A LOCATION-AWARE SOCIAL MEDIA MONITORING SYSTEM

By

Ji Liu, Candidate, M.Sc.

University of Ottawa

November 2014

A thesis

submitted to the School of Faculty of Graduate and Postdoctoral Studies

in partial fulfillment of the requirements

for the degree of

Master of Science in Electronic Business Technologies

Abstract

Social media users generate a large volume of data, which can contain meaningful and useful information. One such example is information about locations, which may be useful in applications such as marketing and security monitoring. There are two types of locations: location entities mentioned in the text of the messages and the physical locations of users.

Extracting the first type of locations is not trivial because the location entities in the text are often ambiguous. In this thesis, we implement a sequential classification model with conditional random fields followed by a rule-based disambiguation model, we apply them to Twitter messages (tweets) and we show that they handle the ambiguous location entities in our dataset reasonably well.

Only very few users disclose their physical locations; in order to automatically detect their locations, many approaches have been proposed using various types of information, including the tweets posted by the users. It is not easy to infer the original locations from text data, because text tends to be noisy, particularly in social media. Recently, deep learning techniques have been shown to reduce the error rate of many machine learning tasks, due to their ability to learn meaningful representations of input data. We investigate the potential of building a deep-learning architecture to infer the location of Twitter users based merely on their tweets. We find that

stacked denoising auto-encoders are well suited for this task, with results comparable to state-of-the-art models.

Finally, we combine the two models above with a third-party sentiment analysis tool and obtain a intelligent social media monitoring system. We show a demo of the system and that it is able to predict and visualize the locations and sentiments contained in a stream of tweets related to mobile phone brands – a typical real world e-business application.

Acknowledgements

I would like to express my gratitude to Professor Diana Inkpen, my supervisor, for offering me the opportunity to research something I am interested in, for teaching me everything it takes to complete this thesis, and for her helpful advices when I got stuck.

I would like to also thank my parents, who have always been unconditionally supportive of me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The recent rise of social media (SM) and progress in natural language processing (NLP) opened the way for studies on how to mine beneficial information from social media data. Semantic analysis in social media is important for applications such as understanding social networks, natural language interfaces and human behaviour on the Web, e-learning environments, cyber communities and educational or online shared workspaces. The information about a user's **location** and **opinion** is particularly important for applications such as marketing and security monitoring. Therefore, we would like to propose a system that monitors social media activities for E-business applications that can extract, utilize, and present these two types of information, as displayed in Figure 1.

In this work, the word *location* denotes two different types of entities, which will be used throughout the whole thesis: the locations mentioned in text and the physical location of the user. Detecting the first type of location is useful in text

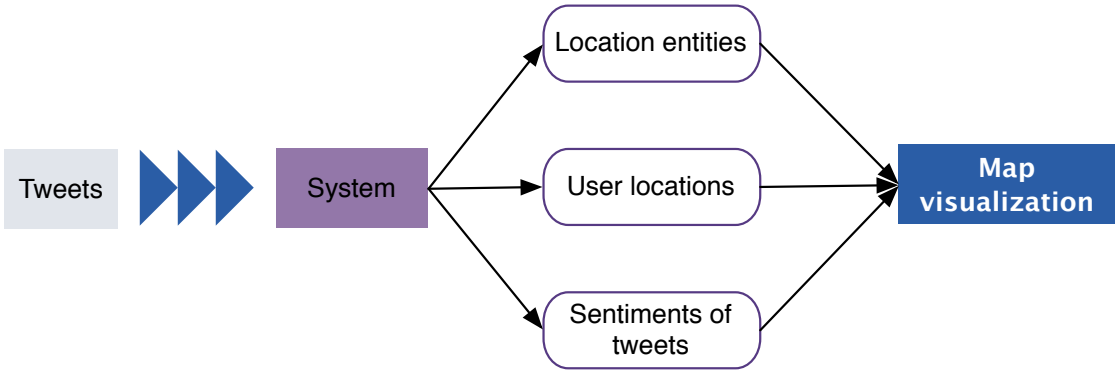


Figure 1: The flow map of the system as a whole.

mining applications where people are interested in finding entities of potential value in text. For example, automatic detection of event locations for individuals or group of individuals with common interests is important for marketing purposes, and also for detecting potential threats to public safety. Therefore, techniques for inferring the location from the communication network infrastructure or from the text content are needed.

In addition, many real-world applications require the knowledge of the actual locations of users. For example, online advertisers would like to target potential buyers in particular regions. There are easy ways to obtain user locations, for example, social media service providers allow users to provide their locations, mostly through GPS locating or by manual specification. However, due primarily to privacy and security concerns, only a small proportion of users actually provide location information. The proportion of users who specify their locations in their profiles is reported to be 14.3% by Abrol et al. (2012); self-reported locations also tend to be unreliable because users can practically type anything they want, such as *In your backyard* or *Wonderland*.

When it comes to per-tweet GPS tagging, only 1.2% of all users use this functionality (Dredze et al., 2013).

1.2 Goals

To construct the e-business monitoring system, we decompose it into three parts.

To begin with, in order to get around the sparsity of location information on social media, we propose one method each to extract the location entities and to infer the locations of users using the text data they generate.

The first method corresponds to a special case of named entity recognition (NER) tasks. Named entities are phrases that contain the names of persons, organizations, locations, times, and quantities (Tjong Kim Sang, 2002).

The second method is related to the field of study called dialectology (Petyt, 1980; Chambers, 1998); its basic idea is that people in different locations use their language differently, in terms of either lexicon, syntax, phonology or morphology. When it comes to written language, people tend to mention things that are local, such as sport teams, landmarks and places nearby. In recent years, computational techniques have been applied to this area by a number of researchers, allowing quantitative and statistical analyses of dialectology.

Another dimension we propose to research in my thesis project is Opinion Mining from social media messages. Social media postings constitute a messy and highly heterogeneous media that nonetheless represent a highly valuable source of information about the attitudes, interests and expectations of citizens and consumers everywhere. This fact has driven recent research and development efforts aimed at managing and interpreting such information for a wide spectrum of commercial applications, among

them: reputation management, branding, marketing design, etc. (Rodríguez-Penagos et al., 2012)

Therefore, this thesis can be seen as consisting of the following subtasks:

1. Detecting location entities mentioned in social media data.
2. Estimating users' locations from the data they generate on social media.
3. Analyzing the sentiment of a given social media text.

By combing the three subtasks, in this thesis, we propose an intelligent system which is capable of analyzing people's sentiment in different regions. We present a demo of this system and show how it can be used in e-business applications.

1.3 Contributions

This thesis has three main contributions, listed below:

1. A novel hybrid approach combing a machine learning model and a rule-based model, which to our knowledge has not been applied to location recognition in social media data before.
2. A successfully trained deep learning model as an approach that infers social media users' locations, which was not applied to this task before; it is also shown to produce results comparable to state-of-the-art models.
3. An annotated dataset that can be made available to and used by other researchers. The dataset is about smartphones and it has every location mentioned in the tweets marked as a city, a state/province, or a country.

4. A visual E-business application demo, which can be of commercial value.

1.4 Outline

The rest of this thesis comes in the following structure: Chapter 2 introduces the background one might need to know in order to read the rest of the thesis; Chapter 3 presents the model that detects location entities from tweets; Chapter 4 describes the model that estimates locations of users; in Chapter 5 we show the demo of the system that combines all modules in this thesis; in the last chapter, we summarize our work and mention some future work.

Chapter 2

Background

In this chapter, we first introduce some general background of natural language processing and machine learning related to this thesis; then, we describe briefly the various machine learning models used in the subsequent chapters, either as the main approaches or as the baseline models.

2.1 Natural Language Processing

Natural language processing (NLP) seeks automatic understanding and generation of human languages with computational techniques (Jurafsky and Martin, 2000). It can be seen as a subfield of artificial intelligence (AI). NLP research generally requires the knowledge of computer science and linguistics. Historically, the NLP methods can be roughly categorized into the following two paradigms:

The **rule-based** paradigm tries to design a set of rules that could explain and predict the phenomena in natural language, including morphology, syntax and semantics. For example, a context-free grammar (Chomsky, 1956) describes how formal language

is generated using production rules. Early natural language understanding systems also used pattern matching and keyword searching. Rules tend to get increasingly complicated as the size of the system grows; complication is often accompanied by the large amount of time and expertise needed to design such rules.

The **stochastic** paradigm builds probabilistic models to process natural language. An important feature of stochastic approaches is the use of language data, also called *corpus*, to estimate the parameters of the probabilistic distribution of the language units (words, morphemes, etc.) people are interested in. The popularity of stochastic approaches is also associated with the development of faster computer hardware and statistical machine learning algorithms (see Section 2.2); since the 1990s, they have been the quite standard approaches in NLP research and application (Jurafsky and Martin, 2000).

There are several fundamental problems in NLP, including but not limited to:

- *Tokenization*, which divides text of natural language into tokens. Tokenization is of great importance to languages whose writing system does not including blank spaces between words such as Chinese and Japanese. However, in languages that do include blank spaces, tokenization is not trivial in the sense that simply tokenizing the text by blank spaces can be problematic sometimes; consider this example in English *the Los Angeles-San Fransisco flight*, where this trivial tokenization method fails badly.

- *Part-of-Speech (POS) tagging*, which assigns each token a tag that corresponds to its part-of-speech tag, such as a verb or a noun. The list of POS tags may vary for different languages; there are different lists of tags even for the same language. In English, the most commonly used POS tags list is the that used

in the Penn Treebank Project (Marcus et al., 1993).

- *Parsing*, which decomposes a sentence into hierarchical structures based on the syntactic role of each component. Parsing is important for natural language understanding. There are several statistical parsers proposed, such as the Lexicalized Probabilistic Context-Free Grammars Parser (Collins, 2003), the Stanford Dependency Parser (De Marneffe et al., 2006).

- *Semantic analysis*, which deals with the "meaning" of natural language by extracting semantics from it.

- *Phonology processing*, which involves automatically understanding and generating the phonology of natural language.

- *Speech recognition*, which aims at recognizing the human-generated speech by processing the acoustic signals.

NLP techniques have a wide range of applications; each of them involves one or more fundamental problems in NLP; a few of them are listed as follows:

- *Machine translation (MT)*, which builds models that automatically translate a language into another. An MT model that works perfectly is of huge value, because human translation is extremely labor-intensive. However, current models are still not satisfying, especially in terms of language pairs that are linguistically remote, such as Chinese and English.

- *Question answering (QA)*, which enables a computer to answers questions, usually asked by a human. This type of system must be able to not only understand but also generate natural language.

- *Sentimental analysis*, which analyzes the sentiments contained in natural language. It is widely applied to marketing and social media analysis.
- *Information extraction*, which extracts relevant information from a collection of text data and presents it in a structured way.

2.2 Machine Learning

Mitchell (1997) defines *machine learning (ML)* in the following way: *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.* Machine learning is widely applied to a number of areas, such as computer vision, NLP, bioinformatics and financial market analysis. To build a practically useful machine learning system, one needs to update its parameters by utilizing some reasonable amount of data; this process is called *training* and these data are called *training data*; to evaluate the performance of the learned model after training is finished, one applies it to a different set of data, known as *test data*. Sometimes, people also use part of the whole dataset as *validation dataset* for model selection, i.e., select from all the models obtained from the training process the one model with the best performance on the validation dataset.

There are several subcategories of machine learning, the commonest among which are unsupervised learning and supervised learning.

Unsupervised Learning

Unsupervised learning tasks have no specific goals, which means the data are not labeled. Unlabeled data can be written as $\{(x^{(1)}, \cdot), (x^{(2)}, \cdot), \ldots, (x^{(m)}, \cdot)\}$, where $x^{(i)}$ is the input of the i-th training example, $\cdot$ indicates the absence of the label, m is the number of training examples. Although unsupervised learning has no explicit purposes, it tries to discover the hidden characteristics of the data. The typical example of unsupervised learning is *clustering*, which aims to divide the whole dataset into several subsets such that one training example is more similar to training examples in the same subset than to those in other subsets.

Directly related to this work is *unsupervised pre-training*, another example of unsupervised learning techniques. Pre-training is done before actually training a supervised learning (see the paragraph below) model, this is why it is called *pre*-training. Its purpose is to learn a representation of the input data; the representation will hopefully help the supervised learning task later on. Typical unsupervised pre-training models are auto-encoders (Bourlard and Kamp, 1988; Hinton and Zemel, 1994) and Restricted Boltzmann machines (RBMs) (Hinton and Sejnowski, 1986; Hinton, 2002). Unsupervised pre-training is the essential component of deep learning architectures, which will be discussed in Section 2.3.3.

Supervised Learning

Supervised learning tasks, by contrast, are often associated with a specific goal. For example, in a object recognition task, the input is an image, and the output is a specific object (e.g., a cat). Therefore, labeled data are necessary to train a supervised learning system; labeled data can be written as $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$,

where $x^{(i)}$ is the input of the i-th training example, $y^{(i)}$ is the label of the i-th training example, m is the number of training examples. The goal of the training process is to find a optimal set of parameters that minimize the loss function defined as the difference between the output labels of the model $\hat{y}$ and the given labels y (also called the *ground truth*), i.e., $\ell(y, \hat{y})$, defined on the training dataset. This definition reduces the training of supervised learning systems to numerical optimization problems. Often, the loss function is defined as a continuously differentiable function; in this case, one can train the corresponding supervised system with *gradient methods* such as gradient descent.[1]

Depending on the nature of the output labels, supervised learning can be divided into classification and regression, detailed as follows:

Classification A classification model predicts the most likely class a given input belongs to. There are finite number of discrete classes. Such a model is called a *classifier*. Specifically, a classifier learns from the data the conditional probability distribution $P(Y|X)$ or the function $Y = f(X)$, where the random variable Y is the target class and the random variable X is the input. An example can be object recognition where X is the input image and Y is the object in the image. There exist many classification algorithms, such as logistic regression, k-nearest neighbours, decision trees, support vector machines, Naive Bayes, artificial neural networks, etc.

By generalizing classification problem to sequential data, we get *sequential classifiers*. Common sequential classification algorithms include Hidden Markov Models (HMMs) (Rabiner, 1989) and conditional random fields (CRFs) (Lafferty et al., 2001).

[1]The details of algorithms used to train machine learning models will not be further discussed since they are beyond the scope of this work. A specific gradient method used in this work will be, however, discussed in Section 4.3.6.

Sequential classifiers are widely applied to information extraction, NLP and bioinformatics – where the data are sequential by nature, such as a sentence or a DNA sequence. Below is an example of a *named entity recognition* (NER) system:
Input: Tim Cook of Apple Inc. gives a talk in San Francisco.
Output: Tim/Person Cook/Person of/O Apple/Organization Inc./Organization gives/O a/O talk/O in/O San/Location Francisco/Location.

Regression A regression model produces a numerical output given the input. It learns from the data the function $Y = f(X)$ where the random variable Y is the target output value and the random variable X is the input. Regression models can be linear or non-linear; if there are more than one output, this type of models are called multivariate regression.

2.3 Machine Learning Models Used in this Thesis

2.3.1 Hidden Markov Models and Conditional Random Fields

Section 2.2 lists two common sequential classifiers: HMM and CRF. This section discusses the difference and the relationship between these two algorithms and explains why we choose CRF for our work in Chapter 3.

Generative Models and Discriminative Models

For a classification task, let Y be the target output class, X be the input. If a model learns the joint probability distribution $P(Y, X)$, this model is called a *generative* model; if a model learns the conditional probability distribution $P(Y|X)$, this model

is called a *discriminative* model. The inference function of a generative model can be defined as:

$$y = \arg\max_{y} P(y, x) \tag{1}$$

Similarly, the inference function of a discriminative model is:

$$y = \arg\max_{y} P(y|x) \tag{2}$$

The advantages of generative models include faster convergence in terms of learning, but it requires modelling the distribution of the input features $P(X)$. It is common that features are interdependent of each other, so modelling $P(X)$ is difficult and intractable (Sutton and McCallum, 2006). One way to solve this problem is to make the assumption that features are independent, then we can easily get $P(X)$ by simply computing:

$$P(X) = \prod_{i} P(X_i) \tag{3}$$

where X_i is the i-th feature of the input. It is worth noting that this independence assumption is a strong one which is likely to harm the expressive power and robustness of the model.

A discriminative model, however, does not have to worry about the interdependence of the features because it does not have to model $P(X)$; therefore the correlation between features is retained. For this reason, discriminative models usually have better classification performance than generative models. Besides, in a classification task, $P(X)$ is actually not required, which makes it even more reasonable to use a discriminative model if we merely care about the classification itself.

We described generative and discriminative models because, as we will later show,

HMM and CRF are such models respectively in terms of sequential classification.

