Partly sunny with a chance of Hashtags: Weather and sentiment prediction by analyzing tweets

Namita Mhatre
110929172
Grad Student, CSE, SBU
nmhatre@cs.stonybrook.edu

Asmita Negi
110937409
Grad Student, CSE, SBU
anegi@cs.stonybrook.edu

Apoorv Agarwal
110932431
Grad Student, CSE, SBU
aagarwal@cs.stonybrook.edu

Abstract

Now a days, Social media is the primary channel of communication. While much of the data here is useless, we can use state of the art machine learning techniques to mine and make this information useful. There are approximately 200 million active users on Twitter and 400 million tweets are posted on a daily basis. With such a large volume of tweets there is huge potential for gathering information about anything. This challenge is about analyzing these tweets to help us give some useful information to a user. By analyzing tweets related to weather, we can get information about the weather, people's reaction about that particular weather, time when the weather occurred (past, present, future), the location, etc. This information can be used to predict the weather of a given location by analyzing the current data-set.

1 Introduction

This challenge which was hosted on Kaggle by Crowdflower, involves the analysis of a tweet and determining 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.

Problem Definition

This challenge involves the analysis of a tweet and determining 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.

Lets consider a tweet:
"@mention another storm? At least we're getting rain. How bad is this one?"

- This tweet was recorded from location Austin in state Texas.
- The sentiment S2 got the maximum score of 0.795, thus rating the sentiment on this tweet as negative.
- The value of w1 came out to 1, which means that the tweet is talking about the weather of the same day.
- As the keywords are picked from the tweet, the classifier predicted the kind of weather to be Stormy and Rainy.

We have 77,947 tweets in train set which are labeled and 42,157 tweets in the test set. For the training data-set, the confidence score for each label was based on the review of multiple raters.
The score is weighted on the reliability of raters. We have total 24 labels, 5 for sentiments, 4 for time and 15 for kind of weather. While sentiment and when can have only one type of label for a tweet, kind can have many labels assigned to one tweet.
The input to our classifier will be 77,947 tweets which are labeled. Features will be extracted. Now, for the two applications for our system :

- You can either input a tweet and get the confidence score for the 24 labels with the sentiment, time and weather predicted.
- You can input the location and the time and the weather will be estimated for that particular time and location from a quantitative analysis of the test data-set.

Motivation

With this huge amount of data available, by applying suitable machine learning techniques it is possible to put this data to use.
Language on social media is not always standard. And the information is many time useless or unreliable. But with proper pre-processing and filtering; and a suitable machine learning model, we can turn this unrefined data into something very useful such as our weather prediction. Thus, we will learn how to actually apply machine learning in our day-to-day lives to get the best out of something trivial.
Using this data from twitter, we can predict the sentiments, and the kind of weather for a particular tweet. Also, we have an added feature where the user inputs the location he wants the information about and after an analysis of the output labels of training and testing data, we can give a comprehensive information about the weather of that location. Thus, we will be doing a quantitative analysis and will be giving out a report which states the type of weather in that location for a given time.
Hence, by analyzing a good amount of tweets from one location, we can know the weather of the location currently

or in future and also know if it is pleasant to the people or not. Also, someone traveling to a new location will be able to get a quantitative analysis about the weather there.
Unlike other weather prediction applications which usually depend on the previous day's weather, current weather conditions and natural occurrences, this depends on feedback by people which is reliable (mostly) as they themselves feel it, given that the labels have been rated by reliable people (as mentioned on the Kaggle website).

Contribution

For this challenge, we built this weather prediction application which predicts the kind of weather and a person's sentiments about it from a tweet. We used four classifiers and a variety of parameters and filters for feature extraction and selection. After working on many combinations of different classifiers with different vectorizers, we came up with a best configuration which used the Ridge Classifier with a TFIDF Vectorizer. We have taken help from the sklearn libraries from python for this implementation. The evaluation was done on Kaggle itself.
Apart from the Kaggle challenge, we have implemented our very own feature of weather prediction based on the location. After learning the classifier from the training data, we predict the test data. Now we have 42,333 labeled tweets in our test data. We do a comprehensive analysis of this for the prediction task. Consider a user wants to know the current weather of New York and how it actually feels. By running an analysis on the labeled test set, we can segregate the current tweets from NYC and we can find the average sentiment people are having about it's weather and the current weather. Thus, this feature will be more like a weather prediction app in general, unlike the task which is about predicting the weather from a single tweet.

