EECS 445: Final Report

Nand Dalal
University of Michigan
nndalal@umich.edu

Charles Lewis
University of Michigan
noodle@umich.edu

Bo Chen
University of Michigan
boch@umich.edu

Eric Taseki
University of Michigan
taseskie@umich.edu

Jeffrey Lin
University of Michigan
yulong@umich.edu

Abstract

Our team has chosen to solve the 'Partly Sunny with a Chance of Hashtags' challenge hosted by Kaggle. The goal of the challenge is to provide accurate predictions for twenty-four labels associated with tweets related to the weather. According to Kaggle, "The challenge is to analyze the tweet and determine whether it has a positive, negative, or neutral sentiment, whether the weather occurred in the past, present, or future, and what sort of weather the tweet references."

The primary objective of the competition is to achieve the lowest root mean square error (RMSE). Our first step in achieving this objective was finding a reasonable way to process the raw tweets and represent the data in an intuitive way. The nature of a tweet can be quite different from one to the next in regards to spelling, grammar, and punctuation, which was a large hurdle to overcome. Initially, we created our own preprocessing stage resulting in around 7,000 unique tokens but ended up using Scikit-Learn's feature extraction module which was much more flexible and ran considerably faster in computing. Second, our task was to decide on appropriate models to fit our data. Given the nature of the labels, we used a different set of machine learning algorithms to classify the three different categories. A multilayer perceptron was used to classify sentiment and time period of a given tweet, given these categories followed a probability distribution. Multilabel classification and linear regression, when combined with bootstrapping, were used to predict the types of weather corresponding to each tweet.

Although the Kaggle competition has come to a close, Kaggle still allows submission and reports a hypothetical rank and RMSE. Using the above mentioned machine learning algorithms, our team placed around 170th place with a RMSE of about 18%; the top submission achieved an RMSE of 14%.

1 Introduction

The time frame and subject matter of the 'Partly Sunny with a Chance of Hashtags' challenge hosted by Kaggle makes it an attractive competition for our team to take part in. The problem is to be solved using the analytics of nearly 70,000 tweets for the prediction of a confidence score for twenty-four labels indicating sentiments, time periods, and the weather types. Our provided training data by CrowdFlower Open Data is an attempt to facilitate the use of large datasets through the creation of labels paired with each data entry for machine learning applications. Through competing in this challenge, we are hoping to generate methods for accurate prediction of scores for data labels derived from a large data set and generalize these machine learning algorithms for other such potential applications. Twitter is a platform for collecting data that can be leveraged, via machine learning algorithms, to derive meaningful insights pertaining to a wide variety of subject matters. Upon successful completion of this challenge, not only have we developed algorithms for accurately predicting the sentiment, time, and weather condition given a tweet, but more imporantly we have achieved the validation of such data organization approach, which can be applied to existing datasets for efficient machine learning applications.

2 Proposed Method

In order to understand the methods being used to solve this classification problem, we will start by examining the data provided by Kaggle.

Kaggle has provided both training and testing data. The testing data has not been labled, and will be used for ranking in the competition. The training data consists of three major items: the raw tweet, locations, and a confidence score corresponding to each of twenty-four labels. The twenty-four labels are further split into three categories: sentiment of the tweet (sentiment), corresponding time period (when), and type of weather (kind). The sentiment and when categories follow probability distributions in that the labels associated with each category add up to one. The kind category however does not follow a probability distribution and can have any confidence score between zero and one inclusive associated with its corresponding labels.

Category	Type	Labels
Sentiment	Probability Distribution ($\sum labels = 1$)	I can't tell, Negative, Neutral / author is just sharing information, Positive, Tweet not related to weather condition
When	Probability Distribution ($\sum labels = 1$)	current (same day) weather, future (forecast), I can't tell, past weather
Kind	Confidence Score (each label $\in [0, 1]$)	clouds, cold, dry, hot, humid, hurricane, I can't tell, ice, other, rain, snow, storms, sun, tornado, wind

The people who rated these tweets were allowed to pick only one label for the sentiment and when categories, but were allowed multiple selections for the kind category. Having a single person rating each tweet introduces a bias, so each tweet has been reviewed by multiple people, who may not agree on the assigned labels for a particular tweet.

