Table of Contents

1. Abstract

Nowadays, a daily increase of online available data leads to a growing need for that data to be organized and regularized. Textual data is all around us starting from web pages, e-books, media articles to emails or user comments. There are a lot of cases where automatic text classification would accelerate processing time (for example, detection of spam pages, personal email sorting, tagging products or document filtering). We can say that all organizations (e.g. academia, marketing or government) that deal with a lot of unstructured text, could handle that data much easier if it was standardized by categories/tags. This Dataset is a collection newsgroup documents. The 4 newsgroups collection can be used for experiments in text applications of machine learning techniques, such as text classification and text clustering.

2. Introduction

Text classification or text categorization is an activity of labelling natural language texts with relevant predefined categories. The idea is to automatically organize text in different classes. It can drastically simplify and speed-up your search through the documents or texts!

Imagine, you own a large e-commerce website which shows relevant products to a user based on his/her search and preferences. Every time you want to add new products you have to read their descriptions and manually assign a category to them. This procedure can cost you too much time and money, especially if you have a high fluctuation of the available products. But, if you develop an automatic text classifier, you can easily add many new products and tag them automatically without actually reading the descriptions! You can also create a classifier to link search texts to the item categories for a better user experience.

3. Model/Algorithm Detail

While training and building a model keep in mind that the first model is never the best one, so the best practice is the “trial and error” method. To make that process simpler, you should create a function for training and in each attempt save results and accuracies.

3.1 Exploratory Data Analysis

I decided to sort the EDA process into two categories: general pre-processing steps that were common across all vectorizers and models and certain pre-processing steps that I put as options to measure model performance with or without them. Accuracy was chosen as a measure of comparison between models since greater the accuracy, better the model performance on test data.

The following general pre-processing steps were carried out since any document being input to a model would be required to be in a certain format:

1. Converting to lowercase
2. Removal of stop words

3. Removing alphanumeric characters

4. Removal of punctuations

5. Vectorization: TfVectorizer was used. The model accuracy was compared with those that used TfIDFVectorizer. In all cases, when TfVectorizer was used, it gave better results and hence was chosen as the default Vectorizer.

The following steps were added to the pre-processing steps as optional to see how model performance changed with and without these steps:

1. Stemming

2. Lemmatization

3. Using Unigrams/Bigrams

I analyzed the spread of output labels in the training data to check for whether it was skewed towards any of the class. Figure 1 below shows this class distribution. As seen in the figure, the spread of variables is reasonably even thus showing a near equal distribution of each output variable, hence the classification models won't be biased towards any particular class and subsequently no resampling was required. Also, from Figure 2 we see that the class distribution in the test.

