Predicting Geo-location using Twitter API

Khyati Kansagra (514110640)
Roshni Ramakrishnan (726838330)
Aniruddh Garge (863559754)

Abstract:

Social media blogging apps and websites such as Twitter have provided a platform for constant communication between people all over the world. It has made access to a million different things extremely fast and easy and has subsequently brought the world closer. This in turn also makes it possible to market and advertise various products and services to different groups of people as well as provide recommendations. With the advent of mobile phones and applications, Twitter is now on everyone's phones and so is everything else. For the above use cases, it is also important to know the location of users so the advertisements and the recommendations can cater to a specific set of people based on their surroundings. It is also important to know the location of a user in case of emergencies.

In this project, we are trying to deal with this exact problem of finding location of people so that it will become easier to categorize them. This will be done by analyzing the content of their tweets to find the location and analyze at different granularities right from city and state to latitudes and longitudes. Knowledge based system was used to generate parameters for the location prediction of each user. We used Support Vector Machines to actually predict the geo-location for each tweet incoming from the Twitter stream.

Introduction:

In today's world, data derived from social media holds much more value than it did before. The data extracted is one of the biggest resources today for descriptive and predictive analysis of data.

This is because people on Twitter post daily updates on their profiles, and discuss various global and local events, right from politics to celebrity trends. Since all of these contain some kind of information, they can be used for a lot of information which can be used for location based recommendations, handling emergency events, which help first responders reach the location of the disaster as soon as possible. Another application is that companies can make customized advertisements based on location detection. We are implementing text based geo-location predictor which would predict location by analyzing the contents of the user's tweets.

In order to achieve this, we will be performing non-linear classification using Support Vector Machine(SVM). Our output will be in the form of map co-ordinates, essentially the latitudes and longitudes after which we will be able to classify the tweets into cities closest to the derived co-ordinates.

Literature Review:

In terms of prior work, there have been many research projects in this field. Most of them explored very different techniques to predict the location based on tweets. One of them was to analyze a user's tweets and compare them with words that were previously used for the place or that described the place.

Another one talked about the analysis of a user's friends and followers on Twitter which would be used to predict the location based on their friend's and follower's tweets.

Another technique that we studied about was actually a combination of a couple of methods. It relied purely on the contents of the tweet without gaining access to the user IP, login information, or any external knowledge base. It also used a classification component which identified certain words in tweets which has a very strong local geo-scope automatically. Lastly, it used a smoothing model in order to refine the estimated location of the user and was essentially lattice-based.

These were some of the vast majority of techniques that have been used previously for prediction of geo-location using user tweets.

In this project, we have tried two approaches for location prediction of a user. One approach uses Naive Bayes and Logistic Regression, and the other is using Knowledge bases and SVM.

Approach 1:

In this approach, we have used UCLA's dataset of tweets for location prediction. This dataset essentially has tweets gathered from different users and their locations. We use SVD, Naive Bayes, and Logistic Regression on this dataset in order to predict the location of any other incoming tweet.

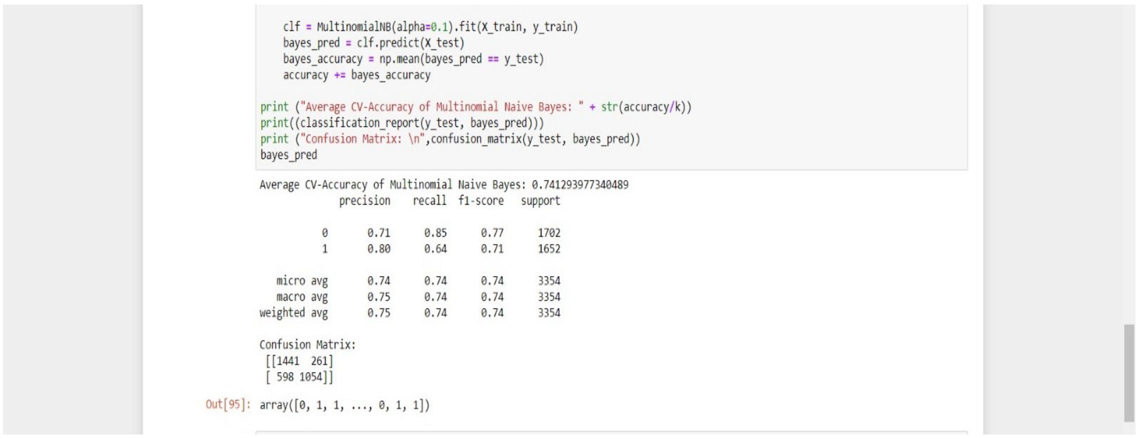
In terms of pre-processing, we have first tokenized the entire dataset into individual tokens. Stemming is used so that the words are essentially converted to their root forms so it gets easier to classify. Then, we have managed to remove all the stopwords from the list of tokens.