Now that we understand the organization of the data, we can proceed with the proposed method for processing the input.

Feature extraction from the raw tweets is arguably the most imporant part of the project. Upon inspection of the training data, we realized that words in the tweets are often spelt with the intention of a four year old. Also, punctuation and other characters in the tweets often times do not affect the tweet's classification.

Initally, we rolled our own preprocessing code. We started the preprocessing stage by tokenizing each tweet. This involved two major steps. First, we used the NLTK library's Lancaster Stemmer to stem each word. Second, we use Peter Norvig's toy spell checker to check if the word is spelled correctly.

These methods resulted in throwing out all mis-spelt words and thus decreasing the potential power given by the data. These two steps in the preprocessing stage proved to reduce the number of features from a few hundred thousand to just under seven thousand. However, as discussed in a later section, even seven thousand features proved too computationally complex.

Upon re-evaluation of various feature extraction methods, we decided to make use of Scikit-Learn, a machine learning library for Python. We used Scikit-Learn's feature extraction module to implement a bag of words representation of the tweets. This involved two major steps. First, we vectorized the data which involved tokenizing, counting, and normalizing the tweets. This allowed us to extract key words from all the tweets as well as count the number of occurrences of these extracted words for each tweet. One of the features of the vectorization step is the option to provide a document frequency requirement for the extracted words. We will explore how we chose the values for the minimum and maximum document frequency in a later section. Second we used Tfidf term weighting to re-weight the count features into floating point values suitable for usage by a classifier. Using these methods from Scikit-Learn's feature extraction module, we were able to decrease our features to around a thousand, achieve significant performance improvements, and decrease the number of lines of code substantially.

After performing feature extraction on our data, we fed our preprocessed data matrix to multiple machine learning algorithms. We used separate algorithms for the sentiments/when labels and the kind labels due to the nature of the distribution. As mentioned previously, sentiments and when labels follow a probability distribution while kind labels just need to be between zero and one inclusive.

nand To model the sentiment and time labels, instead of using a black-box library, we implemented our own algorithms in Theano. This allowed us fit the model to the form of our data, and made it possible to test changes made to the algorithm itself, rather than just the hyperparameters of a black-box library.

The first and fastest algorithm implemented was simple logistic regression. Given input x of size n and output y of size m, weight matrix W of size n by m and bias vector b of size m were trained. The prediction function was simply softmax(W*x+b). (Since both the sentiments and times always sum to one, this constraint was enforced by taking the softmax). The cost function was the negative log likelihood of y, given x, and optimization was performed by gradient descent on W and b with respect to this cost. Root Mean Squared Error was also implemented, to cross validate the performance of the algorithm on a holdout validation set.

To get better performance, a hidden layer was added, with its own W matrix and b vector. The activation function and size of the hidden layer could both be parameterized, but the choice of 500 neurons and tanh activation was made without properly testing these hyperparameters. A multilayer perceptron / neural network was then constructed by connecting the output of this hidden layer to the input of the original logistic regression. This led to higher accuracy in all cases, so all further modifications were made to this algorithm.

As input to this model, simple document-frequency / bag-of-words vectorization was used on the tweets, with a high-pass filter to exclude terms that didn't appear often enough to be useful. The minimum document frequency for inclusion was the hyperparameter most thoroughly tested. The values 800, 400, 200, and 100 were tested; smaller values exhausted the ram on development machines, and were not tested. Of these, min_df = 100 gave the best results, cutting 41 thousand words down to just 1113.