Hidden Markov Model

In a sequential tagging problem, an HMM estimates the joint probability of the hidden state X and the observation Y at time $1 \dots n$:

$$P(y, x) = P(x) \times P(y|x) = P(x_1, x_2, \dots, x_n) \times P(y_1, y_2, \dots, y_n|x) \tag{4}$$

according to the chain rule, this is equal to:

$$\begin{aligned} P(y, x) =& P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \dots P(x_n|x_1, x_2, \dots, x_{n-1}) \times \\ & P(y_1|x)P(y_2|y_1, x)P(y_3|y_1, y_2, x) \dots P(y_n|, y_1, y_2, \dots, y_{n-1}, x) \end{aligned} \tag{5}$$

HMM makes two assumptions; first, the hidden state at time t only depends on the hidden state at its previous time:

$$P(x_t|x_1, x_2, \dots, x_{t-1}) = P(x_t|x_{t-1}) \tag{6}$$

and second, the observation at time t only depends on the hidden state at this time:

$$P(y_t|, y_1, y_2, \dots, y_{t-1}, x) = P(y_t|x_t) \tag{7}$$

therefore, Equation (5) can be simplified to:

$$P(y, x) = \prod_{t=1}^{n} P(x_t|x_{t-1})P(y_t|x_t) \tag{8}$$

where we specifically dictate that $P(x_1|x_0) = P(x_1)$, the initial probability of a hidden state.

Conditional Random Fields

A CRF is a *undirected graphical model*. In a CRF, or more specifically, a linear chain CRF, if we denote the input variables by X and the output labels Y, the conditional probability distribution $P(Y|X)$ obeys the *Markov property*:

$$P(y_i|y_1, y_2, \ldots, y_{i-1}, y_{i+1}, \ldots, y_n, x) = P(y_i, y_{i-1}, y_{i+1}, x) \tag{9}$$

Given some specific input variables x, the conditional probability of some output label y is:

$$P(y|x) = \frac{1}{Z(x)} exp(\sum_{i,k} \lambda_k t_k(y_{i-1}, y_i, x, i) + \sum_{i,l} \mu_l s_l(y_i, x, i)) \tag{10}$$

where $Z(x) = \sum_y exp(\sum_{i,k} \lambda_k t_k(y_{i-1}, y_i, x, i) + \sum_{i,l} \mu_l s_l(y_i, x, i))$ is the *normalizing constant*, t_k and s_l are *feature functions* that map input tokens and labels into features, λ_k and μ_l are the corresponding weights.

Generally speaking, a CRF is a better classification model than an HMM because it does not make the strong assumptions. It has been shown to work better than an HMM on named entity recognition and information extraction tasks (McCallum and Li, 2003; Razavi et al., 2014).

2.3.2 Naive Bayes Classifiers and Support Vector Machines

We hereby introduce Naive Bayes classifiers and support vector machines (SVMs), as they will be implemented as baseline models later in Chapter 4.

Naive Bayes Classifier

A Naive Bayes classifier is a supervised classification model. It makes the assumption that the input features are independent of each other conditioned on the class, i.e.:

$$\begin{aligned} P(X = x|Y = y) &= P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n|Y = y) \\ &= \prod_{i=1}^{n} P(X_i = x_i|Y = y) \end{aligned} \tag{11}$$

This assumption makes parameterizing the model much easier, since estimating $P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n|Y = y)$ is more difficult than estimating $P(X_i = x_i|Y = y)$. In the next step, given a certain input x, the conditional probability of the kth output class y_k is computed according to Bayes' theorem:

$$P(Y = y_k|X = x) = \frac{P(X = x|Y = y_k)P(Y = y_k)}{\sum_k P(X = x|Y = y_k)P(Y = y_k)} \tag{12}$$

Because of Equation (11), we can rewrite Equation (12) as:

$$P(Y = y_k|X = x) = \frac{P(Y = y_k)\prod_{i=1}^{n} P(X_i = x_i|Y = y_k)}{\sum_k P(Y = y_k)\prod_{i=1}^{n} P(X_i = x_i|Y = y_k)} \tag{13}$$

We then return the class that maximizes this probability as the model's prediction, i.e.:

$$\hat{y} = \arg\max_{y_k} P(Y = y_k|X = x) \tag{14}$$

Naive Bayes classifiers, despite their simplicity, are known for performing well on text data and on tasks like document classification and sentiment classification (Lewis, 1998; Joachims, 1998; Pang et al., 2002). We denote this model as *baseline-Naive*

Bayes.

Support Vector Machines

Like Naive Bayes classifiers, support vector machines (SVMs) (Cortes and Vapnik, 1995; Boser et al., 1992) are also classification models. Introduction of complex machine learning models like SVMs is beyond the scope of this thesis, so in this section we only briefly describe the theoretical foundations of SVM without going into too many details.

Prediction Function SVM is essentially a binary classification (i.e., there are exactly 2 possible output classes) model. The parameters of the SVM $\theta = (W, b)$ correspond to a hyperplane in the feature space. Given an input x, the prediction is made by the following function:

$$\hat{y} = f(x) = sign(W * x + b) \tag{15}$$

where $sign(x)$ is the *signum* function defined as:

$$sign(x) = \begin{cases} -1 & \text{if } x < 0, \\ 1 & \text{if } x \geq 0. \end{cases} \tag{16}$$

Training Given a training example $(x^{(i)}, y^{(i)}), y^{(i)} \in \{-1, 1\}$, we define the *functional margin* as:

$$\hat{\gamma_i} = y_i(W * x_i + b) \tag{17}$$

We also define the *minimal functional margin*, which is the functional margin of the training example in the whole training dataset that minimizes $\hat{\gamma}_i$:

$$\hat{\gamma} = \min_{i=1,\ldots,m} \hat{\gamma}_i \tag{18}$$

$\hat{\gamma}$ measures how confident the model is when it makes predictions. However, we could increase $\hat{\gamma}$ by simply proportionately increasing W and b, whereas the model's predictive power does not practically get better. This motivates us to normalize the weights W so that $||W|| = 1$, then the functional margin becomes *geometric margin*, mathematically:

$$\gamma_i = y_i(\frac{W}{||W||} * x_i + \frac{b}{||W||}) \tag{19}$$

Similarly, we define the *minimal geometric margin*:

$$\gamma = \min_{i=1,\ldots,m} \gamma_i \tag{20}$$

Now the next thing to do is simply to find the optimal parameters $\theta^* = (W^*, b^*)$ that maximize γ and to make predictions using Equation (15).

Kernel Functions Note that the training method above works only on *linearly separable* data; if linear separability is not the case, *kernel functions* are typically used to "project" original data into a new feature space where they become linearly separable. Common kernel functions include the polynomial kernel function and the Gaussian (RBF) kernel function.

SVMs generally outperform Naive Bayes classifiers in most machine learning tasks,

since they do not make the strong assumption in Equation (11) as Naive Bayes classifiers do. We denote and later refer to this model as *baseline-SVM*.

2.3.3 Neural Networks and Auto-encoders

In this section, we present the models that shall appear in Chapter 4: artificial neural networks and auto-encoders.

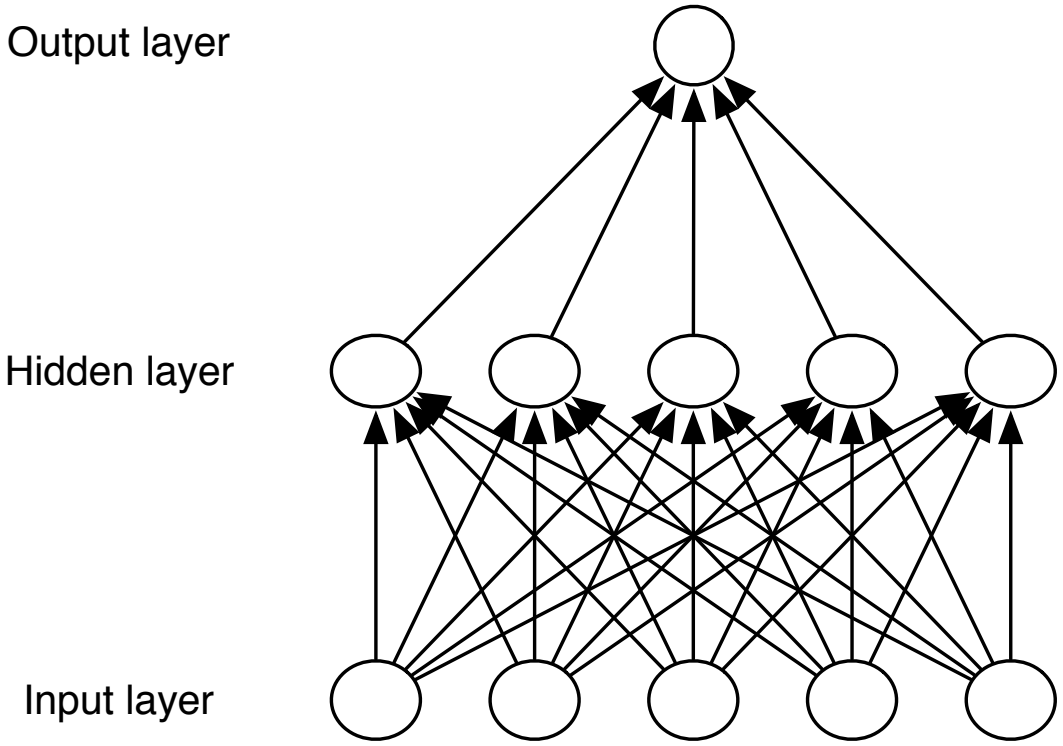


Figure 2: A multilayer perceptron with one hidden layer.

Feedforward Artificial Neural Networks

In this section we present the models that shall appear in Chapter 4, namely artificial neural networks. Feedforward artificial neural networks deal with supervised learning

problems. They map input feature vectors into corresponding outputs. In this work, we focus on *perceptrons* (Rosenblatt, 1961), which are a special class of feedforward neural networks (Haykin, 1994).

A feedforward neural network usually has an input layer and an output layer. If the input layer is directly connected to the output layer, such a model is called a *single-layer perceptron*. A more powerful model has several layers between the input layer and the output layer; these intermediate layers are called *hidden layers*; this type of model is known as a *multi-layer perceptron* (MLP). In a perceptron, neurons are interconnected, i.e., each neuron is connected to all neurons in the subsequent layer. Neurons are also associated with activation functions, which transform the output of each neuron; the transformed outputs are the inputs of the subsequent layer. Typical choices of activation functions include the identity function, defined as $y = x$; the hyperbolic tangent, defined as $y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and the logistic sigmoid, defined as $y = \frac{1}{1+e^{-x}}$. Figure 2 shows the architecture of a simple MLP.

To train a MLP, the most commonly used technique is *back-propagation* (Rumelhart et al., 1985). Specifically, the errors in the output layer are back-propagated to preceding layers and are used to update the weights of each layer.

Deep Neural Networks

An artificial neural network (ANN) with multiple hidden layers, also called a Deep Neural Network (DNN), mimics the deep architecture in the brain and it is believed to perform better than shallow architectures such as logistic regression models and ANNs without hidden units. The effective training of DNNs is, however, not achieved until the work of Hinton et al. (2006) and Bengio and Lamblin (2007). In both

cases, a procedure called *unsupervised pre-training* is carried out before the final supervised fine-tuning. The pre-training significantly decreases error rates of Deep Neural Networks on a number of ML tasks such as object recognition and speech recognition.

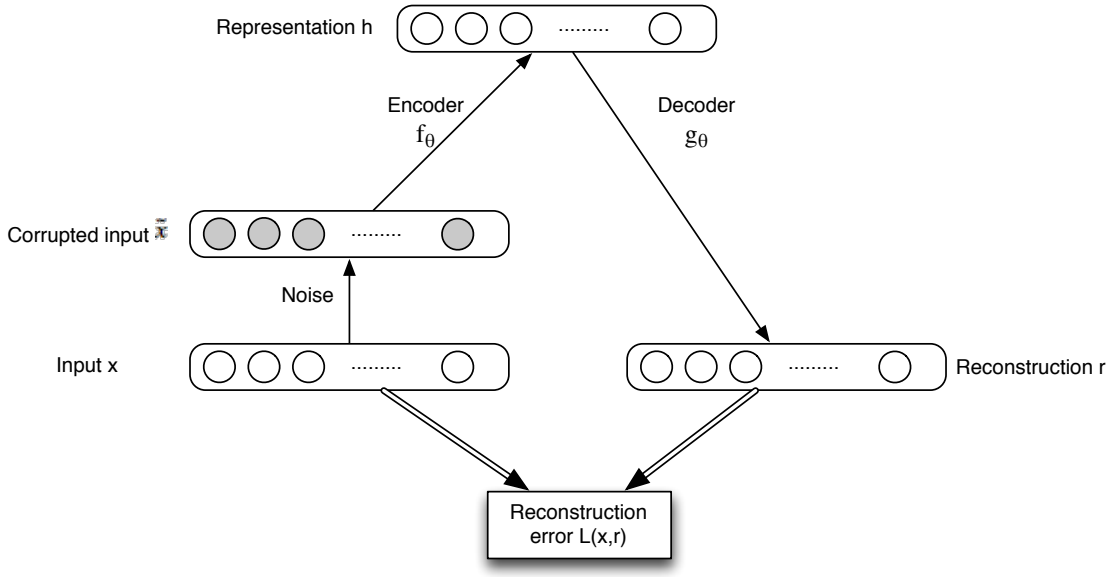


Figure 3: A typical denoising auto-encoder.

Deep Neural Networks with Auto-encoders

An auto-encoder framework includes a feature-extraction function, or encoder, f_θ that takes a feature vector x as input and produces a new feature vector h, which is often called a representation.

$$h = f_\theta(x) = s_f(Wx + b) \tag{21}$$

where s_f is the encoder activation function (usually logistic sigmoid, hyperbolic tangent or identity function), W is the encoder weight matrix and b is the encoder bias.

A reconstruction function, or decoder, g_θ then computes a reconstruction r of the previously learned representation h.

$$r = g_\theta(h) = s_g(W'x + d) \tag{22}$$

where s_g is the decoder activation function, W' is the decoder weight matrix (a common choice is to constrain W' so that $W' = W^T$) and d is the decoder bias.

Another essential component of an auto-encoder framework is a reconstruction error $\ell(x, r)$, which measures the similarity between the original input x and the reconstruction r. The goal of training an auto-encoder is to find a set of parameters $\theta = \{W, b, W', d\}$ that minimize the reconstruction error (Bengio et al., 2013).

The relationship between auto-encoders and DNNs is straightforward: by stacking a number of auto-encoders, we obtain a neural network with the same number of hidden layers. Because each auto-encoder learns an abstract representation of the input, stacking a number of auto-encoders is likely to generate even more abstract representations, which could improve the performance of a supervised ML task. The step of learning representations from raw input is also named *unsupervised pre-training*. Bengio and Lamblin (2007) implemented a greedy layer-wise pre-training algorithm, which was first introduced in Deep Belief Networks (DBNs), another variant of DNNs (Hinton et al., 2006). Specifically, layers of auto-encoders are first trained in a sequence to minimize their individual reconstruction error; then the whole architecture is perceived as a multi-layer feedforward neural network; the weights of each hidden layer are the encoding weights of the corresponding auto-encoder (W and b, as defined in Equation 21). These weights are further optimized to minimize the error function with respect to the supervised ML task; this step is also called fine-tuning.

A variant of auto-encoders is called denoising auto-encoders (DAs). A denoising auto-encoder is identical to an auto-encoder, except the input x is corrupted by applying statistical noise to it (usually masking noise or Gaussian noise); the corrupted input, denoted by $\tilde{x}$, is then fed to the encoder.

$$h = f_\theta(\tilde{x}) \tag{23}$$

The corruption forces the denoising auto-encoder to denoise an artificially corrupted input, therefore to learn a more robust representation of the raw input (Bengio et al., 2013). Figure 3 illustrates how a denoising auto-encoder works.

Just like regular auto-encoders, denoising auto-encoders can also be stacked to construct a deep learning architecture. The work by Vincent et al. (2008) proves on several benchmarking datasets that stacked denoising auto-encoders (SDAs) generally perform better than stacked auto-encoders.

Chapter 3

Detecting Location Entities from Text

3.1 Introduction

In this chapter, we focus on identifying and disambiguating the locations mentioned in the text of the messages and in the user location field (as a textual string), not the user's physical location in particular, which is the focus of Chapter 4. Of course, the locations mentioned in the messages could be locations of events that take place anywhere in the world (e.g., Olympic Games), or the locations near the user's home, or sometimes users could be travelling and describing the locations that they are visiting.

