A Survey of Short-text Classification Using Modified Naïve Bayes and Multiclass Support Vector Machines*

Michael Peven
Johns Hopkins University
225 W. 29th St
Baltimore, MD 21211, USA
mpeven1@jhu.edu

Christian Reotutar
Johns Hopkins University
3700 N. Charles St.
Baltimore MD, USA
creotut1@jhu.edu

Abstract

Electronic messaging services, including Short Message Service (SMS) over phone networks, Instant Messaging (IM) such as Facebook Chat[1] or Whats App[2], and microblogging platforms such as Twitter[3] have become more popular than ever before. This massive amount of raw data is unparalled in size and while it may be overwhelming to users, but it gives way to many possibilities for the field of machine learning.

While the majority of all these ”short-text” documents are unlabeled, the text posted on the social network known as Twitter comes naturally labeled through the use of ”hashtags” (see section 1). To process the labeled data in this typically multi-class problem, the Naïve Bayes classifier is a popular model with results rivaling that of the Support Vector Machine classifier with shorter training time. In this survey, we seek to find improvements in the Naïve Bayes classifier such that accuracy is improved and compare this to another classifier used in the problem space.

*This document has been adapted from the instructions for earlier ACL and NAACL proceedings, including those for NAACL-HLT-09 by Joakim Nivre and Noah Smith, for ACL-05 by Hwee Tou Ng and Kemal Oflazer, for ACL-02 by Eugene Charniak and Dekang Lin, and earlier ACL and EACL formats. Those versions were written by several people, including John Chen, Henry S. Thompson and Donald Walker. Additional elements were taken from the formatting instructions of the *International Joint Conference on Artificial Intelligence*.

[1]https://facebook.com

[2]https://whatsapp.com

[3]https://twitter.com/

1 Introduction

The classification of documents, spanning from text documents and articles to images and videos, has been a prominent problem in information and computer science. The problem is to assign a category or classification to a given document that gives some indication to relationships between documents or information about the document itself.

Twitter is an on-line social networking microblogging application launched in 2006 that allows a user to post a ”tweet”, a 140 character-message about anything. An important aspect about these tweets relating to this project is the format in which they are posted. If a word in the tweet is prefixed with the pound-sign (#), also known as a ”hash”, the word is considered a ”hashtag”. The purpose of hashtags is to convey the topic of the tweet in which it is used. Because of this hashtag function, twitter data can be exploited by machine learning algorithms in a way that allows the hashtag to serve as the topic, and the body of the tweet to serve as the feature space. This allows for the development of text classification algorithms.

Text classification (referred to here as short-text classification because of the limited size of the text document) is the task of classifying a document into a defined topic or category. A formal definition relating to a twitter post (tweet): If t_i is a tweet in the set of tweets T (with hashtags removed from the body), and $h_1, h_2, ..., h_j$ is the set of hashtags (topics), then we propose to classify each tweet t_i by assigning one hashtag h_j to it.

1.1 Problems in Short-text Classification

Most of the difficulty in getting meaningful results of short-text or chat-message topic classification stems from the incorrect grammar and use of slang in electronic human conversations. In full text documents, subject indexing and classification techniques as well as summarization algorithms have seen success due to advancements in natural language processing algorithms. However, the same algorithms do not apply well to human chat messages where there is lack of sentence structure and correct spelling. This is why we apply supervised learning techniques at a massive scale rather than carefully selected artificial intelligence algorithms designed through the engineering and reverse-engineering of lexical semantics, sentence structure, narrative understanding, and other syntax related theory.

1.2 Topic Classification

This survey deals with the problem of short-text classification in relation to summarization of chat. That is, given a line of text, the goal is to predict the topic of the short-text (chat message) based on a bag-of-words approach. We do this in the domain of Twitter messages: given a tweet body, we wish to predict the associated hashtag for the tweet. The assumption being that the associated hashtag represents the topic of the tweet.

2 Data Description

The data we are using for this project comes from Twitter. More specifically, the tweets were downloaded from the Twitter streaming API v1.1[4] using Mark Dredze's Twitter Stream Downloader [5].

The tweets were collected on a remote server, and the raw data were in JSON format. The attribute in a raw tweet included many attributes in addition to the hashtags and the body of the tweet, including info about the user (username, followers, language preference), info about the tweet (whether it was a retweet, whether it was retweeted, when it was created, user mentions).

[4]https://dev.twitter.com/streaming/

[5]https://github.com/mdredze/twitter_stream_downloader

3 Machine Learning Techniques

Here give a broad overview of what we're comparing, what techniques we used and why

3.1 Naïve Bayes

http://people.csail.mit.edu/jrennie/papers/icml03-nb.pdf http://www2.hawaii.edu/ chin/702/sigir99.pdf The Naïve Bayes Classifier is a probabilistic model based on Bayes' theorem for conditional probabilities but with a naïve independence assumption. The basic idea is to find the probability of a classification given the data (features). This is calculated with probabilities based on the entire training set. The independence assumption assumes the features conditionally independent. For example, features like time of day written and timezone are independent of each other and the presence of one value does highly correlate to the presence of another value. This assumption simplifies calculations and often makes Naïve Bayes a quickly trained classifier.

As it relates to our project, we use a bag of words as our features and a single word as the classification for the bag of words. We assume presence of a word is independent of other words.

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

3.2 Modified Naïve Bayes

In this survey we introduce a variation to the standard Naïve Bayes classifier modified to solve the document classification problem more accurately. There are three main modifications: (1) smoothing of the conditional probabilities in the words and (2) use of Google's word2vec toolkit to make relationships between words in the data.

