Walkthrough of NLP Custom Functions

Jack Ogozaly

2/2/2021

The Goal:

This document walks through how the custom functions developed for this project can be used. This document will utilize a public dataset on airline sentiment as a proof of concept.

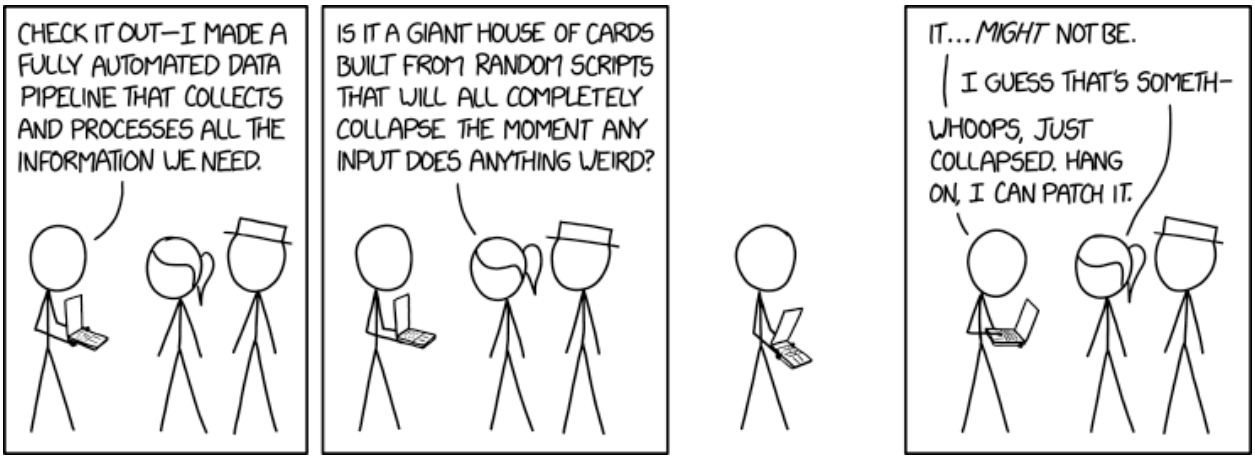


Figure 1: "Data Pipeline" by xkcd

The goal is to avoid the situation described in the comic above. These functions have been tailor made for a specific project but should still be robust enough to be used on unconventional data. These functions should automate much of the training, testing, and predicting phases of a project. The information contained in this document may change as the project progresses.

The Functions:

The functions in this document require the following packages:

```
library(tm) #Used for text cleaning
library(tidytext) #Used for text cleaning
library(caret) #Used for data balancing
library(e1071) #Used for naive bayes model
library(matrixStats) #Used to extract key values from prediction matrix
```

This first function is used to clean the text that should be the field you are trying to make predictions off of.

```
text_clean <- function(text_input, # should be a columm from a dataframe
                       dataframe #should be the dataframe you want the clean text to be attached to
){
  #These lines clean the text input
  tidy_text <- text_input
  tidy_text <- removePunctuation(tidy_text)
  tidy_text <- tolower(tidy_text)
  tidy_text <- removeNumbers(tidy_text) #This line is optional
  tidy_text <- removeWords(tidy_text, stop_words$word)
  #These lines create a new dataframe, attaches the clean text, and exports
  tidy_text_df <- as.data.frame(dataframe)
  tidy_text_df$tidy_text <- tidy_text
  assign("tidytext_df", tidy_text_df, envir = globalenv())
}
```

