COMP90042 Project 2020: Climate Change Misinformation Detection

Chiyu Chen
901265

1 Introduction

Machine learning which is a branch of Artificial Intelligence is about how to extract valuable information and knowledge from dataset. It can be considered as an intersection of statistical learning and analysis, Artificial Intelligence and computer science. It is also known to make prediction in data mining, computer visualization and natural language processing (NLP). NLP is a research field that involves Artificial Intelligence and linguistics. This field mainly focuses on cognition, understanding and generation. Recent years, NLP has reached a great development because of improvements of deep learning such as top achievements on language model and language parsing.

In this project, what we mainly focuses on is the classification of misinformation about climate change. This project aims to build a detection system that checks if a document implies misinformation about climate change. Therefore, the problem can be considered as a binary classification problem: whether this piece of information misleads people on climate change or not.

2 Dataset

The data used in this project is extracted from Twitter, which is be presented in sentences or paragraphs. All data is in `.json` files and expressed by the same format:

`"train(dev)-number":`
`{"text":"content","label:"0/1"}`
dataset details:

`train.json`: a dataset for training data which only has positive label for each piece of information.

`dev.json`: a development dataset that contains positive or negative label, which helps our system to get improvement on making decisions and tuning the related parameters.

`test-unlabelled.json`: a test file without labels. It is for codalab competition and verify how this system model works.

3 Theory and method description

To achieve the best performance of current system, the first consideration is to compare the most well-performed algorithm in this project. There are general seven algorithms for options.

KNeighbors classifier: the main concept on KNeighbors is that a sample belongs to a certain category where there are most of k most similar samples in a feature space. Namely, a sample with a unknown category would be classified into the majority of its k nearest neighbors. Commonly speaking, the value of k is normally less than 20.
DecisionTree classifier: it is a tree-like model to make decisions. Each node represents a decision or deliberation of an output from a feature. Due to its stability and easy interpretation, it is one of the most widely used decision making tool.
RandomForest classifier: Random Forest is a group of DT in some ways. RF builds on bagging integration based on DT as a base learner and further adds a selection of random attributes in training process of DT. It normally has three phases: learning based on individual DT, sampling training samples and random sampling of attributes.
Multinomial Naïve Bayes: Naïve Bayes is widely used in spam detection, emails sorting and text classification. This theory considers each of feature is independent. MNB is one of variations of Naïve Bayes. It takes into account the term frequency.
Support vector machine: SVM has several related models in NLP. In this project, Radial Basis Function (RBF) kernel and linear support vector

machine are applied in comparison. Basically, SVM model aims to represent each instance as a dot in a space, so that each single classification is be separated as widely as possible. The new mapped instance would be classified due to its location in this space.

Logistic Regression: Logistic Regression is essentially a kind of Linear Regression. Unlike the latter, Logistic Regression maps continuous values of Linear Regression to (0,1) space by logistic function, which avoids limitation of Linear Regression: can only get good performance on problems with linear boundaries.

Then, the next is to choose a model which get better performance in the comparison because not all models are suitable for binary classifications. This is a rough filter. Finally, tuning parameter on the most well-performed model is the top priority.

During the final phase, grid search is applied for optimizing parameters. Grid search means looping all combinations of candidate parameters and attempting every possibilities. The one with best performance will be selected.

4 Experimental set-up and implementation

4.1 Experimental set-up

Experimental set-up is also the preprocessing of dataset, which mainly contains:format converting, lemmatisation and stopword removal.

Format converting: In format converting, there are three tasks to complete: `UTF-8` format converting, lowercasing and words separation. The original json files represent with `UTF-8`, which might mislead classifiers and are unnecessary for emotion expression. So, they have to be converted to common expression formats properly. Besides, as far as I inspected, each sentence normally ends with `|endoftext|`, which could be removed as well.

Lemmatisation: Lemmatisation and stemming are to reach the uninflected forms. The packages `WordNetLemmatizer` and `PorterStemmer` are applied to achieve this goal.

Stopword removal Stopword means the words with high frequency but no contribution to classification. It is completed by referring `stopwords` in `nltk.corpus`.

4.2 Implementation

As mentioned above, there are three dataset applied in this project: `train.json, dev.json` and `test.json`, which indicates corresponding three phases: training, development and testing.

Before training, vectorization of text is essential. Classification cannot process text without vectorization. Basically, there are three vectorizer: `DicVectorizer`, `CountVectorizer` and `TfidfVectorizer`. `DicVectorizer` is suitable for object with `dict` which is not applied in this project. `CountVectorizer` focuses on Term Frequency (TF). `TfidfVectorizer` considers not only TF but Inverse Document Frequency (IDF) which means it could cut down some meaningless words with high frequency. However, it does not perform well for current dataset because of limited volume of data. So, `CountVectorizer` is applied in project.

What training could contribute to the classification is to help classifier collect the features and improve the rules because of its data only with positive labels. To achieve this, `OneClassSVM(gamma='scale', nu=0.01)` is used, which is specific for unbalanced dataset. It can find outliers and extract positive features.

In comparison,`cross_validation()` is imported from `sklearn.cross_validation` to avoid contingency with `cv = 10`. It also can show the stability of each method.

