Text Classification Competition - Twitter Sarcasm Detection

CS410 Fall 2020

Siyuan Zhang(siyuan3@illinois.edu)

Introduction

The goal of our project is to design implement and test models that can classify the existence of sarcasm in twitter messages. Since this is a competition that focuses on the performance of the model we built, there is a baseline requirement of a f1 score greater than 0.723.

Proposed Methods

Our methods mainly focusing on using machine learning techniques since the problem we try to tackle here is mainly a text classification problem. A large part of applied machine learning is about feature engineering so a large part of our project is about discovering and experimenting different feature extraction methodologies. After the features are generated from the dataset, we then deployed different machine learning algorithms to train and classify the data set provided. We compare the performance of all the classifiers we experimented.

Preprocessing

We implemented couple standard text preprocessing functions to clean our data. Methods we used includes special character processing, upper/lower case process, punctuation signs, stemming and lemmatization, as well as stop words. We apply these data cleaning process before the data is fed into the classifiers for training. However, later we determined that such data cleaning actually degraded the test time performance by couple percent so we took this part out in the final version of our testing code. The reason for this is that we suspect in twitter message there are lots of special words and characters that actually play an important role in expressing the

message and data cleaning takes a lot of these important information out which makes it harder to classify the sarcasm in the messages.

Feature Engineering

We considered different features to use in this project. Since the context data is provided in the dataset in addition to the responses, we leverage both to extract useful features. We considered word count vectors, TF-IDF vectors, word embeddings and NLP features for this task. Among those features, we found the TF-IDF vectors to be great features that can achieve good classification performance on the test set so it was selected as out feature. We extract TF-IDF weights from both the response string as well as the concatenated context string across the data. We adopted these weights for both the unigram model and the bigram model.

Machine learning classifiers

The classifiers we considered here includes support vector machines, random forests, Boosting from the classical machine learning techniques as well as multilayer perception model from the deep learning models. For all models we believe since they were used extensively in many applications and were able to achieve good performance, they should deliver promising results on the task as well. For each of the models, we performed random searches and grid searches on all the model hyper-parameters to select the best parameters for the actual training and testing process.

Implementation Details

Thanks to the extensive amount of research and development done in this area in the past decades, we were able to make good use of many standard packages and libraries to achieve a lot of functionalities that we require in this project. We mainly used pandas data frame to load the dataset for easier processing. The TF-IDF vectors are extracted using TfidfVectorizer from the scikit learn library. We set it to use both the unigram and bigram from the data and extracts features from both the response and

the context column. After both vectors are calculated, we concatenate them for each data point and use as the features for the machine learning classifiers. We also leveraged different classifiers mainly from the eras, sklearn and xgboost library. There are very well written versions of these algorithms in these libraries and we use them to train and test these classsifiers. After picking the best model, we then producing the test time output using the Test dataset as input and saved the result into the Answer text file for benchmarking.

Experiments Detail

Since the test set is reserved for grading, we split the train set into a 4 1 split and leave 20% of our training data as validation set during out model selection and tuning process. We first perform random search on hyper parameters of each of out models to narrow down the search range of the parameters. Then we performed grid search over the parameters to determine the best performing hyper parameters for all the models. Afterwards, all models are trained on the train set and their accuracy were compared to determine the best model. Although some model tested very well during the training phase, their test time performance is not as good due to potential overfitting so we had to do models selection manually by uploading to GitHub and compare the test set performance on the livedatalab server. Below is the detailed performance of all out models on different metrics.

	Training Accuracy	Test Precision	Test Recall	Test F1
SVM	0.8272	0.658	0.688	0.6725
Random forest	0.9998	0.615	0.894	0.7288
Boosting	0.9216	0.621	0.718	0.6656
Multilayer Percep	0.9182	0.520	0.923	0.6612

From the experiment data above, we can see that the random forest classifier not only have a very high training time performance, but also a very good test time f1 score and

is above the 0.723 baseline f1 performance. Other 3 models especially the boosting and the multilayer perception model perform well during training time put didn't achieve similar performance on the test and the reason can be potential overfitting. The final parameter we used on the random forest is 1000 estimator with a decision threshold of 0.465.

Conclusion

In this project we experimented many techniques on text classification including feature engineering and machine learning classifiers. The final performance of our best model using TF-IDF feature vectors combined with random forest classifier together is able to achieve a good F1 score of 0.7288 which is beyond the baseline. We are happy with the result and this project has been an amazing learning experience for me.

Software Used

Installation guide

The following software packages needs to be installed in order to run our code:

python
pandas
nltk
tensorflow
keras
numpy
scipy
scikit_learn
xgboost

These dependencies can be downloaded and installed automatically by first cloning our repo to local and then run the following line in the terminal:

Install -r requirements.txt

Our code is located in the sarcasm_detection.py, simply run the code and generated prediction result of the test set will be saved in Answer.txt

All of out code is fully documented, please refer to our code for more implementation details.

Reference

https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

https://www.mfz.es/machine-learning/an-end-to-end-machine-learning-project-part-i-text-classification-in-python/