

**FIT 5149 APPLIED DATA ANALYSIS**

Assignment 2 – Data Analysis Challenge



Group 79

Vandhana Gopalakrishnan(29406242)

Vijaya Rohin Periaiah(29289467)

Rishupal Singh Chhabra(29413818)

Contents

[1.Task: 2](#_Toc527928238)

[2.Pre-processing: 2](#_Toc527928239)

[3.Feature Generation: 2](#_Toc527928240)

[Stemming: 3](#_Toc527928241)

[Generating Dense Matrix: 3](#_Toc527928242)

[4.Model Selection: 4](#_Toc527928243)

[5.Support Vector Machine Algorithm: 4](#_Toc527928244)

[6.About our model: 5](#_Toc527928245)

[7.References 5](#_Toc527928246)

# 1.Task:

The purpose of this task is to classify news article into 23 different groups based on its content.

# 2.Pre-processing:

Pre-processing involved reading the given file and extracting text for each of the document. This was done easily with the help of Regular expressions package in Python.

The first step is to extract the document ID for all the training and test documents (training\_docs.txt and testing\_docs.txt respectively) using the following regular expressions:

***tr\_doc\_id = re.findall(r"ID (.\*?)\nTEXT",training\_data)***

***te\_doc\_id = re.findall(r"ID (.\*?)\nTEXT",testing\_data)***

Following this, the lines between the words **TEXT** and **EOD**, were extracted as a string using the following regular expressions:

***tr\_doc\_contents = re.findall(r"ID tr\_doc\_[0-9]+[\n]TEXT(.\*)[\n]EOD", training\_data)***

***te\_doc\_contents = re.findall(r"ID te\_doc\_[0-9]+[\n]TEXT(.\*)[\n]EOD", testing\_data)***

The next step was to extract the labels for the relevant training document set, which was done by reading the given training\_labels\_final text file.

# 3.Feature Generation:

We have chosen to use TF-IDF as the features to build the model upon. The two options that were considered were: 1. Term Frequency 2. Term Frequency – Inverse Document Frequency. Based on several references, one of them being a post of the website “Stack “Bag-of-Words for Text Classification: Why not just use word frequencies instead of TFIDF?” we chose to build the model with TF-IDF as features.

The TF-IDF feature generation was done in Python owing to the convenience of packages available for this purpose. The two packages that were used were 1. NLTK 2. SKLEARN.

The Sklearn package offers a class TfidfVectorizer which is a very convenient way to generate tf-idf values for documents in a corpus. Some of the parameters that were explicitly specified in the tf-idf vector generation are:

***stop\_words='english'***

Passing the parameter as ‘english’ automatically removed the built-in English stop words from the documents.

***analyzer='word'***

This parameter determines whether the feature should be made of characters or words. The features for the purpose of text classification are chosen to be “words”.

***max\_df=0.98***

The max\_df parameter instructs the vectorizer to ignore terms that have document frequency higher than a threshold value. The threshold value chosen in this case is 98%. We found that choosing a lower threshold such as 95% significantly reduces the number of features. Therefore, we decided that 98% would be optimum. This process is nothing but removal of corpus specific stop words.

***min\_df=0.02***

The min\_df parameter instructs the vectorizer to ignore terms that have a document frequency lower than the threshold value.

***ngram\_range = (1, 2)***

This parameter specifies the range of n-grams that needs to be extracted from the documents. We tried this for both bi-grams and tri-grams and found that all of the tri-grams get removed as they have a very low document frequency and the min\_df parameter removes words that occur in less than 2% of the documents in the corpus. In fact, we found that very few bi-grams survived into the final set of features.

## Stemming:

Since stemming is not part of the TfidfVectorizer class in Sklearn, the analyzer in the TfidfVectorizer was overridden to call the Snowball Stemmer from the NLTK package.

## Generating Dense Matrix:

Since not all models run using a sparse matrix, the sparse matrix generated by the TfidfVectorizer was converted to a dense matrix in python. This dense matrix was written to a CSV file and and subsequently read in R to run various models.

# 4.Model Selection:

The following models were tried on the training data set to come up with the best classifier. Since R runs on the RAM completely, several memory issues were encountered. Despite these memory errors, we were able to execute the following models by randomly sampling a subset of the training and test data set.

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Training Set Size** | **Validation Set Size** | **Accuracy** |
| Linear Discriminant Analysis | 10.00% | 10.00% | 45.00% |
| Naïve Bayes | 1.00% | 0.50% | 54.00% |
| K-Nearest Neighbours with K=20 | 20.00% | 10.00% | 57.50% |
| K-Nearest Neighbours with K=30 | 20.00% | 10.00% | 61.04% |
| Random Forest n=500 | 100.00% |  | 70.80% |
| Random Forest with xgboost rounds=50 | 80.00% | 20.00% | 70.32% |
| Support Vector Machine | 80.00% | 20.00% | 71.89% |

# 5.Support Vector Machine Algorithm:

It is a supervised machine learning algorithm that can is used mostly in classification challenges. Objective of SVM is to find a hyperplane in n dimensional space(n – number of features) that classifies the data points distinctly.

To classify the data points, Hyperplanes also called decision boundary are used.

Then we have a concept of support vectors that are data points which are closer to hyperplane and influence its position. Using these vectors, margin of classifier is maximized. Hence, It helps building our SVM model.

It is implemented in practice using kernels.

Some of the advantages of using SVM are as:

1. Effective in high dimensional space
2. Uses a subset of training points in the decision function (support vectors).
3. Memory efficient.
4. Works really well with clear margin of separation.

# 6.About our model:

*Before running this model*, we have removed the features which are in numerals. This resulted in reducing the number of features by 36. Finally, for the SVM we have 930 features.

Since, SVM is giving the best accuracy we have chosen this model for our classifying.

*svm\_model.r* file attached with this report clearly shows the detailed implementation of SVM model in R to predict the class label for testing article.

*tf\_idf\_features.csv* is our features file

*testing\_labels\_pred.txt* file has the prediction of label and is stored in this file.

Due to need to process huge dataset and model using large number number of predictors(>500). We were in the need to have a processor that can compute large amount of data – in memory(RAM > 32 gig).

In normal systems model was taking long time for computations and memory error pops out most of the time.

So, to solve this we have created an AWS instance ec2 of m4\*16 large – 256gig RAM for processing. This instance is used for parallel computation i.e., processing data at different processor’s cores(for our instance it was 64 cores).

# 7.References

https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47. (n.d.).

https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/. (n.d.).