Text Mining in Social Media for Security Threats

Diana Inkpen

Abstract We discuss techniques for information extraction from texts, and present two applications that use these techniques. We focus in particular on social media texts (Twitter messages), which present challenges for the information extraction techniques because they are noisy and short. The first application is extracting the locations mentioned in Twitter messages, and the second one is detecting the location of the users based on all the tweets written by each user. The same techniques can be used for extracting other kinds of information from social media texts, with the purpose of monitoring the topics, events, emotions, or locations of interest to security and defence applications.

Keywords Information extraction · Natural language processing · Social media · Text mining · Automatic text classification · Conditional random fields · Deep neural networks

1 Introduction

There is a huge amount of user-generated content available over the Internet, in various social media platforms. An important part of this content is in text form. Humans can read only a small part of these texts, in order to detect possible threats to security and public safety (such as mentions of terrorist activities or extremist/radical texts). This is why text mining techniques are important for security and defence applications. Therefore, we need to use automatic methods for extracting information from texts and for detecting messages that should be flagged as possible threats and forwarded to a human for further analysis.

Information extraction from text can target various pieces of information. The task could be a simple key phrase search (with focus on key phrases that could be

D. Inkpen (✉)
School of Electrical Engineering and Computer Science, University of Ottawa,
800 King Edward, Ottawa, ON K1N 6N5, Canada
e-mail: Diana.Inkpen@uOttawa.ca

R. Abielmona et al. (eds.), *Recent Advances in Computational Intelligence in Defense and Security*, Studies in Computational Intelligence 621,
DOI 10.1007/978-3-319-26450-9_19

relevant for detecting terrorist threats) or a sophisticated topic detection task (i.e., to classify a text as being about a terrorism-related topic or not). Topic detection was studied by many researchers, while only a few focused on social media texts [32]. Emotion detection from social media texts could also be of interest to security applications, in particular anger detection. Messages that express anger at high intensity levels could be flagged as possible terrorist threats. Combined with topic detection, anger detection could lead to more accurate flagging of the potential threats. Emotion classification was tested on social media messages, for example on a blog dataset [14] and on the LiveJournal dataset [21].

Location detection from social media texts is the main focus of in chapter. There are two types of locations: location entities mentioned in the text of a message and the physical locations of the users. We present experiments that show that location mentions can be extracted from Twitter messages: in particular, what cities, states/provinces, or countries are mentioned in a tweet [20]. This is useful in order to detect events or activities located in specific places that are mentioned by people. For example, potential terrorist plots can target specific geographic areas. For the second kind of locations, we present experiments that predict the physical location of a Twitter user based on all the messages written by the user [26]. Only a few users declare their location in their Twitter account profile. We used this data (tweets annotated with user location) as training data for a classifier that can be used to prediction the location of any user. The classifiers catch subtle differences in the language (dialect) and the types of entities mentioned. User location can be of interest to defence applications in cases when many disturbing messages are posted by a user, in order to estimate the possible location of this user.

The first task discussed (called task 1 bellow) detects location mentioned and it needs to extract spans of one or often several words. Another example of task that extracts spans of text is risk detection. In particular, information about maritime situation awareness, from textual reports (risk spans, type of risk, type of vessel, location, etc.) was extracted using a similar technique [33].

The second task that we present in detail in this chapter (called task 2 bellow) is detecting the location of the Twitter users based on their messages. This is a classification task in which the classifier needs to choose one of the possible locations for which the classifier was trained. It is not a sequential classification task, this is why we experimented with standard classifiers that choose one class for each text, as well as with new models based on Deep Neural Networks.

The two tasks together would allow an intelligent system to analyze information posted on Twitter in real time. It can analyze each tweet in order to spot locations mentioned in it (task 1) and to visualize these locations on a map. If many tweets at a given time mention a specific location, it might be the case that some event (such as natural disaster or terrorist attack) just happened somewhere in the world. Or the system can monitor only a region of interest. The system could also keep track of individual users that might have a suspicious behaviour (for example possible terrorist activities, or cyber bullying). If the user does not have a declared location in his/her Twitter profile, the system can collect all the recent tweets from that user, then apply our models from task 2 in order to compute the location of the user.

The novelty of the computational intelligence methods proposed in this chapter consists in the way we address task 1, via a sequence-based classifier followed by disambiguation rules, and the way we address task 2, via Deep Neural Networks, which were not applied yet to this task.

