The University of Melbourne
COMP90051 Statistical Machine Learning, Semester 2 2019
Project 1 Report - Group 146

Introduction to problem

The purpose of this project is to predict authors of test tweets from a large number of authors found in training tweets. Twitter tweets are relatively self-contained and have small sentence length variance, thus the authorship attribution is challenging regarding the limitation of texts. This project targets to train a machine learning (ML) model with suitable training approaches and methods for predicting the right author, also identifying appropriate raw features from numerous tweets to facilitate the training.

Description of features

Features are an important part of any machine learning algorithm. Good features, when combined with the most basic classifiers, will give better results rather than using a complex classifier with bad features. With the authorship attribution of tweets, the features usually fall into three categories: (1) Lexical features, such as the word count of each sentence or each tweet; (2) Syntactical features,such as the number of title words, the number of punctuation in each sentence and the number of all uppercase words; (3) Content-specific features, such as the presence of re-tweets (indicated by "RT" at the beginning of a tweet), hashtags, user mentions (represented by "@handle" in the dataset) and URLs [1]. Regarding the relatively small size of the dataset, the features should be chosen carefully so that they can best represent the authors' writing pattern without leading to overfitting.

For this project, after removing tweets with length less than 5 and greater than 30 and the users who have less than 20 tweets in training data, we extracted multiple features. Since machine learning algorithms don't work on textual data, the extracted features must be converted in the numerical form. The extracted features were - character counts of each tweet, word count, the number of punctuation is each tweet, the number of letters which were in upper case of any word in the tweet, the total number of uppercased words in the tweets these are the numeric counts. We extracted features like the presence of- '@handle', URL and Retweet- these features were extracted in binary form i.e. if these features were present in the tweet the value was set to 1 if not it was set to 0. The numerical features were then normalized so that there won't be any sort of outliers in the data which might affect the model training. We used count vectors for words with max number of features set to 7000 and characters count vectors with max features set to 500. We have also used TFIDF vectors. The max features count was set based on a trial and error method. We have also used Doc2Vec.

ML methods

1) Naive Bayes classifier for multinomial models

The multinomial Naive Bayes classifier is suitable for classification with discrete features, such as the number of words for text classification. It requires integer feature counts vectors for generating multinomial distribution. We have used the scikit-learn Python library to build the Naive Bayes classifier, with tuning the alpha to smoothing parameter

2) Linear Support Vector Machines (Linear SVM)

Linear SVMs are widely used for text classification, they are a set of supervised learning methods, including classification, regression, and outlier detection. It is an effective tool in high dimensional spaces, it can use a subset of training points in the decision functions, so it is also memory efficient.

a) Linear SVM with stochastic gradient descent (SGD) learning

The gradient of the loss is estimated at each sample at a time and the model is updated along the way with a decreasing learning rate. For best results using the default learning rate schedule, the data should have zero mean and unit variance.

b) Perception

Perception is a classification algorithm that shares the same underlying implementation with the SGD classifier. In the scikit-learn library, *perceptron()* is equivalent to *SGDClassifier(loss="perceptron", eta0=1, learning_rate="constant", penalty=None)*.

c) *Linear Support Vector Classification*

It is a type of Linear SVM classification method that has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples. In the scikit-learn library, this class supports both dense and sparse input and the multiclass support is handled according to a one-vs-the-rest scheme.

3) *Logistic Regression*

It is a simple machine learning classifier, which provides constant output. Multinomial logistic regression is a form of logistic regression used for the model that has more than 2 classes. It performs multi-class classification using the binary classification logistic regression algorithm, either one-vs-rest(OvR) or one-vs-one approach. In the scikit-learn library, the training algorithm uses the OvR scheme in the multinomial situation. Meanwhile, regularization is applied by default to fit a model appropriately on the given training set and avoid overfitting.

4) *K-Nearest Neighbors (k-NN) Algorithm*

K-NN is an instance-based learning algorithm that is easy to apply and less time consuming compared to other machine learning algorithms. The K Neighbors Classifier implemented in the scikit-learn library is used for model fitting, and the normalized feature set is used as the training set. In this attempt, the *k* value (i.e. the parameter *n_neighbors* in scikit-learn API function) has been varied to examine how it affects the accuracy of predictions.

5) *Random Forest Classifier*

Regarding conventional text classification, the random forest classifier has been an effective approach because of its low classification error rate and relatively shorter training time [2]. In this case, the random forest classifier implemented in the scikit-learn library is adopted for modeling fitting. The set of normalized features is used as the training set. During the training process, parameters including *n_estimators* (i.e. the number of trees developed in the model) and *max_depth* (i.e. the maximum depth of trees) have been varied in order to optimize the results.

Model selection

Model selection was done based on the ease of training the model and the tuning of hyperparameters. The first approach was to use the K-NN algorithm, we used K-NN with numerical features which gave us an accuracy of 6.6% on Kaggle and after normalizing the numerical features like the counts of words, characters etc we had an improvement of 2% making the final score to 8%. We trained K-NN on the TFIDF vectors but the accuracy was lower than what we were scoring for numerical data. The 'K' hyperparameter was tried on different values ranging from 1 to 4. For the Kaggle submission we kept the final K value as 3, the value 1 for K was resulting in overfitting of model so we didn't use it, the overall runtime for training and predicting was less than 7 minutes. This does imply that numerical data with multiple features are easier to determine the class for a testing object.