This second function balances the data with upsampling, and divides the data into train and test datasets.

```
balanced_data <- function(x, #Dataframe
                          y, #Labels to be balanced
                          train_percent = .75, #What percent of the data do you wish to train on? 75% b
                          upsample= T) #If true upsamples the data, if false downsamples. True by defau
{
  set.seed(42)
  df <- as.data.frame(x)
  df$label <- as.factor(y)
  rows <- sample(nrow(df))
  df <- df[rows, ]
  rownames(df) <- NULL
  observations <- as.numeric(nrow(df))
  observations <- ceiling(observations)
  train_upper_limit <- train_percent * observations
  train_upper_limit <- ceiling(train_upper_limit)
  train_data <- df[1:train_upper_limit, ]
  test_data <- df[train_upper_limit:observations, ]
  if(upsample==T){
    new.df <- as.data.frame(upSample(train_data, train_data$label))
    number <- nrow(new.df)
    new.df <- new.df[,-ncol(new.df)]
    colnames(test_data) <- colnames(new.df)
    new.df <- rbind(new.df, test_data)
    assign("balanced_df", new.df, envir = globalenv())
    print("The row to stop training at is:")
    print(number)
  }
  else{
    new.df <- as.data.frame(downSample(train_data, train_data$label))
    number <- nrow(new.df)
    new.df <- new.df[,-ncol(new.df)]
    colnames(test_data) <- colnames(new.df)
    new.df <- rbind(new.df, test_data)
    assign("balanced_df", new.df, envir = globalenv())
    print("The row to stop training at is:")
    print(number)
```

```
  }
}
```

The third function trains and tests the model and outputs the results of it.

```
naive_bayes <- function(x, #independent variable; should be a vector of text
                        y, #dependent variable, should be a vector of labels
                        train_rows, #What row num should the model train up to?
                        plot = TRUE, #Prints a confusion matrix by default
                        print_stats = TRUE) #Do you want the model to print confusion matrix results?
{
  #Convert text data to a corpus and then a document term matrix so that the model can be trained off i
  function_corpus <- VCorpus(VectorSource(x))
  function_dtm <- DocumentTermMatrix(function_corpus,
                                     control = list(tolower = TRUE,removeNumbers = TRUE,
                                                    stopwords = TRUE,
                                                    removePunctuatio = TRUE,
                                                    stemming = TRUE))
  x_dataframe <- as.data.frame(x)#Converts column to df

  if (is.factor(y) != TRUE) {
    y <- as.factor(y) #Converts labels to a factor so that classification can happen
  }
  y_dataframe <- as.data.frame(y) #Converts column of labels to df

  #Divides the data into train and test
  observations <- as.numeric(nrow(x_dataframe))
  function_dtm_train <- function_dtm[1:train_rows,]
  function_dtm_test <- function_dtm[train_rows:observations,]
  function_train_labels <- y_dataframe[1:train_rows,]
  function_test_labels <- y_dataframe[train_rows:observations,]
  #Counting freq of words and training the naive bayes model
  function_freq_words <- findFreqTerms(function_dtm_train,5)
  function_dtm_freq_train <- function_dtm_train[,function_freq_words]
  function_dtm_freq_test <- function_dtm_test[,function_freq_words]
  convert_counts <- function(x){
    x <- ifelse(x>0,"Yes","No")
  }
  function_train <- apply(function_dtm_freq_train,MARGIN = 2,convert_counts)
  function_test <- apply(function_dtm_freq_test,MARGIN = 2,convert_counts)
  #Trains the naive Bayes model
  function_classifier <- naiveBayes(function_train, as.factor(function_train_labels))
  #Testing the classifier on the test data
  function_test_pred <- predict(function_classifier, newdata=function_test)
  #Exporting the classifier
  assign("naive_bayes_classifier", function_classifier, envir=globalenv())
  #Creating statistical measures to evaluate model
  cfm <- as.data.frame(table(function_test_pred, function_test_labels))
  print("Model Accuracy is:")
  print(1-mean(function_test_pred != function_test_labels))
  cfm_stats <- confusionMatrix(function_test_pred, function_test_labels,positive = "pos")
  #If the user selected it this prints the cofusion matrix stats
  if(print_stats == TRUE){
```

```
    assign("confusion_matrix_stats", cfm_stats, envir=globalenv())
  }
  #if the user wants a plot this code runs
  if(plot == TRUE){
    #Plots Predicted vs Actual labels from our test data
    print(ggplot(data = cfm,
                 mapping = aes(x = function_test_pred,
                               y = function_test_labels)) +
            geom_tile(aes(fill = Freq)) +
            geom_text(aes(label = sprintf("%1.0f", Freq)), vjust = 1) +
            scale_fill_gradient(low = "blue",
                                high = "red",
                                trans = "log") +ylab("Actual Labels\n") +
            scale_x_discrete(guide = guide_axis(n.dodge=3))+
            xlab("\nPredicted Labels"))
  }
}
```