Figure 1: Class Distribution of Training Data Set

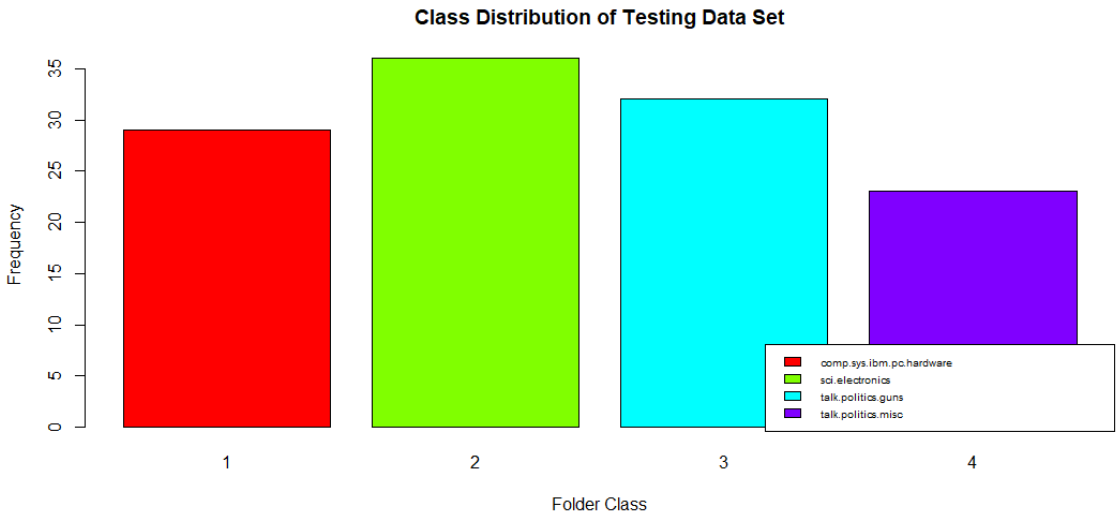

Figure 2: Class Distribution of Testing Data Set

Further, the data (text) that was available had header, footer and quotes included, and looked something like:

```
From: lerxst@wam.umd.edu (where's my thing)
Subject: WHAT car is this!?
Nntp-Posting-Host: rac3.wam.umd.edu
Organization: University of Maryland, College Park
Lines: 15

 I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.

Thanks,
- IL
   ---- brought to you by your neighborhood Lerxst ----
```

I extracted the main body of text by removing header, footer and quotes and eventually the output looked something like:

```
I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.
```

I was getting better accuracies with the initial version (~10% more) but I believe that it was because of the bias introduced into the data. Most writers write on a particular set of topics and our models were using that information also to classify. This meant that the classifier gave less weightage to what exactly the text talked about. This could be a major issue when we see a test

data that doesn't have such information (author, server, organization) included. So, to train a good general model that takes into account only the words used I did the extraction.

3.2 Exploration of Dataset

After the Preprocessing in 3.1, I have removed the Emails and I am also removing the most common terms in the Clean_data.R file.

```
> # Function to remove emails
> removeEmail <- function(X) {
+             str_replace_all(X, "[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+", "")
+ }
> # Funtion to remove most common terms by percentage
> removeCommonTerms <- function(X, percentage) {
+             x <- t(X)
+             t <- table(x$i) < x$ncol * percentage
+             X[, as.numeric(names(t[t]))]
+ }
```

Using pipes to not allocate extra memory. Using Corpus to create the bag of words and then using it for the rest of the evaluation process. I created two files using the Corpus

1. Bag of Words with Document Term Index Weight- TF

2. Bag of Words with Document Term Index Weight- TFIDF

3.2.1 Preprocessing Techniques: Using TF instead of TF-IDF

The motivation behind using IDF as a weighting factor in TF-IDF is:

1. To suppress the more frequently occurring terms in dataset. (eg stop words), if a word occurs in all the document of the corpus then IDF is 0.
2. To give importance to terms that occur in few documents (eg if a word occurs in only 2 documents among 100 then IDF is log (100/2)).

If these two conditions fail i.e.
1. If important words occur throughout the document in your corpus, and frequency of these words are important for your task.
2. If the frequency of rare words those occur only in few documents of your corpus, are not much useful for your task then naturally TF-IDF will not be useful much as TF

3.3 Tokenization and Compute the Top 200 most Popular Words

Like in every machine learning problem, you have to extract features in order to train a model. These algorithms can read-in just numbers and you have to find a way to convert the text into the numerical feature vectors.

If you think about it, a text is just a series of ordered words that usually carry some meaning. If we take each unique word from all the available texts, we'll create our own vocabulary. And every word in a vocabulary can be one feature. For each text, feature vector will be an array where feature values are simply the numbers of unique word repetition in a specific text, i.e. just the count of each word in one text. And if some word is not in the text, its feature value is zero.

Therefore, the word order in a text is not important, just the number of repetitions. This method is called a “Bag of words” and it’s quite common and simple to use.

