Location Classification on Tweets

Erdi Çallı e.calli@student.ru.nl

Abstract

We extracted textual features from tweets which hint to the location it was sent from. Unlike some of the previous studies, we have only used individual tweet texts and geolocation of those tweets. We have also simplified the question to a binary classification problem. Our results are showing up to 74% recall rates in classifying if a tweet is sent from Manhattan, NY.

1. Introduction

Twitter is an online platform that lets users share posts with their followers, limited to at most 140 characters, called tweets. Tweets can be almost anything that we share on the internet, for example: a personal message stating our views, a link to an article or a picture. There are no boundaries except the character limit. In 2009, Twitter introduced an option to include geographic location (geolocation) information with tweets[1]. Since then, a lot of twitter users have been sharing their geolocation with their tweets (or geotagging their tweets). Depending on the context of the tweet, having the geolocation may be very valuable. For example a tweet about a Pizza franchise or a political campaign, if geotagged, may point to an improvement with exact location or contribute to the statistics of a specific area. Unfortunately, geolocation of a tweet is not shared by default. In other words, most of the tweets are not geotagged, unless explicitly specified. Inferring the geolocation of a tweet is what this paper is about.

To be able to infer this information, we can try to use linguistic habits shared by the people who are in the same location. As humans, our spatial cognition deals with updating and managing the knowledge about our environment. Spatial cognition changes when we move between environments, but as we stay within the boundaries of a high-level environment(i.e. state, city, borough), there may be some high-level constants that doesn't change. If some of these constants are shared commonly between people, it may lead them to have similar linguistic habits, in this case, textual features in their tweets. If we group the geotagged tweets sent within a high-level location, it may be possible to extract these features, if they exist.

Using similar features, previous research tries to determine the geolocation of a tweet using multi-class classification algorithms. These methods are trying to determine the most likely geolocation for a tweet from a list(e.g. cities in a country). It is sometimes hard to do that, because we need data from all cities. Also the algorithms and the implementations become more complicated. Thus, in cases where we just want to know if someone is within a state/city/borough, this approach becomes unnecessarily heavy. In this research, we

1. `https://blog.twitter.com/2009/location-location-location`

are simplifying this problem by using binary classification algorithms, which are trying to determine if a tweet is sent within the boundaries of a location(i.e. Manhattan, NY).

Revolving around our hypothesis, the research question we're trying to answer is: "Are there textual features in tweets, that are common for people in the same location? And if there are, can they help us determine if a tweet is sent within that high-level location?"

2. Related Work

Similar to our work, using textual features in tweets and geotags, Cheng et al.approached this question as a regression problem[CCL10]. They have located users within a 100 mile radius and achieved up to 51% accuracy. To do that, they used only the textual features and introduced a method that identifies these features(i.e. unique words) with strong local geo-scope.

Hecht et al. introduced the *CALGARI* classification algorithm specific to this problem, focusing on a higher granularity of country and state prediction[HHSC11]. They got a 72% accuracy for classifying 4 countries and 30% accuracy on classifying 18 states. Their method was depending on the location information revealed by users in their tweets, and classifying user's location based on their past tweets.

Eisenstein et al. created a model using textual features in different topics(e.g. Sports, Entertainment)[EOSX10]. Their research got a median distance error of 494 km for individual tweets.

Wing et al. created another model by combining geotagged Wikipedia articles and tweet texts[WB11]. They have created geodesic grids varying in size, and they have obtained a median error of 479 km for individual tweets.

Mahmud et al. introduced hierarchical classification methods combined with statistical methods[MND12]. They have used tweet texts as well as tweeting behaviour to achieve 58%, 66% and 78% accuracy on city, state and timezone levels, respectively. Same researchers revisited the problem in, trying to classify the home location of users, achieving similar results and also introducing movement and location prediction[MND14].