The fourth function converts a vector of text into a matrix so that the model is able to make predictions off of it. The function takes this input and creates an object in the global environment that we can then use. (This is the consequence of training the model on a document term matrix)

```
predict_input <- function(x) #Should be a vector of text
{
  #Converts to a document term matrix
  function_corpus <- VCorpus(VectorSource(x))
  function_dtm <- DocumentTermMatrix(function_corpus,
                                     control = list(tolower = TRUE,removeNumbers = TRUE,
                                                    stopwords = TRUE,
                                                    removePunctuatio = TRUE,
                                                    stemming = TRUE))
  #Creates bag of words matrix
  function_freq_words <- findFreqTerms(function_dtm,1) #This three here specifies how many times a word
  function_dtm_freq <- function_dtm[,function_freq_words]
  convert_counts <- function(x){
    x <- ifelse(x>0,"Yes","No")
  }
  function_test <- apply(function_dtm_freq,MARGIN = 2,convert_counts)
  assign("predict.input", function_test, envir=globalenv())
}
```

The fifth and final function converts a column of text into document term matrix (the same as function #4) but it also then runs the model on that input, and then exports the model's best guess, its confidence in the best guess, and then the second best guess.

```
confidence_interval <- function(x, #Should be a model
                                y) #Should be the model's input
{
  #Function that converts the input into a bag of words matrix
  predict_input <- function(x)
  {
    function_corpus <- VCorpus(VectorSource(x))
    function_dtm <- DocumentTermMatrix(function_corpus,
```

```
                                        control = list(tolower = TRUE,removeNumbers = TRUE,
                                                       stopwords = TRUE,
                                                       removePunctuatio = TRUE,
                                                       stemming = TRUE))

    function_freq_words <- findFreqTerms(function_dtm,3)
    function_dtm_freq <- function_dtm[,function_freq_words]
    convert_counts <- function(x){
      x <- ifelse(x>0,"Yes","No")
    }
    function_test <- apply(function_dtm_freq,MARGIN = 2,convert_counts)
    assign("predict.input", function_test, envir=globalenv())
  }
  y <- predict_input(y)
  #Runs prediction on on the text and generates probability of every label
  df <- as.matrix(predict(x, y, type="raw"))
  #Picks the highest value in the predicted dataframe
  max_row <- as.data.frame(rowMaxs(df, value = FALSE))
  #Change column name
  colnames(max_row) <- c("confidence")
  #Attach a column to our df with the best prediction from the model
  max_row$Prediction <- predict(x,y)
  #Find and attach the column name for cell that has the second highest probability
  max_row$second_most_probable <- apply(df[,-ncol(df)], 1,
                                        FUN = function(x) which(x == sort(x, decreasing = TRUE)[2]))
  max_row$second_most_probable <- colnames(df)[max_row$second_most_probable]
  #Export to global enviornment so that it can be viewed
  assign("confidence_interval.df", max_row, envir=globalenv())
}
```

Airline Tweet Sentiment Example:

These functions were made specifically for entity resolution for a supply chain dataset, but they are versatile enough that they can also be used for other NLP supervised learning projects. This section will use an airline tweet sentiment dataset to demonstrate how the functions should be used. This dataset can be found at: https://www.kaggle.com/crowdflower/twitter-airline-sentiment