One must realize that this is not a trivial task; we cannot simply apply keyword matching due to two levels of ambiguities defined by Amitay et al. (2004): *geo/non-geo ambiguity* and *geo/geo ambiguity*. Geo/non-geo ambiguities happen when a location entity is also a proper name (e.g., *Roberta* is a given name and the name of a city

in *Georgia, United States*) or has a non-geographic meaning (e.g., *None* is a city in Italy in addition to the word *none* when lower case is ignored or when it appears at the beginning of a sentence). A geo/geo ambiguity occurs when several distinct places have the same name, as in *London, UK; London, ON, Canada; London, OH, USA; London, TX, USA; London, CA, USA*, and a few more in the USA and other countries. Another example is the country name *China* being the name of cities in the United States and in Mexico.

As a consequence of the ambiguities, an intelligent system smarter than simple keyword matching is required. Specifically, we propose to address the geo/non-geo ambiguities by defining a named entity recognition task which focuses on locations and ignores other types of named entities. To deal with geo/geo ambiguities, we implement several heuristic disambiguation rules which are shown to perform reasonably well.

The outline of the remainder of this chapter is detailed as follows: Section 3.2 lists previous work related to ours; Section 3.3 describes how we collect and annotate the dataset; Section 3.4 details the methods we adopt to extract location entities; in Section 3.5, we present the experimental results; and finally, in Section 3.6 we summarize the work in this chapter.

3.2 Related Work

Before the social media era, researchers tried to detect locations from online contents such as news and blogs. Li et al. (2002) named this type of work *location normalization*. Their approach comes in the following steps:

1. Extract locations from the text using a Maximum-entropy Markov model (MEMM). MEMM is a sequential classifier like HMM and CRF.

2. Use local context. Several patterns are designed, for example: location+comma+NP (headed by city) e.g., *Chicago, an old city.*

3. Use the global context. Find the all other locations in the same document; for each potential candidate, compute a score that measures the likelihood of this candidate with a maximum spanning tree approach; the candidate with the best score is returned.

4. If none of the steps above can lead to a decision, the default sense, derived from several similarity feature, is returned.

Their system is reported to have an overall precision of 93.8% on several news datasets.

Amitay et al. (2004) do not use a supervised learning approach; instead they implement a score based approach to address both geo/non-geo and geo/geo ambiguities. Specifically, lexical evidences supporting the likelihood of a candidate location will increase its score. When applied to Internet contents, their algorithm is reported to have an accuracy of 81.7%.

Users on social media represented by Twitter are generating substantial amount of data, which contain valuable information, including locations. However, social media text, especially Twitter messages (called tweets), is very different from traditional text, since it usually contains misspellings, slangs and is short in terms of length. Consequently, detecting locations from social media is more challenging. Paradesi (2011) trains a simple log-near model with just 2 features for each type of ambiguity to disambiguate locations in Twitter messages; the model achieved a precision of 15.8%. Bouillot et al. (2012) look at how to exploit information about location mentions in texts of French tweets related to medical issues. The locations were detected by gazetteer lookup and pattern matching to map them to physical locations

using a hierarchy of countries, states/provinces and cities. In case of ambiguous names, they did not fully disambiguate, but relied on users' time zones. Gelernter and Mushegian (2011) explore how to identify location mentions in tweets about disasters, they applied the Stanford NER software to this task and made several suggestions to improve the results.

3.3 Dataset

Annotated data are required in order to train our supervised learning system. Our work is a special case of the Named Entity Recognition task, with text being tweets and target Named Entities being locations. To our knowledge, a corresponding corpus does not yet exist.

3.3.1 Data Collection

We used the Twitter API[1] to collect our own dataset. Our search queries were limited to six major cell phone brands, namely iPhone, Android, Blackberry, Windows Phone, HTC and Samsung. Twitter API allows its users to filter tweets based on their languages, geographic origins, the time they are sent, etc. We utilized such functionality to collect only tweets written in English. Their origins, however, were not constrained, i.e., we collected tweets from all over the world.

We ran the crawler from June 2013 to November 2013, and eventually collected a total of over 20 million tweets.

[1] https://dev.twitter.com

3.3.2 Manual Annotation

The amount of data we collected is overwhelming for manual annotation, which is essential for any supervised learning task for location detection. We therefore randomly selected 1000 tweets from each subset (corresponding to each cellphone brand) of the data, and obtained 6000 tweets for the manual annotation (more data would have taken too long to annotate).

We have defined annotation guidelines to facilitate the manual annotation task. Mani et al. (2008) defined spatialML: an annotation schema for marking up references to places in natural language. Our annotation model is a sub-model of spatialML and it is described as follows.

The location entities are annotated with location tags. A location tag has an attribute *locType* which indicates the type of a particular location; the value of it is either *country, SP* (indicating that this location is a state of the USA or a province of Canada[2]) or *city*. If the location is a state or province outside North America, or it cannot be categorized into any location type, the value of this attribute is set to *other*.

The following examples show the annotation scheme used for our corpus:

1. Mon Jun 24 23:52:31 +0000 2013

   <**location locType='city'**>Seguin </**location**><**location locType='SP'**>Tx </**location**>

   RT @himawari0127i: #RETWEET#TEAMFAIRYROSE #TMW #TFBJP #500aday #ANDROID #JP #FF #Yes #No #RT #ipadgames #TAF #NEW

[2]We do not include states or provinces elsewhere because 1. not all countries have states or provinces; 2. our application focuses on North America

#TRU #TLA #THF 51

2. Wed Sep 11 09:09:21 +0000 2013

   Worldwide

   no_location

   BlackBerry lays off dozens of < **location locType='country'**>US <**/location**>sales staff: WSJ - Reuters Canada: MobileSyrup.comBlackBerry lays off dozens ... http://t.co/GZcO3H3wro

Next, the process of manual annotation is described as follows:

Gazetteer Matching

A gazetteer is a list of proper names such as people, organizations, and locations. Since we are interested only in locations, we only require a gazetteer of locations. We obtained such a gazetteer from GeoNames[3], which includes additional information such as populations and higher level administrative districts of each location. We also made several modifications, such as the removal of cities with populations smaller than 1000 (because otherwise the size of the gazetteer would be very large, and there are usually very few tweets in the low-populated areas); we also allowed the matching of alternative names for locations. For instance, ATL, which is an alternative name for Atlanta, will be matched as a city. Table 1 lists two examples of locations from the modified gazetteer.

We then used GATE's gazetteer matching module (Cunningham, 2002) to associate each entry in our data with all potential locations it refers to, if any. Note that,

[3]http://www.geonames.org

Location ID	Name	Type	Higher level admin. dist.	Population	Latitude and longitude	Alternative names
136616	Los Angeles	city	California, United States	3792621	34.05223, -118.24368	Angelopolis, L.A.,LA,LAX, Lok-chham-ki,Los-Andzeles,Los-Andzheles,Los-Angeleso,Los-Anjeles,Los-Anzheles, Losandzelosa, los-anjelesi, loseuaenjelleseu, rosanzerusu
13289	Toronto	city	Ontario, Canada	4612191	43.70011, -79.4163	Torontas,Toronto, Torontu, Toron-tum, Toront, YTO,roranro, taronto,teareantea, tolonto,toramto, toranto,toronto, twrntw,twrwntw

Table 1: Examples of entries in the gazetteer. An location ID uniquely corresponds to one and only one location.

in this step, the only information we need from the gazetteer is the name and the type of each location. The details of how the potential location entities are matched are described in Section 3.5.3.

Manual Filtering

It is important to note that the first step is merely a coarse matching mechanism without any effort made to disambiguate candidate locations. E.g., the word *Georgia* would be matched to both the state of Georgia and the country in Europe. Another example is that *on* inside a shortened URL *http://t.co/fdJonDf* would be matched to the province of Ontario. In the next phase, we arranged for two annotators, who are graduate students with adequate knowledge of geography, to go through every entry matched to at least one of locations in the gazetteer list. The annotators are required to identify, first, whether this entry is a location; and second, what type of location this entry is. In addition, they are also asked to mark all entities that are location entities, but not detected by GATE due to misspelling, all capital letters, all small letters, or other causes.

We split the dataset so that each annotator was assigned one fraction. In addition, both annotators annotated one subset of the data containing 1000 tweets, which corresponds to the search query of Android phone, in order to compute an inter-annotator agreement, which turned out to be 88%. The agreement by chance is very low, since any span of text could be marked, therefore the kappa coefficient that compensates for chance agreement is close to 0.88. It is worth noting that the matching between the manual annotations and those of the initial GATE gazetteer matcher in the previous step was 0.56 and 0.47, respectively for each annotator (these

values are the F-measure between the initial GATE annotation and the annotators' final annotations). This shows the amount of manual work our annotators had to do in order to correct the labels, to add missing locations, or to remove false detections.

Annotation of True Locations

Up to this point, we have identified locations and their types, i.e., geo/non-geo ambiguities are resolved, but geo/geo ambiguities still exist. For example, we have annotated the token *Toronto* as a city, but it is not clear wether it refers to *Toronto, Ontario, Canada* or *Toronto, Ohio, USA*. Therefore we randomly choose 300 tweets from the dataset of 6000 tweets and further manually annotated the locations detected in these 300 tweets with their actual location. The actual location is denoted by a numerical ID (as displayed in Table 1) as the value of an attribute named *trueLoc* within the XML tag. For example, if we annotate actual locations in the previous example, we would get:

Mon Jun 24 23:52:31 +0000 2013

<location locType='city', trueLoc='22321'>Seguin **</location><location locType='SP', trueLoc='12'>**Tx **</location>**

RT @himawari0127i: #RETWEET#TEAMFAIRYROSE #TMW #TFBJP #500aday #ANDROID #JP #FF #Yes #No #RT #ipadgames #TAF #NEW #TRU #TLA #THF 51

3.4 Methods

3.4.1 Location Detection

We looked into methods designed for sequential data, because the nature of our problem is sequential. The different parts of a location such as country, state/province and city in a tweet are related and often given in a sequential order, so it seems appropriate to use sequential learning methods to automatically learn the relations between these parts of locations.

The main difference between regular classifiers such as decision trees and sequencial classifiers such as CRF is that the former work on individual instances and are trained to classify instances into two or more classes, whereas the latter are trained based on a sequence of objects. Hence, sequence classifiers need to find the spans (i.e., portions of the sequence with variable length) to be matched to any of the training annotations. In Section 2.3.1, we discussed two typical sequential classifiers: HMM and CRF and the advantages of CRF over HMM in terms of CRF's higher accuracy in named entity recognition tasks. We therefore decided to use CRF as our main machine learning algorithm.

Designing Features

Features that are good representations of the data are important to the performance of a machine learning task. The features that we design for detecting locations are listed below:

- Bag-of-Words: To start with, we defined a sparse binary feature vector to represent each training case, i.e., each token in a sequence of tokens; all values of

the feature vector are equal to 0 except one value corresponding to this token is set to 1. This naive feature representation is a common starting point in machine learning tasks for NLP applications, often referred to as *Bag-of-Words* or unigram features. We will use *Bag-of-Words Features* or *BOW features* to denote them, and the performance of the classifier that uses these features will be considered as the baseline in this work. This is a high baseline, because the words tend to be strong features in many text classification tasks. A lower baseline could be the initial location detection by lookup in the GATE gazetteers (F-measures are reported in Section 3.3.2).

This set of features can be denoted by $X_{BOW} \in \mathbb{B}^D$, where D is the size of the vocabulary. A BOW feature vector is a D-dimensional binary vector.

- Part-of-Speech: The intuition for incorporating Part-of-Speech tags in a location detection task is straightforward: a location can only be a noun or a proper noun. Similarly, we define a binary feature vector, where the value of each element indicates the activation of the corresponding POS tag (1 means active, 0 means inactive; exactly one element is active). We later on denote these features by *POS features*.

  This set of features can be denoted by $X_{POS} \in \mathbb{B}^P$, where P is the number of all POS tags. A POS feature vector is a P-dimensional binary vector.

- Left/right: Another possible indicator of whether a token is a location is its adjacent tokens and POS tags. The intuitive justification for this features is that locations in text tend to have other locations as neighbours, i.e., *Los Angeles, California, USA*; and that locations in text tend to follow prepositions, as in

the phrases *live in Chicago*, *University of Toronto*. To make use of information like that, we defined another set of features that represent the tokens on the left and right side of the target token and their corresponding POS tags. These features are similar to Bag-of-Words features, but instead of representing the token itself they represent the adjacent tokens. These features are later on denoted by *Window features* or *WIN features*.

This set of features can be denoted by $X_{WIN} \in \mathbb{B}^D$, where D is the size of the vocabulary. A WIN feature vector is a D-dimensional binary vector.

- Gazetteer: Finally, a token that appears in the gazetteer is not necessarily a location; by comparison, a token that is truly a location must match one of the entries in the gazetteer. Thus, we define another binary feature which indicates whether a token is in the gazetteer. This feature is denoted by Gazetteer feature or GAZ feature in the succeeding sections.

  This set of features can be denoted by $X_{GAZ} \in \mathbb{B}^1$. A GAZ feature vector is a 1-dimensional binary vector.

In order to obtain BOW features and POS features, we preprocessed the training data to tokenize and POS tag all the tweets. This step was done using Carnegie Mellon University's Twitter NLP and Part-of-Speech Tagging tool (Owoputi et al., 2013).

For experimental purposes, we would like to find out the impact of each set of features has on the performance of the model. Therefore, we test different combinations of features and compare the accuracies of resulting models. To combine features, we simply concatenate the feature vectors, e.g., $X_{BOW+POS} = X_{BOW}||X_{POS}$, as illustrated in Figure 4.

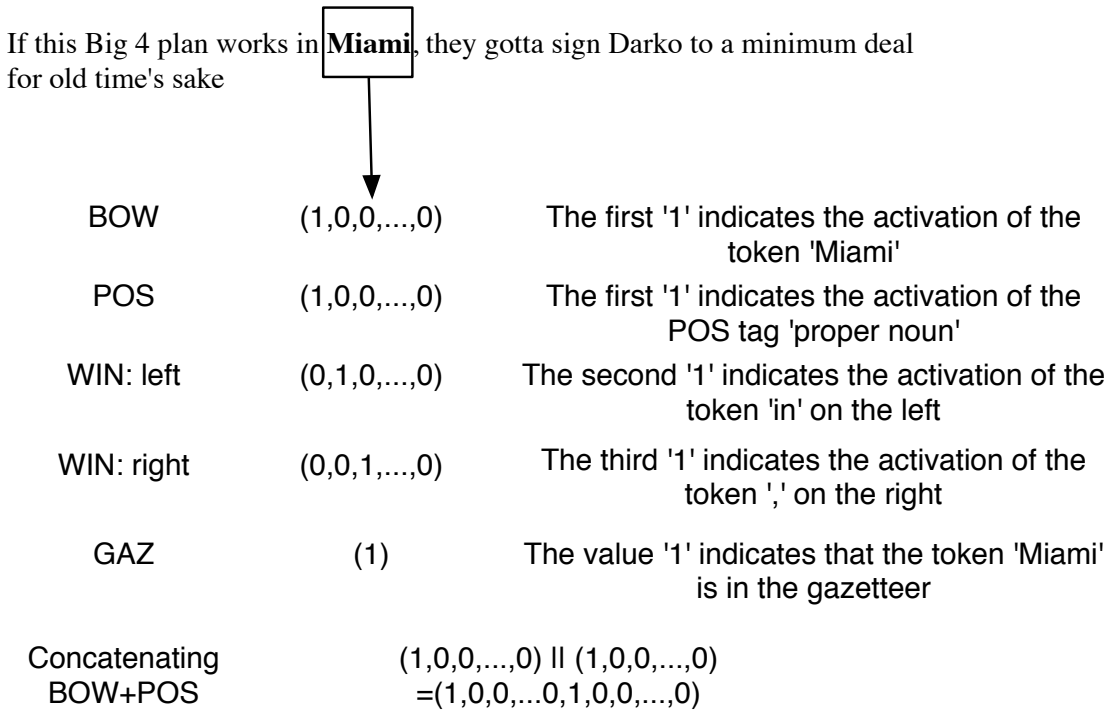


Figure 4: Example of different feature vectors and how to concatenate them. On the top there is a tweet and we extract features for the token "Miami".

Feature Extraction

Feature extraction transform text data into feature vectors and labels. A pair of feature vector and label is called a training example. In our case, a tweet is transformed into a sequence of training examples, i.e.:

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

where x_i is the extracted feature vector of the i-th token, y_i is its label (indicating whether this token is a location), n is the lengthen of this tweet (number of tokens).

Training the CRF Classifiers

After the features are extracted from the data, the training examples are used to train the model. For each type of location (i.e., city, SP, country) a sequential classifier is trained respectively. The results of testing the models will be report in Section 3.5.

3.4.2 Location Disambiguation

In the previous subsection, we have identified the locations in Twitter messages and their types; however, the information about these locations is still ambiguous, because of location names shared by different locations in the world. In this section, we describe the heuristics that we use to identify the unique actual location referred to by an ambiguous location name. The disambiguation process is divided into 5 steps, as follows:

1. **Retrieving candidates**. A list of locations whose names are matched by the location name we intend to disambiguate are selected from the gazetteer.