Looking at the original Naïve Bayes, we see that the probability of a classification is determined as that product of the conditional probabilities of the all the words given the labels. If we have a probability of 0 for any of the words, we see that this automatically makes the product 0 and thus the probability of the classification is 0. This is not our goal, however, because there will be many documents that have words not associated with a label in the training data but that does not necessarily mean it will never happen and we should not discount the possibility immediately. Thus we add smoothing to probabilities by add a nonzero positive number to the probability. This number is very small and has no affect on the probability because classification prediction relies only on proportionalities not exact values.

In order to increase connections between words and labels that might not be explicitly in the training data, we use an external toolkit, word2vec, which takes documents and texts as input and then clusters single words with each based on meaning and context from the training documents. We used this k-means clustering tool in order to add conditional probabilities between words not found in the training data to labels based on their association to words actually in the training data.

3.3 Multi-class SVM

An Support Vector Machine (SVM) is a model that constructs a hyperplane through a set of samples for the purpose of separating them into different classes. The SVM model is a representation of these samples in which the samples of separate categories are divided by a margin that is as large as possible. Formally, given a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where $\mathbf{x}_i \in \mathbb{R}^M$ and $y_i \in \{+1, -1\}$, we want to find the minimizer of the problem:[6]

$$\min_{\mathbf{w}} \lambda \frac{1}{2} ||\mathbf{w}||^2 + \frac{1}{N} \sum_{i=1}^{N} l(\mathbf{w}; (\mathbf{x}_i, y_i)) \qquad (1)$$

where λ is the regularization parameter,

$$l(\mathbf{w}; (\mathbf{x}, y)) = \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x} \rangle\} \qquad (2)$$

Previously, we implemented a Support Vector Machine for a binary classification problem. The approach was called Pegasos[7]. Pegasos performs an optimization function for minimizing an object function called stochastic gradient descent. However, our Pegasos implementation was for a binary classification task. Even though this SVM implementation was binary, there are a variety of strategies for these to be turned into multinomial classifiers. We used a method called One-Versus-All, in which our classifier is trained on one class at a time, treating its own class of samples as positive, and every other class as a negative sample, thereby turning it into a binary classification task. This procedure is done for each unique class/label (or in our case - hashtag). This technique produces a learner for each of the classes in the set. After training is finished, to predict an unlabeled sample, all of these classifiers are then run over the sample, and the one with the largest confidence is selected as the label.

[6]CS475 Machine Learning: Homework 3

[7]see references

4 Methods

4.1 Data Description

The stream downloader was set to run in the background on a remote server, and over the course of a month, ended up collecting over 100 million tweets. This was more than we could even hope to process, and so we only ran our algorithm on only a small subset of those.

4.2 Data filtering

Because we had so much raw data, we were able to be pretty liberal about filtering tweets out of the dataset. Therefore, we added to filtering options to a python script [8] and were able to achieve greater accuracy with the ”cleaned” tweets. The steps we took were:

- transforming all words to lowercase
- removing URLs (*www.example.com*)
- removing at-mentions (@*JohnDoe*)
- removing hashtags from the body (*#pizza*)
- removing punctuation
- removing white-space

Here we talk about the filtering done on the data, the importance of feature vector formation and attribute picking, and the problems we came upon (e.g. language, more than one hash tag).

4.3 Code for Naïve Bayes

In this survey, we use the multinomial Naïve Bayes model used by [Yang and Liu, 1999]. We run our classifier

[8]seen in our code in tweet_parser.py

4.4 Code for Modified Naïve Bayes

4.5 SVMs: Scikit-learn & Multi-class Pegasos

We wanted to compare our Naïve Bayes impelementation with the results from an SVM. We tested a few SVMs from Python's Scikit-learn Machine Learning package[9]

5 Results

5.1 Comparisons

Lots of tables and graphs

5.2 Explanations

It is important that we relate our results to the classifiers we used and other ML topics.

5.3 How we can improve

Any improvements for the classifiers we did not have time to implement

6 Conclusion

Comparison to Proposal

Here is a list of what was done differently from what was stated in the proposal:

- We used a different twitter API in conjunction with Professor Dredze's twitter stream downloader
- We were unable to achieve as clear of results as we thought we could have, but after looking at how poorly structured the tweet bodies were and how the hashtags were often not good classification labels, this was to be assumed ("garbage in, garbage out" as they say)
-

How the time was spent on this project

Coding the Naïve Bayes classifier took longer than expected due to debugging, it was around 20-30 hours worth of work. Adding in the modifications with the word2vec library took an additional 20 hours. Getting the raw data was really easy because of the API, but after achieving bad results, we spent 5-10 hours working with python to filter it. There was an additional 10 hours spent writing python scripts to transform the data into the different forms required (SVM light for Naïve Bayes, but a two-dimensional sparse array). Trying to transform Pegasos into a multi-class SVM algorithm took around 15 hours of work. Using the scikit-learn SVM packages was about 20 hours worth of tinkering. Additional research done for this project was approximately 50 hours. Writing this proposal took another 10 hours.

Acknowledgments

- We would like to acknowledge Ilya Shpitser for giving feedback on our project and offering suggestions. We would also like to acknowledge Professor Shpitser for the skeleton used in the code.
- We would like to thank Mark Dredze for providing access to his twitter stream downloader [10].

References

Wang, Crammer and Vucetic *Multi-Class Pegasos on a Budget*
http://webee.technion.ac.il/people/koby/publications/ICML10_BPegasos.pdf

Shalev-Shwartz, S., Singer, Y., Srebro, N., & Cotter, A. (2011). *Pegasos: primal estimated sub-gradient solver for SVM.* Mathematical Programming, 127(1), 3-30.

Crammer, Koby; and Singer, Yoram (2001). *"On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines"* (PDF). J. of Machine Learning Research 2: 265292.

[9]http://scikit-learn.org/stable/index.html

[10]https://github.com/mdredze/twitter_stream_downloader