Outcomes

This application works on any type of tweet that is entered as an input. Considering that a tweet is a type of sentence (up to 140 characters), we can also input any sentence as an input to find what kind of weather it conveys and a person's sentiment behind it.
Our application works successfully for all types of tweets in the test set as well as for any manual input. For the weather prediction based on the location part, it mostly works well for all location except if we do not have enough data in the labeled test set for that location. Also, while mentioning the location, many people tend to write acronyms instead of the entire name or even use slang (like The Empire State instead of New York). So it is difficult to work with a small data-set to predict weather on basis of location.

2 Description

Data-set

This is a multi-label classification problem where we have 24 different types of labels which belong to three different categories.

- Sentiment of the tweet
- when : the time to which the tweet refers to (past, present, future)
- Kind of weather the tweet is talking about.

While sentiment and when can have only one type of label for a tweet, kind can have many labels assigned to one tweet.
Following shows the labels of the three classes:

Sentiments:
```
s1,"I can't tell
s2,"Negative"
s3,"Neutral "
s4,"Positive"
s5,"Tweet not related to weather
condition
```
When:
```
w1,"current (same day) weather"
w2,"future (forecast)"
w3,"I can't tell"
w4,"past weather
```
Kind of weather:
```
k1 = "clouds"
k2 = "cold"
k3 = "dry"
k4 = "hot"
k5 = "humid"
k6 = "hurricane"
k7 = "I can't tell"
k8 = "ice
k9 = "other"
k10 = "rain"
k11 = "snow"
k12 = "storms"
k13 = "sun"
k14 = "tornado"
k15 = "wind
```

The score is weighted on the reliability of raters. We have total 24 labels, 5 for sentiments, 4 for time and 15 for kind of weather. While sentiment and when can have only one type of label for a tweet, kind can have many labels assigned to one tweet.

Methodology and Algorithm

This is how our algorithm for this problem works (Learning and classification):

- The train and test data present in the CSV files are read.
- First, is the process of feature extraction. We process the input data by vectorizing it. We have two vectorizers, Count Vectorizer and TFIDF Vectorizer. We used 5 combinations of these to try and extract better features for a better model. Following are the two combinations which were better than others:

```
vector = TfidfVectorizer(max_features=10000,
    strip_accents='unicode', analyzer='char_wb',
    sublinear_tf=1,ngram_range=(2, 9))

vector = TfidfVectorizer(max_features=80000,
    strip_accents='unicode',
    analyzer='word',token_pattern=r'\w{1,}',
    sublinear_tf=1,ngram_range=(1, 3),
    tokenizer = SnowballTokenizer())
```

- Two tokenizers are initialized; the Snowball tokenizer and the Wordnet Tokenizer. As the Snowball Tokenizer gave better results, it was used in the final configuration.

```
class SnowballTokenizer(object):
    def __init__(self):
        self.wnl = nltk.stem.SnowballStemmer
        ("english")
    def __call__(self, doc):
        return [self.wnl.stem(t) for t in
        wordpunct_tokenize(doc)]
```

- Then, we have three different types of classifiers declared, so as to compare them and analyze the results. We have used Ridge, Extra tree Regressor and Random Forest Regressor. After tuning the hyper parameters, following were the most optimized configurations we found:
  - Extra Tree Regressor:

```
if choice == 1:
        classifier = ExtraTreesRegressor(
        max_depth = 25, n_estimators = 28,
        max_features = 100, n_jobs = 5)
```

  - Ridge Classifier:

```
if choice == 2:
        classifier = Ridge()
```

  - Random Forrest Regressor:

```
if choice == 3:
        classifier = RandomForestRegressor(
        max_depth = 30, n_estimators = 30,
        max_features = 100, n_jobs = 5)
```

- Now after declaring all the feature extractors and classifiers, we need to train our model.
  We decided to use cross-validation. 20% of the train data was used for cross-validation.
- Tweets were extracted from the training data and were transformed according to the current chosen vectorizer.

```
X = tfid.transform(trainTweets)
```

- Using LSA and truncatedSVD the dimentionality was reduced.

```
LSA = TruncatedSVD(n_components = 500)
```

- The data was then normalized and fit with the chosen classifier. The classifier is now ready for predicting the test set. The cross validation set was predicted to find the cross validation error.
- This classifier and the vectorizer were dumped into a pickle file which was then used in our application for predicting the weather and sentiment of a single tweet.