2 Proposed Computational Intelligence Solution

2.1 Extracting Expressions Using Conditional Random Fields

Early information extraction techniques were based on identifying patterns that can extract information of interest [7]. The patterns were often manually formulated, though it is possible to automatically learn patterns. Modern methods of information extraction are based on the latter idea, and even deeper on automatic text classification [1]. In this chapter, we discuss on the latest advances in information extraction from text, based on classifiers such as Support Vector Machines (SVM) [9], Deep Neural Networks [5], and Conditional Random Fields (CRF) [22].

The first two classifiers are applied to a text as a whole and are able to predict a class from a set of pre-determined classes. SVM classifiers were shown to obtain high performance on text data, including the emotion classification tasks. The deep neural networks were very recently applied on text data with high success rate [16].

The CRF classifiers were designed specifically for sequence classification. They can be applied to detecting spans of text that are of interest, by classifying each word into one of the following classes: beginning of a span, inside a span, and outside a span. In this way, CRF learns spans of interested from the annotated training data, and can be applied to detect similar spans in new test data. We used this technique for location expressions detection, due to the sequential nature of the task (an expression contains one or more words, and often a city name, followed by a state/province, followed by a country name).

Before using these classification techniques, we applied Natural Language Processing (NLP) techniques to pre-process the texts in order to extract the features needed for the classification. Examples of features are: words, n-grams (sequences of 2, 3, or more words), part-of-speech tags (such as nouns, verbs, adjectives, adverbs), and syntactic dependency relations.

Social media text is particularly difficult because the current NLP tools are trained on carefully edited texts, such as newspaper texts. Therefore they need to be adapted before being able to run on social media texts that are more informal, ungrammatical, and full of abbreviations and jargon. There are two ways to adapt tools to social media texts. One is to normalize the texts, which is rather difficult without loosing useful information about the people who post messages on social media. The second way, that we use here, was to train all the tools and methods on social media texts, in addition to shallow forms of text normalization, which we used in order to extract

better features for the classification tasks. We also added new types of features specific to social media texts, such as hashtags for Twitter messages, emoticons, etc.

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*:

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

Given some specific sequence of input variables $\mathbf{x}$, the conditional probability of some sequence of output label $\mathbf{y}$ is:

$$\begin{aligned} &P(\mathbf{y}|\mathbf{x}) = \\ &\frac{1}{Z(\mathbf{x})} exp(\sum_{i,k} \lambda_k t_k(y_{i-1}, y_i, x, i) + \sum_{i,l} \mu_l s_l(y_i, x, i)) \end{aligned} \tag{2}$$

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

2.2 Classifying Texts Using Deep Neural Networks

In this section, we present the artificial neural network architectures that we will employ in the task of detection user locations based on their tweets. The reason we chose this technology is that other methods, such as SVM classifiers, were already applied for this task in related work (that we will compare with).

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 $y = \frac{1}{1+e^{-x}}$. To train a MLP, the most commonly used technique is *back-propagation* [35]. Specifically, the errors in the output layer are back-propagated to preceding layers and are used to update the weights of each layer.

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 [5, 18]. 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. The details of DNNs are beyond the scope of this chapter; interested readers can refer to [5, 18, 39] and the introduction from [4].

Data representation is important for machine learning [11]. 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 emotion or sentiment classification which uses external lexicons that contain words with emotion or sentiment prior polarities [2, 15, 24, 28]. Despite the usefulness of these hand-crafted features, designing them is time-consuming and requires expertise. We also used hand-crafted features for task 1, while here, for task 2, we let the DNN choose the features automatically, since this in one of the advantages of the method.

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 in [16], 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, [38] 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 [19] 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 %.

3 Datasets

3.1 Location Expressions Data

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 specific kinds of locations. To our knowledge, a corresponding corpus does not yet exist.[1]

We used the Twitter API[2] 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 were posted, 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.

The amount of data we collected is overwhelming for manual annotation, but having annotated training data 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. [27] defined spatialML: an annotation schema for marking up references to places in natural language. Our annotation model is a sub-model of spatialML. The process of manual annotation is described next.

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) and removal of states and provinces outside the U.S. and Canada; 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.

We then used GATE's gazetteer matching module [10] to associate each entry in our data with all potential locations it refers to, if any. Note that, in this step, the only information we need from the gazetteer is the name and the type of each location. GATE has its own gazetteer, but we replaced it with the GeoNames gazetteer which serves our purpose better. The sizes of both gazetteers are listed in Table 1.[4] In addition to a larger size, the GeoNames contains information such as population, administrative division, latitude and longitude, which will be useful later in Sect. 4.4.