In order to filter the data set to get specific locations, we have chosen two locations. Boston, Massachusetts and Seattle, Washington. We have given scores in order to keep these in check. If Boston or Massachusetts occurs, the score is 0, else it is 1.

After this, we had to balance the data set. This was a very important step since if the data set is not balanced, it will always only predict one case. We then generated a count of tweets and used it to convert into scorers. This was done by using the Count Vectorizer which would count the occurrences of words. After counting these, it will generate a TFDIF score for them respectively. This new transformed data is our training data.

We then performed dimensionality reduction on training data. This was done using Singular Value Decomposition(SVD). SVD is nothing but the factorization of the matrix. The benefit of using SVD is that it deals better with a sparse matrix than most other techniques. So we use the same data and transform it in terms of SVD.

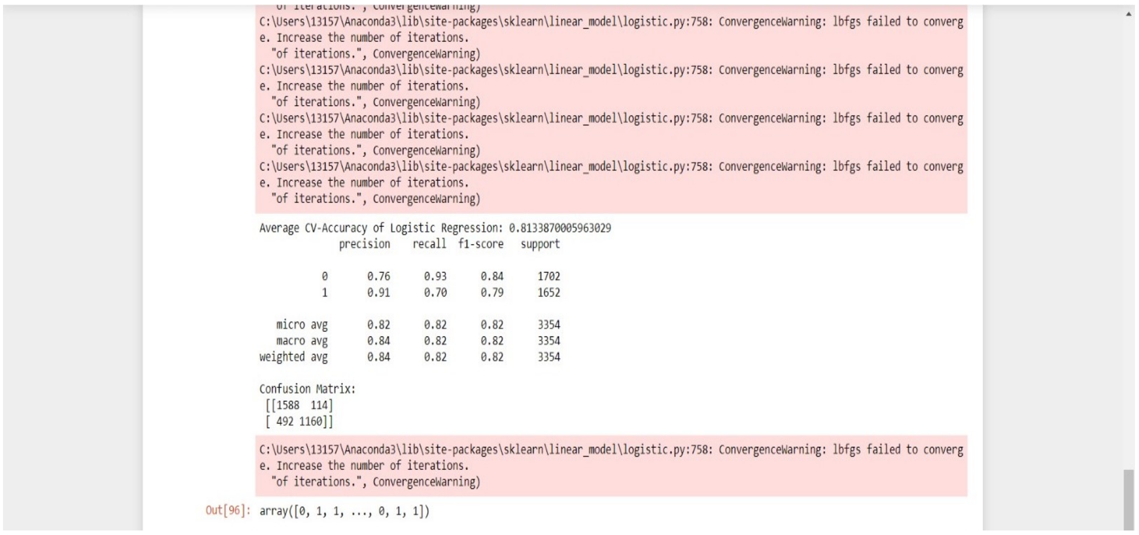
The output is scaled using the min max scaler. This min max scaler is specifically used because it does not give negative values. After this, we used ten-fold cross-validation on it and finally fit Multinomial Naive Bayes on it.
We then calculated the accuracy, classification report(precision, recall), and average accuracy.