The last 'hyperparameter' tested is related to the cost of the original logistic regression code, and has the most mysterious behavior. Last-minute inspection of the code revealed a flaw in the neg_log_likelihood function. The original code took the element-wise multiplication of the predicted and actual values, and then took the sum across the m dimension to get a real number for each test case. This is a bug; a moment's thought makes it clear that this code rewards the wrong behavior. For example, suppose the actual distribution is 0,0.8,0,0,0.2. Maximizing according to the original algorithm peaks at 0,1,0,0,0. Replacing this snippet of code with (1 abs(actual predicted)), at the very least, fixes this bug; the maximum of this snippet is to match the training data. Unfortunately, performing this substitution only helps for some of the data. Performance on sentiments increases appreciably when this bug is fixed. When training on the time data, though, the new code trains for 4 steps in the wrong direction and gives up. In consequence, the final codebase includes an explicit override, forcing the old, incorrect algorithm to be used for the time data. It's ugly, but it's better to go with what works, rather than what theoretically should be correct. In addition, this shows how important cross-validation is; accurate performance measurements make it possible to choose the algorithms that give the best results. nand

Multiple algorithms were considered while approaching how to classify the type of weather.

First, we applied a simple logistic regression from Theano as described above. We used this as a toy algorithm, and only applied it to a single label, namely 'cloudy'. The purpose of applying this toy algorithm was to test our feature extraction code when we were initially rolling our own.

We then looked at the data intuitively to determine another algorithm to predict the type of weather. We determined that labels such as clouds, rain, and, storms may be associated with similar sets of tweets. Other such subsets of labels may be grouped together as well such as cold, ice, and snow. These subsets of labels that may be associated the same sets of tweets and therefore suggested the use of an algorithm that assigns a datum to multiple algorithms. Scikit-Learn provides two similar types of algorithms: multiclass and multilabel classification. Multiclass is able to assign a sample to a single class while multilabel is able to assign to each sample a set of target labels. This can be useful when attempting to predict properties of a datum that are not mutually exclusive. This algorithm fits the intuition behind predicting the type of weather and therefore was chosen. Multilabel classification uses the one-vs-all strategy and is implemented by Scikit-Learn's multiclass one versus the rest module. The strategy involves fitting one classifier per label. For each classifier, the label is fitted against all the other labels. For the classifier, Scikit-Learn provides a stochastic gradient descent module that implements a wide variety of linear models. We decided to use the linear support vector classifier, which is simply a support vector machine with a linear kernel. Even though multilabel classification seems to match the requirements for classifying type of weather, the one caveat is that multilabel classification can only process binary input and output. Therefore confidence scores less than half and more than half were considered zero and one respectively.

Lastly, we decided to apply simple linear regression to each label for type of weather. Since the data for type of weather does not follow a probability distribution and does not resemble binary labels, it made sense to correlate the tweets to the confidence score of a single label using linear regression. Scikit-Learn provides a linear regression implementation in its linear models module.

For both the multilabel classification and linear regression methods, we implemented bootstrapping. We split the training data a certain number of times, and for each batch of the training data we used the corresponding generated model to predict the labels of all the test data, and at the end took the average of all these predictions. This particularially made sense for the binary multilabel classification since the average would result in a non-binary value for each label. Before submitting to Kaggle, we also performed hold out cross validation by splitting the given training data into training and testing data. While submitting to Kaggle, we ensured that the reported RMSE was close to our own calculated error from hold out cross validation to prevent over-fitting.

3 Related Work

There are many different models used in extracting sentiment data from tweets. The most widely used ones include Naive Bayes classiffer, Maximum Entropy, and SVM. Naive Bayes happens to be the most popular of those most likely due to it's simplicity and ease of implementation. The results of a simple algorithm on a very large data set has been shown to produce better results than a more complicated algorithm (Lin Kolcz). Preprocessing of the data is an integral factor in sentiment analysis. There are a multitude of ways to process and tokenize each tweet. Features can be individual words or a sequence of n-grams.