[1] [25] recently released a dataset of various kinds of social media data annotated with generic location expressions, but not with cities, states/provinces, and countries.

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

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

[4] The number of countries is larger than 200 because alternative names are counted; the same for states/provinces and cities.

Table 1 The sizes of the gazetteers

Gazetteer	Number of countries	Number of states and provinces	Number of cities
GATE	465	1215	1989
GeoNames	756	129	163285

The first step is merely a coarse matching mechanism without any effort made to disambiguate candidate locations. For example, the word "Georgia" would be matched to both the state of Georgia and the country in Europe.

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. Ultimately, from the 6000 tweets, we obtained 1270 countries, 772 states or provinces, and 2327 cities.

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, corresponding 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. The agreement 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. The fully-annotated dataset (as well as our source code for task 1) can be obtained through this link.[5]

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 whether 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 the value of an attribute named *trueLoc* within the XML tag. An example of annotated tweet is displayed in Table 2.

3.2 User Location Data

For the second task, we need data annotated with users' locations. We choose two publicly available datasets which have been used by several other researchers. The

[5] https://github.com/rex911/locdet.

Table 2 An example of annotation with the true location

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

first one is from [13].[6] 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. The second one is much larger and we obtained it from [34].[7] It contains 38 million tweets from 449,694 users, all from North America. We regard each user's set of tweets as a training example (labelled with location), i.e., $(x^{(i)}, y^{(i)})$ where $x^{(i)}$ represent all the tweets from the ith user and $y^{(i)}$ is the location of the ith user. Meta-data like user's profile and time zone will not be used in our work.

4 Task 1: Detecting Location Expressions

For this subtask, we propose to use methods designed for sequential data, because the nature of the 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. We decided to use CRF as our main machine learning algorithm, because it achieved good results in similar information extraction tasks.

4.1 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

[6] http://www.ark.cs.cmu.edu/GeoTwitter.

[7] https://github.com/utcompling/textgrounder/wiki/RollerEtAl_EMNLP2012.

to 1. This feature representation is 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 can be considered as the baseline in this work.

- 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. We later on denote these features by *POS features*.
- 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 and POS 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*.
- 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 *G*azetteer feature or *GAZ* feature in the next sections.

In order to obtain BOW features and POS features, we preprocessed the dataset by tokenizing and POS tagging all the tweets. This step was done using the Twitter NLP and Part-of-Speech Tagging tool [29].

For experimental purposes, we would like to find out the impact 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.

4.2 Experiments

Evaluation 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 3 with respect to our task. We will present separate results for each type of locations (cities, states/provinces, and countries).

Table 3 Definitions of true positive, false positive, true negative and false negative

Model	Ground truth	
	Location	¬ Location
predicted as location	TP	FP
predicted as ¬ location	FN	TN

Precision measures how correctly the model makes predictions; it is the proportion of all positive predictions that are actually positive, computed by:

$$precision = \frac{TP}{TP + FP} \tag{3}$$

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

$$recall = \frac{TP}{TP + FN} \tag{4}$$

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

$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha)\frac{1}{R}} \tag{5}$$

where P is the precision and R is the recall; α is the weighting coefficient. In this work, we shall use a conventional value of α, which is 0.5; one can interpret it as equally weighting precision and recall.

We report precision, recall and F-measure for the extracted location expressions, 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 consecutive tokens. The evaluation at the span level is stricter. 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 stricter.

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. We report results for 10-fold cross-validation, because a conventional choice for n is 10. In addition, we report results for separate training and test data (we chose 70 % for training and 30 % for testing). Because the data collection took several months, it is 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 Sect. 4.1 to the performance of the model. To achieve a comprehensive comparison, we tested all possible combinations of features plus the BOW features. In addition, a baseline model which simply predicts a token or a span as a location if it matches one of the entries in the gazetteer.

We implemented the models using an NLP package named MinorThird [8] that provides a CRF module [36] 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.

4.3 Results for Location Expressions

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. To our knowledge, there is no previous work that extracts locations at these three levels, thus comparisons with other models are not feasible.

Discussion The results from Tables 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 160,000 cities, but only 756 countries and 129 states/provinces, as detailed in Table 1. 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 names contain only one word.