Also, Graham et al. studied the problems with the baselines accepted by recent studies and explained why location and language classification is not an easy problem and how some of the assumptions are incorrect, such as accepting the device location as user's location[GHG14].

3. Methods

3.1 Data Collection and Filtering

Using Twitter's sample status streaming api[2], we have sampled the public tweets. This api samples approximately 1% of all the public tweets around the world. To be able to find the tweets that are relevant to our task, we have applied some filters to this api.

First of all, we have applied a filter to ignore non-geotagged tweets. The high-level location information contained within each geotag will be used as the label of a tweet.

2. https://dev.twitter.com/streaming/reference/get/statuses/sample

These labels will be used to train our algorithms, and they will be compared with the predicted results to see how accurate our methods are.

Also, we are not interested with tweets in different languages. Having tweets from different languages would introduce a complicated variable and make our results hard to interpret. Therefore, we choose English as our language of interest, because we assumed that English is the most common language that people tweet, thus it would yield the biggest dataset(compared to other languages). Samples from twitter contain the language information, we have used this information to filter tweets which are not English.

Considering the language assumption, having tweets from non-English speaking countries would be unnecessary. Also, having tweets from different countries would introduce another variable. Assuming that United States would yield the biggest dataset(compared to other countries), we have limited the tweets we collected to United States.

Running the data collector for 1 month, we have collected a dataset consisting of 33112 tweets, geotagged within United States, tagged as English and each from a unique user.

Assuming we had a dataset, big enough for the task, we examined a small sample of it. In this sample, we have seen many examples which were not composed by humans, but bots(e.g. "*Police department activity on #I95 SB at 3rd Avenue; Exit 3*"). Twitter also gives the information on which application is used to send a tweet. We have examined this information, called *source*, to filter these tweets. Grouping and sorting them by the number of tweets, we were able to see which sources are most relevant for our study. Annotating the most relevant 20(e.g. *Twitter for iPhone*, *Facebook* and etc.) we have created the whitelist filter for tweet sources.

In our examination, we have also seen a variety of geotag names which represent the high-level locations we're interested in(e.g. *Mississippi, USA*, *Manhattan, NY* and etc.). When we checked the number of tweets per location, we have seen that there are 3306 unique locations in the whole dataset with different granularities(e.g. state, city, borough and etc). Upon examining these locations, we have seen that it was not worth merging these locations with small granularities(e.g. boroughs) to higher levels(e.g. cities). Also we are not interested in location definitions higher than cities. Therefore, we have picked 8 of the most common locations(e.g. *Manhattan, NY*, *Los Angeles, CA*, *Houston, TX* and etc.) ignoring states.

When we applied all these filters, we ended up having a dataset containing 8 locations, each of them having between 976 to 212 tweets, summing up to 3176.

Since each tweet is limited by 140 characters, and our dataset is shrank to only 4751 tweets, our dataset might not be enough to extract the textual features we are interested in. To enrich our database, we assumed that our users are always sending tweets from the same location. Therefore, we have collected past 40 tweets of these users and attached them to the sampled tweet as *past tweets*.

In all our experiments, we have splitted our datasets 0.5 by 0.5 training and test datasets.

3.2 Tools

sklearn python mongo

3.3 Empirical Bayes

In the process of filtering data, we have implemented an Empirical Bayes[CL97] classifier using MAP estimation[DeG] to perform multi-class classification. We chose Empirical Bayes because it was easy to implement and it was not assuming any independency between data points, which was required when using past tweets, which may be dependent to each other. Implementing this algorithm also helped us to understand the nature of data, and also we somehow tried to reproduce results similar to previous research.

$$\begin{aligned} \mathbf{T}_i &= \text{tokens of } i\text{th tweet} \\ t_{ni} &= n\text{th token of } i\text{th tweet} \\ L &= \mathit{Location} \\ p(L|\mathbf{T}_i) &= \frac{p(\mathbf{T}_i|L)p(L)}{p(\mathbf{T}_i)} \end{aligned}$$

To train this algorithm, we created a likelihood($p(t|L)$) and an empirical prior. To calculate the likelihood, we have tokenized each tweet using the *twokenize* module[GSO+11], labelled them with the location and calculated the "probability of a token given location". As for the prior we have generated the empirical prior, which is the occurrences of a location divided by the number of data points. As a result of training, we have stored the likelihood and prior.