Let's load in our data

```
Airlines_Tweet_Dataset <- read.csv("Tweets.csv")
Airlines_New_Data <- Airlines_Tweet_Dataset[2001:2500,]
Airlines_Tweet_Dataset <- Airlines_Tweet_Dataset[1:2000,]
```

Then, let's clean the text field we want to predict off of.

```
text_clean(Airlines_Tweet_Dataset$text, Airlines_Tweet_Dataset)
```

Next, since tweets are biased towards being negative, we should balance the data. Our function balanced_data automatically splits the data 75 train 25 test but you can change this by typing : train_percent=X

```
balanced_data(tidytext_df, tidytext_df$airline_sentiment)
```

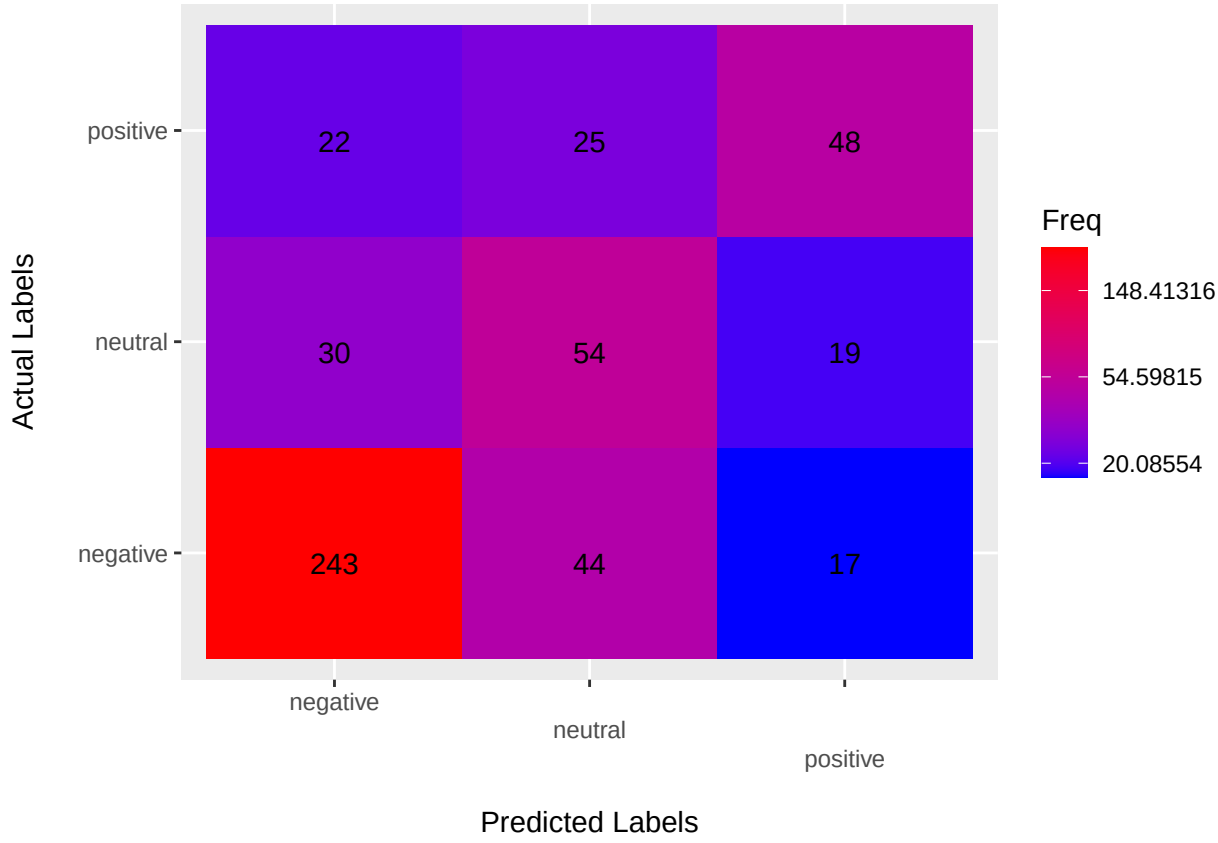
```
## [1] "The row to stop training at is:"
## [1] 2862
```

Notice that the function returns a balanced dataframe but it also prints at what row the training data stops. This is important because the function only upsampled our training data and inputting this number into our naive bayes function will ensure the train and test data stays entirely seperate and also makes sure the test data is not upsampled.