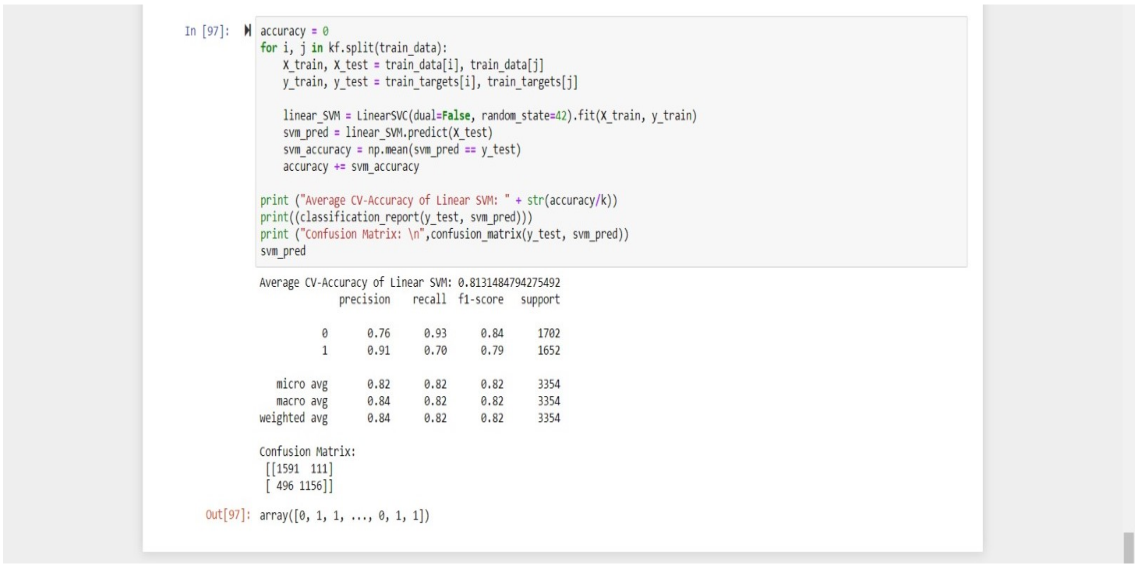
```
    clf = MultinomialNB(alpha=0.1).fit(X_train, y_train)
    bayes_pred = clf.predict(X_test)
    bayes_accuracy = np.mean(bayes_pred == y_test)
    accuracy += bayes_accuracy

print ("Average CV-Accuracy of Multinomial Naive Bayes: " + str(accuracy/k))
print((classification_report(y_test, bayes_pred)))
print ("Confusion Matrix: \n",confusion_matrix(y_test, bayes_pred))
bayes_pred
```

```
Average CV-Accuracy of Multinomial Naive Bayes: 0.741293977340489
              precision    recall  f1-score   support

           0       0.71      0.85      0.77      1702
           1       0.80      0.64      0.71      1652

   micro avg       0.74      0.74      0.74      3354
   macro avg       0.75      0.74      0.74      3354
weighted avg       0.75      0.74      0.74      3354

Confusion Matrix:
 [[1441  261]
 [ 598 1054]]

Out[95]: array([0, 1, 1, ..., 0, 1, 1])
```

We perform a similar process with logistic regression, and linear SVM too. Even in this case we use a ten-fold cross validation and then fit multinomial regression on it. We then find the performance metrics.



```
C:\Users\13157\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarning: lbfgs failed to converg
e. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
C:\Users\13157\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarning: lbfgs failed to converg
e. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
C:\Users\13157\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarning: lbfgs failed to converg
e. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
C:\Users\13157\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarning: lbfgs failed to converg
e. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
Average CV-Accuracy of Logistic Regression: 0.8133870005963029
              precision    recall  f1-score   support

           0       0.76      0.93      0.84      1702
           1       0.91      0.70      0.79      1652

   micro avg       0.82      0.82      0.82      3354
   macro avg       0.84      0.82      0.82      3354
weighted avg       0.84      0.82      0.82      3354

Confusion Matrix:
 [[1588  114]
 [ 492 1160]]
C:\Users\13157\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarning: lbfgs failed to converg
e. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
Out[96]: array([0, 1, 1, ..., 0, 1, 1])
```



```
In [97]:  accuracy = 0
          for i, j in kf.split(train_data):
              X_train, X_test = train_data[i], train_data[j]
              y_train, y_test = train_targets[i], train_targets[j]

              linear_SVM = LinearSVC(dual=False, random_state=42).fit(X_train, y_train)
              svm_pred = linear_SVM.predict(X_test)
              svm_accuracy = np.mean(svm_pred == y_test)
              accuracy += svm_accuracy

          print ("Average CV-Accuracy of Linear SVM: " + str(accuracy/k))
          print((classification_report(y_test, svm_pred)))
          print ("Confusion Matrix: \n",confusion_matrix(y_test, svm_pred))
          svm_pred

          Average CV-Accuracy of Linear SVM: 0.8131484794275492
                        precision    recall  f1-score   support

                     0       0.76      0.93      0.84      1702
                     1       0.91      0.70      0.79      1652

             micro avg       0.82      0.82      0.82      3354
             macro avg       0.84      0.82      0.82      3354
          weighted avg       0.84      0.82      0.82      3354

          Confusion Matrix:
           [[1591  111]
           [ 496 1156]]

Out[97]:  array([0, 1, 1, ..., 0, 1, 1])
```

The output will essentially be in the form of 1s and 0s since, as mentioned earlier, we have given scores for the locations – 0 for Boston and 1 for Seattle.