Table 4 Performance of the classifiers trained on different features for countries

Features	Token			Span			Separate train-test sets	
	P	R	F	P	R	F	Token F	Span F
Baseline-Gazetteer Matching	0.26	0.64	0.37	0.26	0.63	0.37	–	–
Baseline-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

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. (The same in Tables 5 and 6)

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
Baseline-Gazetteer Matching	0.65	0.74	0.69	0.64	0.73	0.68	–	–
Baseline-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 6 Performance of the classifiers trained on different features for cities

Features	Token			Span			Separate train-test sets	
	P	R	F	P	R	F	Token F	Span F
Baseline-Gazetteer Matching	0.14	0.71	0.23	0.13	0.68	0.22	–	–
Baseline-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

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.

The baseline model not surprisingly produces the lowest precision, recall and f-measure; it suffers specifically from a dramatically low precision, since it will predict everything contained in the gazetteer to be a location. 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.

We conducted t-tests on the results of models trained on all combinations of features listed in Tables 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 significant; 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 significantly; by contrast, they always significantly improve when GAZ features or WIN features are added. These are consistent with our previous observations.

Error Analysis Some of the predictions errors were due to partial detection of some names, for example "Korea" was predicted as a country, instead of "South Korea". Another source of errors was due to misspellings and to non-standard nicknames that were not in our gazetteers. We went through the predictions made by the location entity detection model, picked some typical errors made by it, and looked into the possible causes of these errors.

Example 1:

Mon Jul 01 14:46:09 +0000 2013
Seoul
yellow cell phones family in South Korea #phone #mobile #yellow #samsung
http://t.co/lpsLgepcCW

Example 2:

Sun Sep 08 06:28:50 +0000 2013
minnesnowta.
So I think Steve Jobs' ghost saw me admiring the Samsung Galaxy 4 and now is messing with my phone. Stupid Steve Jobs. #iphone

In Example 1, the model predicted "Korea" as a country, instead of "South Korea". A possible explanation is that in the training data there are several cases containing "Korea" alone, which leads the model to favour "Korea" over "South Korea". In Example 2, the token "minnesnowta" is quite clearly a reference to "Minnesota", which the model failed to predict. Despite the fact that we allow the model to recognize nicknames of locations, these nicknames come from the GeoNames gazetteer; any other nicknames will not be known to the model. On the other hand, if we treat "minnesnowta" as a misspelled "Minnesota", it shows that we can resolve the issue of unknown nicknames by handling misspellings in a better way.

4.4 Location Disambiguation

In the previous section, we have identified the locations in Twitter messages and their types; however, the information about these locations is still ambiguous. In this section, we describe the heuristics that we use to identify the unique actual location referred to by an ambiguous location name. These heuristics rely on information about the type, geographic hierarchy, latitude and longitude, and population of a certain location, which we obtained from the GeoNames Gazetteer. 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 *SP* 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". 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. 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 reset.
4. **Checking global context**. Locations mentioned in a document are geographically correlated [23]. 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. 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).

4.5 Experiments and Results for Actual Locations

We ran the location disambiguation algorithm described above. 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 deactivated optional steps and compared the results.

Example 3:

> Fri Jul 19 16:35:29 +0000 2013
> NYC and San Francisco
> You Have to See this LEOPARD phone HTC 1 case RT PLS http://t.co/Ml6zH3Yp2b

The results of different location disambiguation configurations are displayed in Table 7, where we evaluate the performance of the model by accuracy, which is defined as the proportion of correctly disambiguated locations. By analyzing them, 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 [23] 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.

Error Analysis Similar to Sect. 4.3, this section presents an example of errors made by the location disambiguation model in Example 3. In this example, the disambigua-

Table 7 Location disambiguation results

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

tion rules correctly predicted "NYC" as "New York City, New York, United States"; however, "San Francisco" was predicted as "San Francisco, Atlantida, Honduras", which differs from the annotated ground truth. The error is caused by step 4 of the disambiguation rules that uses contextual locations for prediction; San Francisco of Honduras is 3055 km away from the contextual location New York City, while San Francisco of California, which is the true location, is 4129 km away. This indicates the fact that a more sophisticated way of dealing with the context in tweets is required to decide how it impacts the true locations of the detected entities.

5 Task 2: Detecting User Locations

We define our work as follows: first, a classification task puts each user into one geographical region (see Sect. 5.5 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.