We call these locations candidates. After step 1, if no candidates are found, disambiguation is terminated; otherwise we continue to step 2.

2. **Type filtering**. The actual location's type must agree with the type that is tagged in the previous step where we apply the location detection model; therefore, we remove any candidates whose types differ from the tagged type from the list of candidates. E.g., if the location we wish to disambiguate is *Ontario* tagged as a city, then *Ontario* as a province of Canada is removed from the list of candidates, because its type is *SP* which differs from our target type. After step 2, if no candidates remain in the list, disambiguation is terminated; if there is only one candidate left, this location is returned as the actual location; otherwise we continue to step 3.

3. **Checking adjacent locations**. It is common for users to put related locations together in a hierarchical way, e.g., Los Angeles, California, USA. In this step, we check adjacent tokens of the target location name; if a candidate's geographic hierarchy matches any adjacent tokens, this candidate is added to a temporary list. To illustrate, if the target is *Ottawa* and it has an adjacent token *Ontario*, the candidate *Ottawa, ON, Canada* is added to the temporary list, while other candidates such as *Ottawa, OH, U.S.* is not. After step 3, if the temporary list contains only one candidate, this candidate is returned as the actual location. Otherwise we continue to step 4 with the list of candidates received in step 3.

4. **Checking global context**. Locations mentioned in a document are geographically correlated (Li et al., 2002). In this step, we first look for other tokens tagged as a location in the Twitter message; if none is found, we continue to

step 5; otherwise, we disambiguate these context locations. Note that we cannot include step 4 in the disambiguation, because it causes an infinite loop. After we obtain a list of locations from the context, we calculate the sum of their distances to a candidate location and return the candidate with minimal sum of distances.

5. **Default sense**. If none of the previous steps can decide a unique location, we return the candidate with largest population (based on the assumption that most tweets talk about large urban areas).

3.5 Experiments and Results

3.5.1 Metrics

We compute the precision, recall and F-measure, which are the most common evaluation measures used in most information retrieval tasks. Specifically, the prediction of the model can have four different outcomes: true positive (TP), false positive (FP), true negative (TN) and false negative (FN), as described in Table 2.

	Ground truth	
Model	target	¬ target
predicted	TP	FP
¬ predicted	FN	TN

Table 2: Definitions of true positive, false positive, true negative and false negative.

Precision measures how correctly the model makes predictions; it is the proportion

of all positive predictions that are actually positive, computed by:

$$precision = \frac{TP}{TP + FP} \tag{24}$$

Recall measures the model's capability of recognizing positive test example; it is the proportion of all actually positive test example that the model successfully predicts, computed by:

$$recall = \frac{TP}{TP + FN} \tag{25}$$

Once precision and recall are computed, we can therefore calculate the F-measure by:

$$F = \frac{1}{\alpha\frac{1}{P} + (1 - \alpha)\frac{1}{R}} \tag{26}$$

where P is the precision and R is the recall; α is the weighting coefficient. In this thesis, we shall use a conventional value of α, which is 0.5; one can interpret it as equally weighting precision and recall. By setting $\alpha = 0.5$, we can rewrite Equation (26) as:

$$F = \frac{2PR}{P + R} \tag{27}$$

This equation shows a nice property of the F-measure from a mathematical perspective: the only way to obtain a high F-measure is by achieving reasonably high values with respect to both precision and recall. By contrast, for example, a system that predicts every test example to be positive has a recall of 1, but a very low precision of, say, 0.001; then its F-measure would be merely $F = \frac{2\times0.001\times1}{0.001+1} \approx 0.002$, showing how poorly this naive system performs.

We report precision, recall and F-measure at both the token and the span level,

to evaluate the overall performance of the trained classifiers. A token is a unit of tokenized text, usually a word; a span is a sequence of adjacent tokens. At the token level, we calculate the listed measures based on the number of tokens that belong to location spans (city, SP, or country) that are annotated. In other words, if a token belongs to the span and is tagged by the classifier the same as the location label, we count it as a true positive; otherwise, we count it as false positive; the same strategy is taken for the negative class. At the span level, we evaluate our method based on the whole span; if our classifiers correctly detects the start point, the end point and the length of the span, this will be counted as a true positive; however, if even one of the three factors was not exact, we count it as a false positive. It is clear that evaluation at the span level is more strict.

The metrics we use to evaluate the location disambiguation module is simply the accuracy of its predictions.

3.5.2 Experiments

In our experiments, one classifier is trained and tested for each of the location labels city, SP, and country. For the learning process, we need to separate training and testing sets. For this purpose, we can either randomly split the annotated data into training and testing subsets (e.g., 70% and 30%) or apply the *n-fold cross-validation* procedure. In a n-fold cross-validation the classifier is trained on n-1 folds of the data and tested on the remaining fold, then this is repeated n times for different splits, and the results are averaged over the n experiments. We report results for 10-fold cross-validation, because a conventional choice for n is 10. In addition, because the data collection took several months, it's likely that we have both new and old tweets

in the dataset; therefore we performed a random permutation before splitting the dataset for training and testing.

We would like to find out the contribution of each set of features in Section 3.4.1 to the performance of the model. To achieve a comprehensive comparison, we tested all possible combinations of features plus the BOW features.

We also run the location disambiguation algorithm described in Section 3.4.2. In order to evaluate how each step (more specifically, step 3 and 4, since other steps are mandatory) contributes to the disambiguation accuracy, we also deactivate optional steps and compared the corresponding results.

3.5.3 Implementation Notes

GATE's Gazetteer Matching Module

The GATE gazetteer matching module adopts a simple keyword matching mechanism, which does not require tokenization of the text. We first prepare three files that list the locations from each category; then a file named *lists.def* is required so that GATE would understand the contents of these files. The format of the definition is:

file name:matching type:specific type

for example, the definition:

countries.lst:LOCATION:country

would mark anything that matches an entry from the file *countries.lst* a *LOCATION*, with its specific type of location being *country*. The contents of all the required files are shown in Table 3.

File name	Content
countries.lst	List of all countries; one entry per line.
sp.lst	List of all states and provinces; one entry per line.
cities.lst	List of all cities; one entry per line.
lists.def	countries.lst:LOCATION:country sp.lst:LOCATION:SP cities.lst:LOCATION:city

Table 3: The contents of gazetteer files as per the requirements of GATE.

CRF Classifiers

We used an NLP package called MinorThird (Cohen, 2004) that supports different versions of sequence-based classification algorithms such as: CRF, HMM. It is an open-source tool which is available for both commercial and research purposes. It is also a combination of tools for annotating and visualizing text with a wide range of learning methods.

More importantly, the MinorThird package provides a CRF module (Sarawagi and Cohen, 2004) easy to use; the loss function is the log-likelihood and the learning algorithm is the gradient ascent. The loss function is convex and the learning algorithm converges fast, usually within 20 minutes on a computer with Intel Core i5 CPU and 4G RAM, when using all the features and the whole dataset (6000 tweets).

The location disambiguation module is implemented as a Java class and wrapped in the same package with the CRF classifiers.

3.5.4 Results

The results are listed in the following tables. Table 4 shows the results for countries, Table 5 for states/provinces and Table 6 for cities. The results of different location disambiguation configurations are displayed in Table 7.

Features	Token			Span			Separate train-test sets	
	P	R	F	P	R	F	Token F	Span F
BOW	0.93	0.83	0.88	0.92	0.82	0.87	0.86	0.84
BOW+POS	0.93	0.84	0.88	0.91	0.83	0.87	0.84	0.85
BOW+GAZ	0.93	0.84	0.88	0.92	0.83	0.87	0.85	0.86
BOW+WIN	0.96	0.82	0.88	0.95	0.82	0.88	0.87	0.88
BOW+POS+ GAZ	0.93	0.84	0.88	0.92	0.83	0.87	0.85	0.86
BOW+WIN+ GAZ	0.95	0.85	0.90	0.95	0.85	0.89	0.90	0.90
BOW+POS+ WIN	0.95	0.82	0.88	0.95	0.82	0.88	0.90	0.90
BOW+POS+ WIN+GAZ	0.95	0.86	0.90	0.95	0.85	0.90	0.92	0.92

Table 4: Performance of the classifiers trained on different features for countries. Column 2 to column 7 show the results from 10-fold cross validation; the last two columns show the results from random split of the dataset where 70% are the train set and 30% are the test set. (Same in Table 5 and Table 6)

3.6 Discussion

The results from Table 4, 5 and 6 show that the task of identifying cities is the most difficult, since the number of countries or states/provinces is by far smaller. In our gazetteer, there are over 140,000 cities, but only 252 countries and 73 states/provinces. A lager number of possible classes generally indicates a larger search space, and consequently a more difficult task. We also observe that the token level F-measure and the span level F-measure are quite similar, likely due to the fact that most location

Features	Token			Span			Separate train-test sets	
	P	R	F	P	R	F	Token F	Span F
BOW	0.90	0.78	0.84	0.89	0.80	0.84	0.80	0.84
BOW+POS	0.90	0.79	0.84	0.89	0.81	0.85	0.82	0.84
BOW+GAZ	0.88	0.81	0.84	0.89	0.82	0.85	0.79	0.80
BOW+WIN	0.93	0.77	0.84	0.93	0.78	0.85	0.80	0.81
BOW+POS+GAZ	0.90	0.80	0.85	0.90	0.82	0.86	0.78	0.82
BOW+WIN+GAZ	0.91	0.79	0.84	0.91	0.79	0.85	0.83	0.84
BOW+POS+WIN	0.92	0.78	0.85	0.92	0.79	0.85	0.80	0.81
BOW+POS+WIN+GAZ	0.91	0.79	0.85	0.91	0.80	0.85	0.84	0.83

Table 5: Performance of the classifiers trained on different features for SP.

Features	Token			Span			Separate train-test sets	
	P	R	F	P	R	F	Token F	Span F
BOW	0.91	0.59	0.71	0.87	0.56	0.68	0.70	0.68
BOW+POS	0.87	0.60	0.71	0.84	0.55	0.66	0.71	0.68
BOW+GAZ	0.84	0.77	0.80	0.81	0.75	0.78	0.78	0.75
BOW+WIN	0.87	0.71	0.78	0.85	0.69	0.76	0.77	0.77
BOW+POS+GAZ	0.85	0.78	0.81	0.82	0.75	0.78	0.79	0.77
BOW+WIN+GAZ	0.91	0.76	0.82	0.89	0.74	0.81	0.82	0.81
BOW+POS+WIN	0.82	0.76	0.79	0.80	0.75	0.77	0.80	0.79
BOW+POS+WIN+GAZ	0.89	0.77	0.83	0.87	0.75	0.81	0.81	0.82

Table 6: Performance of the classifiers trained on different features for cities.

Deactivated steps	Accuracy
None	95.5 %
Adjacent locations	93.7 %
Global context	98.2 %
Adjacent locations + context locations	96.4 %

Table 7: Location disambiguation results.

names contain only one word.

We also include the results when one part of the dataset (70%) is used as training data and the rest (30%) as test data. The results are slightly different to that of 10-fold cross validation and tend to be lower in terms of f-measures, likely because less data are used for training. However, similar trends are observed across feature sets.

By comparing the performance of different combinations of features, we find out that the differences are most significant for the classification of cities, and least significant for the classification of states/provinces, which is consistent with the number of classes for these two types of locations. We also observe that the simplest features, namely BOW features, always produce the worst performance at both token level and span level in all three tasks; on the other hand, the combination of all features produces the best performance in every task, except for the prediction of states/provinces at span level. These results are not surprising.

For each set of features, the results show that POS features improve the overall performance by the smallest margin; occasionally, POS features even lower the F-measure. These facts are contrary to the intuition that part-of-speech tags are relevant to the detection of locations. In addition, WIN features improve the performance of the models, and the improvements are most significant when detecting cities. Therefore, we can conclude that incorporating the local context is helpful in our experiments.

Also, it is worth noting that external resources (the gazetteer) can help improve the recall. Such improvement is dramatic for the city level prediction, from 0.56 when using BOW features, to 0.75 when adding only Gazetteer features; an explanation

of why features that do not leverage information from the gazetteer produce poor recall is that the corresponding classifier could not adequately estimate locations that did not appear in the training data, thus the classifier tends to make conservative predictions, i.e., it only predict tokens it can recognize. This also explains why there is a slight drop with respect to precision when Gazetteer features are included. The improvement for recall is less obvious for other two labels. The explanation could be that among all the locations considered in this work, few are SPs and countries, while the vast majority are cities. The training data is likely to contain many instances of *SP* and *country* labels, but only a small fraction of the cities from the world. As a result, the prediction of cities relies significantly on the external gazetteer.

To better analyze how each feature combination affects the classification, we conducted t-tests on the results of models trained on all combinations of features listed in Table 4, 5 and 6. We found that in *SP* classification, no pair of feature combinations yields statistically significant difference. In *city* classification, using only BOW features produces significantly worse results than any other feature combinations at a 99.9% level of confidence, except BOW+POS features, while using all features produces significantly better results than any other feature combinations at a 99% level of confidence, except BOW+GAZ+WIN features. In *country* classification, the differences are less significant; where using all features and using BOW+GAZ+WIN features both yield significantly better results than 4 of 6 other feature combinations at a 95% level of confidence, while the difference between them is not significantly different; unlike in *city* classification, the results obtained by using only BOW features is significantly worse merely than the two best feature combinations mentioned above.

We further looked at the t-tests results of *city* classification to analyze what impact each feature set has on the final results. When adding POS features to a feature combination, the results might improve, but never statistically significant; by contrast, they always significantly improve when GAZ features or WIN features are added. These are consistent with our previous observations.

By analyzing the results of location disambiguation shown in Table 7, we can see that when going through all steps, we get an accuracy of 95.5%, while by simply making sure the type of the candidate is correct and choosing the default location with the largest population, we achieve a better accuracy. The best result is obtained by using the adjacent locations, which turns out to be 98.2% accurate. Thus we conclude that adjacent locations help disambiguation, while locations in the global context do not. Therefore the assumption made by Li et al. (2002) that the locations in the global context help the inference of a target location does not hold for Twitter messages, mainly due to their short nature.

Chapter 4

Estimating User Locations

4.1 Introduction

In this chapter, our concern is how to estimate users' locations from the data they generate on social media, and in particular infer Twitter users' location using tweets, the messages they post on their Twitter accounts. For each user, we put together all the tweets written by that user, in order to predict his/her physical location.

The relation between geographical location and language has been studied since the 19th century as a sub-field of sociolinguistics known as dialectology (Petyt, 1980; Chambers, 1998). The location of a social media user can be highly valuable. For instance, advertisers would want to target consumers in a particular city or country. Having realized that, social media service providers have added a feature that allow users to provide their locations, mostly through GPS locating or by manual specification. However, due primarily to privacy and security concerns, only a small proportion of users actually provide location information. In view of such extreme sparsity, researchers have developed various ways of inferring users' locations using

information such as interactions between users, locations declared by users in their social media profiles, users' time zones, the text they generate, etc.

In this part of the thesis, we mainly focus on predicting users' locations with a deep learning architecture built with denoising auto-encoders proposed first by Vincent et al. (2008). The chapter is structured as follows: Section 4.2 introduces previous work relevant to our work; Section 4.3.1 explains the dataset we use to train and test our models; in Section 4.3.2, we describe in detail the models we propose; Section 4.4 explains what the evaluation metrics are and how the models are implemented; Section 4.4.6 lists our results and a comparison with previous work; Section 4.5 summarizes our work.

4.2 Related Work

4.2.1 Location Prediction Using Twitter Data

Many methods have been proposed to predict users' locations based on the social media text data they generate. One of the very first is by Cheng et al. (2010), who first learned the location distribution for each word, then inferred the location of users at U.S. city level according to the words in their tweets. Specifically, they estimated the posterior probability of a user being from a city c given his/her tweets t by computing:

$$P(c|t) = \prod_{w \in t} P(c|w) \times P(w) \tag{28}$$

where w is a word contained in this user's tweets. To improve the initial results, they also used several smoothing techniques such as Laplace smoothing and so-called *data-driven geographic smoothing* and *model-based smoothing*. Their best model managed

to make accurate predictions (less than 100 miles away from the actual location) 51% of the time, and the average error distance is 535.564 miles. It is worth noting that the size of the dataset in their work is large, containing 4,124,960 tweets from 130,689 users.

Eisenstein et al. (2010) adopted a topic model approach. They treated tweets as documents generated by two latent variables, i.e., topic and region, and train a system they call *geographic topic model*, which could predict authors' locations based on text alone. Like Cheng et al. (2010), their model also relied on learning regional word distributions. The average distance from the model's prediction to the actual location is 900 kilometres. By comparison, their dataset is much smaller, containing 380,000 tweets from 9,500 users. This dataset is made available and has been used by a number of works.