Approach 2:

Here, we have performed location prediction by fetching tweets from Twitter using the Twitter API and then using Knowledge Bases and applying SVM on it.

Workflow:

For this project, we will be performing a series of steps to attain our desired result. This is a basic workflow of the project:

- Firstly, we will be creating a knowledge base which would have all the links of each city that we are trying to predict.
- Next, we will be creating a knowledge base for each city’s named entities. A named entity is basically a real world object that has a predefined name, like a country, a city, a person, a product, or a book title.

- We will be creating a knowledge base for each of the links that were created for the cities which consist of the named entities.
- Next, we will be creating a graph of all the links that connect to an individual city.
- After this, we will use page ranking on each graph and then calculate the score.
- We will calculate another score based on the local entities that would overlap between each city's local entities and their respective link's local entities.
- Next, we will be finding the local entities and their corresponding geo-text in the user tweet content and check the user's bio.
- The above two scores will be used in calculating the confidence score for every city.
- The above calculate score will then be fed to the SVM as parameters for training.
- Lastly, in order to measure the accuracy, we will also calculate the distances between predicted and actual co-ordinates.

Creating Knowledge Base:

As mentioned above, the first step is creating a knowledge base. This knowledge base will be created for every city in the country. For our use, knowledge base is nothing but a combination of words which hold some meaning in our context and are related to the city in a manner as close as possible. This is essentially a compilation of named entities that are all closely related to each city.

In order to create a knowledge base,

- Local named entities are collected for each city.
- In the city's Wikipedia page, all the links going out of the page are taken into record.
- Local entities that exist in each link for a particular city will be collected.

The Wikipedia API is used in order to fetch the local entities of each city. This API fetches the main page of the city. We used the Python library Spacy in order to extract the named entities. Spacy is an open-sourced library that is mainly used in Natural Language Processing. It can recognize various types of named entities in a document by asking the model for a prediction. We have tuned it a little depending on our use case.

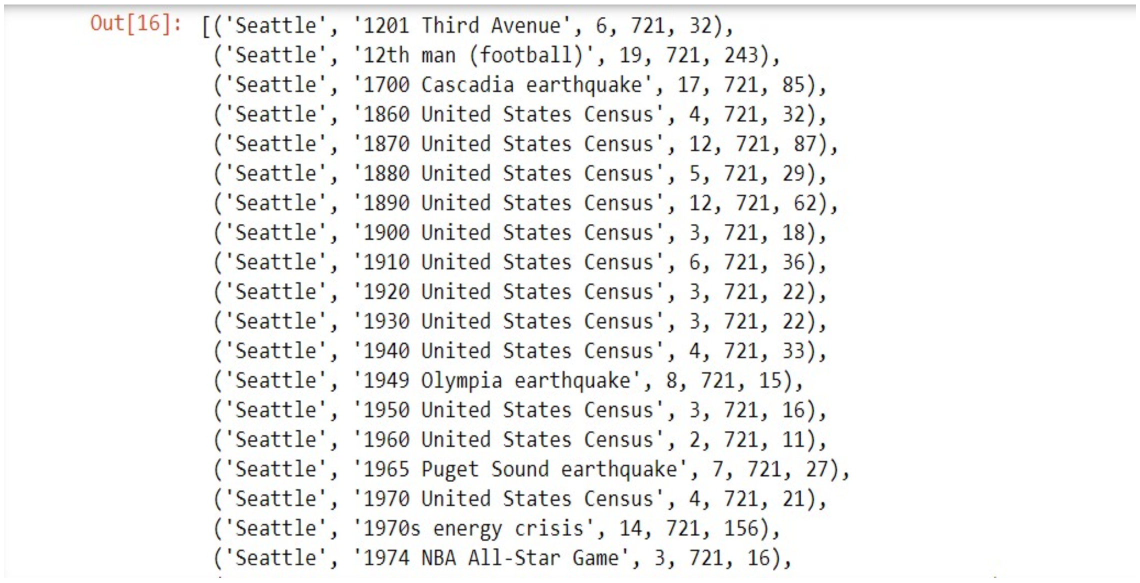
The entire Wikipedia page's content is parsed in order to extract the local entities. This is done specifically for each city. The local entities are then stored in the knowledge database. Named entities can be categorized in a variety of ways such as product, event, person, etc. Of these categories, there are some that we absolutely do not need because if we use more categories, it would create a messy knowledge base with more irrelevant things than relevant. For this, we have not considered certain categories since all we're interested in is finding the words that help in predicting the location. Thus, trivial categories such as quantity, money, ordinal, cardinal, date, percent, have been removed.
During the continuous updating of the knowledge base, it may happen that some words get repeated. We have tried to reduce this kind of redundancy by making sure that such words are not stored more than once in our knowledge base.