5.1 Models

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:

$$softmax_i(\mathbf{z}) = \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^{J} e^{\mathbf{z}_j}} \tag{6}$$

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:

$$
\begin{aligned}
&P(Y = i|x^N, W^{(N+1)}, b^{(N+1)}) \\
&= softmax_i(W^{(N+1)}x^N + b^{(N+1)}) \\
&= \frac{e^{W_i^{(N+1)}x^N + b_i^{(N+1)}}}{\sum_j e^{W_j^{(N+1)}x^N + b_j^{(N+1)}}}
\end{aligned} \tag{7}
$$

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 ith hidden layer (i = 1 ... N), we have:

$$x^i = s(W^{(i)}x^{i-1} + b^{(i)}) \tag{8}$$

where s is the activation function, $W^{(i)}$ and $b^{(i)}$ correspond to the weight matrix and biases of the ith hidden layer. x^0 is the raw input generated from text,[8] as specified in Sect. 5.5. We return the label that maximizes Eq. (7) as the prediction, i.e.:

$$i_{predict} = \arg\max_i P(Y = i|x^N, W^{(N+1)}, b^{(N+1)}) \tag{9}$$

We denote this model as SDA-1.

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:

$$y_i = W_i^{(N+1)}x^N + b_i^{(N+1)} \tag{10}$$

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 ith hidden layer (i = 1 ... N) is computed using Eq. (8), 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 1 shows the architecture of both models.

5.2 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 [4, 16].

[8] Explained in Sect. 5.2.

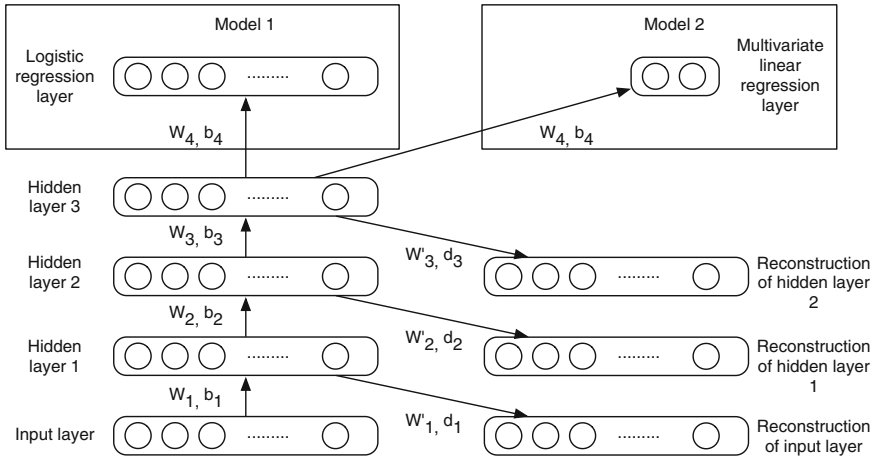

Fig. 1 Illustration of the two proposed models (with 3 hidden layers). The models differ only in the output layers. The neurons are fully interconnected. A layer and its reconstruction and the next layer together correspond to a denoising auto-encoder. For simplicity, we do not include the corrupted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

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.[9] We tokenized the tweets using the *Twokenizer* tool [29].

5.3 Statistical Noises for Denoising Auto-Encoders

An essential component of a DA is its statistical noise. Following [16], 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. 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 distribution has a 0 mean. The

[9]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.

standard deviation of the Gaussian distribution σ decides the degree of corruption; we also use the term *corruption level* to refer to σ.

5.4 Loss Functions

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

$$\ell(x, r) = ||x - r||^2 \tag{11}$$

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.

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:

$$\begin{aligned} &\ell(\theta = \{W, b\}, (x, y)) \\ &= -\log(P(Y = y|x, W, b)) \end{aligned} \tag{12}$$

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 Eq. (12), 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 [37]:

$$\Delta\sigma = \arctan \left(\frac{\sqrt{\left(\cos\phi_2 \sin\Delta\lambda\right)^2 + \left(\cos\phi_1 \sin\phi_2 - \sin\phi_1 \cos\phi_2 \cos\Delta\lambda\right)^2}}{\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos\Delta\lambda} \right) \tag{13}$$

Next, calculate the actual distance:

$$d((\phi_1, \lambda_1), (\phi_2, \lambda_2)) = r\Delta\sigma \tag{14}$$

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:

$$\begin{aligned} &\ell(\theta = \{W, b\}, (x, y)) \\ &= d(Wx + b, y) \end{aligned} \tag{15}$$

where $\theta = \{W, b\}$ are the parameters of the model, x is the input and y is the actual location.[10] Now that we have defined the loss functions for both models, we can train them with back-propagation [35] and Stochastic Gradient Descent (SGD).

