CPTS 575 : DATA SCIENCE, FALL 2018

Assignment 5: Text Classification with Naïve Bayes

Submitted By: Sukhjinder Singh

1. Data Collection

First you will need to gather your data from the web API for your newspaper of choice. Because most web APIs return JSON objects, you may want to use the fromJSON function in the RJSONIO package to make your API requests. I suggest using the Guardian, which has fairly lenient query limits, and makes getting an API key very simple (https://openplatform.theguardian.com/access/), but you may use any paper you like, so long as it includes section headings with its articles and has at least six sections. You may also want to choose a paper that lets you download full article text, as some papers like the New York Times only give you the lead paragraph. To get full credit for this portion you must include your full code and
demonstrate (by showing summaries of your data) that you have collected approximately 6,000 articles from a newspaper in six sections, and that approximately 1,000 articles are in each section

Solution

```
library(httr)
library(rjson)
library(lubridate)
sections <-
c("Books","Football","lifeandstyle","Us-news","Sport","World")

apiKey <- "06e9d99e-81ff-411f-b68c-f4aac4720d1f"
# Create url for each desk
urls <- lapply(sections, function(x)
{
  paste0("http://content.guardianapis.com/search?section=", x,
paste0("&order-by=newest&show-fields=body&page-size=50&api-key=",
apiKey))
})
```

```
# Write a function to get results

getRes <- function(url, pageNum)
  {

     # If pagenum is > 1, then add
  if(pageNum > 1)
     {
     url <- paste0(url, "&page=", pageNum)
     }

  # Query
  res <- content(GET(url), "parsed")

  # Control
  if(length(res) == 0)
     {
     warning("Empty result. Returning  NULL.")
     return(NULL)
     }

     # Simmer down
     res <- res[["response"]]$results


     # Return
     return(res)
}

# Main function
mainCall <- function(url, pages = 50)
{
  # master
  master <- vector("list", pages)
  # Loop
  for(page in 1:pages)
  {
```

```r
      temp <- getRes(url, page)
      # add
      master[[page]] <- temp
      #Sys.sleep(0.5)
  }
  # Return
  return(master)
}

# For each url, call Main
data <- lapply(urls, function(x)
  {
  print(paste0("Section ", x))

  # Set limit (6000 articles)
  upperLim <- 6000
  limitPP <- 120

  # Number of calls needed
  calls <- upperLim / (limitPP)
  getwd()
  # Run main
  main <- mainCall(x, calls)
  # Get name
  nam <- gsub("http://content.guardianapis.com/search?section=",
           "",
           unlist(strsplit(urls[[1]], "&order", fixed=T))[1],
           fixed=T)

  # Write to disk
  save(main, file = paste0(nam,"_guardian.Rdata"))
})

# Load all
folder <- paste0(getwd(),"/")
files <- paste0(folder, list.files(folder))

# Vector
```

```r
master <- vector("list", length(files))

# Loop
for(x in 1:length(files))
  {
  load(file=files[x])
  master[[x]] <- main
}

# To data frame
library(plyr)
library(data.table)

# Function to turn json into df
toDF <- function(json)
  {
  # return in df format
  res <- lapply(json, function(x)
  {
     # lapply
     res <- lapply(x, function(b)
     {
     # lapply
     tmp <- lapply(b, function(y)
     {
     striphtml <- function(htmlString)
           {
           return(gsub("<.*?>", "", htmlString))
           }
     # Return fields
     y$fields <- striphtml(y$fields$body)
     temp <- lapply(y, Filter, f = Negate(is.null))
     temp <- list("url" = ifelse(length(temp$webUrl) > 0,
temp$webUrl, NA),
                      "headline" = ifelse(length(temp$webTitle) > 0,
temp$webTitle, NA),
                      "body" = ifelse(length(temp$fields) > 0,
temp$fields, NA),
                       "section" = ifelse(length(temp$sectionName) > 
0, temp$sectionName, NA))
     # To ASCII
     temp$body <- iconv(temp$body, "latin1", "ASCII", sub="")
     # Return
     return(temp)
     })
     # Bind
     return(rbindlist(tmp, fill=T))
     })
     return(rbindlist(res, fill=T))
  })
  return(rbindlist(res, fill=T))
}

# Convert variables
res$section <- as.factor(res$section)

# Object name
final.data <- res

# Save as Rdata
save(final.data, file = "guardian_final.Rdata")
```

Results:
Since, the data is in the Rdata format, I have six different section containing more than 6000 articles and more than 1000 articles in each sections. All the data is stored in guardian_final.rdata file. All the rdata files has size approximately (6-10 Megabytes).