Roller et al. (2012) used a variant of K-Nearest Neighbours; they divided the geographic surface of the Earth into *grids* and then constructed a pseudo-document for each grid; a location for a test document was chosen based on the most similar pseudo-document. Another type of model is a variant of Gaussian mixture models (GMMs) proposed by Priedhorsky et al. (2014). Their approach resembles that of Cheng et al. (2010) in constructing location-sensitive n-grams; besides tweets, they also used information such as users' self-reported locations and time zones for prediction.

4.2.2 Deep Neural Networks Applied to NLP

Data representation is important for machine learning (Domingos, 2012). Many statistical NLP tasks use hand-crafted features to represent language units such as words and documents; these features are fed as the input to machine learning models. One

such example is sentiment classification which uses external lexicons that contain words with emotion or sentiment prior polarities (Ghazi et al., 2014; Aman and Szpakowicz, 2008; Melville et al., 2009; Li et al., 2009). Despite the usefulness of these hand-crafted features, designing them is time-consuming and requires expertise.

A number of researchers have implemented DNNs in the NLP domain, achieving state-of-the-art performance without having to manually design any features. The most relevant to ours is the work of Glorot et al. (2011), who developed a deep learning architecture that consists of stacked denoising auto-encoders and apply it to sentiment classification of Amazon reviews. Their stacked denoising auto-encoders can capture meaningful representations from reviews and outperform state-of-the-art methods; due to the unsupervised nature of the pre-training step, this method also performs domain adaptation well.

In the social media domain, Tang et al. (2013) extracted representations from Microblog text data with Deep Belief Networks (DBNs) and used the learned representations for emotion classification, outperforming representations based on Principal Component Analysis and on Latent Dirichlet Allocation.

Huang and Yates (2010) showed that representation learning also helps domain adaptation of part-of-speech tagging, which is challenging because POS taggers trained on one domain have a hard time dealing with unseen words in another domain. They first learned a representation for each word, then fed the learned word-level representations to the POS tagger; when applied to out-of-domain text, it can reduce the error by 29%.

4.3 Methods

4.3.1 The Dataset

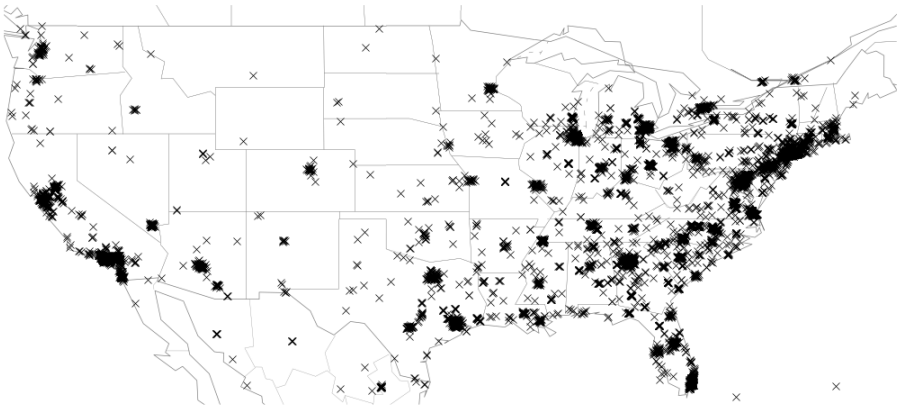


Figure 5: Geographical distribution of users in our dataset; the distribution is clearly skewed across each region, most users are from the West Coast and the East; users also tend to centre around big cities.

In order to compare the performance of our system with that of other systems, we choose a publicly available dataset from Eisenstein et al. (2010), which has been used by several other researchers. It includes about 380,000 tweets from 9,500 users from the contiguous United States (i.e., the U.S. excluding Hawaii, Alaska and all off-shore territories). The dataset also provides geographical coordinates of each user. We regard each user's tweets and location as a training example, i.e., $(x^{(i)}, y^{(i)})$ where $x^{(i)}$ is all the tweets from the i-th user and $y^{(i)}$ is the location of the i-th user. Meta-data

like user's profile and time zone will not be used in our work. Figure 5 shows the distribution of locations of the 9,500 users.

There are several other datasets specifically for localization of tweets, such as the datasets from Roller et al. (2012); Han et al. (2014); but we chose not to test our models on them, because they are not as popular as the Eisenstein dataset, and because their sizes overwhelm the computational resources we had access to.

4.3.2 The Models

We define our work as follows: first, a classification task puts each user into one geographical region (see Section 4.4 for details); next, a regression task predicts the most likely location of each user in terms of geographical coordinates, i.e., a pair of real numbers for latitude and longitude. We present one model for each task.

Model 1

The first model consists of three layers of denoising auto-encoders. Each code layer of denoising auto-encoders also serves as a hidden layer of a multiple-layer feedforward neural network. In addition, the top code layer works as the input layer of a logistic regression model whose output layer is a softmax layer.

Softmax Function The softmax function is defined as:

$$softmax_i(\mathbf{z}) = \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^{J} e^{\mathbf{z}_j}} \tag{29}$$

where the numerator z_i is the ith possible input to the softmax function and the denominator is the summation over all possible inputs. The softmax function produces

a normalized probability distribution over all possible output labels. This property makes it suitable for multiclass classification tasks. Consequently, a softmax layer has the same number of neurons as the number of possible output labels; the value of each neuron can be interpreted as the probability the corresponding label given the input. Usually, the label with the highest probability is returned as the prediction made by the model.

In our model, mathematically, the probability of a label i given the input and the weights is:

$$\begin{aligned} & P(Y = i|x^N, W^{(N+1)}, b^{(N+1)}) \\ & = softmax_i(W^{(N+1)}x^N + b^{(N+1)}) \\ & = \frac{e^{W_i^{(N+1)}x^N + b_i^{(N+1)}}}{\sum_j e^{W_j^{(N+1)}x^N + b_j^{(N+1)}}} \end{aligned} \tag{30}$$

where $W^{(N+1)}$ is the weight matrix of the logistic regression layer and $b^{(N+1)}$ are its biases. N is the number of hidden layers, in our case $N = 3$. x^N is the output of the code layer of the denoising auto-encoder on top. To calculate the output of i-th hidden layer (i = 1 ... N), we have:

$$x^i = s(W^{(i)}x^{i-1} + b^{(i)}) \tag{31}$$

where s is the activation function, $W^{(i)}$ and $b^{(i)}$ correspond to the weight matrix and biases of the i-th hidden layer. x^0 is the raw input generated from text[1], as specified in section 4.4. We return the label that maximizes Equation (30) as the prediction,

[1]Explained in Section 4.3.3

i.e.:

$$i_{predict} = \arg\max_i P(Y = i|x^N, W^{(N+1)}, b^{(N+1)}) \tag{32}$$

We denote this model as SDA-1.

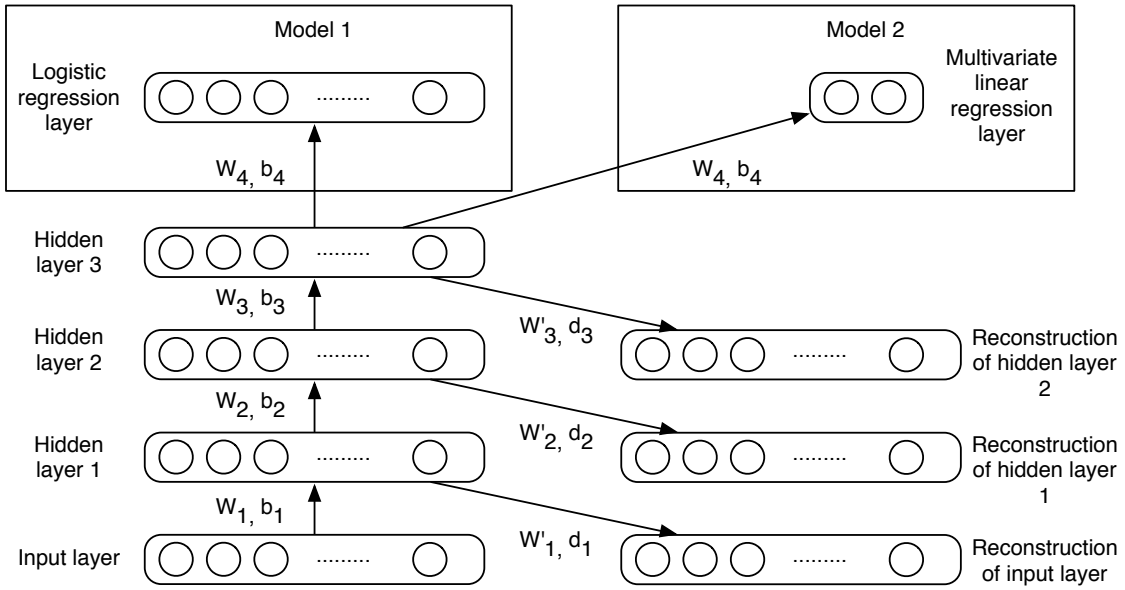


Figure 6: Illustration of the two proposed models (with 3 hidden layers). The models differ only in the output layers. The neurons are fully interconnected. A layer and its reconstruction and the next layer together correspond to a denoising auto-encoder. For simplicity, we do not include the corrupted layers in the diagram. Note that models 1 and 2 are not trained simultaneously, nor do they share parameters.

Model 2

In the second model, a multivariate linear regression layer replaces a logistic regression layer on top. This produces two real numbers as output, which can be interpreted as geographical coordinates. Therefore the output corresponds to locations on the surface of Earth. Specifically, the output of model 2 is:

$$y_i = W_i^{(N+1)} x^N + b_i^{(N+1)} \tag{33}$$

where $i \in \{1, 2\}$, $W^{(N+1)}$ is the weight matrix of the linear regression layer and $b^{(N+1)}$ are its biases, x^N is the output of the code layer of the denoising auto-encoder on top. The output of i-th hidden layer (i = 1 ... N) is computed using Equation (31), which is the same as Model 1. The tuple (y_1, y_2) is then the pair of geographical coordinates produced by the model. We denote this model as SDA-2. Figure 6 shows the architecture of both models.

4.3.3 Input Features

To learn better representations, a basic representation is required to start with. For text data, a reasonable starting representation is achieved with the *Bag-of-N-grams* features (Glorot et al., 2011; Bengio et al., 2013).

N-grams In NLP, the n consecutive tokens in the text are together called a n-gram. For $n = 1, 2, 3$, the corresponding n-gram is called unigram, bigram and trigram, respectively; for $n > 3$, we simply replace the letter n by its numerical value, such as 4-gram, 5-gram, etc. For example, the text *University of Ottawa* contains the unigrams *University, of* and *Ottawa*; the bigrams *University of* and *of Ottawa*; the trigram *University of Ottawa*.

Bag-of-N-grams features are similar to the Bag-of-Words features we used in Section 3.4.1. The elements of a feature vector are binary values which indicate the presence of the corresponding n-grams; alternatively, they can be integers which indicate the frequencies of the corresponding n-grams.

Therefore, the input text of Twitter messages is preprocessed and transformed into a set of Bag-of-N-grams **frequency** feature vectors. We did not use binary feature vectors because we believe the frequency of n-grams is relevant to the task at hand.

For example, a user who tweets *Senators* 10 times is more likely to be from Ottawa than another user who tweets it just once. (The latter is more likely to be someone from Montreal who tweets *Senators* simply because the Canadiens happen to be defeated by the Senators that time.) Due to computational limitations, we consider only the 5000 most frequent unigrams, bigrams and trigrams[2]. We tokenized the tweets using the *Twokenizer* tool from Owoputi et al. (2013).

4.3.4 Statistical Noises for Denoising Auto-encoders

In Section 2.3.3, we introduce denoising auto-encoders (DAs). An essential component of a DA is its statistical noise. Following Glorot et al. (2011), the statistical noise we incorporate for the first layer of DA is the masking noise, i.e., each active element has a probability to become inactive. Mathematically, we get the corrupted input $\tilde{x}$ by:

$$\tilde{x} = x \odot b, x \in \mathbb{N}^D, b \in \mathbb{B}^D, b_i \sim \mathrm{B}(1, p) \tag{34}$$

where x is the original input, b is a binary vector whose elements are independently sampled from a Bernoulli distribution $\mathrm{B}(1, p)$, D is the size of the feature vectors and is equal to 5000 in our case. $x \odot b$ denotes the element-wise multiplication of x and b. Note that the value $q = 1 - p$ is called the masking probability of the masking noise; we can get more corrupted input by increasing the value of q.

For the remaining layers, we apply Gaussian noise to each of them, i.e., a number independently sampled from the Gaussian distribution $\mathcal{N}(0, \sigma^2)$ is added to each element of the input vector to get the corrupted input vector. Note that the Gaussian

[2]Not all of these 5000 n-grams are necessarily good location indicators, we don't manually distinguish them; a machine learning model after training should be able to do so.

distribution has a 0 mean. Mathematically:

$$\tilde{x} = x + g, x \in \mathbb{N}^D, g \in \mathbb{R}^D, g_i \sim \mathcal{N}(0,\, \sigma^2) \tag{35}$$

where x is the original input, g is the vector whose elements are Gaussian distributed, as described above; $x + g$ denotes the element-wise addition of x and g; D is the size of input feature vectors that differs for each layer of denoising auto-encoder. The standard deviation of the Gaussian distribution σ decides the degree of corruption; we also use the term *corruption level* to refer to σ.

4.3.5 Loss Functions

Pre-training

In terms of training criteria for unsupervised pre-training, we use the squared error loss function:

$$\ell(x, r) = ||x - r||^2 \tag{36}$$

where x is the original input, r is the reconstruction. The squared error loss function is a convex function, so we are guaranteed to find the global optimum once we find the local optimum.

The pre-training is done by layers, i.e., we first minimize the loss function for the first layer of denoising auto-encoder, then the second, then the third. We define the decoder weight matrix as the transposition of the encoder weight matrix, as discussed in section 2.3.3.

Fine-tuning

In the fine-tuning phase, the training criteria differ for model 1 and model 2. It is a common practice to use the *negative log-likelihood* as the loss function of models that produce a probability distribution, which is the case for model 1. The equation for the negative log-likelihood function is:

$$\begin{aligned} &\ell(\theta = \{W, b\}, (x, y)) \\ &= -\log(P(Y = y|x, W, b)) \end{aligned} \tag{37}$$

where $\theta = \{W, b\}$ are the parameters of the model, x is the input and y is the ground truth label. To minimize the loss in Equation (37), the conditional probability $P(Y = y|x, W, b)$ must be maximized, which means the model must learn to make the correct prediction with the highest confidence possible. Training a supervised classifier using the negative log-likelihood loss function can be therefore interpreted as maximizing the likelihood of the probability distribution of labels in the training set.

On the other hand, model 2 produces for every input a location $\hat{y}(\hat{lat}, \hat{lon})$, which is associated with the actual location of this user, denoted by $y(lat, lon)$. Given latitudes and longitudes of two locations, their great-circle distance can be computed by first calculating an intermediate value $\Delta\sigma$ with the Haversine formula (Sinnott, 1984):

$$\Delta\sigma = \arctan\left(\frac{\sqrt{\left(\cos\phi_2 \sin\Delta\lambda\right)^2 + \left(\cos\phi_1 \sin\phi_2 - \sin\phi_1 \cos\phi_2 \cos\Delta\lambda\right)^2}}{\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos\Delta\lambda}\right) \tag{38}$$

Next, calculate the actual distance:

$$d((\phi_1, \lambda_1), (\phi_2, \lambda_2)) = r\Delta\sigma \tag{39}$$

where ϕ_1, λ_1 and ϕ_2, λ_2 are latitudes and longitudes of two locations, $\Delta\lambda = \lambda_1 - \lambda_2$, r is the radius of the Earth. Because d is a continuously differentiable function with respect to ϕ_1 and λ_1 (if we consider (ϕ_1, λ_1) as the predicted location, then (ϕ_2, λ_2) is the actual location), and minimizing d is exactly what model 2 is designed to do, we define the loss function of model 2 as the great-circle distance between the estimated location and the actual location:

$$\begin{aligned} &\ell(\theta = \{W, b\}, (x, y)) \\ &= d(Wx + b, y) \end{aligned} \tag{40}$$

where $\theta = \{W, b\}$ are the parameters of the model, x is the input and y is the actual location. Alternatively, we also tried the loss function defined as the average squared error of output numbers, which is equivalent to the average euclidean distance between the estimated location and the true location; this alternative model did not perform well.

Now that we have defined the loss functions for both models, we can train them with back-propagation (Rumelhart et al., 1985).

