Author: Tanmay Pawar

tanmaymsp@gmail.com

Abstract

In this report, I have trained a Naïve Bayes classifier for text classification using python and talked in brief about general techniques of data-preprocessing needed for any natural language processing problem. Also used SVM to build the model using same dataset, How class imbalance effects the predictions and why oversampling worked for Naïve Bayes but not for SVM

Report of   
text classification problem

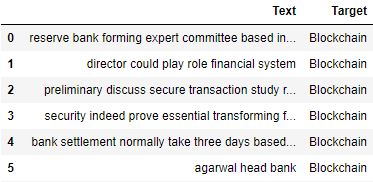
Classifying text according to topics

**Introduction:**

Text classification is widely done using some machine learning algorithms. It is a part of Natural Language Processing. In text classification, basically, by reading the text (news title, paragraph, etc.), it is decided that the text belongs to which class known by our machine learning algorithm.

As text data is very different than numerical and categorical data that we generally saw in our dataset, it needs to be processed before applying machine learning algorithm on it. The pre-processing of text data consists of **Tokenization, stemming, word vectorization**, etc. techniques. The popularly used algorithms for text classification are **Naive Bayes, SVM and algorithms of deep learning**. I have used Naïve Bayes for the classification. We will see how it performs on our data. We will be facing problem of class imbalance also try to tackle it.

**Problem Statement:**

 The given dataset consists of two features, one is ‘Text’ and other is ‘Target’.

**Text** – each value of Text consist of snippets like a news title (Independent feature)

**Target** – corresponding Target value for every Text is the topic to which the text is based.(class to which text belongs)(Dependent feature)

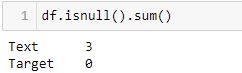
Build a model to classify the text to its right class.

**Requirements:**

* Dataset <https://docs.google.com/spreadsheets/d/1DLL6BTXiHHsn1w9NvVi0BZbass0QU0RSvKEXtJlwfCM/>
* Python
* Jupyter notebook
* Libraries
  + Pandas, Numpy (for manipulating the data)
  + Seaborn (for visualization)
  + nltk.stem (for stemming)
  + Sklearn (for model selection, feature extraction, TFIDF vectorization)
  + Imblearn (for dealing with class imbalance)

**Approach:**

The dataset have two features (one dependent and one independent) and 22,704 instances. So, the dataset looks pretty large.

 Starting with data pre-processing, I first checked missing values in it. There are 3 instances where text data is missing.

After dropping these instances, the dataset have no missing values.

Natural language processing problems have some basic steps for data pre-processing, to make data ready for model building. I have performed these steps as follows,

**Stemming:**

In a text data, same word can be with different prefixes, postfixes, etc. E.g. perform, performing, performed. The machine will consider them totally different, even if they are not significantly different. This will increase the complexity (as these words will be different features after vectorization)

In stemming, the program tries to remove prefixes, postfixes making perform, performing, performed same i.e. perform.

Stemming reduces complexity, but rarely it can also change the information. By converting any word witch changes their meaning significantly after having some prefix, postfix, etc.

e. g. Sample data after stemming

Note: accuracy of the model is increased after stemming.

After splitting the dataset into training and testing data, I have performed text vectorization.

**Text Vectorization:**

The machine learning algorithms will not understand the text data. So I converted it into numerical data. Vector is formed for words in the text, where a value is assigned to a word depending on its occurrence in the text.

I have used **TF-IDF** in our model for text vectorization.

For each word in the vector (text converted into vector), TF-IDF assigns a value which is equivalent to

“**Term Frequency\* Inverse Document Frequency** “

- Term frequency (TF) = frequency of word in the text/total no of words in text

- Inverse document frequency (IDF) = log (no. of text data/ no. of text data containing the word)

TF-IDF assigns value to the words such that the vector formed have values according to semantic importance of the word in the feature considering whole dataset. Unlike, bag of words assign only either 0 or 1 to the word.

Note: Tfidfvectorizer method from sklearn also provides argument for stop word i.e. using stop\_words we can eliminate common pronouns present in text.

Now that pre-processing is done on our data, I applied machine learning algorithm to build the model.

**Naïve Bayes:**

Naïve Bayes is proved to be a good choice for text classification problems. Naïve Bayes is based on 2 basic assumptions that,

* Each feature is totally independent of other.
* Each feature contributes equally to the predictions.