5.5 *Experiments*

Evaluation 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 [13] 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[11] 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.[12] 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 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 mean 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 Eqs. (13)–(14) for its computation. In Sect. 5.6, we applied two additional metrics, which are the median error distance and the percentage of predictions less than 100 miles away from the true locations, to comply with previous work. Similarly, we implement a baseline model which is simply a multivariate linear regression layer on top of the

[10] 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.

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

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

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.

Early Stopping We define our loss functions without regularizing the weights; to prevent overfitting, we adopt the early-stopping technique [41]; i.e., training stops when the model's performance on the validation set no longer improves [3].

To make the comparisons fair, we split the Eisenstein dataset in the same way as [13] did, i.e., 60 % for training, 20 % for validation and 20 % for testing. The Roller dataset was provided split, i.e., 429,694 users for training, 10,000 users for validation and the rest 10,000 users for testing; this is the split we adopted.

Tuning Hyper-parameters One of the drawbacks of DNNs is a large number of hyper-parameters to specify [3]. The activation function we adopt is the sigmoid function $y = \frac{1}{1+e^{-x}}$, which is a typical choice as the non-linear activation function. For 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. As for the corruption level, 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. The Mini-batch size chosen for stochastic gradient descent is 32, which is a reasonable default suggested by Bengio [3]. For the learning rates, we explore different configurations in the set {0.00001, 0.0001, 0.001, 0.01, 0.1} for both pre-learning rate and fine-tuning learning rate. Lastly, the 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 Sect. 5.5, this number is rarely reached.

Implementation Theano [6] 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 Eq. (13). We implemented SDA-1, SDA-2[13] and the baseline multivariate linear regression model with Theano. Scikit-learn [30] 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.

[13] Our code is available at https://github.com/rex911/usrloc.

Table 8 Classification accuracy for SDA-1 and other models

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

5.6 Results for User Locations

Evaluation on the Eisenstein Dataset 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 [13] 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.[14]

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 [13] 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).

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 %.[15] 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.

Evaluation on the Roller Dataset Table 10 compares the results from various models on the Roller dataset. The model in [17], which included extensive feature engineering, outperformed other models. In addition it achieves the best results by uti-

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

[15] 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.

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

Model	Mean error distance (km)
Eisenstein [12]	845
SDA-2	**855.9**
Priedhorsky [31]	870
Roller [34]	897
Eisenstein [13]	900
Wing [40]	967
Baseline-MLR	1268

Table 10 Results from SDA-2 and the best models of previous work

Model	Mean error (km)	Median	Acc. error (km)
Roller [34]	860	463	34.6
Han [17]	NA	260	45
Han [17] using top 3 % features (6420)	NA	NA	10
SDA-2	733	377	24.2

NA indicates *Not Available*

lizing about 90 % of all 214,000 features; when using the top 3 % (6420) features, the Accuracy was 10 %.[16] The SDA-2 model, despite the computational limitation, achieved better results than [34] using just 5,000 features.

Error Analysis The datasets we used for this task do not have a balanced distribution. 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. This would explain some of the prediction errors made by our models.

6 Conclusion and Future Work

In this chapter, we looked at techniques that allow us to extract information form texts. This information can be useful ins security applications by allowing to monitoring od locations, topics, or emotions mentioned in texts. Of particular interest are social media messages, which are more difficult to process than regular texts.

We examined two tasks in detail. The first task was extracting location entities mentioned in tweets. We extracted different types of features for this task and did experiments to measure their usefulness. We trained CRF classifiers that were able to achieve a very good performance. We also defined disambiguation rules based on a few heuristics which turned out to work well. In addition, the data we collected and

[16] Only this metric was reported by the author in the top 3 % features configuration.

annotated for task 1 is made available to other researchers to test their models and to compare with ours.

We identify two main directions of future work. First, the simple rule-based disambiguation approach does not handle issues like misspellings well, and can be replaced by a machine learning approach, although this requires more annotated training data. Second, since in the current model, we consider only states and provinces in the United States and Canada, we need to extend the model to include states, provinces, or regions in other countries as well.

For the second task, user location detection, we proposed models based on DNN. Our experimental results show that our SDA-1 model outperformed other 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 that require location detection, in addition to their recent success in natural language processing tasks and 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 [4].

In terms of improvement in the future, 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 [16]. 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.

The most important direction of future work in regard to applications for security and defence is to use the information extracted from texts, about topics, emotions, and locations, in order to flag social media messages for possible security threats. These pieces of information can be combined via a rule-based approach. Alternatively, classification techniques, such as the ones we used for task 2, can be employed; but annotated training data (marked with security threat labels) would be needed.