Then, tuning parameter of selected models would curve models to be suitable for dataset features. We will discuss results in detail in the next section.

5 Results and error analysis

5.1 Models comparison

We will fist discuss about the performance of each model. The results has been shown in Table 1. F1 score is selected to indicate how each model performs.

Models	**Avg. F1 score**
KNeighbors	0.706
DecisionTree	0.857
RandomForest	0.667
MultinomialNB	0.737
LinearSVC	0.778
LogisticRegression	0.737
GaussianNB	0.615

Table 1: Models comparison results

As the Table 1 shows, KNeighbors and RandomForest perform poorly comparing with others. One of the reasons for that is KNeighbors is a lazy learning classifier and it does not well on interpretability, especially when samples are unbalanced. For example, when sample A size is significantly larger than others, a new coming sample would be very likely to classified into A because most of its neighbors are A. For RF, it could be overfitting samples that have high noise. Text often contains many noise in terms of classification.

DT and LinearSVC have the best results among these seven models. DT does not demand highly on data preparation. Instead, other models often need normalized data before processing. SVM benefits depend on the size of data. The size of data in this project is around 1000 documents which is small size and suitable for SVM.

However, DT and LinearSVC perform differently in testing.

Models	**Avg. F1 score**
LinearSVC	0.55
DecisionTree	0.51

Table 2: LinearSVC and DT in ongoing codalab

According to Table 2, DT does not perform better than LinearSVC as Table 1 shows. The reason could be that, DT is difficult to handle test data where there are new features. It hardly to classify them to correct labels.

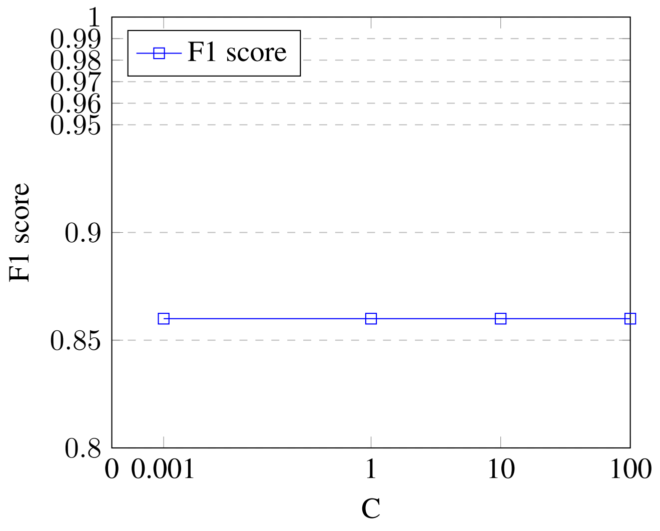
Then, LinearSVC is selected and will be tuned with parameters.

5.2 Tuning parameters

Inspired by the official document about LinearSVC, the key parameter is C, which is a regulation parameter. In other words, when C becomes larger, there will be less regulation for data features. If C goes smaller, LinearSVC will has a strict regulation of samples, which will cause a bad performance.

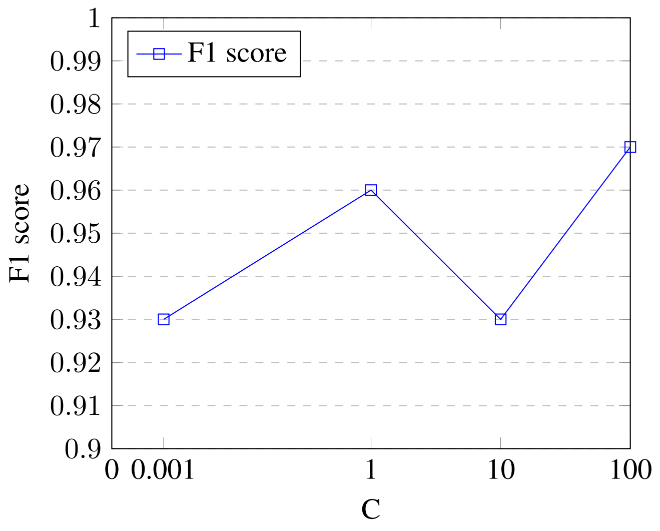
An interesting find during tuning, at the beginning of optimization, I did not realize the importance of RandomState when train the dev data. Therefore, I got the result as Figure 1. Clearly, it does not show any improvement when C goes up.

Figure 1:LinearSVC with different C, RandomState =0



Then, after I change RandomState to 42, it shows a rough up trend overall as shown in Figure 2.

Figure 2:LinearSVC with different C, RandomState =42



5.3 Performance in final codalab

The best result of performance in final Codalab is around F1 = 0.60, which is a not-surprising result among other students. There are many improvements that need to take during the experiment, such as adding penalty rule in LinearSVC, optimization of data features, adding more weights on some specific words which would clearly show people's emotion trends.

Overall, there are still many excellently-performed and inspired models need to try, such as RNN, LSTM,GRU and so on. This is not only a peer competition but a self-challenging task to myself. It helps me deeply understand how those models work in real instead of on theoretical layer.

References

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. arXiv preprint arXiv:1508.05326.

Qian Chen, Xiaodan Zhu, Zhenhua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2016. Enhanced LSTM for natural language inference. arXiv preprint arXiv:1609.06038.