After this, each city's Wikipedia page is checked and all the links that are going outward from the page are stored in the knowledge base. The nodes are created for the city and the places around it and the edges connect the nodes to the city. All these edges are directed edges. Therefore, the outward going links from a city's Wikipedia page are checked for these directed edges. If there is a link that is somehow directed back to the city, then a backward edge is also created between the link and city. This leads to the creation of a cycle in the graph.

Calculation of Scores:

Use of local entities:
Here, the local entities which have been extracted from the city's Wikipedia page is actually compared with the local entities which exist in each individual link that is present in the city's Wikipedia page. These are used to calculate the scores.

The scores are calculated for each city as follows- First, the number of common words occurring in the city is calculated. Next, the union of the total number of local entities that are present in that city's Wikipedia page and number of local entities in each link of that city is calculated. Finally, the number of common words is divided by the union, and that is how we get the scores using local entities.



```
Out[16]: [('Seattle', '1201 Third Avenue', 6, 721, 32),
          ('Seattle', '12th man (football)', 19, 721, 243),
          ('Seattle', '1700 Cascadia earthquake', 17, 721, 85),
          ('Seattle', '1860 United States Census', 4, 721, 32),
          ('Seattle', '1870 United States Census', 12, 721, 87),
          ('Seattle', '1880 United States Census', 5, 721, 29),
          ('Seattle', '1890 United States Census', 12, 721, 62),
          ('Seattle', '1900 United States Census', 3, 721, 18),
          ('Seattle', '1910 United States Census', 6, 721, 36),
          ('Seattle', '1920 United States Census', 3, 721, 22),
          ('Seattle', '1930 United States Census', 3, 721, 22),
          ('Seattle', '1940 United States Census', 4, 721, 33),
          ('Seattle', '1949 Olympia earthquake', 8, 721, 15),
          ('Seattle', '1950 United States Census', 3, 721, 16),
          ('Seattle', '1960 United States Census', 2, 721, 11),
          ('Seattle', '1965 Puget Sound earthquake', 7, 721, 27),
          ('Seattle', '1970 United States Census', 4, 721, 21),
          ('Seattle', '1970s energy crisis', 14, 721, 156),
          ('Seattle', '1974 NBA All-Star Game', 3, 721, 16),
```

The above image shows the number of common local entities between two things. If we take the example of 'Seattle' and '1201 Third Avenue':

Total number of local entities for 'Seattle' = 721

Total number of local entities for '1201 Third Avenue' = 32

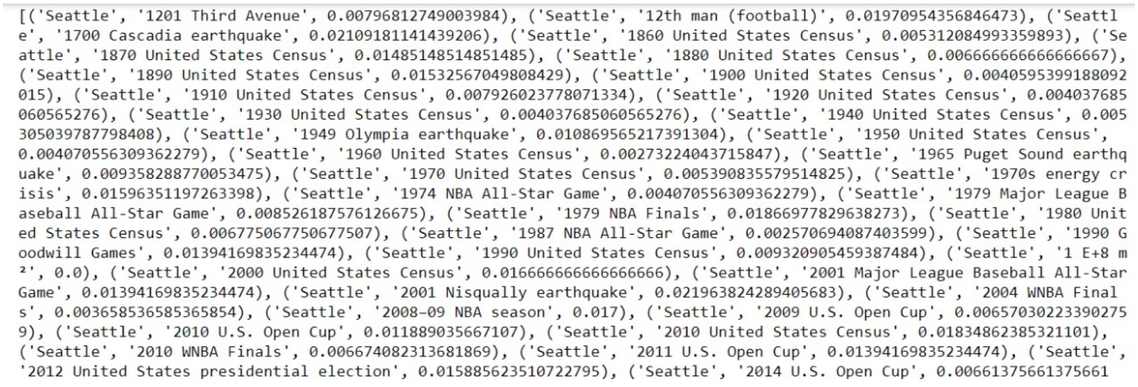
Number of common total local entities = 5

