Naive Bayes Classifier Documentation

Stefan Kotevski - 171101

1 Introduction

This is a small homework project for a Pattern Recognition Course I am attending. The main task is to implement a Naive Bayes Classifier. In this document i explain some of the inner-workings of the project, explain the usage and show the results.

2 Implementation and Usage

The project is implemented as a .NET Core Console application. It can be compiled into a windows executable easily and be modified to be executed by CLI. In the current state I do not provide a way to pass arguments from the CLI but this can be changed in future versions. The whole project can be found in the 'full_project' directory. This i a Visual Studio project that can be loaded, run and modified. My source Code is located in the 'source' directory. The project is very well documented in the source code itself, but this can be used as a helper document.

The main part of the project is the [NaiveBayesClassifier] class. This is the implementation of the classifier and includes the training and prediction logic. A method called [Fit] is used to fit the model with a given training dataset. Once an instance of [NaiveBayesClassifier] is made, the first method that should be called is the [Fit] method. The parameters are well documented in the project code itself. This method calculates the statistical likelihood from the dataset. The calculations are fairly simple and include counting the occurrence of unique values for each feature and class. This implementation of the Naive Bayes Classifier is not limited to only binary class datasets and can be used to train models on multiclass datasets as well.

Once the model is trained, the [Predict] method can be called with a testing dataset. This method will do the predicting for all the feature vectors in the dataset and return a list of predictions as integers. Again as I mentioned above the method is fairly well documented in the source code so explanation for the parameters can be found there.

2.1 Metrics

After the model is trained and the predictions have been made on the test dataset the performance of the model can be evaluated with the [ModelMetrics] class. An instance of this class can be instantiated and used to get some basic model metrisc like: model accuracy, precision, specificity, f1_score and a confusion matrix (this is only for 2 classes since the problem that I was tasked with was for 2 classes). I used this class to determine my model metrics and configure my parameters.

3 Training the model

The second part of the task was to train a model on the "SMS Spam Collection Dataset". I used my classifier and some preprocessing to achieve this. This is all done in the [main] method of the 'Program.cs' file. First I load the dataset. This is a bit specific for Visual Studio since I loaded the training and test datasets as .NET Resources in my VS project. This was done so that the project is portable. However if only the source code is used then you will have to load the dataset another way. The dataset can be downloaded here.

After the sets are loaded into lists of [_DataSetItem] objects they can be separated into feature vectors and targets. The [_DataSetItem] class is a small helper data structure to wrap the raw data that is read from the dataset. To create the feature vectors I opted to use a Bag of Words representation for its simplicity.

To do this firs of all a vocabulary of the dataset is calculated. This is done with the helper function [ExtractVocabulary]. There are a few helper functions in the 'Program.cs' file to help with code separation and management. The vocabulary is extracted by separating each SMS into words. This is done by RegEx separation and only keeping words consisting of alphanumeric characters. After this further filtration is done in order to remove words made up of single characters (these don't really carry any information since they are either artifacts or article words). After this each word occurrence is counted and words that occur lest then a specified 'cut_off' value are also discarded.

Now it is easy to calculate the bag of words. Here I also use a helper function called [BagOfWordsFromVocabularyAndSMS]. This is pretty straight forward so I will not explain it.

3.1 Priors

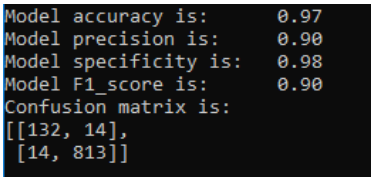
For the priors that I used I tested 3. Reading in [2] it is mentioned that according to some statistics about 80% of mail is spam so one of the priors, named [wiki_prior] is this exact prior. The second is an unbiased prior that gives equal probability to each class. Finally for a third prior a statistical prior from the dataset itself is calculated. This prior is also almost unbiased since the dataset is fairly balanced.

3.2 Training and Prediction

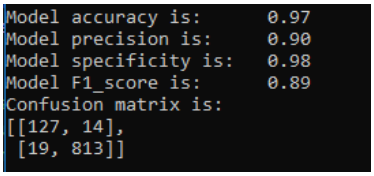
In the next section of the code I train my classifier and use it to classify the test dataset. After this I use my [ModelMetric] class to calculate the the model performance. Then the results are stored in a '.tsv' file.

4 Results

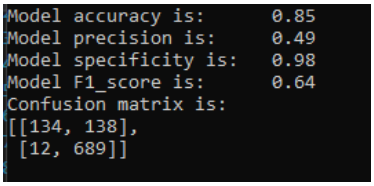
In Figure 1 below you can see the results I achieved with my methods with the 3 different priors. As can be seen the [wiki_prior] prior performs the worst but still outputs a fairly good accuracy rate. This proves that even a simple model like Naive Bayes can be used quite efficiently for SPAM filtering. I forgot to mention but the decision rule in the [Predict] function uses max likelihood estimation to determine the class once the posterior probabilities are determined.



(a) Unbiased Prior



(b) Dataset Prior



(c) Wikipedia Prior

Figure 1: Model Metrics

References

[1] UCI Machine Learning Repository *SMS Spam Collection Dataset)*

[2] Wikipedia: *Naive Bayes spam filtering*

[3] GeeksForGeeks: *Naive Bayes Classifiers*