2. Data cleaning (10%)

The data you have now collected is likely very dirty. Examples of issues you are likely to run across are invalid characters, HTML tags, links and other non-article text, varied cases and punctuation. All of these will cause problems for the tokenization step that comes next. For this portion, you will clean up the article bodies you collected by removing the above mentioned messiness, and any other messiness you come across in your data. When you are finished, each article body should contain only plain, lower case text and no punctuation.

Code

```
# Clean wd
rm(list=ls())
# Load final data
load(file="guardian_final.Rdata")
# To df
final.data <- as.data.frame(final.data)
final.data <- unique(final.data)
# Rearrange
final.data <- final.data[,c(1,4,2,3)]

# Function to fix whitespace
whiteSpaceFix <- function(string)
  {

  # Strip punctuation
  temp <- gsub("[[:punct:]]", "", string)
  # Take sentence apart
  temp <- unlist(strsplit(temp, " "))
  # Control statement. If the result of the above is an empty
character, then return NULL
  if(length(temp) == 0) {
      # Print message
      print("Empty character. Moving on . . . ")
      # Return empty character
      return("")
  } else{
      # Take out whitespaces
      temp <- temp[sapply(temp, function(b) b != "")]
      # Reconstruct and take out punctuation + newlines etc.
      checkF <- function(z) grepl("[[:punct:]]", z) |
grepl("[\r\n\t]", z)
      temp <- temp[!checkF(temp) == TRUE]
      # Paste & collapse
      paste0(temp, collapse = " ")
  }
}
```

```r
# Set character encoding
final.data$body <- iconv(final.data$body, "latin1", "ASCII", sub="")
final.data$headline <- iconv(final.data$headline, "latin1", "ASCII",
sub="")
# Take out NA's
final.data <- final.data[!is.na(final.data$body), ]
final.data <- final.data[!is.na(final.data$headline), ]
final.data <- final.data[!nchar(final.data$headline) < 2, ]
final.data <- final.data[!nchar(final.data$body) < 2, ]
# whitespaces
final.data$body <- sapply(final.data$body, whiteSpaceFix)
final.data$headline <- sapply(final.data$headline, whiteSpaceFix)
# Shuffle rows
set.seed(215)
final.data <- final.data[sample(nrow(final.data)),]
# Split data
library(caret)
set.seed(9)
index <- createDataPartition(final.data$section, p=0.9, list=F)
# Training data
samp <- final.data[index,]
summary(samp$section)
# Write to file
write.table(samp, file = "train.txt", sep="\t", row.names=F,
col.names = F, fileEncoding = 'UTF-8')

# Test data
sampT <- final.data[-index,]
write.table(sampT, file = "test.txt", sep="\t", row.names=F,
col.names = F)
```

Results:

The data is split into a training set (90%), and a test set (10%) to evaluate the performance of the classifier.

SECTION	NUMBER.POS	PROPORTION
Books	1893	0.157
Football	1904	0.171
Lifeandstyle	1872	0.161
Us-news	1897	0.163
Sport	1905	0.173
World	1911	0.175