Acknowledgments I would like to thank my collaborators who helped with the location detection project: Ji Rex Liu, Diman Ghazi, Atefeh Farzindar and Farzaneh Kazemi for task 1 and Ji Rex Liu for task 2.

References

1. Aggarwal, C., Zhai, C.: A survey of text classification algorithms. In: Aggarwal, C.C., Zhai, C. (eds.) Mining Text Data, pp. 163–222. Springer (2012). http://dx.doi.org/10.1007/978-1-4614-3223-4_6
2. Aman, S., Szpakowicz, S.: Identifying expressions of emotion in text. In: Text, Speech and Dialogue, pp. 196–205. Springer (2007)
3. Bengio, Y.: Practical recommendations for gradient-based training of deep architectures. Neural Networks: Tricks of the Trade 7700, pp. 437–478 (2012). http://link.springer.com/chapter/10.1007/978-3-642-35289-8_26
4. Bengio, Y., Courville, A., Vincent, P.: Representation learning: a review and new perspectives. Pattern Anal. Mach. Intell. **35**(8), 1798–1828 (2013). http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6472238
5. Bengio, Y., Lamblin, P.: Greedy layer-wise training of deep networks. Adv. Neural Inf. Process. Syst. **19**(153) (2007). https://papers.nips.cc/paper/3048-greedy-layer-wise-training-of-deep-networks.pdf
6. Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., Bengio, Y.: Theano: a CPU and GPU math expression compiler. In: Proceedings of the Python for Scientific Computing Conference (SciPy). vol. 4, p. 3 (2010)
7. Chang, C.H., Kayed, M., Girgis, M.R., Shaalan, K.F.: A survey of web information extraction systems. Knowl. Data Eng. IEEE Trans. **18**(10), 1411–1428 (2006)
8. Cohen, W.W.: Minorthird: methods for identifying names and ontological relations in text using heuristics for inducing regularities from data (2004)
9. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995). http://dx.doi.org/10.1007/BF00994018
10. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: A framework and graphical development environment for robust nlp tools and applications. In: Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02). Association for Computational Linguistics (2002)
11. Domingos, P.: A few useful things to know about machine learning. Commun. ACM **55**(10), 78–87 (2012)
12. Eisenstein, J., Ahmed, A., Xing, E.P.: Sparse additive generative models of text. In: Proceedings of the 28th International Conference on Machine Learning (ICML'11), pp. 1041–1048 (2011)
13. Eisenstein, J., O'Connor, B., Smith, N.A., Xing, E.P.: A latent variable model for geographic lexical variation. In: Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, pp. 1277–1287. ACL (2010)
14. Ghazi, D., Inkpen, D., Szpakowicz, S.: Hierarchical versus flat classification of emotions in text. In: Proceedings of the NAACL HLT 2010 workshop on computational approaches to analysis and generation of emotion in text, pp. 140–146. Association for Computational Linguistics, Los Angeles, (June 2010). http://www.aclweb.org/anthology/W10-0217
15. Ghazi, D., Inkpen, D., Szpakowicz, S.: Prior and contextual emotion of words in sentential context. Comput. Speech Lang. **28**(1), 76–92 (2014). http://dx.doi.org/10.1016/j.csl.2013.04.009
16. Glorot, X., Bordes, A., Bengio, Y.: Domain adaptation for large-scale sentiment classification: a deep learning approach. In: Proceedings of the 28th International Conference on Machine Learning (ICML'11). pp. 513–520 (2011)