The next approach was to use SVC, but since the time complexity of SVC is O(n^3) it was a better option to not choose it as it would take very long to train the model, instead of choosing SVC we used the SGD classifier which is an optimizer for linear models and part of SVM class, we trained SGD on the count features and the count vectors and as expected the accuracy value for count vectors was much higher than the numerical count features. The angle accuracy score was 13% with count vectors having a maximum of 5000 features, the number of max features were set after cross validating the training and testing data. The overall run time for SGD was around 1hour in which the training the model took 99% of the time. The next approach was to use perceptron, it was used with numerical count features, the accuracy score was less even in cross validation stage with 'train_test_split' method so we didn't go ahead with it.

The next model we used was Multinomial Naive Bayes, once we dropped multiple outliers from the training data, it was very easy to fit the whole model in the memory, which was one of our challenges as it was impossible to fit the whole training data without any removal of outliers. The Multinomial Naive Bayes was trained on the numerical count features, count vectors and TFIDF vectors. As it turns out, count vectors were more efficient than the other features. This is because

the values of TFIDF and numerical count features were normalized which isn't good for the Multinomial Naive Bayes. The max features for count vectors were set to as 7000 for word count vectors and 500 for character count vectors. The total number of features in the end were around 7277, stating that there were very few single character usage after removing the stopwords. The final shape of our matrix was (211222, 7277). The hyperparameter for Multinomial NB which is '***alpha'*** was tried on different values of - [1,0.5,0.1,0.08,0.05,0.01,0]. Alpha is a smoothing parameter in the model, where the value 0 stands for no smoothing, when using the value 1 for alpha we found that the model was underfit and resulting in low accuracy for training set than the test set, for the alpha value of 0.01 we found the model was overfitting, in order to find a fitting alpha value we used ***'gridsearchCV'.*** We were also able to see the difference in the accuracy score for different alphas with different number of features. The range of features was- [3500,5500,7500,15000], with 7500 features having the highest accuracy at not much lower alpha value of 0.08. We were able to score 19% on Kaggle with this score. The total training time was less than 5 minutes. The plot 1 will depict the different cross validation scores on ***train_test_split*** within the model.

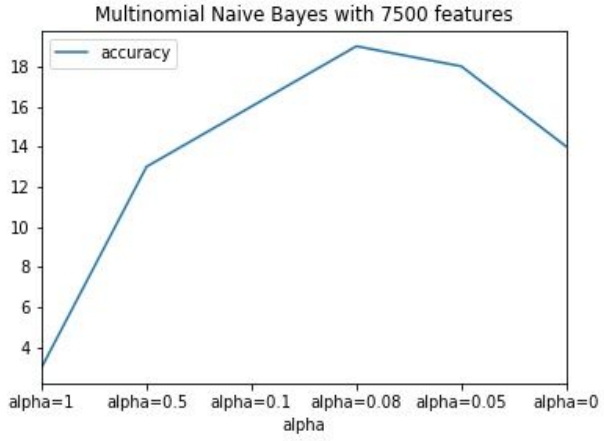
The next approach was to use Logistic Regression with Doc2Vec, but again the training the model was taking very long even after 10 hours it wasn't trained so we didn't give much importance to it. The Random Forest classifier was a challenging one as there were a lot of features and limited memory to train the model. We trained the Random Forest model on the numerical count features, with max depth set at 17. This was the max we could use so that we have enough memory left to predict the values. Since the depth was very shallow, the overall accuracy score on Kaggle was very low.

Summarise and explain results with critical analysis

Multinomial Naive Bayes is one of the most simplest and easiest text classification models which can be used for multiclass classification as well. Given the results of all the model in the previous section, we chose Multinomial Naive Bayes with count vectors as features where the max features were set to 7500- which consisted of 7000 word vectors and 500 character vectors. When using numerical count features like the character count, word count etc… it is not possible for a Multinomial NB to predict the probability of it, rather using count vectors which works on Bag of Words model can be used to store the word vectors and we can calculate its probability of occurring next. The effect of the hyperparameter ***'alpha'*** also plays an important role, if the value of it is very low, it is going to overfit the model and if it is going to be high it might underfit the model. Choosing the correct ***'alpha'*** value was done with the help of ***'gridsearchCV'*** and with the help of cross-validation results from it. Our final model has an ***'alpha'*** value of 0.08 and max number of features set to 7500. We had to also choose over TFIDF vectors and count vectors, since the TFIDFtransformer normalizes the vectors it was best in favour to not use it, as Multinomial Naive Bayes doesn't require it.

We plan on implementing the SVM using Shougun library which was linked on the discussion board. Due to the time constraints and lack of experience in deep learning we weren't able to implement, one of the main goals is to implement this project using ***fast.ai*** and ***BERT*** combined. Random Forest model can be used in combination with parallel programming to get better results.

Graph comparing the accuracy and hyperparameter



Plot 1

References

[1] M. Bhargava, P. Mehndiratta, and K. Asawa, "Stylometric Analysis for Authorship Attribution on Twitter," *Big Data Analytics Lecture Notes in Computer Science*, pp. 37–47, 2013.

[2] P. Maitra, S. Ghosh, and D. Das, "Authorship Verification – An Approach based on Random Forest," *Notebook for PAN at CLEF 2015*, 2015.