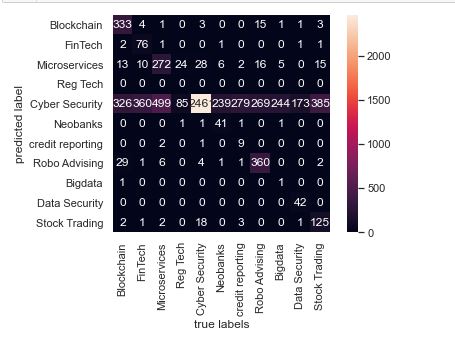
In text classification, features are the words in our dataset and their values are according to their occurrence / importance in the instance. Each instance is a vector.

Naïve Bayes is based on Bayes theorem, which gives conditional probability. I.e. probability of class Y=y given, features have values, X = (x1, x2…).

For text classification, it is used as, probability that text belongs to class Y=y given, the words in the text are X = (x1, x2 …).

So I will built the model using Naïve Bayes classifier. Using “MultinomialNB()” from sklearn library, I trained the Naïve Bayes’ classifier on training data.

Note: training data is vectorised now.

Training Naïve Bayes algorithm on the vectorised training data have done predictions on test data. But it showed poor results.

Most of the test data is classified to one class only.

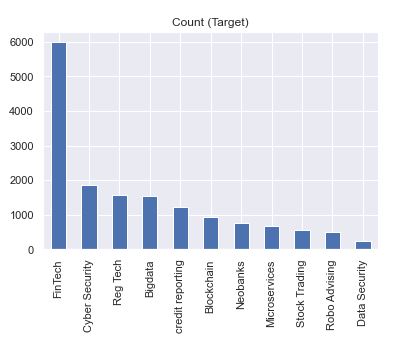
Accuracy score for this model is,

**Training accuracy = 0.6267**

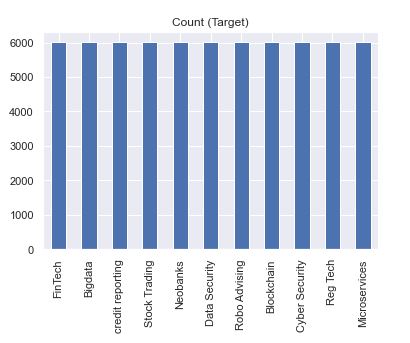
**Test accuracy = 0.546**

While analysing this problem, this may be due to imbalance in class.

I calculated F1 score, as it is a better evaluation for classes with extreme imbalance

**F1 score = 0.3369**

*Class distribution in training data*

After oversampling,

*Class distribution after oversampling in training data*

Due to imbalance in data, the results were totally in favour of any single class, making the classifications poorer. So, we can use multiple techniques to overcome problem of class imbalance and boost performance of the classifier. One of the most basic technique for it is, under-sampling or oversampling.

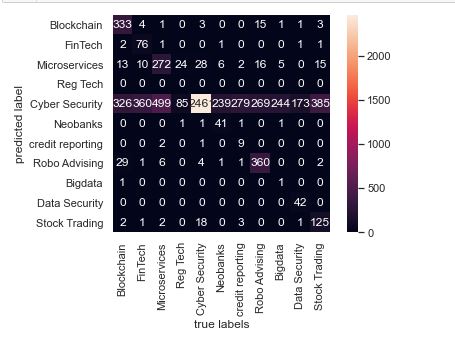
Under-sampling reduces instances of class which have maximum occurrence to balance it with class having minimum instances. But it may lead to loss of any important information. So, I used oversampling. In oversampling, occurrence of classes with minimum instances is increased

* Oversampling drastically increased the performance of the classifier.

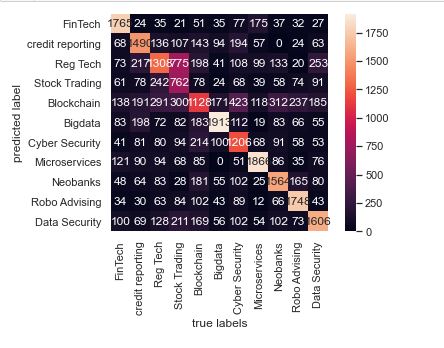
Accuracy score for the model (after oversampling),

**Training accuracy = 0.8230**

**Test accuracy = 0.5914**

**F1 score = 0.5938**

*Confusion matrix for Naïve Bayes without oversampling*



*Confusion matrix for Naïve Bayes with oversampling*

Note that, F1 score for naïve Bayes improved drastically after oversampling.