17. Han, B., Cook, P., Baldwin, T.: Text-based Twitter user geolocation prediction. Artif. Intell. Res. **49**(1), 451–500, (Jan 2014). http://dl.acm.org/citation.cfm?id=2655713.2655726
18. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Comput. **18**(7), 54–1527, (Jul 2006). http://dl.acm.org/citation.cfm?id=1161603.1161605
19. Huang, F., Yates, A.: Exploring representation-learning approaches to domain adaptation. In: Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing, pp. 23–30 (2010). http://dl.acm.org/citation.cfm?id=1870530
20. Inkpen, D., Liu, J., Farzindar, A., Kazemi, F., Ghazi, D.: Detecting and disambiguating locations in Twitter messages. In: Proceedings of the 16th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2015). Cairo, Egypt (2014)
21. Keshtkar, F., Inkpen, D.: A hierarchical approach to mood classification in blogs. Nat. Lang. Eng. **18**(1), 61–81 (2012)
22. Lafferty, J.D., McCallum, A., Pereira, F.C.N.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01, pp. 282–289. Morgan Kaufmann Publishers Inc., San Francisco (2001). http://dl.acm.org/citation.cfm?id=645530.655813
23. Li, H., Srihari, R.K., Niu, C., Li, W.: Location normalization for information extraction. In: Proceedings of the 19th international conference on Computational linguistics, vol. 1, pp. 1–7. Association for Computational Linguistics, Morristown, (Aug 2002). http://dl.acm.org/citation.cfm?id=1072228.1072355
24. Li, T., Zhang, Y., Sindhwani, V.: A non-negative matrix tri-factorization approach to sentiment classification with lexical prior knowledge. In: 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. pp. 244–252. Association for Computational Linguistics, (Aug 2009). http://dl.acm.org/citation.cfm?id=1687878.1687914
25. Liu, F., Vasardani, M., Baldwin, T.: Automatic identification of locative expressions from social media text: A comparative analysis. In: Proceedings of the 4th International Workshop on Location and the Web. LocWeb '14, pp. 9–16. ACM, New York (2014). http://doi.acm.org/10.1145/2663713.2664426
26. Liu, J., Inkpen, D.: Estimating user locations on social media: a deep learning approach. Technical Report. University of Ottawa (2014)
27. Mani, I., Hitzeman, J., Richer, J., Harris, D., Quimby, R., Wellner, B.: SpatialML: annotation scheme, corpora, and tools. In: Proceedings of the 6th international Conference on Language Resources and Evaluation (2008), p. 11 (2008). http://www.lrec-conf.org/proceedings/lrec2008/summaries/106.html
28. Melville, P., Gryc, W., Lawrence, R.D.: 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), p. 1275. ACM Press, New York (June 2009). http://dl.acm.org/citation.cfm?id=1557019.1557156
29. Owoputi, O., O'Connor, B., Dyer, C., Gimpel, K., Schneider, N., Smith, N.A.: Improved part-of-speech tagging for online conversational text with word clusters. In: Proceedings of NAACL-HLT, pp. 380–390 (2013)
30. 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., Duchesnay, E.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
31. Priedhorsky, R., Culotta, A., Del Valle, S.Y.: 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), pp. 1523–1536. ACM Press, New York (Feb 2014). http://dl.acm.org/citation.cfm?id=2531602.2531607
32. Razavi, A.H., Inkpen, D., Brusilovsky, D., Bogouslavski, L.: General topic annotation in social networks: A latent dirichlet allocation approach. In: Advances in Artificial Intelligence, Lecture Notes in Computer Science, vol. 7884, pp. 293–300. Springer, Berlin (2013). http://dx.doi.org/10.1007/978-3-642-38457-8_29

33. Razavi, A.H., Inkpen, D., Falcon, R., Abielmona, R.: Textual risk mining for maritime situational awareness. In: 2014 IEEE International Inter-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), pp. 167–173. IEEE (2014)
34. Roller, S., Speriosu, M., Rallapalli, S., Wing, B., Baldridge, J.: 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. pp. 1500–1510. Association for Computational Linguistics (Jul 2012). http://dl.acm.org/citation.cfm?id=2390948.2391120
35. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. Technical Report. DTIC Document (1985)
36. Sarawagi, S., Cohen, W.W.: Semi-markov conditional random fields for information extraction. NIPS **17**, 1185–1192 (2004)
37. Sinnott, R.W.: Virtues of the haversine. Sky Telesc. **68**, 158 (1984)
38. Tang, D., Qin, B., Liu, T., Li, Z.: Learning sentence representation for emotion classification on microblogs. Natural Language Processing and Chinese Computing, vol. 400, pp. 212–223 (2013). http://link.springer.com/chapter/10.1007/978-3-642-41644-6_20
39. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th International Conference on Machine Learning (ICML'08), pp. 1096–1103 (2008). http://portal.acm.org/citation.cfm?doid=1390156.1390294
40. Wing, B.P., Baldridge, J.: Simple supervised document geolocation with geodesic grids. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL HLT '11), pp. 955–964. Association for Computational Linguistics (June 2011). http://dl.acm.org/citation.cfm?id=2002472.2002593
41. Yao, Y., Rosasco, L., Caponnetto, A.: On early stopping in gradient descent learning. Constr. Approx. **26**(2), 289–315 (2007)