3.3.1 Sparse Terms and Common Terms Removal

You can use different approaches for word scores/values, but the most popular one is TF-IDF (Term Frequencies times Inverse Document Frequency), which calculates the frequency of a word in a document and reduces the weight of such common words like “the” or “is”. But here we’ll just use tm build-in function TF Vectorizer and highlight the most important words from each text.

```
+ removeSparseTer4ms(0.99) %>%          # at least in 1% documents
+ removeCommonTerms(0.50) %>%            # maximun 50% documents
```

The Total Document Term frequency is 1738 after doing all the preprocessing techniques. Using term frequency and inverse document frequency allows us to find words that are characteristic for one document within a collection of documents, whether that document has a email or anyother text or punctuations.

```
> names(frequency)
  [1] "use"                "one"                "can"
  [4] "will"               "peopl"              "wire"
  [7] "know"               "get"                "like"
 [10] "just"               "think"              "univers"
 [13] "gun"                "make"               "sender"
 [16] "want"               "time"               "nntppostinghost"
 [19] "right"              "work"               "also"
 [22] "say"                "need"               "system"
 [25] "xref"               "cantaloupesrvcscmuedu" "new"
 [28] "good"               "two"                "drive"
 [31] "state"              "govern"             "thing"
 [34] "power"              "said"               "well"
 [37] "problem"            "control"            "ground"
 [40] "presid"             "way"                "may"
 [43] "law"                "tri"                "even"
 [46] "now"                "much"               "talkpoliticsgun"
 [49] "believ"             "question"           "news"
 [52] "connect"            "look"               "circuit"
 [55] "scielectron"        "mani"               "compsysibmpchardwar"
 [58] "differ"             "talkpoliticsmisc"   "distribut"
 [61] "anyon"              "mean"               "case"
 [64] "run"                "point"              "protect"
 [67] "number"             "inform"             "post"
 [70] "group"              "take"               "comput"
 [73] "card"               "person"             "requir"
 [76] "sure"               "year"               "see"
 [79] "help"               "sinc"               "possibl"
 [82] "fire"               "come"               "reason"
 [85] "must"               "realli"             "repli"
 [88] "set"                "day"                "respons"
 [91] "thank"              "seem"               "actual"
 [94] "start"              "first"              "made"
 [97] "find"               "general"            "anyth"
[100] "chang"              "someth"             "buy"
[103] "consid"             "cabl"               "etc"
[106] "pleas"              "world"              "call"
[109] "pay"                "public"             "tax"
[112] "usa"                "instal"             "pin"
[115] "data"               "fact"               "support"
```

```
[118] "servic"      "box"       "weapon"
[121] "anoth"       "place"     "part"
[124] "softwar"     "opinion"   "happen"
[127] "electr"      "never"     "probabl"
[130] "lot"         "compani"   "might"
[133] "got"         "hous"      "fbi"
[136] "still"       "claim"     "current"
[139] "kill"        "tell"      "bit"
[142] "program"     "read"      "back"
[145] "last"        "without"   "signal"
[148] "give"        "interest"  "caus"
[151] "polic"       "suggest"   "ask"
[154] "usenet"      "input"     "david"
[157] "follow"      "certain"   "devic"
[160] "order"       "usual"     "provid"
[163] "enough"      "forc"      "children"
[166] "clinton"     "better"    "least"
[169] "copi"        "talk"      "end"
[172] "includ"      "health"    "fri"
[175] "switch"      "put"       "batf"
[178] "note"        "file"      "long"
[181] "port"        "yes"       "cours"
[184] "money"       "insur"     "specif"
[187] "price"       "show"      "wrong"
[190] "firearm"     "around"    "john"
[193] "abort"       "nation"    "less"
[196] "second"      "tue"       "keep"
[199] "far"         "mark"      "code"
[202] "standard"    "old"       "chip"
```

3.4 Top 200 most popular words with tokens length (min. size = 4 and max. size = 20)

Exploring term frequency on its own can give us insight into how language is used in a collection of natural language, and tm verbs like count and rank gives us tools to reason about term frequency. The tm package uses an implementation of tf consistent with data principles that enables us to see how different words are important in documents within a collection or corpus of documents.

```
> frequency[1:200]
             will              peopl               wire               know
              335               269               265                262
             like               just              think             univers
              238               221               218                209
             make             sender              want               time
              206               196               191                190
   nntppostinghost              right              work               also
              186               184               182                180
             need             system              xref               good
              168               168               152                143
            drive              state            govern              thing
              138               138               138                131
            power               said              well            problem
              130               128               127                125
          control             ground            presid               even
              125               125               125                117
            much     talkpoliticsgun            believ           question
              114               113               111                111
             news            connect              look            circuit
              111               108               107                107
      scielectron             mani  compsysibmpchardwar             differ
              107               106               105                105
  talkpoliticsmisc         distribut             anyon               mean
              105               104               104                103
             case             point           protect             number
              102                98                97                 96
           inform              post             group               take
               96                95                95                 94
           comput              card            person              requir
               93                92                92                 92
             sure              year              help               sinc
               90                90                87                 87
         possibl              fire              come             reason
               87                87                86                 86
             must            realli             repli            respons
               86                85                85                 84
            thank              seem            actual              start
               83                83                83                 83
```