Next, let's finally train our model and evaluate its performance.

```
naive_bayes(balanced_df$tidy_text, balanced_df$label, 2862)
```

```
## [1] "Model Accuracy is:"
## [1] 0.687251
```



Our naive_bayes function has trained and exported a model, and has exported the statistics for this model. As we can see we have ~68% accuracy on the test dataset. If we want more advanced statistics we can simply view the confusion_matrix_stats object the function returned.

```
head(confusion_matrix_stats)
```

```
## $positive
```

```
## [1] "pos"
##
## $table
##           Reference
## Prediction negative neutral positive
##   negative      243      30       22
##   neutral        44      54       25
##   positive       17      19       48
##
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   6.872510e-01   4.436986e-01   6.446830e-01   7.275968e-01   6.055777e-01
## AccuracyPValue  McnemarPValue
##   8.870578e-05   2.500507e-01
##
## $byClass
##                  Sensitivity Specificity Pos Pred Value Neg Pred Value Precision
## Class: negative    0.7993421   0.7373737      0.8237288      0.7053140 0.8237288
## Class: neutral     0.5242718   0.8270677      0.4390244      0.8707124 0.4390244
## Class: positive    0.5052632   0.9115479      0.5714286      0.8875598 0.5714286
##                    Recall        F1 Prevalence Detection Rate
## Class: negative 0.7993421 0.8113523  0.6055777     0.48406375
## Class: neutral  0.5242718 0.4778761  0.2051793     0.10756972
## Class: positive 0.5052632 0.5363128  0.1892430     0.09561753
##                  Detection Prevalence Balanced Accuracy
## Class: negative             0.5876494         0.7683579
## Class: neutral              0.2450199         0.6756698
## Class: positive             0.1673307         0.7084055
##
## $mode
## [1] "sens_spec"
##
## $dots
## list()
```

Our next function, predict_input can be used to convert a column of text into a format that the model can predict off of. For the following two examples we'll use new data the model has never seen. Now we just clean the data and then put it through the predict function.

```
text_clean(Airlines_New_Data$text, Airlines_New_Data)
predict_input(tidytext_df$tidy_text)
```

Now if we want to just make a prediction with base R we can run the following code.

```
prediction <- predict(naive_bayes_classifier, predict.input)
head(prediction)
```

```
## [1] negative neutral  negative negative neutral  neutral
## Levels: negative neutral positive
```

However, the much better way to predict using the model is by using the confidence_interval function.

```
confidence_interval(naive_bayes_classifier, tidytext_df$tidy_text)
head(confidence_interval.df)
```

```
##   confidence Prediction second_most_probable
## 1  0.9891621   negative              neutral
## 2  0.5939431    neutral             negative
## 3  0.5089072    neutral             negative
## 4  0.9237834   negative              neutral
## 5  0.5084372    neutral             negative
## 6  0.5244099    neutral             negative
```

Now that we have this prediction dataframe, we can do stuff like add the actual label, text, get the accuracy, etc.

```
confidence_interval.df$Actual_Label <- tidytext_df$airline_sentiment
confidence_interval.df$is.match <- ifelse(confidence_interval.df$Prediction == confidence_interval.df$A
head(confidence_interval.df)
```

