouns. *New Scientist*, 211(2828):42–45, 2011.

James W Pennebaker, Roger J Booth, and Martha E Francis. Linguistic inquiry and word count: Liwc [computer software]. *Austin, TX: liwc. net*, 135, 2007.

Jeffrey Pennington. Glove: Global vectors for word representation. https://nlp.stanford.edu/projects/glove/, 2014.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

Robi Polikar. Ensemble based systems in decision making. *IEEE Circuits and systems magazine*, 6(3):21–45, 2006.

Martin Potthast, Sebastian Köpsel, Benno Stein, and Matthias Hagen. Clickbait detection. In *European Conference on Information Retrieval*, pages 810–817. Springer, 2016.

Daniel Preoţiuc-Pietro, Maarten Sap, H Andrew Schwartz, and Lyle Ungar. Mental illness detection at the world well-being project for the clpsych 2015 shared task. In *Proceedings of the 2nd Workshop on Computational Linguistics and Clinical Psychology: From Linguistic Signal to Clinical Reality*, pages 40–45, 2015.

Philip Resnik, William Armstrong, Leonardo Claudino, and Thang Nguyen. The university of maryland clpsych 2015 shared task system. In *Proceedings of the 2nd Workshop on Computational Linguistics and Clinical Psychology: From Linguistic Signal to Clinical Reality*, pages 54–60, 2015.

Joseph Rocca. Ensemble methods: bagging, boosting and stacking. https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205, 2019.

Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

Bernard Roy. *Multicriteria methodology for decision aiding*, volume 12. Springer Science & Business Media, 2013.

Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. 2014.

Chris Seiffert, Taghi M Khoshgoftaar, Jason Van Hulse, and Andres Folleco. An empirical study of the classification performance of learners on imbalanced and noisy software quality data. *Information Sciences*, 259:571–595, 2014.

Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Gregoire Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *CIKM*, November 2014. URL https://www.microsoft.com/en-us/research/publication/a-latent-semantic-model-with-convolutional-pooling-structure-for-information-retrieval/.

Sonia Singh and Priyanka Gupta. Comparative study id3, cart and c4. 5 decision tree algorithm: a survey. *International Journal of Advanced Information Science and Technology (IJAIST)*, 27(27):97–103, 2014.

Diego P Sousa, Guilherme A Barreto, Charles C Cavalcante, and Cláudio MS Medeiros. Lvq-type classifiers for condition monitoring of induction motors: A performance comparison. In *International Workshop on Self-Organizing Maps*, pages 130–139. Springer, 2019.

Zhucheng Tu. An experimental analysis of multi-perspective convolutional neural networks. Master's thesis, University of Waterloo, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Peng Wang, Jiaming Xu, Bo Xu, Chenglin Liu, Heng Zhang, Fangyuan Wang, and Hongwei Hao. Semantic clustering and convolutional neural network for short text categorization. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 352–357, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-2058. URL https://www.aclweb.org/anthology/P15-2058.

Gary M Weiss. Mining with rarity: a unifying framework. *ACM Sigkdd Explorations Newsletter*, 6(1):7–19, 2004.

Gary M Weiss and Foster Provost. Learning when training data are costly: The effect of class distribution on tree induction. *Journal of artificial intelligence research*, 19: 315–354, 2003.

Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative study of CNN and RNN for natural language processing. *arXiv preprint arXiv:1702.01923*, 2017.

Hao Zhang, Alexander C Berg, Michael Maire, and Jitendra Malik. hat. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2126–2136. IEEE, 2006.

Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

Jiang Zhiyuan, Huang Qingyi, and Zhu Huiying. Urban land cover data classification prediction using multiclass rbf-svm model.