To predict where a tweet has been sent from we have applied MAP estimate to the Empirical Bayes. For each tweet in our test dataset, we have tokenized the tweet, queried the likelihood values for each token and location, combined these probabilities per location, applied the empirical prior and choose the location with the maximum probability.

$$\hat{L} = \arg\max_{L} \{p(\mathbf{T}_i|L)p(L)\}$$

$$p(\mathbf{T_i}|L) = \prod_{n} p(t_{ni}|L)$$

To test our assumption about past tweets, this experiment was ran with 2 configurations one using past tweets and other using sampled tweets. In the first configuration, we combined each past tweet to one document and labelled them with the high-level location of the related sample(tweet). We have used this configuration to test our assumption of users are always sending tweets within same high-level location. In the second configuration, we only used the sample tweets and their high-level locations as labels.

First run of the experiment, without applying filters to the dataset, resulted with 0.02 and 0.04 macro-average precision rates for the first and second configurations, respectively. After this experiment, we have seen that the assumption about past tweets did not improve our results. The rest of this Empirical Bayes experiment was ran only with the second configuration, sampled tweets.

One problem with this experiment was, if a token(t_x) didn't appear in the training dataset, we didn't know the likelihood of that token. So if there is a token did not appear in the training, the tweets containing that token got a probability of 0 for all locations, regardless of the other tokens. To solve this problem, we have ignored those tokens in our calculations. This solution, is not mathematically correct because ignoring a token means

setting it's probability to 1. Still, replacing those probabilities improved the macro-average precision to 0.05.

When we applied the location whitelist, we have achieved a macro-average precision of 0.11. Which doesn't seem better than making random classifications for 8 locations or picking the location which occurred the most.

So our experiment with Empirical Bayes classifier failed to produce results better than any baseline, but it helped us understand the nature of data and what kind of filters we need to apply.

3.4 Scikit Learn - Experiment Setup

After failing to produce results better than making random choices, we decided to use the Scikit Learn [PVG+11](sklearn, v0.17) library to actually test our hypothesis using binary classification and compare our results with multi-class classification.

In this set of experiments, we have re-define our experiment setting. Since we started using a library, we have access to more complicated tools which helps us to define better experiment settings. This experiment uses sparse arrays to represent tweets. For each tweet, there is a 1 dimensional sparse array with columns representing tokens(in our dataset). Combining these, we have created two 2 dimensional sparse arrays, representing our training and test datasets. As for the values representing tokens, we have used tf-idf scores.

We chose tf-idf scores because we wanted to reduce the importance of stopwords and frequently used words. This experiment learns idf scores from the training dataset, and uses these scores to calculate the tf-idf scores for training and test datasets.

Using tf-idf scores and sparse arrays, we can't capture textual features created by a sequence of words. A good example to that would be *The Empire State*. *The*, *Empire* and *State* are words that could be used without a spatial context. Our sparse array would only contain individual information about the tf-idf scores of these tokens. So we are losing the information of using them together, which may be a textual feature, pointing to Manhattan, New York.

To overcome this limitation, we included word ngram tf-idf scores in our sparse arrays. Doing so we got information about word sequences, which would possibly solve the problem. We have experimented with 2 configurations, first one using only the tokens or word unigrams, second one using word unigrams with word bigrams.

We also tried to make use of past tweets by labelling them individually with the location of their sample tweet. We created 2 configurations for this, one using the sample tweets, other using the sample tweets.