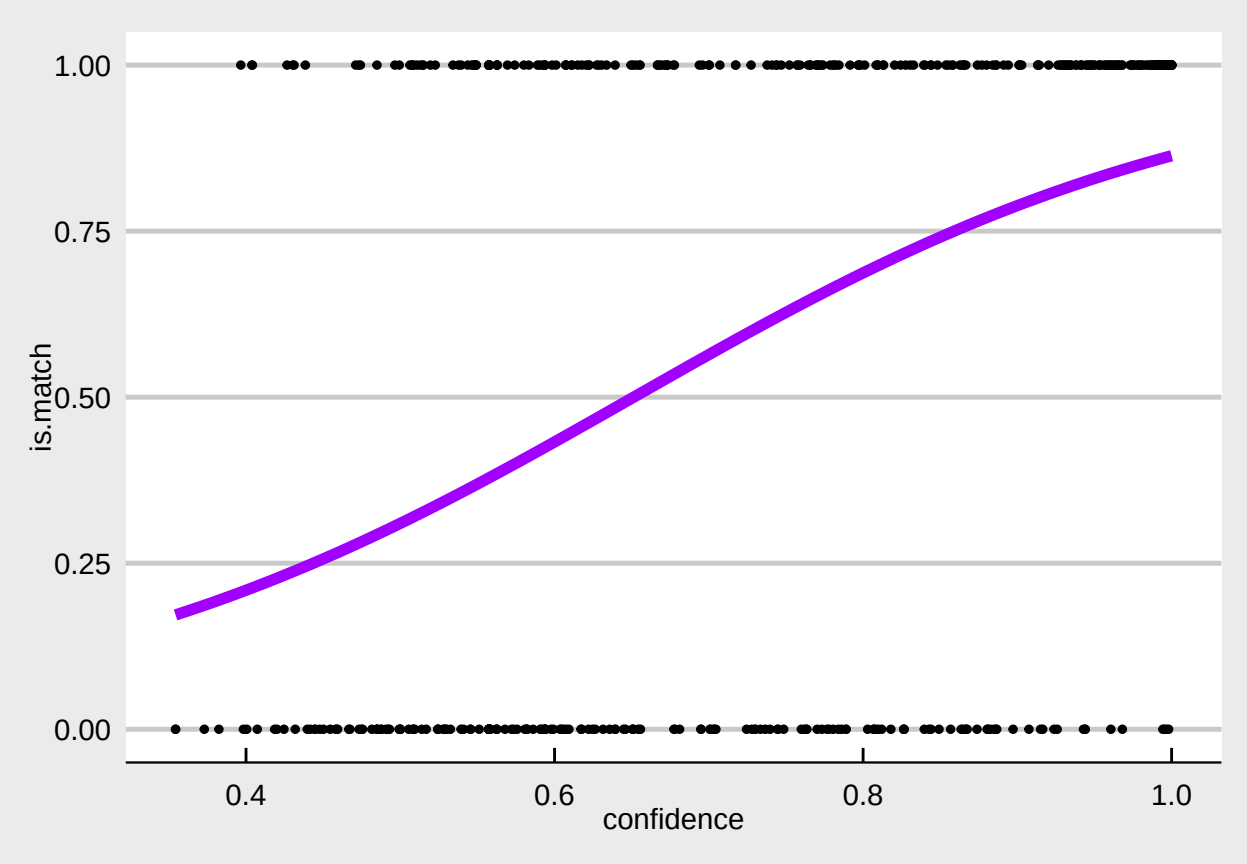
```
##   confidence Prediction second_most_probable Actual_Label is.match
## 1  0.9891621   negative              neutral     negative        1
## 2  0.5939431    neutral             negative     negative        0
## 3  0.5089072    neutral             negative      neutral        1
## 4  0.9237834   negative              neutral      neutral        0
## 5  0.5084372    neutral             negative     negative        0
## 6  0.5244099    neutral             negative     negative        0
```

Finally, this enables us to do some visualizations of the data.

```
library(ggthemes)

g <- ggplot(confidence_interval.df, aes(confidence, is.match)) +
  geom_point(size=1) +
  stat_smooth(method="glm", se=FALSE, method.args = list(family=binomial),
              color="#A100FF", size=2) + theme_economist_white()
plot(g)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



And that's all. To access the code please visit the github repository.