Thus, the overlapping score can be calculated as follows:

Total overlapping score = Common total local entities / (Number of local entities for the city + Number of local entities for the link)

Total overlapping score = 6 / (721 + 32)
= 0.00796

As seen in the image below, that is exactly the score we achieved in our first instance of the overlapping score between the city and its link.



```
[('Seattle', '1201 Third Avenue', 0.00796812749003984), ('Seattle', '12th man (football)', 0.01970954356846473), ('Seattl
e', '1700 Cascadia earthquake', 0.02109181141439206), ('Seattle', '1860 United States Census', 0.005312084993359893), ('Se
attle', '1870 United States Census', 0.01485148514851485), ('Seattle', '1880 United States Census', 0.006666666666666667),
('Seattle', '1890 United States Census', 0.01532567049808429), ('Seattle', '1900 United States Census', 0.0040595399188092
015), ('Seattle', '1910 United States Census', 0.007926023778071334), ('Seattle', '1920 United States Census', 0.004037685
060565276), ('Seattle', '1930 United States Census', 0.004037685060565276), ('Seattle', '1940 United States Census', 0.005
305039787798408), ('Seattle', '1949 Olympia earthquake', 0.010869565217391304), ('Seattle', '1950 United States Census',
0.004070556309362279), ('Seattle', '1960 United States Census', 0.00273224043715847), ('Seattle', '1965 Puget Sound earthq
uake', 0.009358288770053475), ('Seattle', '1970 United States Census', 0.005390835579514825), ('Seattle', '1970s energy cr
isis', 0.01596351197263398), ('Seattle', '1974 NBA All-Star Game', 0.004070556309362279), ('Seattle', '1979 Major League B
aseball All-Star Game', 0.008526187576126675), ('Seattle', '1979 NBA Finals', 0.01866977829638273), ('Seattle', '1980 Unit
ed States Census', 0.006775067750677507), ('Seattle', '1987 NBA All-Star Game', 0.002570694087403599), ('Seattle', '1990 G
oodwill Games', 0.01394169835234474), ('Seattle', '1990 United States Census', 0.009320905459387484), ('Seattle', '1 E+8 m
²', 0.0), ('Seattle', '2000 United States Census', 0.016666666666666666), ('Seattle', '2001 Major League Baseball All-Star
Game', 0.01394169835234474), ('Seattle', '2001 Nisqually earthquake', 0.021963824289405683), ('Seattle', '2004 WNBA Final
s', 0.003658536585365854), ('Seattle', '2008–09 NBA season', 0.017), ('Seattle', '2009 U.S. Open Cup', 0.00657030223390275
9), ('Seattle', '2010 U.S. Open Cup', 0.011889035667107), ('Seattle', '2010 United States Census', 0.01834862385321101),
('Seattle', '2010 WNBA Finals', 0.006674082313681869), ('Seattle', '2011 U.S. Open Cup', 0.01394169835234474), ('Seattle',
'2012 United States presidential election', 0.015885623510722795), ('Seattle', '2014 U.S. Open Cup', 0.00661375661375661
```

Using Page Rank:

Page rank is a technique that is used for analysis of links where every node in the graph is assigned a numerical score between 0 and 1. This score is called the page rank. The page rank of a node will depend upon the graph and the link structure.

In our case, page rank will show how important a node is based on the structure of our graph that we have created for the city and its links. Since there exists a link which goes back to the city, leading to the formation of a backward edge, the page rank of that node will be more. The reason behind this is that if the user uses a word in their tweet which matches one of these nodes, it would drastically increase the probability that the user is tweeting from that place or somewhere close by. Therefore, we keep checking the page rank of each node in the graph and then calculate the score.