4.3.6 Training the Models

Stochastic Gradient Descent

Gradient descent is a numerical optimization method. To start with, we describe its simplest form: *online gradient descent* (Robbins and Monro, 1951). When a loss function $\ell(\theta, (x, y))$ is defined, we update the parameters θ proportional to the negative of the partial derivatives with respect to θ[3], i.e.:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \alpha \frac{\partial \ell(\theta, (x^{(t)}, y^{(t)}))}{\partial \theta} \tag{41}$$

where α is called the *learning rate* which dictates how much we update the parameters, $(x^{(t)}, y^{(t)})$ is the training example we used to update the parameters at the t-th iteration. Usually we start by initializing θ with some values $\theta^{(0)}$, by applying Equation (41) repeatedly, we obtain a series of parameters

$$(\theta^{(0)}, \theta^{(1)}, \theta^{(2)}, \dots)$$

We would expect this series to converge to a local minimum, i.e., parameters that minimize the loss function; these parameters are called the *optimal parameters* and are applied to the test data.

Note that in online gradient descent, we consider exactly one training example at each iteration. Alternatively, we can consider a different number of training examples (let's denote this number by B), and update the parameters based on the average of

[3]For MLPs, the parameters of the hidden layers except the topmost one are computed by back-propagation (Rumelhart et al., 1985)

the partial derivatives:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \alpha \frac{1}{B} \sum_{Bt+1}^{B(t+1)} \frac{\partial \ell(\theta, (x^{(t)}, y^{(t)}))}{\partial \theta} \tag{42}$$

If $B = 1$, it is equivalent to online gradient decent; if B is the size of the training dataset, then the method is called *standard/ordinary/batch/full-batch gradient descent*; when B is an intermediate value, the method is then called *mini-match gradient descent*. Online gradient descent and mini-batch gradient descent together are known as *stochastic gradient descent*. An appropriate value of B is beneficial to the model in terms of computation efficiency and fast convergence; however, it generally does not influence the performance of the model (Bengio, 2012).

Early Stopping

We define our loss functions without regularizing the weights; to prevent overfitting, we adopt the early-stopping technique (Yao et al., 2007); i.e., training stops when the model's performance on the validation set no longer improves. Specifically, we adopt the *patience* approach (Bengio, 2012), which is illustrate in pseudocode:

initialization

patience=20, iteration=1;

while *iteration <patience* **do**

  update parameters;

  if *the performance improves* **then**

    patience := max(patience, iteration*2);

  end

  iteration +=1

end

Algorithm 1: Early stopping.

4.4 Experiments and Results

4.4.1 Metrics

We train the stacked denoising auto-encoders to predict the locations of users based on the tweets they post. To evaluate SDA-1, we follow Eisenstein et al. (2010) and define a classification task where each user is classified as from one of the 48 contiguous U.S. states or Washington D.C. The process of retrieving a human-readable address including street, city, state and country from a pair of latitude and longitude is known as *reverse geocoding*. We use MapQuest API [4] to reverse geocode coordinates for each user. We also define a task with only four classes, the West, Midwest, Northeast and South regions, as per the U.S. Census Bureau.[5] The metric for comparison is the classification accuracy defined as the proportion of test examples that are correctly classified. We also implement two baseline models, namely a Naive Bayes classifier

[4] http://www.mapquest.com

[5] `http://www.census.gov/geo/maps-data/maps/pdfs/reference/us_regdiv.pdf`

and an SVM classifier (with the RBF kernel); both of them take exactly the same input as the stacked denoising auto-encoders.

To evaluate SDA-2, the metric is simply the error distance in kilometres from the actual location to the predicted location. Note that this is the distance on the surface of the Earth, also known as the great-circle distance. See Equations (38)-(39) for its computation. Similarly, we implement a baseline model which is simply a multivariate linear regression layer on top of the input layer. This baseline model is equivalent to SDA-2 without hidden layers. We denote this model as baseline-MLR. After we have obtained the performance of our models, they will be compared against several existing models from previous work.

4.4.2 Splitting the Data

To make the comparisons fair, we split the dataset in the same way as Eisenstein et al. (2010) did, i.e., 60% for training, 20% for validation and 20% for testing.

4.4.3 Baseline Models

We also implemented three baseline models which will be compared against the SDA-1 and SDA-2 models. These three baseline models are relatively simple and have shallow architectures; their input features are exactly the same as those of SDA-1 and SDA-2 described in Section 4.3.3.

Baseline Models for SDA-1

The two baseline models implemented are a Naive Bayes classifier and a SVM, described in Section 2.3.2.

Baseline Models for SDA-2

Recall that SDA-2 has a multivariate linear regression layer on the top of three denoising auto-encoders. To obtain a shallow architecture, we simply take away all the DAs and feed the input directly into the multivariate linear regression layer, i.e., we compute the output by:

$$y_i = W_i x^0 + b_i \tag{43}$$

where $i \in \{1, 2\}$, (W, b) are the parameters of the model, x^0 is the raw input described in Section 4.3.3. We denote this model as *baseline-MLR*.

4.4.4 Tuning Hyper-parameters

One of the drawbacks of DNNs is a large number of hyper-parameters to specify (Bengio, 2012). In this section, we list all these hyper-parameters required for our models.

- The activation function, as described in Section 2.3.3. We adopt is the sigmoid function $y = \frac{1}{1+e^{-x}}$, which is a typical choice as the non-linear activation function.

- The size (the number of neurons) of each hidden layer. Usually a larger size indicates better performance but higher computational cost. Since we do not have access to extensive computational power, we set this hyper-parameter to 5000, which is equal to the size of the input layer;

- The corruption level, as described in Section 4.3.4. The masking noise probability for the first layer is 0.3; the Gaussian noise standard deviation for other

layers is 0.25. These two values are chosen because they appear to work well in our experiments based on the validation dataset.

- Mini-batch size. We train the DNN with stochastic gradient descent; the size of mini-batches is 32, which is a reasonable default suggested by Bengio (2012).

- Learning rates. We explore different configurations in the set {0.00001, 0.0001, 0.001, 0.01, 0.1} for both pre-learning learning rate and fine-tuning learning rate.

- Training epochs. Pre-training stops after 25 epochs, which usually guarantees the convergence. Fine-tuning stops after 1000 epochs; because of the early stopping technique described in Section 4.3.6, this number is rarely reached.

4.4.5 Implementation Notes

The Theano Library

Theano (Bergstra et al., 2010) is a scientific computing library written in Python. It is mainly designed for numerical computation. A main feature of Theano is its symbolic representation of mathematical formulas, which allows it to automatically differentiate functions. We train our model with stochastic gradient descent which requires the computation of gradients, either manually or automatically. Since Theano does automatic differentiation, we no longer have to manually differentiate complex functions like Equation (38).

We implemented SDA-1, SDA-2 and the baseline multivariate linear regression model with Theano.

The Scikit-learn Package

Scikit-learn (Pedregosa et al., 2011) is a machine learning package written in Python. It includes most standard machine learning algorithms. The two baseline models compared against SDA-1 (Naive Bayes and SVM) are implemented using the Scikit-learn package.

In addition, scikit-learning has a module that automatically extract Bag-of-N-grams features from text data. We use this module to obtain features described in Section 4.3.3.

Running Time

Trained on a computer with Intel Core i5 CPU and 4G RAM, SDA-1 and SDA-2 normally took a few hours to converge to a local minimum. The other models were usually trained instantly.

4.4.6 Results

The SDA-1 model yields an accuracy of 61.1% and 34.8%, for region classification and state classification, respectively. The results of all models are shown in Table 8. Among all previous works that use the same dataset, only Eisenstein et al. (2010) report the classification accuracy of their models; to present a comprehensive comparison, all models from their work, not just the best one, are listed. Student's t-tests suggest that the differences between SDA-1 and the baseline models are statistically significant at a 99% level of confidence[6].

[6]We are unable to conduct t-tests on the Eisentein models, because of the unavailability of the details of the results produced by these models.

It can be seen that our SDA-1 model performs best in both classification tasks. It is surprising to find that the shallow architectures that we implemented, namely SVM and Naive Bayes, perform reasonably well. They both outperform all models in (Eisenstein et al., 2010) in terms of state-wise classification. A possible explanation is that the features we use (frequencies of n-grams with n = 1, 2, 3) are more indicative than theirs (unigram term frequencies).

	Model	Classification Accuracy(%) Region (4-way)	State (49-way)
Eisenstein et al. (2010)	Geographical topic model	58	24
	Mixture of unigrams	53	19
	Supervised LDA	39	4
	Text regression	41	4
	K-nearest neighbors	37	2
Our models	**SDA-1**	**61.1**	**34.8**
	Baseline-Naive Bayes	54.8	30.1
	Baseline-SVM	56.4	27.5

Table 8: Classification accuracy for SDA-1 and other models

Table 9 shows the mean error distance for various models trained on the same dataset. The difference between SDA-2 and the baseline model is statistically significant at a level of confidence of 99.9% [7]. Our model has the second best results and performs better than four models from previous work. In addition, the fact that SDA-2 outperforms the baseline model by a large margin shows the advantages of a deep architecture and its ability to capture meaningful and useful abstractions from input data.

[7] We are unable to conduct t-tests on the other models, because of the unavailability of the details of the results produced by these models.

Model	Mean Error Distance(km)
Eisenstein et al. (2011)	845
SDA-2	**855.9**
Priedhorsky et al. (2014)	870
Roller et al. (2012)	897
Eisenstein et al. (2010)	900
Wing and Baldridge (2011)	967
Baseline-MLR	1268

Table 9: Mean error distance of predictions for SDA-2 and models from previous work.

4.5 Discussion

The experimental results show that our SDA-1 model outperformed empirical models; our SDA-2 model's performance is reasonable. We demonstrate that a DNN is capable of learning representations from raw input data that helps the inference of location of users without having to design any hand-engineered features. The results also show that deep learning models have the potential of being applied to solve real business problems, in addition to their well-established success in computer vision and speech recognition.

We believe a better model can yet be built. For example, our exploration for hyper-parameters is by no means exhaustive, especially for the mini-batch size and the corruption levels, due to the very high running time required. It would be interesting to find out the optimal set of hyper-parameters. More computational capacity also allows the construction of a more powerful DNN. For example, in our SDA the hidden layers have a size of 5000, which is equal to the size of input layer; however, a hidden layer larger than the input layer learns better representations (Bengio et al., 2013).

The dataset we use does not have a balanced distribution, as shown in Figure 5. Users are densely distributed in the West Coast and most part of the East, whereas very few are located in the middle. Such label imbalance has a negative effect on statistical classifiers, and adversely affects regression models because many target values will never be sampled.

Chapter 5

E-Business Application

5.1 Introduction

In this chapter, we illustrate how the models from the previous chapters can be combined and applied in a real-world e-business scenario, as mentioned in Chapter 1. Specifically, we implement a system that monitors a stream of tweets obtained by calling the twitter API search queries and returns the following information:

1. The location entities mentioned in these tweets.

2. The physical locations of users.

3. The geographic distribution of the sentiments associated with these tweets.

5.2 Prerequisites

5.2.1 Data for Testing the System

We test the system on the data collected in chapter 3, i.e., the tweets related to mobile phones, because it is a typical e-business application in which companies monitor the social media data about their products.

5.2.2 Retraining the Location Estimation Model

In this chapter, we would like to estimate user locations using the stacked denoising auto-encoders from Chapter 4. However, applying the trained SDAs directly to our data is problematic because the training data of the SDAs are quite different: the training data from Eisenstein et al. (2010) are not associated to any topics, i.e., the users tweet about virtually anything; in our data, all tweets are strictly limited to mobile phones. From a machine learning point of view, since the probability distribution of the data $P(X)$ has changed, the original prediction function $P(Y|X)$ is less suitable.

Dataset

Fortunately, among all 20 million tweets in our data 563,314 are tagged with geographic coordinates. We then grouped these geotagged tweets by user and obtained 1462 users who have more than 10 tweets in the data. Furthermore, for simplicity we consider only users from the contiguous United States, so we filtered out users who are not from these regions. We ultimately got a dataset of 929 users. Its size is about the tenth of that of the Eisenstein et al. (2010) dataset. This dataset will be used to

retrain the model.

Model Modifications

We made two modifications to the SDAs to adapt to the new data. The first modification is regarding the input features of the first layer. In Chapter 4, the features are n-gram frequencies; they do not raise serious issues because the numbers of tweets per user in the Eisenstein et al. (2010) dataset are moderately constant (mostly between 30 and 50), they have a *coefficient of variation*, defined as the ratio of the standard deviation to the mean, of 0.61; whereas in our data set, users either have few (below 15) or a lot (hundreds or even thousands) of tweets; the coefficient of variation is a much larger 3.10. In this situation, using frequencies as features becomes doubtful because higher frequencies no longer indicate higher likelihood to mention the corresponding uni-grams, but simply a large number of tweets. To remedy this, we propose two alternative feature sets:

1. Binary features instead of frequency features.

2. Normalized frequency features; i.e., the frequencies are divided by the number of tokens in the corresponding tweets.

For the first alternative feature set, despite the loss of information, we can take advantage of binary features to define a better-suited loss function when pre-training the first layer of the DA. In Chapter 4, this loss function is the squared error loss; if we interpret the input feature vectors as bit vectors, the binary cross-entropy loss is defined as (Bengio et al., 2013):

$$\ell(x, r) = -\sum_i x_i log(r_i) + (1 - x_i) log(1 - r_i) \tag{44}$$

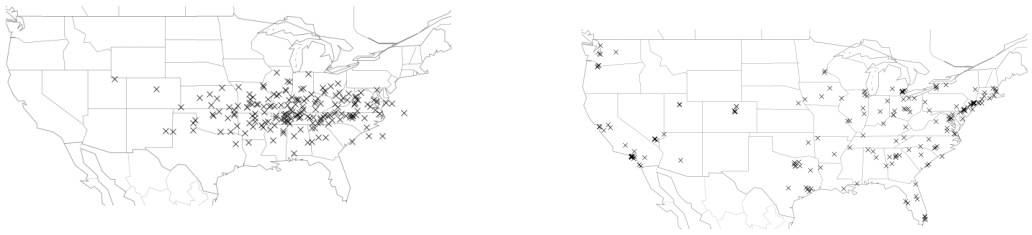
where x is the input and r is the reconstruction. This loss function forces the elements of the reconstruction vector to be as close to 1 or 0 as possible when the input elements are 1 or 0; otherwise, the model suffers a larger penalty when using the binary cross-entropy loss than using the squared error loss.

Experiments and Results

We follow the same experimental procedures as in Chapter 4 and the results are displayed in Table 10. The dataset is split randomly into three subsets: 60% for training, 20% for validation and 20% for testing. The accuracies and mean error distance are obtained by applying the model to the test set. A significant drop in performance is noticed by comparing it to Table 8 and Table 9. Possible explanations include the smaller size of the dataset and the fact that tweets in this dataset are all related to mobile phones, and while talking about some central topic, users are less likely to share strong evidences of their whereabouts. We also observe that binary features yield slightly better results than normalized frequency features. Figure 7 shows the predicted distribution and the true distribution of the users in the test dataset, we can see that the predictions from Figure 7a are not very good, because they are the result of the model's effort to minimize the average error distance, since the East is where most users are; as a consequence, the predicted locations are not widespread across the map.

SDA-1	Accuracy(%)	
	Binary features	Normalized frequency features
Region classification	39.1	34.4
State classification	14.1	14.0
SDA-2	Mean error distance (km)	
Regression	1282.264	1304.158

Table 10: Performance of the SDAs trained on our data.



(a) Predicted locations of users in the test set. (b) True locations of users in the test set.

Figure 7: Distribution of users in the test set.

5.2.3 Preparing the Demo Dataset

To test the location detection model and the location estimation models, we first have to obtain a dataset specifically for this purpose. We first randomly chose 1500[1] users from the whole dataset (containing 20 million tweets) who have more than 10 tweets and no geographic coordinates, which ensures that the demo dataset and the training dataset have no intersections. Because the location estimation models are trained on tweets from the contiguous United States, naturally the tweets in the demo dataset should also have the same geographic origin. To ensure this, we manually went through the self-claimed profile locations of these 1500 users to check if they explicitly indicate that they are from the contiguous U.S., such as *NYC*, *Memphis, TN*; users whose profile locations do not qualify were discarded. In the end, we obtained a final demo dataset that consists of 596 users and 7997 tweets.

5.3 Applying the Location Entity Detection Model

The location entities identified by the location detection model are plotted in Figure 8. This figure, along with the subsequent plots, is generated with the Basemap toolkit[2] and the matplotlib library (Hunter, 2007). For the sake of illustration, we only plot cities since they can be interpreted as a point on the map and are thus easy to display. It is interesting to notice that although these users are all from the U.S., they tweet about cities all over the world, but unsurprisingly, less often than U.S. cities.

[1] We chose this number because it is a manageable amount for the manual filtering discussed later.

[2] http://matplotlib.org/basemap/.

Figure 8: The locations recognized by the location detection model.