Oversampling also have some disadvantages, by training he model on oversampled data, accuracy of the model can reduce. Because oversampling makes the model more biased to the class which is oversampled.

**Limitations for Naïve Bayes:**

Although, Naïve Bayes have done the predictions. Still it have low accuracy. Naïve Bayes is good for text classification but it have some limitations in the model.

Naïve Bayes assumes every feature (word in text) independent of other. But this is rarely true in the real world.

So, when it gets a new word in the text data, based on its training, Naïve Bayes algorithm calculate probability of that word equal to zero, which makes probability of the whole instance equal to zero. It is called ‘zero-frequency problem’.

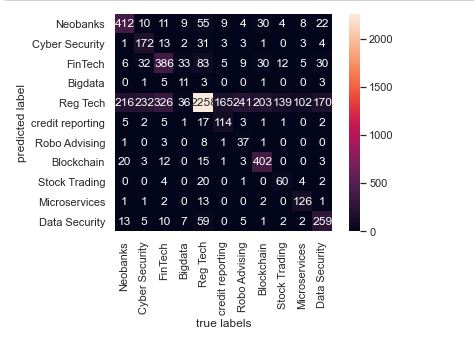
To overcome this problem by a small extent, Laplace smoothing is introduced. It adds a constant term in the Bayes theorem with a coefficient (I have used the value of coefficient as 0.5 in our code) which don’t let probability to be zero for new word. Rather it have some value. But still it can’t capture dependence of the word on others in the same instance.

**Trying out SVM**:

SVM is also one algorithm popularly used for text classification. I have tried using SVM to build our model. As the naïve Bayes consider each feature to be independent, it will not take into account the relation between two different words in an instance. But, SVM do consider the dependence between different features/words. For e.g. when word ‘cricket’ comes into the text, it is more likely to have bowling, batting in the same instance of text. SVM takes it into account but Naïve Bayes will not.

* With oversampling, SVM produced better results than Naïve Bayes,

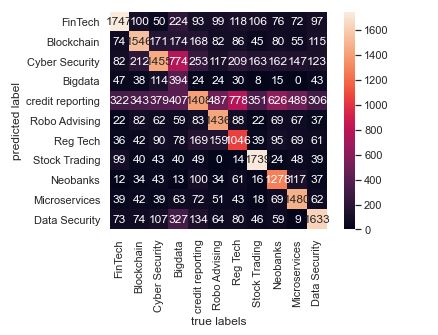
**SVM test accuracy (without oversampling) = 0.632**

**SVM F1 score (without oversampling) = 0.5136**

*Confusion matrix of SVM without oversampling*

* But interestingly, after oversampling the dataset, SVM results accuracy decreases and poorer than Naïve Bayes.

**SVM test accuracy (with oversampling) = 0.5454**

**SVM F1 score (with oversampling) = 0.5468**

*Confusion matrix of SVM with oversampling*

Reduction in accuracy can be due the fact that, training the model on oversampled data makes it more biased towards the class which is oversampled, which in turn can reduces the performance of the classifier.

**For SVM,**

|  |  |  |
| --- | --- | --- |
|  | Accuracy ( on test data) | F1 score |
| Without oversampling | 0.632 | 0.5136 |
| With oversampling | 0.5454 | 0.5468 |

**For Naïve Bayes Classifier,**

|  |  |  |
| --- | --- | --- |
|  | Accuracy ( on test data) | F1 score |
| Without oversampling | 0.546 | 0.3369 |
| With oversampling | 0.5914 | 0.5938 |

**Conclusions:**

Problems like text classification requires data pre-processing which contains techniques such as tokenization, stemming, vectorization.

When data was not oversampled, SVM gave better results than Naïve Bayes. The best accuracy score overall was when SVM is used without oversampling. But after oversampling (to tackle class imbalance), oversampled Naïve Bayes showed better results than oversampled SVM.

As discussed earlier, it may be because of increased bias towards oversampled class, SVM accuracy decreased. (But still F1 score of SVM increased by small after oversampling)

Considering F1 score, Naïve Bayes with oversampling is better choice. Although, SVM without oversampling gave best accuracy.

Also, Naïve Bayes being simple algorithm, it is way faster than SVM.