first	made	find	general
82	82	82	80
anyth	chang	someth	consid
78	78	78	77
cabl	pleas	world	call
75	74	74	74
public	instal	data	fact
74	73	72	72
support	servic	weapon	anoth
72	72	72	71
place	part	softwar	opinion
70	70	70	70
happen	electr	never	probabl
70	69	69	69
compani	might	hous	still
68	68	68	67
claim	current	kill	tell
66	66	66	65
program	read	back	last
65	65	64	64
without	signal	give	interest
64	63	63	63

caus	polic	suggest	usenet
63	63	62	62
input	david	follow	certain
62	62	62	62
devic	order	usual	provid
61	61	61	61
enough	forc	children	clinton
61	61	61	61
better	least	copi	talk
60	60	60	59
includ	health	switch	batf
59	58	57	57
note	file	long	port
56	56	56	56
cours	money	insur	specif
56	56	56	55
price	show	wrong	firearm
54	54	54	54
around	john	abort	nation
53	53	53	53
less	second	keep	mark
53	52	52	52
code	standard	chip	howev
52	52	51	51
hard	packag	oper	situat
51	51	51	51
allow	care	death	access
51	51	50	50
done	issu	build	high
50	50	50	50
american	origin	local	live
50	49	49	49
everi	scsi	hand	idea
49	49	49	49
discuss	author	board	either
49	49	48	48
email	someon	ever	press
48	48	48	48
feder	name	though	cost
48	47	47	47

4. Research Aspect

Now let's build a classifier! I'll start with the most common one: the multinomial Naive Bayes classifier which is suitable for discrete classification. R has a great Class called Pipeline, which allows us to a create pipeline for a classifier, i.e. you can just add the functions that you wanna use on your input data. Here, we are using a Tf Vectorizer as vectorizer and Multinomial as classifier

4.1 More Preprocessing Techniques (Stemming)

Word segmentation (or word tokenizing) – separates continues text into independent words. In most western languages (such as English, Croatian or French) words are separated by space. But for languages like Chinese or Japanese where words aren't delimited, it can be quite a problem!

```
+ tm_map(content_transformer(tolower)) %>%        # no uppercase
+ tm_map(content_transformer(removeEmail)) %>%    # remove email
+ tm_map(removeNumbers) %>%                       # no numbers
+ tm_map(removeWords, stopwords('en')) %>%        # remove stopwords
+ tm_map(removePunctuation) %>%                   # no punctuation
+ tm_map(stripWhitespace) %>%                     # no extra whitespaces
```

```
+ tm_map(stemDocument) %>%                    # reduce to radical
```

StemDocument is used to reduce to the radical and other Preprocessing techniques add up to give a Total Document Term Frequency of 1571

Stemming – is a process of reducing inflected words to their word stem, i.e root. It doesn't have to be morphological; you can just chop off the words' ends. For example, word "solv" is the stem of words "solve" and "solved".

