COMP30027 Report

anonymous

1 Introduction

This project is about to pick suitable machine learning strategy to predict the geotag based on the text generated by user. In this project, I will discuss my work from the following perspective: data pre-precessing[2], Models[4] that I explored.

2 Data Preprocessing

When working on this project, I found out that it is really important to have a good tool to transfer the plain text to useful and suitable new features to feed into the model. Better data preprocessing strategy may produce more useful information and more choice for our later feature engineering.

2.1 *-top* Files Only

Those files were originally provided to us. Even though those words are highly correlated (in terms of all the words) to our labels, whereas, up to 78% of the instances contain only 0 for all the attributes. If I could correctly predict over 70% of the non-zero labels data and use 0-R to assign the others to one label, 34.9% [1] accuracy could be reached overall. However, based on my discovery, it is extremely hard to reach even 50% on the non-zero part. Also, the real world data wouldn't be evenly distributed. In this case, the data could not be very useful. Extra work is required.

2.2 Raw File Text Precessing

Another choice becomes to use the raw tweets and process it to usable data.

2.2.1 Step 1 - Split Word By Space

Plain text is hard to use but words are easier. Therefore, the first step is to split the text by space.

2.2.2 Step 2 - Reduce Redundancy

Although we get some usable words data, there are still a lot of noise and redundancy in it. The primary thing to do is to remove all the punctuation which are widely used but uninformative. After that, by sampling some text from the tweets, I notice that the word after # and might be extremely useful as they may contains information hight correlated to one of the label but rarely occur, for instance, "BrisbaneNews9". What I did for string like this is that I split them into smaller words by capital letter. Notice that I might get a text like this "#LouisWhyAreYouAnEGG", the EGG at the end won't become ["E","G", "G"]. Instead, it will becomes a whole word "EGG". After this process, all words are transfer to lowercase to prevent that people using words in unexpected way.

2.2.3 Step 3 - Reduce Sample Size

Up until now, the sample size are huge, throwing some less informative words is necessary. In this steps, most [2] of stop words and unicode emoji are removed. Those words are widely used but less informative. Roughly 25% features are removed in this step.

3 Feature Engineering

Feature engineering is another very important part. Based on the performance of my preprocessing strategy, same set of data gain a 5% [3] boost in accuracy when comparing to using train-top100. In this section, I will discuss two feature engineering approach that I used. There performance will be mentioned in model section. [4]

[1] $0.7 * 0.22 + 0.25 * 0.78$

[2] more will be removed in later section

[3] figure obtained by 20-fold CV on train-raw and dev-raw by using MNB and Logistic regression(Saga)

3.1 Frequency Based

Since the feature set are still very large, but number of instance are relative small. To avoid overfitting and variance of our model, I prefer to take the top 5000 [4] most frequent words among all tweets. Two difference representations for this method are used, the first one is one-hot (preferred by most of the model) stored in sparse matrices, the other one use the number of ranking in frequency (start from 1) and then use 0 padding to extend to the same length (preferred by word embedding).

3.2 TF-IDF

TF-IDF is a commonly used feature selection tools in NLP, it measure the how important is a word from the documents perspective. It stands for term frequency(TF) and inverse document frequency(IDF). The TF part measure how frequent we could see a word in a document, IDF represent the incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.(Wikipedia contributors (2019)) By using this method, the words in those tweets could be weighted properly. What should not be neglected is that I process the text both in word level and char level (with n-gram range 2-6 for word to char) [5]. The rest of the stop words are removed at this stage. Finally, there are 60,000 feature created in total. This feature engineering method produce the best accuracy among all the method that I tried.

4 Models

In this project, I considered both deep learning approaches and classical statistical machine learning methods and becomes my final choice.

Table1[1] is the accuracy for all the model I tried (CV represent the mean accuracy yield by implementing 5 or 10 fold cross-validation and new represent accuracy tested on dev set) I use the results of cv to fine-tune the hyper-parameters of my model ,then test it on dev set to figure out how well are my models perform on relatively large unseen data. The reason that I don't use dev set for testing primarily is because I prefer dev set as real testing set which could provide me an opportunity to test my model on brand new data. If we only test our data on dev set, we are still trying to learn the hyper-parameters from dev set which will cause over-estimate.

Model	cv	new
MNB	42%	34.3%
Logistic Regression	40.1%	34.1%
Random Forest	40.2%	34.1%
Linear SVM	40.4%	33.1%
Voting	40.6%	34.7%
–	–	–
Dense NN	None	33.8%
Word Embedding	None	31.5%

Table 1: Comparison of accuracy for different models

4.1 Classical Statistical Approach

Initially, I prefer these approaches to be baseline, whereas, they produce a lot of outstanding results and becomes my final choice.

4.1.1 Multinomial Naive Bayes (sklearn)

- NB is normally the first approach people want to try when doing a NLP task. Also, It could handle multi-class problems naturally and yield some decent result. Since we know this is a multi-class prediction and the data distribution follows a multinomial distribution, we chose the MNB classifier.

4.1.2 Logistic Regression (sklearn)

- sag/saga solver works pretty well on multi-class problems and perform fast convergence in problem with huge data set.

- Only change solver to saga for the purpose of saving memory (compare to newton's method) Hessian matrices are not required, convergence grantee for enough training instances (compare to quasi-newton's method), support multi-class instead of one-vs-rest (compare to linear solver) and support more normalization method (compare to sag).

4.1.3 Random Forest (sklearn)

- Ensemble learning normally works well and it could handle multi-class problem naturally. In Random Forest, the variance of the model could be minimized.

[4] 2000-10000 were tested with steps=500, 5000 is reasonably good for my model

[5] optimal hyper-parameters found by loop through multiple possible combinations

4.1.4 LinearSVM (sklearn)

Why this model was chosen (Alexandre KOWALCZYK (2014)):

1. Most of text of NLP problems are linear separable

2. SVM has works well on a lot of different jobs with good interpretability

3. Normal NLP task contains a huge number of instance and features. Since documents contains a lot of unique words and we are trying to map the features to high dimension. In this case, using linear kernel could significantly improve our training time.

4. Less work in fine-tuning, since only regularization parameter.

4.1.5 Voting (sklearn)

- Voting is a robust learning method, works well on combining multiple weak learner. When doing this project, we use softmax function to make a decision based on the how confidence (represent in probability) is the model think their predictions are correct.

- As we already had some trained weak model, we could just combine them and see if they yield some useful results. The final result shows that it does well pretty well. MNB, Logistic Regression and Random Forest are combined for my voting approach, and yield a results which better than any single of them.

4.2 Deep Learning

Deep learning has some advantages comparing to non-deep learning, which require less feature engineering but more on architectural engineering. It could help the machine learner to do some feature selection as it is really good at fitting the data.

4.2.1 Feedforward Neural Network (Keras+Tensorflow)

- Only train-100 was used as train data here, and

4.2.2 Word Embedding (Keras+Tensorflow)

- Why this model was chosen:

5 Conclusions

In conclusion, I found out that the most important part of this project is feature engineering. By using the same processed data, the difference between best model and worst model are smaller than 3% (compare to itself not 100%). Intuitively, better pre-processing method could extract more useful information from the raw data. Consequently, models could gain huge advantages by using this ”extra” information.

References

Alexandre KOWALCZYK. 2014. Linear Kernel: Why is it recommended for text classification? `https://www.svm-tutorial.com/2014/10/svm-linear-kernel-good-text-classification/`. [Online; accessed 19-May-2019].

Wikipedia contributors. 2019. Tf–idf — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Tf%E2%80%93idf&oldid=894112456`. [Online; accessed 18-May-2019].