5.4 Applying the User Location Estimation Model

The models from Section 5.2.2 are applied to the demo dataset. Once again, we only apply SDA-2 because it produces geographic co-ordinates which are easy to plot on a map. Figure 9 is the plot of the locations of the users in the demo dataset predicted by the SDA-2 model. It is obviously a poor prediction because the predicted locations basically form an ellipse in the central eastern U.S. We can see that Figure 9 shows a similar trends to Figure 7a mentioned earlier. In Section 4.5, we discussed how poor data negatively affect machine learning models; the dataset we used in this chapter is as poor as, if not poorer than, that of Eisenstein et al. (2010)

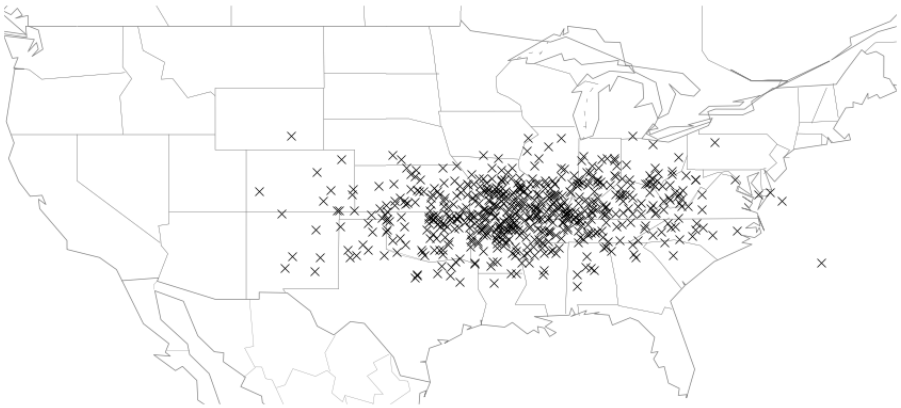


Figure 9: Predicted locations of users in the demo dataset.

5.5 Sentiment Analysis of the Tweets

5.5.1 Related Work

In NLP, sentiment analysis is the field of study that extracts information about sentiments from text data (Liu, 2012). There are various types of goals; the simplest one is to decide whether the author of some text has a *positive* or *negative* opinion about something; a slightly more complicated goal would include *neutral* as well. These positive, negative or neutral opinions are also know as *polarities*. Other goals include identifying the author's emotion as *sad*, *happy*, *surprised*, *angry*, etc.; detecting the target of opinion and opinions about particular aspects or features of the target.

Machine learning techniques are commonly used in sentiment analysis. Early work uses unsupervised learning techniques (Turney, 2002) or standard supervised learning algorithms such as Naive Bayes, SVM and logistic regression (Pang et al., 2002) to classify the sentiments of online reviews of movies and automobiles.

External resources such as lexicons containing the polarities of each lexical entry are often used in sentiment classification; such lexicons include the Word-Net Affect (Strapparava and Valitutti, 2004), the Affective Norms (Stevenson et al., 2007), the NRC Emotion Lexicon (Mohammad and Turney, 2010, 2013), the MPQA Opinion Corpus (Wiebe et al., 2005; Wilson, 2008), the Linguistic Inquiry and Word Count (LIWC) (Pennebaker et al., 2001) and Sentiwordnet (Esuli and Sebastiani, 2006; Baccianella et al., 2010). A number of sentiment analysis models are built using external resources (Aman and Szpakowicz, 2008; Melville et al., 2009; Li et al., 2009).

While the work above utilizes mostly lexical and semantic information from the text, some researchers use the linguistic structure on the sentence level to make more

accurate inference of sentiments. For example, Ghazi et al. (2014) are able to build a model that outperforms their baseline models by taking into account negations, adjectival modifiers, adverbial modifiers and intensifiers.

Deep learning techniques, discussed in Section 2.3.3 and through out Chapter 4, have been successfully applied to sentiment analysis. Glorot et al. (2011) develop a deep learning architecture that consists of stacked denoising auto-encoders and apply it to sentiment classification of Amazon reviews. Another type of models that deserve attention are *recursive neural networks (RNNs)* (Goller and Kuchler, 1996) which have been modified and applied to sentence-level sentiment analysis by Richard Socher in a series of work (Socher et al., 2010, 2011, 2013). RNNs are suitable for learning representations from sentences because they can recognize the structure of these sentences due to the recursive nature of RNNs.

Sentiment analysis has gained popularity in the social media domain mainly because the massive amount of social media data provides a huge potential for mining opinions from them. Mohammad et al. (2013) develop a SVM classifier using features from several external resources to classify the sentiments of tweets and SMS. Speriosu et al. (2011) classify the polarity of tweets using sentiment lexicons and the interactions between users. Kunneman et al. (2014) try to build a model to predict the emotions contained in hashtags in Twitter. Tang et al. (2013) extract representations from data in Weibo, the Chinese counterpart of Twitter, with Deep Belief Network (DBN) and use the learned representations for sentiment classification.

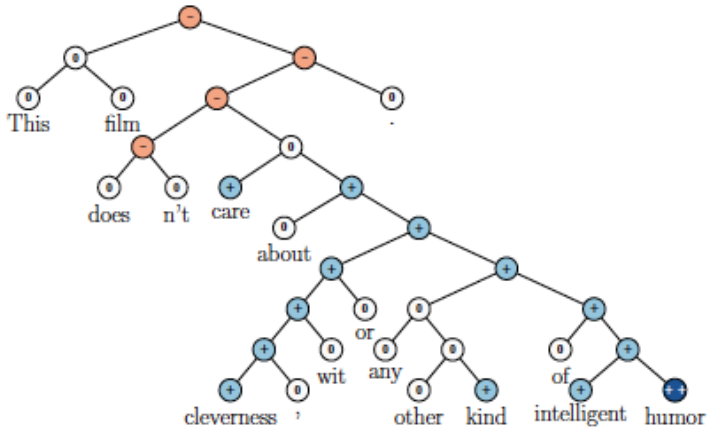


Figure 10: An example of an RNN recursively predicting the sentiment of a sentence. Notice that the subtree on the right is positive; but when it is merged with the subtree on the left, which contains a negation, its positive sentiment is reversed to become the sentiment of the whole sentence. (Socher et al., 2013)

5.5.2 SentiStrength

Apart from machine learning approaches, rule-based approaches are also applied to sentiment analysis. *SentiStrength* (Thelwall et al., 2010, 2012; Thelwall and Buckley, 2013) is one such rule-based sentiment analysis tool. It can produce various outputs, including binary polarities (positive or negative), trinary polarities (positive, neutral or negative) and two numbers that indicate the intensity of the positive and the negative sentiment. We choose to use this tool for our sentiment analysis task in this section because first, it is adapted to informal text in social media; second, it is free for research purposes.

5.5.3 Applying SentiStrength

To every tweet in the demo dataset, we apply SentiStrength and the output is the trinary polarity, i.e., whether the emotion in this tweet is positive, negative or neutral. The number of each polarity with respect to each topic is summarized in Table 11.

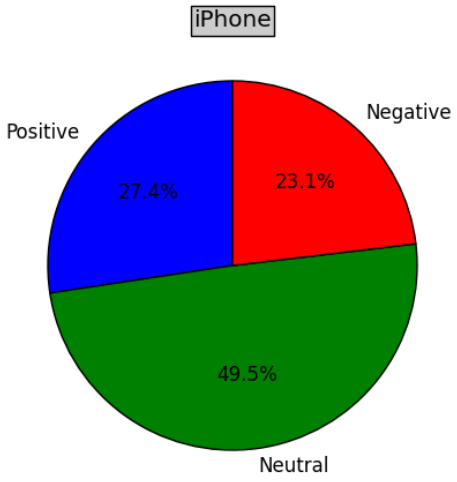
Topic	Positive	Nerutral	Negative
iPhone	1309	2365	1105
Android	668	1008	480
Blackberry	97	190	126
Samsung phone	13	34	21
HTC phone	12	16	9
Windows phone	177	259	105

Table 11: The number of polarities predicted by SentiStrength.

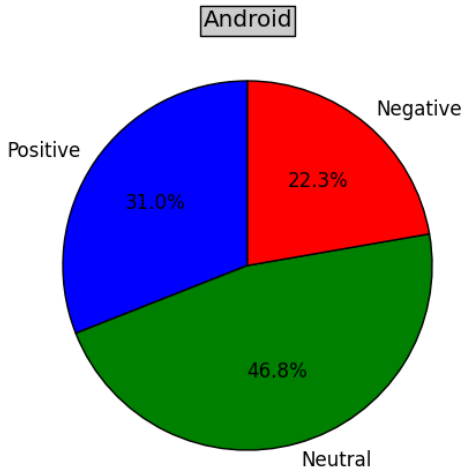
Pie Charts

A more straightforward way to analyze the results above is visualizing them in pie charts, which are displayed in Figure 11. The neutral tweets are approximately the half for each topic; Twitter users appear to be most satisfied with Windows phone, which has the largest fraction of positive tweets and the smallest fraction of negative ones. Similar trends are observed with respect to tweets about HTC phone and Android. Users also generally express positive emotions about iPhone, but the margin between the fraction of positive tweets and that of negative tweets are smaller. By contrast, Blackberry and Samsung are less loved since they received more negative tweets than positive ones.

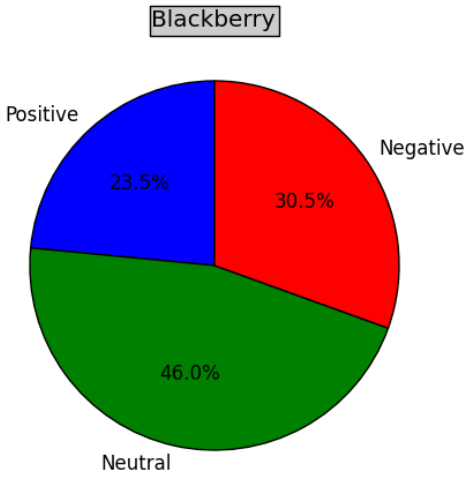
The visualizations are particularly meaningful when companies would like to find out the general reception of their products from the public.



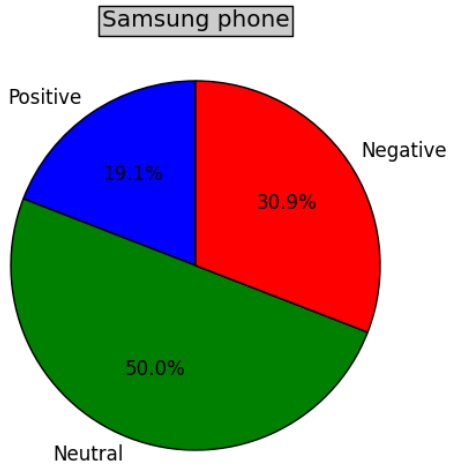
(a) Polarity fractions of tweets related to iPhone.



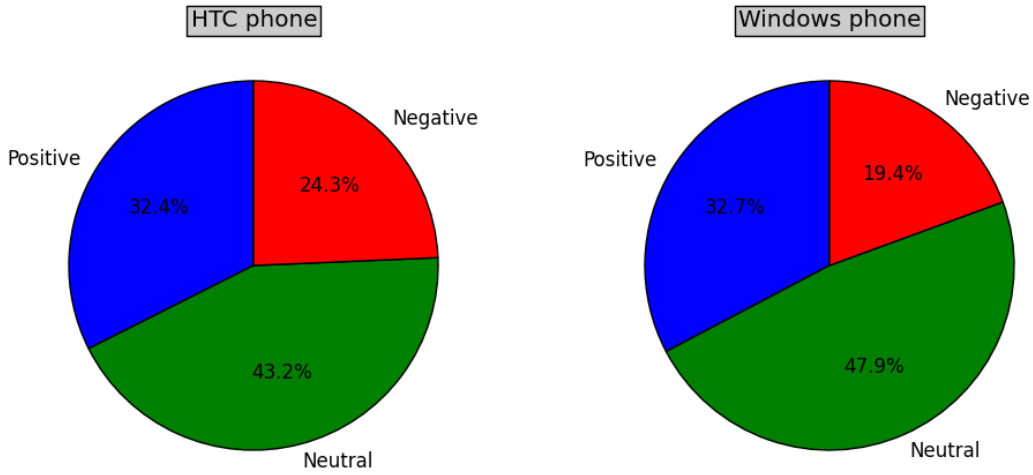
(b) Polarity fractions of tweets related to Android.



(c) Polarity fractions of tweets related to Blackberry.



(d) Polarity fractions of tweets related to Samsung phone.



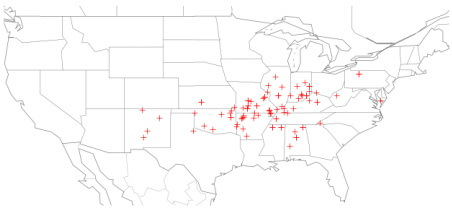
(e) Polarity fractions of tweets related to HTC phone.

(f) Polarity fractions of tweets related to Windows phone.

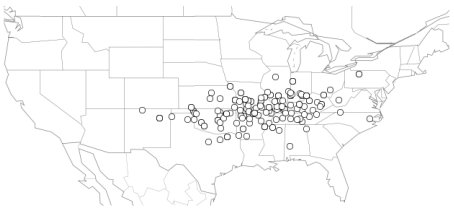
Figure 11: Polarity fractions for each topic.

Geographic Distribution

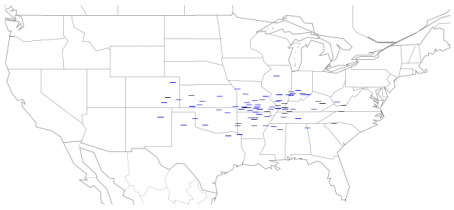
In Section 5.4 we estimated the locations of the users. By assuming that users do not constantly travel long distances, we can simply define the location of a tweet as the location of its author. Therefore, each of the tweets in the demo dataset now corresponds to one location; furthermore, we can plot these tweets on a map according to their **topics, sentiments and locations**. Figure 12 shows how the tweets with different sentiments related to Blackberry are distributed. Unfortunately the poor location predictions limit the information we can learn from these plots, but we can imagine their usefulness if a better location estimation model is available. For example, companies can put emphasis on areas where they receive mostly negative opinions.



(a) Distribution of positive tweets.



(b) Distribution of neutral tweets.



(c) Distribution of negative tweets.

Figure 12: Geographic distribution of tweets related to Blackberry

5.6 Discussion

In this chapter, we show how we can combine the models obtained in previous chapters with a sentiment analysis model and apply the system to real world e-business scenarios. The system can look for location entities mentioned in the tweets and estimate locations of users and visualize them on a map. The sentiment analysis module identifies and opinions contained in each tweet; consequently, we can found out how Twitter users think about some brand and visualize the geographic distribution of their opinions.

Unfortunately, the demo system displayed here has several limitations. For instance, we did not implement a sentiment classification module on our own, but instead the system relies on a third-party toolkit. It has two main drawbacks: first, it causes the whole system to be less coherent; second, the third-party toolkit is not adapted to our application and our data.

Another limitation, and perhaps the most significant one, is the low quality of the training data in Section 5.4 and, consequently, the poor estimations made by the model. To remedy this, we mentioned the possibility of collecting better datasets. Many other improvements, along with this, are necessary to increase the overall usefulness of the system.

Chapter 6

Conclusion and Future Work

6.1 Summary of the Thesis

This thesis can be viewed as a combination of three **sub-tasks**.

The first one looks for location entities in tweets. We extracted different types of features for this task and did experiments to find out their usefulness. We also defined disambiguations rules based on a few heuristics which turned out to work well. We collected and manually annotated the dataset, which can be shared with other researchers.

The second one estimates the physical locations of users base on the tweets they posted. We showed that a deep neural network approach is suitable for this work and that our models yielded results comparable to state-of-the-art models.

The last sub-task is more application-driven. We presented the demo of the system that predicts from tweets the information about locations and sentiments. The results were visualized to provide a straightforward illustration. We also named several real world scenarios where our system can be applied.

We summarize the primary contributions of this thesis as follows: first, we performed location entity extraction from social media data by combing a machine learning approach and a rule-based approach; second, we build a deep neural network to estimate users' locations based on text produced by them. These two models, to the best of knowledge, are new for the respective tasks.

In addition, the data we collected and annotated will be made available to other researchers to test their models and to compare with ours; we also proposed a demo of a social media monitoring system that can be useful in e-business applications.

6.2 Future Work

Sub-task 1

The simple rule-based disambiguation approach can be replaced by a machine learning approach, although this requires more annotated data. Also, in the current model, we consider only states and provinces in the United States and Canada, we plan to extend the model to include states and provinces in other countries as well. Lastly, deep learning models can learn not only document level representations, as we did in Chapter 4, but also word level representations, which can be fed into a sequential tagging model. We plan to implement this approach too.