Putting it all together, we had 8 different configurations, combining the word ngrams, past/sample tweets and binary/multiclass classification. Again using the 0.5 by 0.5 split for train and test datasets.

3.4.1 Multinomial Naive Bayes

To test our experiment setup, we chose MultinomialNB(Multinomial Naive Bayes) implementation for multi-class classification. MultinomialNB considers the

This implementation has achieved higher macro-average precision results compared to our Empirical Bayes experiment. Using unigrams to perform multi-class classification, we

had macro-average precisions 0.19, 0.24 using past tweets and sample tweets, respectively. But when we investigated the predictions, we have seen that the classifier was almost always predicting the location with highest amount of tweets(Manhattan, NY).

3.4.2 Normalisation of Documents per Class

To prevent the problem we faced in MultinomialNB test, we have normalised the dataset number of documents per class. To do so, we normalized the number of documents per class by limiting the *maximum number of documents per class* to the *class with least amount of documents*. That reduced the elements in our training dataset to *number of classes*(8) times *number of documents in class with least documents*(e.g. *Atlanta, GA* with 122 samples) for multi-class configuration. For binary classification, we have used the class with maximum number of documents(e.g. *Manhattan, NY* with 976 samples), and combined it with same amount of random samples for non-class training data. Upon normalisation, our multi-class training dataset shrank considerably, but we have achieved macro-average precision rates of 0.29, 0.42 for past tweets and sample tweets, using unigrams.

3.5 Scikit Learn - Algorithm Comparison

To see which approach is best for our research question and what kind of token based textual features are useful for our task, we compared some classification methods implemented in Scikit Learn[3]. Scikit Learn implements various classification methods such as, Logistic Regression, Perceptron, Support Vector Classifiers and etc.

Scikit Learn implements different model configurations. A good example is, the loss function of SGDClassifier. There are 9 different loss functions implemented that we could use with SGDClassifier. In our experiment, we tried to capture some of these different parameters, and tried to explain them. But in most cases, we didn't try to fine tune the hyper-parameters.

1. LogisticRegressionCV: Uses a logistic function that combines the tf-idf scores with weights in a logistic function. This model is useful if our tf-idf scores have exponential effects.

2. SGDClassifier: Stochastic Gradient Descent creates a linear function and gives initial weights to each token for each class. Based on the loss functions defined below, it tries to optimize these weights.

   (a) SGDClassifier(loss="hinge"): Hinge is a commonly used linear loss function for classification.

   (b) SGDClassifier(loss="log"): Converts SGD to a probabilistic logistic regression classifier.

   (c) SGDClassifier(loss="modified_huber"): Hinge with a setting, $\gamma = 2$

   (d) SGDClassifier(loss="squared_hinge"): Quadratically penalised hinge.

   (e) SGDClassifier(loss="perceptron"): Linear loss function used with the Perceptron algorithm.

   (f) SGDClassifier(loss="squared_loss"): Least squares loss function.

   (g) SGDClassifier(loss="huber"): Similar to squared_loss but switches from squared to linear after a threshold.

   (h) SGDClassifier(loss="epsilon_insensitive"): Linear loss function, ignoring errors less than the threshold, epsilon.

   (i) SGDClassifier(loss="squared_epsilon_insensitive"): Squared version of epsilon_insensitive.

3. Perceptron: Same as SGDClassifier(loss="perceptron"), updates the tf-idf weights with a constant learning rate and has different default parameters.