Unigrams, the '...easiest and most used approach (Pang et al.) reported an accuracy of 81.0%, 80.4%, and 82.9% for Naive Bayes, MaxEnt and SVM respectively in the movie-review domain. This was found to be closely similar to accuracies obtained in twitter classiffcation which were 81.3%, 80.5%, and 82.2% respectively.'

The combination of unigrams and bigrams in their feature set showed an improvement with the Naive Bayes and MaxEnt results but a decline in SVM performance.

Research done by another group was performed on a much larger set who's goals were to implement a machine learning pipeline that was scalable. Features used in the implementation were the four character grams present amongst all tweets. These features were treated as binary values so that a simple logistic regression model would be applicable. The ensemble method was used with various sizes and showed significant improvement in hold-out test accuracy. When training on ten million examples, they achieved an accuracy of 78% with an ensemble size of one and an accuracy of around 80% with 95% confidence intervals[7].

In the previously mentioned research, groups performed analysis on twitter data sets in the millions of examples and computed on machines in some cases using computing clusters such as Hadoop. Our training set has just under 70,000 examples. In our case, we must keep our features to a resonably low number due to memory constraints other groups are not limited to.

4 Experimental Results

As previously metioned in the section pertaining to the proposed method we have implemented a first iteration of tokenizing our input data. This involved applying a word stemmer from the NLTK library and a spell checker developed by Peter Norvig. A combination of these two methods reduced the token count from a raw 70,000 spanning the entire training set to just over 6,000. However, we must keep in mind that this method also involves throwing away mis-spelt words as well as getting rid of punctuation including emoticons, which may affect the sentiment analysis. After preprocessing the data, we have plugged our data into Theano's logistic regression tutorial in order to familiarize ourselves with the Theano library. We chose to only learn about 'cloudy' tweets, so we used 500 tweets and the corresponding 'cloudy' weather label to predict 'cloudy'-ness in another set of five hundred tweets from the training examples. This is because the test data provided by Kaggle has not been classified. Theano's logistic regression model correctly labeled 480 out of 500 of the test examples from the training data, achieving 96% accuracy.

We only trained and tested on less than a thousand examples, which resulted in approximately 92% accuracy, but took way too much time to run. SOME OTHER STUFF.

5 Conclusion

From this challenge, we want to demonstrate that it is possible to classify sentiment, time, and weather conditions with a limited amount of words. Being able to quickly analyze short sentences and small phrases is crucial as the general population is slowly receiving news and information from social medias like Twitter, Facebook, Google+, and etc. The results that we have achieved from Theano's logistic regression thus far is very hopeful. Currently, we have performed a first iteration of tokenizing the raw tweets. We have also been able to apply a machine learning algorithm, namely logistic regression to classify one attribute of the tweets. Moving forward, we wish to analyze the full data set provided by Kaggle. This will entail further tokenizing tricks including PCA and sparse coding as well as the application of more complex algorithms including multilayer perceptrons and GMM.

References

[1]Kaggle, Partly Sunny with a Chance of Hashtags, `http://www.kaggle.com/c/crowdflower-weather-twitter`

[2]Theano Deep Learning, `http://deeplearning.net/tutorial/`

[3]NLTK Lancaster Stemmer, `http://nltk.org/api/nltk.stem.html`

[4]Peter Norvig, *How to Write a Spelling Corrector*, `http://norvig.com/spell-correct.html`

[5]Hannak et al., *Measuring and predicting sentiment on Twitter*, `http://www.janysanalytics.com/ussi/pdf/Measuring-and-predicting-sentiment-on-Twitter.pdf`

[6]Lei Zhang et al., *Combining Lexicon-based and Learning-based Methods for Twitter Sentiment Analysis*

[7]Jimmy Lin & Alek Kolcz, *Large-Scale Machine Learning at Twitter*

[8]Anthony K Jose et al., *Twitter Sentiment Analysis*