Folder_class has been added to the bag of words as the predictor variable with cbind technique

4.2 Methodology

In this study, I compare various features of TF Vectorizer to find the set which works the best under different classification models. I start the modeling procedure by first considering stemming in the vectorizer. Next, I will applied other methods of preprocessing in the vectorizer. After all of settings we use only unigrams as the resulting features from the TF Vectorizer and then compare it with the results of when both TF-IDF which are considered in the feature set. Subsequently, after all of these combinations we consider four different classification models to examine how they perform on the test data. These include

- Naive Bayes,
- k Nearest Neighbors,
- Random Forest,
- SVM
- Random Forest with Random Forest Library
- K Nearest Neighbor with Cross Validation
- Stochastic Gradient Descent Classifier,
- Decision Trees

The analysis is performed using different library in R and they are mentioned below:

- library(tm)- Text Mining Package
- library(dplyr)- A Grammar of Data Manipulation
- library(stringr)- Character manipulation
- library(e1071)- Misc Functions of the Department of Statistics
- library(mlr)- For the machine learning algorithms;
- library(readtext)- Makes it easy to import text files in various formats
- library(caret)- Build all sorts of machine learning models.
- library(randomForest) – Random Forest Implementation
- library(MASS)- Support Functions and Datasets for Venables

4.3 Naïve Bayes

I started with the most commonly used algorithm for text classification, Naive Bayes, to get a sense of how we should proceed.

On the first try without tuning any hyper-parameters, I hit a 51% overall accuracy.

4.3.1 Calculating Conditional Probabilities

Bayes Theorem provides a principled way for calculating a conditional probability.Although it is a powerful tool in the field of probability, Bayes Theorem is also widely used in the field of machine learning. Including its use in a probability framework for fitting a model to a training dataset, referred to as maximum a posteriori or MAP for short, and in developing models for classification predictive modeling problems such as the Bayes Optimal Classifier and Naive Bayes.

Conditional Probability: Probability of one (or more) event given the occurrence of another event, e.g. P(A given B) or P(A | B).

The joint probability can be calculated using the conditional probability; for example:

$$P(A, B) = P(A \mid B) * P(B)$$

This is called the product rule. Importantly, the joint probability is symmetrical, meaning that:

$$P(A, B) = P(B, A)$$

The conditional probability can be calculated using the joint probability; for example:

$$P(A \mid B) = P(A, B) / P(B)$$

The conditional probability is not symmetrical; for example:

$$P(A \mid B) != P(B \mid A)$$

```
naive_bayes <- function(datafr,lambda=0){
  num_features <- ncol(datafr) - 1
  num_obs <- nrow(datafr)
  num_class <- length(unique(datafr[,num_features+1]))
  # Calculating conditional probabilities
  for (i in 1:num_features) {
    tab <- table(datafr[,i],datafr[,num_features+1]) + lambda
    if (i==1){
      all <- apply(tab,2,sum)
      conprob <- sweep(tab,2,all,'/')
    }
    else
    {
      update <- sweep(tab,2,all,'/')
      conprob <- rbind(conprob,update)
    }
  }
  all <- all/sum(all)
  conprob <- rbind(conprob,all)
  # Calculating predictions
  for (i in 1:num_obs) {
    obs <- conprob[c(as.character(datafr[i,1:num_features]),"all"),]
    obsprob <- apply(obs,2,prod)
    if (i==1) {
      preds <- names(which.max(obsprob))
    }
    else{
      preds <- c(preds,names(which.max(obsprob)))
    }
  }
  error <- 100*(1-sum(preds==datafr[,num_features+1])/length(datafr[,1]))
  model <- list(conprob,datafr[,num_features+1],preds,error)
  return(model)
```

4.3.2 Confusion Matrix for Predictions using Naïve Bayes Condition Probabilities

Take a closer look at the performance analysis, we can see that the Naive Bayes algorithm performs the best on "hardware" typed newsgroups, almost all accuracies in the "computer" category achieved above 40%. Whereas for talk related newsgroups, the accuracies range from 45% to 56%.

```
> confusionMatrix(table(pred_dataset[[2]],pred_dataset[[3]]))
Confusion Matrix and Statistics


     1   2   3   4
  1 100   0   0   0
  2  69  31   0   0
  3  98   0   2   0
  4  29   0   0  71

Overall Statistics

               Accuracy : 0.51
                 95% CI : (0.4598, 0.56)
    No Information Rate : 0.74
    P-Value [Acc > NIR] : 1

                  Kappa : 0.3467

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            0.3378   1.0000   1.0000   1.0000
Specificity            1.0000   0.8130   0.7538   0.9119
Pos Pred Value         1.0000   0.3100   0.0200   0.7100
Neg Pred Value         0.3467   1.0000   1.0000   1.0000
Prevalence             0.7400   0.0775   0.0050   0.1775
Detection Rate         0.2500   0.0775   0.0050   0.1775
Detection Prevalence   0.2500   0.2500   0.2500   0.2500
Balanced Accuracy      0.6689   0.9065   0.8769   0.9559
```