Classifiers and other parameters

- **Feature Extraction : TFIDF Vectorizer**
  Using this vectorizer, we extract the features from our training set. We used the following parameters for extraction, we can tune these parameters :
  - 10000/25000/51000/80000/10500 maximum features
  - Unicode accents only
  - Analyzing words/characters only
  - Tokens with word length 2/3 or more
  - n-gram baselines
  - Snowball Tokenizer/Wordnet Tokenizer .
- **Snowball Tokenizer:**
  The language people use in social media gives a lot of weightage to emoticons. Emoticons/Smileys play an important role in conveying sentiments. When we use stop words or some other type of feature filtration, as the emoticons are formed using punctuation symbols; these are filtered and are not counted as features. Same happens when we use a unigram model. A smiley :) is formed using a colon and a space separated closing brace. In unigram model, these will be accounted for separate features. Thus, we have to consider at least bigram model to account for emoticons. Multi gram configuration not only provide weightage to just unigram but also to the context in which the message is being written. So as we are dealing with tweets which have a huge amount of emoticons, we are using this tokenizer to help us predicts the sentiment in a better way. A snowball tokenizer is supported in 19 languages and it implements different language specific flavours. This tokenizer helps extracting features from emoticons.
- **Dimentionality Reduction : TruncatedSVD**
  In order to reduce the dimentionality of vector the approach taken is to compute eigenvalue or singular value decompositions. The task is to diagonalize the matrix and this essentially yields the whole eigenvalue / singular value decomposition (the whole eigenvalue spectrum) at the same time . Basically, a truncated SVD of the term-by-document matrix reduces the m-by-n matrix A to an approximation Ak that is stored compactly with a smaller

number of vectors. If we choose a higher k, we get a closer approximation to A. On the other hand, choosing a smaller k will save us more work. Basically, we must choose between time and accuracy.

- **Cross Validation :**
  Cross validation is where we divide the data in a:b, use b part for validation and rest for training. We have used 20% of the data for cross validation to improve the accuracy.
- **Classification :**
  A classifier is a Supervised function where the learned (target) attribute is categorical. It is used after the learning process to classify new records by giving them the best target attribute (prediction). The target attribute can be one of k class membership.
  - **Ridge regression :**
    This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm.
  - **Random Forrest Regressor:**
    A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the data-set and use averaging to improve the predictive accuracy and control over-fitting.
  - **Extra Tree Regressor :**
    This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the data-set and use averaging to improve the predictive accuracy and control over-fitting. It gives a very good accuracy for multi-label classification.

3 Results and Evaluation

Following were the evaluation criteria, the results obtained on various tests and the analysis done on the results.

Performance evaluation metric

To evaluate the performance of a classifier we can either calculate the accuracy, precision-recall, the running time of the algorithm etc.
The Root Mean Squared Error ("RMSE") is used to measure the accuracy for our classifier. (Refer Figure 1)

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(p_i - a_i)^2}{n}}$$

Figure 1: Evaluation Criteria

Where:

- n is 24 times the total number of tweets.
- p_i is the predicted confidence rating for a given label.
- a_i is the actual confidence rating for a given label.

The RMSE gives the weighted error between the predicted and the actual labels. So, the lower the value of RMSE, better the classification algorithm.
In the Kaggle challenge, the winner had an RMSE value of 0.14314. Deep learning was used in that model with a combination of classifiers.

Data-set for evaluation: For evaluating our model, we trained it on 77,947 tweets and predicted labels for a test set of 42,157 tweets. This labeled test set was then evaluated on the Kaggle website at the challenge to get the Root Mean Square Error.

Comparing Models: We compared three different classifiers with different settings, thus giving us many combinations of classifiers. We used Extra Tree Regressor, Random Forrest Regressor and Ridge classifier as our classification models. We modified and tuned the TFIDF vectorizer parameters for different types of filtration of features (we have stated the best parameters below). Thus we have three combinations (and their RMSE values):

- Extra Tree Regressor with TFIDF_1 and Snowball Tokenizer : 0.17208
- Random Forrest Regressor with TFIDF_1 and Snowball Tokenizer : 0.17651
- Ridge classifier with TFIDF_1 and Snowball Tokenizer : 0.1604

where TFIDF_1 = 10,000 maximum features, unicode accent, 2-9 gram baseline with a character analyzer.