3. http://scikit-learn.org/stable/supervised_learning.html

4. PassiveAggressiveClassifier: Similar to Perceptron. Instead of using a learning rate, uses a regularisation term that makes sure that tf-idf weights don't get too big and prevents overfitting.
   (a) PassiveAggressiveClassifier(loss="hinge")
   (b) PassiveAggressiveClassifier(loss="squared_hinge")
5. SVC: Classifier model using Support Vector Machines. Converts tf-idf scores based on the kernel function to create a high or infinite dimensional hyperplanes by clustering classes on that space.
   (a) SVC(kernel='poly'): Polynomial kernel for SVM.
   (b) SVC(kernel='sigmoid'): Sigmoid kernel for SVM.
6. LinearSVC: SVM with a linear kernel.
7. KNeighborsClassifier: K-Nearest Neighbours classifier, uses weights function to cluster and classify tf-idf scores using the function given in weights. Finds the $n_neighbours$ number of tweets with most similar tokens from training dataset, casts a vote among them make predictions.
   (a) KNeighborsClassifier(n_neighbors=8, weights="uniform"): Uses uniform weights to classify neighbours.
   (b) KNeighborsClassifier(n_neighbors=8, weights="distance"): Inverse of distance weights. Closer neighbours have a greater influence on classification.
8. MultinomialNB: Gives probabilities for each token and class, combines them to find the most likely class.
9. BernoulliNB: Similar to Multinomial, but instead of using tf-idf scores, considers if a token occurs in a tweet or not.
10. DecisionTreeClassifier: Creates a rule based decision tree inferred from data for predict.
11. AdaBoostClassifier(n_estimators=170): Fits N($n\_estimators$) Decision Tree Classifiers to samples from training data. Then assigns them weights based on errors. Using those weights, casts a voting to find the best prediction.

We have tried those 22 models, with 8 configurations to create 176 different combinations.

3.5.1 Execution

To execute those tasks, we have used Google Cloud instances. Google Cloud offers 300$ trial usage for users who want to try out their services and it allows up to 32 cores in this trial period. We have implemented an argument to execute all those tasks on multiple google cloud instances, in parallel. Created a template operating system including an executable version of our implementation. Using 15 two core execution instances from the execution template, 1 MongoDB instance, it took 35 mins to finish the execution of those 176 tasks.

4. Results

4.1 Binary Classification Baselines

We have considered 2 baselines to compare our results for predicting *Manhattan, NY*. First one is making random choices with a baseline macro-average precision/recall and f1-score of 0.5.

The second baseline is through choosing tweets containing important keywords. To do so, we have created a regular expression and ran it through out dataset. This regular expression created a baseline macro-average precision of 0.8. Compared to the high macro-average precision, it had a very low recall 0.37 for predicting our location, *Manhattan, NY*. Since we are interested in being able to say if a tweet is sent within a location, we're not going to use this baseline for comparison.

```
.*(([Mm]anhattan)|([Nn]ew[ ]?[Yy]ork)|(NY)).*
```

Combining the random baseline and regular expression, we have created a new baseline. In this baseline, if the regular expression matches we classify this data as *Manhattan, NY*, otherwise we make random choices. This combined baseline gives us a 0.68 recall, 0.57 precision and 0.62 f1-score for *Manhattan, NY* with 0.59 macro-averaged recall, 0.59 macro-averaged precision and 0.59 macro-averaged f1-score.

4.2 Binary Classification

Including bigrams, BernoulliNB, SGD with modified huber loss function and SGD with hinge loss function have achieved the top 3 macro-averaged f1scores of 0.734, 0.726, 0.72, respectively.

Only with unigrams BernoulliNB, MultinomialNB and PassiveAgressiveClassifier with hinge loss function achieved the top 3 macro-averaged f1scores of 0.726, 0.719 and 0.717, respectively.

4.3 Multi-Class Classification

Random baseline combined with obvious location names, for 8 locations, gives us a macro-averaged f1-score of 0.17. In our experiments with multi-class classification, we were able to improve this score to 150% using SGDClassifier with different loss functions.

4.4 Assumption of Past Tweets

Using past tweets as our dataset reduced the classifier performance of almost all configurations. Overall average decrease in f1scores was 0.076 with values ranging from 0.21 to -0.09. Our assumption was wrong.