4.4 K-Nearest Neighbors

KNN is a Supervised Learning algorithm that uses labeled input data set to predict the output of the data points. It is one of the simplest Machine learning algorithms and it can be easily implemented for a varied set of problems. It is mainly based on feature similarity. KNN checks how similar a data point is to its neighbor and classifies the data point into the class it is most similar to.

Consider the set, (Xi, Ci),

- Where Xi denotes feature variables and 'i' are data points ranging from i=1, 2, ….., n
- Ci denotes the output class for Xi for each i

The condition, Ci ∈ {1, 2, 3, ……, c} is acceptable for all values of 'i' by assuming that the total number of classes is denoted by 'c'.

```
> task = makeClassifTask(data = data.train, target = target)
> learner_knn = makeLearner("classif.knn", k=3)
> train.set = c(1:nrow(data.train))
> model_knn = mlr::train(learner_knn, task, subset = train.set)
> unseen_predictions_knn = predict(model_knn, newdata = data.test, type="folder_class")
> accuracy_knn = performance(unseen_predictions_knn, measures = acc)
> cat("Accuracy KNN: ", accuracy_knn)
Accuracy KNN:  0.925
```

4.5 Random Forest with MLR Package

The random forest algorithm works by aggregating the predictions made by multiple decision trees of varying depth. Every decision tree in the forest is trained on a subset of the dataset called the bootstrapped dataset.

```
> model_randomforest = mlr::train(learner_randomforest, task, subset = train.set)
> unseen_predictions_randomforest = predict(model_randomforest, newdata = data.test)
> accuracy_randomforest = performance(unseen_predictions_randomforest, measures = acc)
> cat("Accuracy Random Forest: ", accuracy_randomforest)
Accuracy Random Forest:  1
```

5. Empirical Evaluation

5.1 Robust Evaluation

5.2 Support Vector Machine

SVM is a supervised learning algorithm. This means that SVM trains on a set of labeled data. SVM studies the labeled training data and then classifies any new input data depending on what it learned in the training phase.

A main advantage of SVM is that it can be used for both classification and regression problems. Though SVM is mainly known for classification, the SVR (Support Vector Regressor) is used for regression problems.

SVM can be used for classifying non-linear data by using the kernel trick. The kernel trick means transforming data into another dimension that has a clear dividing margin between classes of data. After which you can easily draw a hyperplane between the various classes of data.

```
> confusionMatrix(table(predsvm,data.test$folder_class))
Confusion Matrix and Statistics


predsvm  1  2  3  4
       1 31  0  0  0
       2  0 29  6  0
       3  0  3 28  0
       4  0  0  0 23

Overall Statistics

               Accuracy : 0.925
                 95% CI : (0.8624, 0.9651)
    No Information Rate : 0.2833
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8994

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
```

```
Sensitivity          1.0000   0.9062   0.8235   1.0000
Specificity          1.0000   0.9318   0.9651   1.0000
Pos Pred Value       1.0000   0.8286   0.9032   1.0000
Neg Pred Value       1.0000   0.9647   0.9326   1.0000
Prevalence           0.2583   0.2667   0.2833   0.1917
Detection Rate       0.2583   0.2417   0.2333   0.1917
Detection Prevalence   0.2583   0.2917   0.2583   0.1917
Balanced Accuracy      1.0000   0.9190   0.8943   1.0000
```