Why does ridge perform better?
Text classification problems are usually of high dimensionality. High dimensional problems tend to be linearly separable and hence linear classifiers like ridge classifier perform better than most other classifiers.
One of the reasons that ridge regression works well is that non-linear methods are too powerful and it is difficult to avoid over-fitting. There may be a non-linear classifier that gives better generalization performance than the best linear model, but it is too difficult to estimate those parameters using the finite sample of training data that we have. In practice, the simpler the model, the less problem we have in estimating the parameters, so there is less tendency to over-fit, so we get better results in practice.
Another issue is feature selection, ridge regression avoids over-fitting by regularizing the weights to keep them small, and model selection is straight forward as we only have to choose the value of a single regression parameter. If we try to avoid over-fitting by picking the optimal set of features, then model selection becomes difficult as there is a degree of freedom (sort of) for each feature, which makes it possible to over-fit the feature selection criterion and we end up with a set of features that is optimal for this particular sample of

data, but which gives poor generalization performance. So not performing feature selection and using regularization can often give better predictive performance.
The difference between random forest and Extra tree is that the split in Random Forest is deterministic whereas it is random in Extra Trees. Also because of high number of noisy features in this high dimensional problem, Extra tree regressor performs a little worse than Random Forest.

Figure 2: Comparing models on training and cross validation error

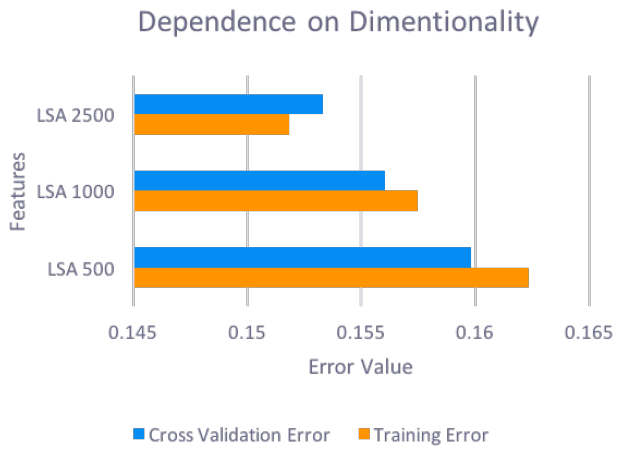

Figure 3: The effect of dimentionality on performance

Examples :

1. For predicting a tweet :

```
TWEET IS: SUNSHINE! It's been so long since ive seen you. Now if you could be warmer than 38 degrees right now I PROMISE I'll stop complaining.
Sentiment prediction:  Positive
When prediction: current (same day) weather
Weather prediction:  sun
```

2. For predicting weather of a location :

```
STATE : Chicago
NUMBER OF TWEETS : 500
When prediction:
Current (same day) weather : Sun
Past weather : Hot
Future (forecast) weather : Sun
```

Results

The best value we obtained for the Root Mean Squared Error is : 0.15998.
This best configuration uses:

- TFIDF vectorizer (10,000 maximum features, unicode accent, 2-9 gram baseline with a character analyzer).
- LSA which uses truncatedSVD for reducing the dimensions of the features.
- a ridge classifier for the multi-class classification.

Different feature extraction parameters and different classifiers were used and following graphs show the comparisons.

4 Conclusion

Different classifiers, feature extractors and selectors were used in plug and play format to learn their relative efficiency. For a model with multi-class classification and such a high amount of data, the Ridge classifier gave the best performance.
Language on social media is not always standard. And the information is many time useless or unreliable. But with proper pre-processing and filtering; and a suitable machine learning model, we can turn this unrefined data into something very useful such as our weather prediction.
For future work, we can use a more complex procedure for predicting the weather based on location. We can add weights to every tweet label based on the confidence score and predict the weather accordingly.

Thus, we learnt how to actually apply machine learning in our day-to-day lives to get the best out of something trivial.

5 References

- http://stats.stackexchange.com/questions/17711/why-does-ridge-regression-classifier-work-quite-well-for-text-classification, Stackoverflow Website
- http://stats.stackexchange.com/questions/175523/difference-between-random-forest-and-extremely-randomized-trees, Stackoverflow Website
- https://www.kaggle.com/c/crowdflower-weather-twitter, Kaggle Challenge