Sub-task 2

As is disused in Chapter 4 and Chapter 5, both datasets we use do not have balanced distributions thus they have negative impact on the performance of the models. We plan to collect a dataset uniformly distributed geographically, and the locations do

not have to be limited to the contiguous United States. Alternatively, one may notice that the distribution of users is similar to that of the U.S. population, therefore it is possible to use the U.S. census data to offset such skewed distribution of users. In addition, the input of our system consists only of tweets, because we are mostly interested in recovering users' location from the language they produce; however, real applications require a higher accuracy. To achieve this, we could also incorporate information such as users' profiles, self-declared locations, time zones and interactions with other users. Another type of stacked denoising auto-encoder is one that only does unsupervised pre-training, then the output of the code layer is regarded as input into other classifiers such as SVM (Glorot et al., 2011). It would be interesting to compare the performance of this architecture and that of an SDA with supervised fine-tuning with respect to our task.

Sub-task 3

We used a third party sentiment analysis tool SentiStrength in this sub-task. Recursive neural networks (Goller and Kuchler, 1996) is a ideal model to represent structured data like languages. We plan to train a RNN suitable for our task and use this RNN instead of SentiStrength.

In general, for the 3 sub-tasks, we would also like to extend all the models in this thesis to languages other than English, such as French and Chinese.

Appendix A

Table of Notations

$x^{(i)}$	The input of the i-th training example.
$y^{(i)}$	The output of the i-th training example.
$(x^{(i)}, y^{(i)})$	The i-th training example.
$\hat{y}$	The prediction of the model.
y_k	The kth possible outcome class in a classification task.
x_i	The i-th element of the input feature vector.
x^i	The output feature vector of the i-th hidden layer of a multi-layer neural network.
$\tilde{x}$	The reconstruction of the input x.
$\arg\max y$	The argument of the function y that maximizes y.
$\prod$	Product.
$\sum$	Sum.
$\mathbb{B}$	The set of binary values, i.e. $\{0, 1\}$.
$\mathbb{N}$	The set of natural numbers.
$\mathbb{R}$	The set of real numbers.
$u\|\|v$	Concatenating two vectors.
$\|\|x\|\|$	The L_2 norm of x.
$u \odot v$	The element-wise multiplication of two vectors.
$x \sim \mathrm{B}(n, p)$	x is drawn from the binomial distribution.
$x \sim \mathcal{N}(\mu, \sigma^2)$	x is drawn from the normal distribution.
ℓ	The loss function.
∂	Partial derivative.
min	The minimal value in a series/of a function.
$\log(x)$	The logarithm of x.

Appendix B

An Example of the Eisenstein Dataset

Tweets:

Now I feel like I am in school, haven't watched TV... Thank God for DVR...

Yeh.. But I just got in n I DVR'd it.. So will watch it later not 2nite dohRT

@USER_d62d87d3: @USER_0dd20b3c u watch BadGirlsClub??

@USER_474a553b I dey oo... School is takin over my life.. How's work n d baby?

#nowplaying dear Mr. president by PINK

@USER_f4ca2bcb yay!! Wen?

@USER_68d414e1 LOL....LOL. Dis ur bird story is hella funny.. 2much!!

@USER_f4ca2bcb which is wen?????

@USER_1d197eb1 duh!!!! Lol.. Nvm.. Wats goodie?

@USER_1d197eb1 cool.. I see dat,aint nuthin wrng with dat, just headin home from school

@USER_17ec85ee LOL...LOL.. Hope u sha enjoy her when she/he is #gboko e..

LOL...

@USER_d35d3a6c dats a lie.. I have been in d lib err day since school resumed.. N it hasn't been dis packed!!

lunch flow.. jollof rice with chicken and pieces of shaki(tripe).. yummy!!!!

Omg!!! Godfrey is def reppin naija!!!

Haven't been been tweetin much lately... Dats wat happens wen school takes over ur life.. Good morin yalll.. Thank u God 4 dis day!!!

Just gimme d mula..RT @USER_1644c912: please i need to spend tonight, where the party at?

@USER_d57ac466 did it work?

@USER_d57ac466 LOL.. Nah but I have had pretty bad cramps n dats all I take

@USER_d57ac466 LOL...LOL. Den again u aint a female...

@USER_f4ca2bcb Justina- omo2 sexy, Ololufe-wande coal.. Dats all I can think of now..

@USER_2782049d u should.. Lovely day to go rock climbing

@USER_d62d87d3 I kno. Had no idea

Geographic coordinates: 40.667075 -73.766048

Reverse geocoded location: Belt Parkway, Rochdale Village, NYC, Kings, New York, 11413, United States of America

Appendix C

Source Code and Datasets

The source code of **location entities detection** is written in Java and archived at: https://github.com/rex911/locdet, along with the datasets and additional resources. Specifically:

- The folder **lib** contains the external libraries this project is dependent on.
- The file **instruction.pdf** is the manual to read if one would like to run this program.
- The folder **src** contains the source code and the data, detailed as follows:
  - src/crf: CRF classifiers.
  - src/crf/fe: feature extractors.
  - src/disambiguate: the location disambiguation module.
  - src/gazetteer: classes to handle gazetteer and the locations in it.
  - src/resources: the folder **train** contains all the annotated data for training and testing the CRF classifiers; the folder **disam** contains the additional

tweets annotated with true locations for testing the location disambiguation module. The file **hyer.txt** stores the gazetteer.

The source code of of **user locations estimation** is implemented in Python and archived at: https://github.com/rex911/usrloc. Specifically:

- **GeoLearn.py** and **utils.py** wrap several helper function.
- **eisenstein.data.gz** is the dataset from Eisenstein et al. (2010)
- **phone.data.gz** is the dataset collected by us.
- **exdA.py, exSdA.py, exSdA_reg.py, logistic_sgd.py, regression_sgd.py, mlp.py** are the fundamental components used to build the models.
- **tSdA_ei.py, tSdA_ei_reg.py, tSdA_phone.py, tSdA_phone_reg.py** actually train and test the models.

Bibliography

Abrol, S., Khan, L., and Thuraisingham, B. (2012). Tweecalization: Efficient and intelligent location mining in twitter using semi-supervised learning. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on*, pages 514–523.

Aman, S. and Szpakowicz, S. (2008). Using roget's thesaurus for fine-grained emotion recognition. In *IJCNLP*, pages 312–318.

Amitay, E., Har'El, N., Sivan, R., and Soffer, A. (2004). Web-a-Where: Geotagging Web Content. In *Proceedings of the 27th annual international conference on Research and development in information retrieval - SIGIR '04*, pages 273–280, New York, New York, USA. ACM Press.

Baccianella, S., Esuli, A., and Sebastiani, F. (2010). Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In *LREC*, volume 10, pages 2200–2204.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. *Neural Networks: Tricks of the Trade*, 7700:437–478.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review

and new perspectives. *Pattern Analysis and Machine Intelligence*, 35(8):1798 – 1828.

Bengio, Y. and Lamblin, P. (2007). Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19(153).

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, volume 4, page 3.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM.

Bouillot, F., Poncelet, P., and Roche, M. (2012). How and why exploit tweet ' s location information ? In Jérôme Gensel, D. J. and Vandenbroucke, D., editors, *AGILE'2012 International Conference on Geographic Information Science*, pages 24–27, Avignon.

Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294.

Chambers, J. K. (1998). *Dialectology*. Cambridge University Press.

Cheng, Z., Caverlee, J., and Lee, K. (2010). You are where you tweet: a content-based approach to geo-locating Twitter users. In *CIKM '10 Proceedings of the 19th ACM international conference on Information and knowledge management*, volume October 26, pages 759–768, Toronto.

Chomsky, N. (1956). Three models for the description of language. *Information Theory, IRE Transactions on*, 2(3):113–124.

Cohen, W. W. (2004). Minorthird: Methods for identifying names and ontological relations in text using heuristics for inducing regularities from data.

Collins, M. (2003). Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Cunningham, H. (2002). GATE, a general architecture for text engineering. *Computers and the Humanities*, 36(2):223–254.

De Marneffe, M.-C., MacCartney, B., Manning, C. D., et al. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87.

Dredze, M., Paul, M. J., Bergsma, S., and Tran, H. (2013). Carmen: A twitter geolocation system with applications to public health. In *AAAI Workshop on Expanding the Boundaries of Health Informatics Using AI (HIAI)*.

Eisenstein, J., Ahmed, A., and Xing, E. P. (2011). Sparse additive generative models of text. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1041–1048.

Eisenstein, J., O'Connor, B., Smith, N. A., and Xing, E. P. (2010). A latent variable model for geographic lexical variation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 1277–1287, Stroudsburg, PA, USA. Association for Computational Linguistics.

Esuli, A. and Sebastiani, F. (2006). Sentiwordnet: A publicly available lexical resource for opinion mining. In *Proceedings of LREC*, volume 6, pages 417–422.

Gelernter, J. and Mushegian, N. (2011). Geo-parsing messages from microtext. *Transactions in GIS*, 15(6):753–773.

Ghazi, D., Inkpen, D., and Szpakowicz, S. (2014). Prior and contextual emotion of words in sentential context. *Computer Speech & Language*, 28(1):76–92.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Domain Adaptation for Large-Scale Sentiment Classification : A Deep Learning Approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 513–520.

Goller, C. and Kuchler, A. (1996). Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, pages 347–352. IEEE.

Han, B., Cook, P., and Baldwin, T. (2014). Text-based twitter user geolocation prediction. *J. Artif. Intell. Res.(JAIR)*, 49:451–500.

Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–54.

Hinton, G. E. and Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. *Cambridge, MA: MIT Press*, 1:282–317.

Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, minimum description length, and Helmholtz free energy. *Advances in neural information processing systems*, pages 3–3.

Huang, F. and Yates, A. (2010). Exploring representation-learning approaches to domain adaptation. In *Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing*, pages 23–30.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3):90–95.

Joachims, T. (1998). *Text categorization with support vector machines: Learning with many relevant features*. Springer.

Jurafsky, D. and Martin, J. H. (2000). *Speech and language processing an introduction to natural language processing, computational linguistics, and speech*. Pearson Education, 1st edition.

Kunneman, F., Liebrecht, C., and van den Bosch, A. (2014). The (un) predictability of emotional hashtags in Twitter. In *Proceedings of the 5th Workshop on Language Analysis for Social Media (LASM)@ EACL*, pages 26–34.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of*

the Eighteenth International Conference on Machine Learning, ICML '01, pages 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Lewis, D. D. (1998). Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval. *Machine Learning: ECML-98*, 1398:4–15.

Li, H., Srihari, R. K., Niu, C., and Li, W. (2002). Location normalization for information extraction. In *Proceedings of the 19th international conference on Computational linguistics*, volume 1, pages 1–7, Morristown, NJ, USA. Association for Computational Linguistics.

Li, T., Zhang, Y., and Sindhwani, V. (2009). A non-negative matrix tri-factorization approach to sentiment classification with lexical prior knowledge. In *ACL '09 Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 244–252. Association for Computational Linguistics.

Liu, B. (2012). Sentiment analysis and opinion mining. *Synthesis Lectures on Human Language Technologies*, 5(1):1–167.

Mani, I., Hitzeman, J., Richer, J., Harris, D., Quimby, R., and Wellner, B. (2008). SpatialML: Annotation Scheme, Corpora, and Tools. In *Proceedings of the 6th international Conference on Language Resources and Evaluation (2008)*, page 11.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

McCallum, A. and Li, W. (2003). Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 188–191. Association for Computational Linguistics.

Melville, P., Gryc, W., and Lawrence, R. D. (2009). Sentiment analysis of blogs by combining lexical knowledge with text classification. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, page 1275, New York, New York, USA. ACM Press.

Mitchell, T. M. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45.

Mohammad, S. M., Kiritchenko, S., and Zhu, X. (2013). Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets. *arXiv preprint arXiv:1308.6242*.

Mohammad, S. M. and Turney, P. D. (2010). Emotions evoked by common words and phrases: Using mechanical turk to create an emotion lexicon. In *Proceedings of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*, pages 26–34. Association for Computational Linguistics.

Mohammad, S. M. and Turney, P. D. (2013). Crowdsourcing a word–emotion association lexicon. *Computational Intelligence*, 29(3):436–465.

Owoputi, O., OConnor, B., Dyer, C., Gimpel, K., Schneider, N., and Smith, N. A. (2013). Improved part-of-speech tagging for online conversational text with word clusters. In *Proceedings of NAACL-HLT*, pages 380–390.

Pang, B., Lee, L., and Vaithyanathan, S. (2002). Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics.

Paradesi, S. (2011). Geotagging tweets using their content. In *Proceedings of the Twenty-Fourth International Florida*, pages 355–356.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pennebaker, J. W., Francis, M. E., and Booth, R. J. (2001). Linguistic inquiry and word count: Liwc 2001. *Mahway: Lawrence Erlbaum Associates*, 71:2001.

Petyt, K. M. (1980). *The study of dialect: An introduction to dialectology.* Andre Deutsch.

Priedhorsky, R., Culotta, A., and Del Valle, S. Y. (2014). Inferring the origin locations of tweets with quantitative confidence. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing - CSCW '14*, pages 1523–1536, New York, USA. ACM Press.

Rabiner, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Razavi, A. H., Inkpen, D., Falcon, R., and Abielmona, R. (2014). Textual risk mining for maritime situational awareness. In *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2014 IEEE International Inter-Disciplinary Conference on*, pages 167–173. IEEE.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

Rodríguez-Penagos, C., Grivolla, J., and Fibá, J. C. (2012). A hybrid framework for scalable opinion mining in social media: detecting polarities and attitude targets. In *Proceedings of the Workshop on Semantic Analysis in Social Media*, pages 46–52. Association for Computational Linguistics.

Roller, S., Speriosu, M., Rallapalli, S., Wing, B., and Baldridge, J. (2012). Supervised text-based geolocation using language models on an adaptive grid. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1500–1510. Association for Computational Linguistics.

Rosenblatt, F. (1961). Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, DTIC Document.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, DTIC Document.

Sarawagi, S. and Cohen, W. W. (2004). Semi-markov conditional random fields for information extraction. In *NIPS*, volume 17, pages 1185–1192.

Sinnott, R. W. (1984). Virtues of the haversine. *Sky and telescope*, 68:158.

Socher, R., Manning, C. D., and Ng, A. Y. (2010). Learning Continuous Phrase Representations and Syntactic Parsing with Recursive Neural Networks. *Advances in Neural Information Processing Systems*, pages 1–9.

Socher, R., Pennington, J., Huang, E. H., Ng, A. Y., and Manning, C. D. (2011). Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 151–161. Association for Computational Linguistics.

Socher, R., Perelygin, A., and Wu, J. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1631–1642.

Speriosu, M., Sudan, N., Upadhyay, S., and Baldridge, J. (2011). Twitter polarity classification with label propagation over lexical links and the follower graph. In *Proceedings of the First workshop on Unsupervised Learning in NLP*, pages 53–63. Association for Computational Linguistics.

Stevenson, R. A., Mikels, J. A., and James, T. W. (2007). Characterization of the Affective Norms for English Words by discrete emotional categories. *Behavior Research Methods*, 39(4):1020–1024.

Strapparava, C. and Valitutti, A. (2004). Wordnet affect: an affective extension of WordNet. In *LREC*, volume 4, pages 1083–1086.

Sutton, C. and McCallum, A. (2006). *An introduction to conditional random fields for relational learning*, volume 2. Introduction to statistical relational learning. MIT Press.

Tang, D., Qin, B., Liu, T., and Li, Z. (2013). Learning Sentence Representation for Emotion Classification on Microblogs. *Natural Language Processing and Chinese Computing*, 400:212–223.

Thelwall, M. and Buckley, K. (2013). Topic-based sentiment analysis for the social web: The role of mood and issue-related words. *Journal of the American Society for Information Science and Technology*, 64(8):1608–1617.

Thelwall, M., Buckley, K., and Paltoglou, G. (2012). Sentiment strength detection for the social web. *Journal of the American Society for Information Science and Technology*, 63(1):163–173.

Thelwall, M., Buckley, K., Paltoglou, G., Cai, D., and Kappas, A. (2010). Sentiment strength detection in short informal text. *Journal of the American Society for Information Science and Technology*, 61(12):2544–2558.

Tjong Kim Sang, E. F. (2002). Introduction to the conll-2002 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2002*, pages 155–158. Taipei, Taiwan.

Turney, P. D. (2002). Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 417–424. Association for Computational Linguistics.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 1096–1103.

Wiebe, J., Wilson, T., and Cardie, C. (2005). Annotating expressions of opinions and emotions in language. *Language resources and evaluation*, 39(2-3):165–210.

Wilson, T. A. (2008). *Fine-grained subjectivity and sentiment analysis: recognizing the intensity, polarity, and attitudes of private states*. ProQuest.

Wing, B. P. and Baldridge, J. (2011). Simple supervised document geolocation with geodesic grids. In *HLT '11 Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 955–964. Association for Computational Linguistics.

Yao, Y., Rosasco, L., and Caponnetto, A. (2007). On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315.