Thus, the overlap method will return the score for all tweets from all cities. This will be done by comparing the content of the tweet and the user's bio whose knowledge base is currently being generated.

Twitter API:

For fetching tweets we used Tweepy, which is a python library built for accessing the Twitter API. Here, Twitter is accessed with the help of OAuth. It does not reveal the password of the user, and makes it easier to customize the tweets with any string we want. We made a developer account on Twitter and managed to get the consumer key, secret, access token, and access token secret.

First we created an empty dataframe and appended all the useful information gathered from the tweets in there. We made sure that the tweets were only in English, by setting the language to "en". We first fetched the raw tweets and then used Natural Language Processing to clean them. We also eliminated trivial categories such as quantity, date, ordinal, cardinal, percent, and money.
After this, we had a dataframe with all the information extracted from the tweets that would help us predict user location.

Support Vector Machine:

Support Vector Machine (SVM) is an algorithm that helps us determine the best decision boundary among different vectors that would eventually belong to a particular category. In our implementation, now that we have the two scores namely, the non duplicated words and the geo-text, we add them together to make our final list. This final list is then put in a data frame and an overlap of these scores, so this final list is sent in as a parameter for the function which calculates overlaps. This function is not an inbuilt function but is rather defined by us which maintains a final score by calculating the distances between the words.

This will be appended to our final data frame. Our latitude and longitude columns are defined and added to the final data frame. Next our training sets are made and here we have made two sets of training sets by splitting our data. Finally we train the model using SVM. Here we have used the svm.src package from scikit-learn library. Scikit-learn is an open source library for predictive data analysis. It is originally built on NumPy, SciPy, and matplotlib.
After passing the parameters into our SVM, there will be two outputs, one for latitude and one for longitude. These outputs are appended in a dataframe. The dataframe is made in order to calculate accuracy after this. The dataframe also has the actual latitude and longitudes as well.

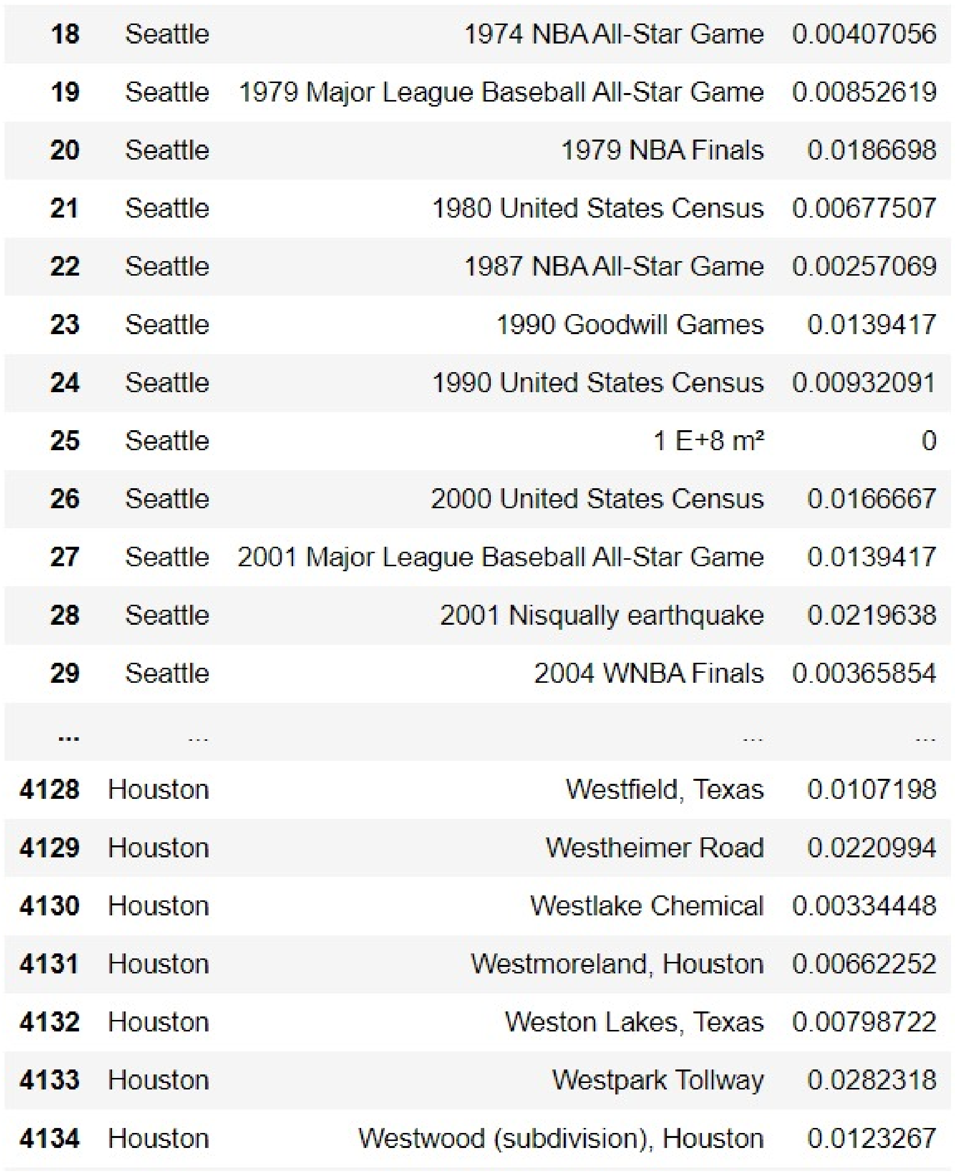
Results:

The main goal of this project was to find the location of the user from where he/she has posted the tweet. We have taken into consideration the contents of the tweet and the twitter bio data of that user. The local entities were extracted from the tweets and the output is as shown:

After this, we extracted the geo-text in the tweet content which in turn led to an increase in confidence level of that user that it will be in the right place by a considerable margin. We later passed the list of words having local entities and geo-text as parameters in our function that calculated the overlapping scores and produced an output in the form of a dataframe which would have column names as cities and rows as the scores calculated for each city for the list of words that we had given as input.

The values obtained in the dataframe are the input parameters for our Support Vector Machine. Here, we are using a regressor to achieve our results.

However, before passing the parameters, we split the derived data so far into training and testing and then the model is trained such that we get our predicted latitudes and longitudes. Now, the distance between the actual latitudes and longitudes and the predicted latitude and longitudes are calculated.



18	Seattle	1974 NBA All-Star Game	0.00407056
19	Seattle	1979 Major League Baseball All-Star Game	0.00852619
20	Seattle	1979 NBA Finals	0.0186698
21	Seattle	1980 United States Census	0.00677507
22	Seattle	1987 NBA All-Star Game	0.00257069
23	Seattle	1990 Goodwill Games	0.0139417
24	Seattle	1990 United States Census	0.00932091
25	Seattle	1 E+8 m²	0
26	Seattle	2000 United States Census	0.0166667
27	Seattle	2001 Major League Baseball All-Star Game	0.0139417
28	Seattle	2001 Nisqually earthquake	0.0219638
29	Seattle	2004 WNBA Finals	0.00365854
...	...	...	...
4128	Houston	Westfield, Texas	0.0107198
4129	Houston	Westheimer Road	0.0220994
4130	Houston	Westlake Chemical	0.00334448
4131	Houston	Westmoreland, Houston	0.00662252
4132	Houston	Weston Lakes, Texas	0.00798722
4133	Houston	Westpark Tollway	0.0282318
4134	Houston	Westwood (subdivision), Houston	0.0123267

Above is the similarity measures calculated by the overlapping scores based on our function for each city and it’s links.

Finally, we measure the accuracy. This is done by calculating the distance between the coordinates. If the distance is anywhere less than a hundred miles, the prediction is assumed to be correct. However, if the distance between the coordinates is anywhere more than a hundred miles, it is safe to assume that the model has predicted the wrong location.

Findings:

After the results we found above, we found that the input lists in which there existed a larger number of local entities and geo-texts gave a more accurate prediction, whereas the ones with lesser local entities did not always give accurate results. The solution to improve this accuracy would be to increase the overall total number of local entities while calculating the scores of each and every city. In continuation, in order to increase the number of local entities, we would have to fetch and analyze more tweets of each user such that we can thereby gather more number of local entities. Our method to extract tweets will likely fetch almost a thousand tweets for each user and will then create local entities for each user. These words will then be used to generate a vast enough data set for our SVM model which will further improve the accuracy of the model.