Support Vector Machine has an Accuracy of 92.5%

5.3 Random Forest

The portion of samples that were left out during the construction of each decision tree in the forest are referred to as the Out-Of-Bag (OOB) dataset. As we'll see later, the model will automatically evaluate its own performance by running each of the samples in the OOB dataset through the forest.

```
> confusionMatrix(table(round(predrandomforest), data.test$folder_class))
Confusion Matrix and Statistics


   1  2  3  4
 1 31  0  0  0
 2  0 32  0  0
 3  0  0 34  1
 4  0  0  0 22

Overall Statistics

               Accuracy : 0.9917
                 95% CI : (0.9544, 0.9998)
    No Information Rate : 0.2833
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9888

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity          1.0000   1.0000   1.0000   0.9565
Specificity          1.0000   1.0000   0.9884   1.0000
Pos Pred Value       1.0000   1.0000   0.9714   1.0000
Neg Pred Value       1.0000   1.0000   1.0000   0.9898
Prevalence           0.2583   0.2667   0.2833   0.1917
Detection Rate       0.2583   0.2667   0.2833   0.1833
Detection Prevalence   0.2583   0.2667   0.2917   0.1833
Balanced Accuracy      1.0000   1.0000   0.9942   0.9783
```

Random Forest has an Accuracy of 99.17%

5.4 K- Nearest Neighbor with Cross- Validation

k-Nearest-Neighbors (k-NN) is a supervised machine learning model. Supervised learning is when a model learns from data that is already labeled. A supervised learning model takes in a set of input objects and output values. The model then trains on that data to learn how to map the inputs to the desired output so it can learn to make predictions on unseen data.

k-NN models work by taking a data point and looking at the 'k' closest labeled data points. The data point is then assigned the label of the majority of the 'k' closest points.

For example, if k = 5, and 3 of points are 'green' and 2 are 'red', then the data point in question would be labeled 'green', since 'green' is the majority

```
> cv.knn <- crossval(learner = learner_knn,task = task,iters = 3,stratify = TRUE,measures = acc,show.info = F)
> cv.knn$aggr
acc.test.mean
    0.942843
> cv.knn$measures.test
  iter       acc
1    1 0.9354839
2    2 0.9462366
3    3 0.9468085
> unseen_predictions_knn = predict(model_knn, newdata = data.test, type="folder_class")
> accuracy_knn = performance(unseen_predictions_knn, measures = acc)
> cat("Accuracy KNN: ", accuracy_knn)
Accuracy KNN:  0.925
```

KNN with Cross Validation has an Accuracy of 94.28%

5.5 Hyper-parameter Tuning for Implementing it with Decision Trees

Tuning a model often requires exploring the impact of changes to many hyperparameters. The best way to approach this is generally not by changing the source code of the training script, but instead by defining flags for key parameters then training over the combinations of those flags to determine which combination of flags yields the best model

```
[Tune-x] 997: minsplit=37; minbucket=50; cp=0.2
[Tune-y] 997: acc.test.mean=0.5924274; time: 0.0 min
[Tune-x] 998: minsplit=41; minbucket=50; cp=0.2
[Tune-y] 998: acc.test.mean=0.5924274; time: 0.0 min
[Tune-x] 999: minsplit=46; minbucket=50; cp=0.2
[Tune-y] 999: acc.test.mean=0.5924274; time: 0.0 min
[Tune-x] 1000: minsplit=50; minbucket=50; cp=0.2
[Tune-y] 1000: acc.test.mean=0.5924274; time: 0.0 min
[Tune] Result: minsplit=32; minbucket=30; cp=0.0894 : acc.test.mean=0.9857775
```

Decision Trees with Hyper-parameter tuned has an Accuracy of 98.57%

5.6 Gradient Boosting

Now you are entering the territory of boosting algorithms. GBM performs sequential modeling i.e after one round of prediction, it checks for incorrect predictions, assigns them relatively more weight and predict them again until they are predicted correctly.

```
> target = colnames(data.train)[ length(colnames(data.train))]
> task = makeClassifTask(data = data.train, target = target)
> learner_gbm = makeLearner("classif.gbm")
> train.set = c(1:nrow(data.train))
> model_gbm = mlr::train(learner_gbm, task, subset = train.set)
Distribution not specified, assuming multinomial ...
Warning message:
In gbm.fit(x = x, y = y, offset = offset, distribution = distribution,  :
  variable 1146: frequent has no variation.
> unseen_predictions_gbm = predict(model_gbm, newdata = data.test, type="folder_class")
> accuracy_gbm = performance(unseen_predictions_gbm, measures = acc)
> cat("Accuracy GBM: ", accuracy_gbm)
Accuracy GBM:  1
```