4.5 Textual Features

When we investigated why BernoulliNB is achieving the highest performance, we have seen that in most of the cases(around 90% of the correct predictions), there was a building or place name included in the tokens(e.g. *Central Park*, *Empire State*). In the rest of the cases(around 10%), there were no obvious textual features. In those cases we assumed that there was a unique token that only appeared in tweets labelled as *Manhattan, NY*. BernoulliNB is the simplest of all, it only considers if a token appears in a location or not.

5. Conclusion

In our research question, we asked if there are any textual features that are shared by the people who are within a high-level location. In our experiments, we couldn't find any specific linguistic or textual features in English that hints to the geolocation(i.e. *Manhattan, NY*) of a tweet, except the location names.

Using the location names included in tweets, were able to improve the combined baseline(macro-averaged f1score) for binary classification around 0.16.

6. Discussion

The fact of all our data was geotagged may be adding a bias to our dataset and compromising the validity of our results. It may be possible for geotagged tweets to include more location names. Refining the training dataset with more user generated content(i.e. removing Foursquare or Yelp from source whitelist) and collecting more data would yield more authentic results and more textual features.

We have seen that BernoulliNB had one of the top results in binary classification. This hints to having tf-idf scores may not be a good idea, because this method ignores these scores and uses binary values. Instead of having tf-idf scores, having binary values in the sparse array and trying different configurations on this dataset might yield to a better improvement on the baseline. But as we have concluded, we couldn't see any unique words that identify the high-level location of user.

To improve Empirical Bayes, we have ignored tokens that are not appearing in a location. This solution was wrong because it sets the likelihood to 1 when it can't find a token on a location. A solution would be setting the likelihood as if that token appeared only once.

References

[CCL10] Zhiyuan Cheng, James Caverlee, and Kyumin Lee. You are where you tweet: a content-based approach to geo-locating twitter users. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 759–768. ACM, 2010.

[CL97] Bradley P Carlin and Thomas A Louis. Bayes and empirical bayes methods for data analysis. *Statistics and Computing*, 7(2):153–154, 1997.

[DeG] Morris DeGroot. H (1970): Optimal statistical decisions.

[EOSX10] Jacob Eisenstein, Brendan O'Connor, Noah A Smith, and Eric P Xing. A latent variable model for geographic lexical variation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1277–1287. Association for Computational Linguistics, 2010.

[GHG14] Mark Graham, Scott A Hale, and Devin Gaffney. Where in the world are you? geolocation and language identification in twitter. *The Professional Geographer*, 66(4):568–578, 2014.

[GSO+11] Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A Smith. Part-of-speech tagging for twitter: Annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 42–47. Association for Computational Linguistics, 2011.

[HHSC11] Brent Hecht, Lichan Hong, Bongwon Suh, and Ed H Chi. Tweets from justin bieber's heart: the dynamics of the location field in user profiles. In *Proceedings*

of the SIGCHI Conference on Human Factors in Computing Systems, pages 237–246. ACM, 2011.

[MN+98] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.

[MND12] Jalal Mahmud, Jeffrey Nichols, and Clemens Drews. Where is this tweet from? inferring home locations of twitter users. *ICWSM*, 12:511–514, 2012.

[MND14] Jalal Mahmud, Jeffrey Nichols, and Clemens Drews. Home location identification of twitter users. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(3):47, 2014.

[MTL78] Robert McGill, John W Tukey, and Wayne A Larsen. Variations of box plots. *The American Statistician*, 32(1):12–16, 1978.

[PP10] Alexander Pak and Patrick Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In *LREC*, volume 10, pages 1320–1326, 2010.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[SKB12] Adam Sadilek, Henry Kautz, and Jeffrey P Bigham. Finding your friends and following them to where you are. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 723–732. ACM, 2012.

[WB11] Benjamin P Wing and Jason Baldridge. Simple supervised document geolocation with geodesic grids. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 955–964. Association for Computational Linguistics, 2011.