Geo-location of German Tweets

Partu Ana-Maria

1.Dataset

The task for this project is to predict the locations (latitude, longitude) of German Tweets. The dataset has 3 .txt files with data for training, validation and testing. The training and validation files contain an id on the first column, the latitude on the second column, the longitude on the third column and the tweet on the fourth column. In the training set there are 22,583 labeled examples. The validation set consists of 3,044 examples. The “test.txt” file has only 2 columns, one for the id and one for the tweet. The test set contains 3,138 examples and the labels are not provided for the data.

2. Data processing

Since the data in the files was split into columns, I could easily use Pandas Dataframes to read the data and split it into arrays containing the data and the labels.

Because the data was in text format I had to apply NLP on it. This includes:

* Lowercase-ing
* Stopwords removal
* Stemming
* Tokenization
* N-grams
* Tf-idf (Term frequency – invers document frequency)

I applied lowercase-ing even though German nouns start with an uppercase letter because I found this to be irrelevant to the task. I chose to not remove stopwords as the task implied also checking the style of writing, which would be lost by stopwords removal, as proven by running both versions (with and without stopwords) of the algorithm and comparing the MAE. Same reason applies to not using stemming. I used word level N-grams to conserve the context of words. By trial and error I found that using both unigrams and bigrams together yields the lowest MAE in this case. For tokenization I tried CountVectorizer and HashingVectorizer provided by sklearn package. The HashingVectorizer proved to be better for the task in every case I tested. Its advantages over the CountVectorizer are that it uses very little memory because it does not store a dictionary (which is not needed for the task because we do not need the inverse transformation). One advantage of the CountVectorizer is that it is able to use idf weighting which I did not end up using anyway. For the HashingVectorizer I used the highest number of features I could use (2\*\*28). It uses l2 norm and I made it output only positive numbers. The Tf-idf transformer weights the terms outputted by the vectorizer in a tf-idf manner. By trial and error I noticed a decrease in MAE when not using the idf part. All these and the machine learning algorithm were implemented inside a Pipeline provided by sklearn for ease of use.

3. Machine Learning

The 2 models I chose were LinearRegression and LinearSVR as these yielded the best results in testing. Due to the nature of the algorithms (LinearSVR working faster) I used different number of features (2\*\*24 for LinearRegression and 2\*\*28 LinearSVR). LinearSVR is linear kernel SVR which is the regression version of SVM. I tried changing the hyperparameters from the default to see if I could improve the results but none of the changes yielded a better MAE. The default loss function for LinearSVR is epsilon insensitive. In the case of LinearSVR I had to train the latitude and the longitude separately as it does not support multi-label regression.

Results on validation set:

CountVectorizer + LinearRegression + Stopwords = 0.614 MAE

CountVectorizer + LinearRegression = 0.608 MAE

HashingVectorizer + LinearRegression + Stopwords = 0.600 MAE

HashingVectorizer + LinearRegression = 0.598 MAE

CountVectorizer + LinearSVR + Stopwords = 0.599 MAE

CountVectorizer + LinearSVR = 0.591 MAE

HashingVectorizer + LinearSVR + Stopwords = 0.592 MAE

HashingVectorizer + LinearSVR = 0.588 MAE

The model used for the submission used HashingVectorizer and LinearSVR which yielded a MAE of 0.588 and a MSE of 0.620 on the validation set.

I conclude that for this type of geo-location based on tweets the best approach is to leave the text as close to its original form as possible and extract as many features as possible from the text and then apply regression to this data.