GBM has an Accuracy of 100%

5.7 XGBoost

Xgboost is considered to be better than GBM because of its inbuilt properties including first and second order gradient, parallel processing and ability to prune trees. General implementation of xgboost requires you to convert the data into a matrix. With mlr, that is not required.

```
> target = colnames(data.train)[ length(colnames(data.train))]
> task = makeClassifTask(data = data.train, target = target)
> learner_xgboost = makeLearner("classif.xgboost")
> train.set = c(1:nrow(data.train))
> model_xgboost = mlr::train(learner_xgboost, task, subset = train.set)
> unseen_predictions_xgboost = predict(model_xgboost, newdata = data.test, type="folder_clas
s")
> accuracy_xgboost = performance(unseen_predictions_xgboost, measures = acc)
> cat("Accuracy XGBOOST: ", accuracy_xgboost)
Accuracy XGBOOST:  0.9916667
```

XGBoost has an Accuracy of 99.16%

Table1: Model vs Accuracy

Model	Accuracy (%)
Naïve Bayes	51
KNN	92.5
Random Forest	100
SVM	92.5
KNN with Cross Validation	94.28
Decision Trees with Tuning	98.57
Gradient Boosting	100
XGBoost	99.16

6. Conclusion and Future Work

Overall, we saw that the best performing classifiers were, Gradient Boosting, XGBoost and Random Forest. We also noticed that using TF Vectorizer gave better results than TF-IDF Vectorizer. The best accuracy for Random Forest was achieved by using Stemming along with both unigrams and bigrams extracted in TF Vectorizer. For KNN better results were achieved using Lemmatization and limiting the parameters which significantly increased the speed of training. Also, unigrams helped

us achieve high accuracy most of the time, so we don't really need to extend the features and increase the time taken by the classifiers. Finally, we were able to achieve an accuracy of 51% for 4 classes using the techniques learned in class and training different models and optimizing their parameters using grid search.

6.1 Discussion about an Interesting Deduction

The most interesting deduction is that the more specific the newsgroup topic is, the more accurate that the Naïve Bayes classifier can determine what newsgroup a document belongs to and the converse is also true where the less specific the newsgroup is, the accuracy rate plummets.

We can see this in Accuracy where every newsgroup that isn't a misc will always have an accuracy rate of at least 50%. The bottom newsgroups for terms of accuracy rate are all misc which includes a 0.25% accuracy rate for talk.politics.misc.

A reason for this is that the posts that are written in misc newsgroups are rarely related to the actual root of the newsgroup. The misc section caters to other topics of discussion other than the "root newsgroup" meaning that it is much easier for the classifier to confuse a document from a misc newsgroup with another newsgroup and much harder for the classifier to even consider the root newsgroup since topics regarding the root newsgroup at posted there instead.

For example, a post about guns is posted in talk.religion.misc can be easily classified as being talk.politics.guns because it would have to use similar words found in the posts found in talk.politics.guns. Likewise, posts about politics in talk.politics.misc are less likely because you are more likely to post in or talk.politics.guns (where wildcard is the relevant section for the type of politics to be discussed).

7. References

1. Why Term Frequency is better than TF-IDF for text classification: https://www.quora.com/Why-does-TF-term-frequency-sometimes-give-better-F-scores-than-TF-IDF-does-for-text-classification

2. Naïve Bayes Classification for 20 News Group Dataset: https://github.com/Loc-Tran/NaiveBayes20NewsGroup

3. Analyzing word and document frequency: tf-idf : https://www.tidytextmining.com/tfidf.html

4. Natural Language Processing : https://krakensystems.co/blog/2018/nlp-syntax-processing

5. K Nearest Neighbor in R: https://www.edureka.co/blog/knn-algorithm-in-r/

6. MLR Package : https://www.analyticsvidhya.com/blog/2016/08/practicing-machine-learning-techniques-in-r-with-mlr-package/