3. Tokenization (20%)

Solution:

```
import sys
from collections import Counter, defaultdict
import re
import math
from stemming.porter2 import stem
import nltk
```

```
from itertools import islice, izip, tee
from sklearn.metrics import confusion_matrix
from nltk.metrics import BigramAssocMeasures

""" TOKENIZATION """

def tokenize_porter(title, body):
    """ Break text into words and stem user porter stemmer """
    # break up words & remove stopwords
    title_break = stopWords(nltk.word_tokenize(title), lower_case =
True)
    body_break = stopWords(nltk.word_tokenize(body), lower_case =
True)
    #print title_break
    return(["title:" + stem(title) for title in title_break] +
["body:" + stem(body) for body in body_break])

""" TEXT PRE-PROCESSING """

def stopWords(text, lower_case = False):
    """ Remove stop words """
    # If lower_case == true
    if(lower_case == True):
      text = [word.lower() for word in text]
    # Remove stopwords
    return([word for word in text if word not in
nltk.corpus.stopwords.words('english') + [unicode("youre"),
unicode("may")]])

def numbers(text):
    """ remove numbers and percentages """
    return(re.sub(r'[0-9%]','', text))

def clean(line):
    """ Remove whitespace and numbers """
    # Strip whitespace
    text = re.sub(r'[^\w\s]','', line)
    # Strip numbers
    text = numbers(text)
    # strip and join
    return(" ".join(text.split()))

""" FEATURE SELECTION METHODS """

def chiSQ(priors, likelihood, keep):
    """ Extract the 10000 most informative features using chi-square
"""
    words = {}
    # Total word count
    twc = sum(priors.values())
    # All words in the counter
    words_unique = [likelihood[section].keys() for section in
likelihood.keys()]
    words_unique = set(sum(words_unique, []))
    for word in words_unique:
      # Go past each class
      scores = []
      for c in priors.keys():
            # Class word count
            cwc = priors[c]
            # Get word occurrence over all classes
            totalFreq = sum([likelihood[section][word] for section in
priors.keys()])
            # Word count within class
            wc = likelihood[c][word]
            # Get chi-sq
            score = BigramAssocMeasures.chi_sq(wc, (totalFreq, cwc),
twc)
            # Append
            scores.append(score)
      # Add to dict
      words[word] = sum(scores)
    # Select best words
    bestWords = sorted(words.iteritems(), key=lambda (w,s): s,
reverse=True)[:keep]
    # Save
    with open("chiSQ.txt", 'w') as f:
      print >> f, bestWords
    # Get names
    bestWords = [b[0] for b in bestWords]
    # Filter likelihood
    for c in priors.keys():
      for key in list(likelihood[c]):
            if key not in bestWords:
                 del likelihood[c][key]
    # Return
    return(likelihood)

""" NAIVE BAYES ALGORITHM """

def priors_and_likelihood(lines, keep, word_filter = "none"):
    """ Calculate priors (== group sizes) and likelihood of each word
(== occurrences) """
    # Open counters
    priors = Counter()
    likelihood = defaultdict(Counter)
    # For each headline & snippet, tokenize and add word to
likelihood counter
    for line in lines:
      priors[line[1]] += 1
      for word in tokenize_nltk(clean(line[2]), clean(line[3])):
            likelihood[line[1]][word] += 1
    # Filter
    if word_filter == "none":
      return(priors, likelihood)
    if word_filter == "chi-sq":
      likelihood = chiSQ(priors, likelihood, keep)
    # Return
    return(priors, likelihood)

def naive_bayes(tokens, priors, likelihood):
    """ Return the class that maximizes the posterior """
    # Get prediction
    max_class = (-1E6, '')
    for c in priors.keys():
      p = math.log(priors[c])
      n = float(sum(likelihood[c].values()))
      for word in tokens:
            p = p + math.log(max(1E-6, likelihood[c][word]) / n)

      if p > max_class[0]:
            max_class = (p,c)

    return max_class[1]

""" HELPERS """

def read_training_file(filename):
    """ Open tab-delimited file, read each line, split and return """
    with open(filename) as f:
      return([line.split("\t") for line in f])

def tokenize_testset(line):
    # Tokenize
    tokenized_text = tokenize_nltk(clean(line[2]), clean(line[3]))
    # Return
    return(tokenized_text)

def read_testing_file(filename):
    """ Open tab-delimited file, read each line, split and return """
    with open(filename) as f:
      return([line.split("\t") for line in f])

def savePredictions(filename, labels, predictions):
    # Open filename and write
    with open(filename, 'w') as f:
      for f1,f2 in zip(labels, predictions):
            print >> f,f1 + "\t" + f2

""" CROSS-VALIDATION """

def KfoldCV(data, folds):
    """ Split data in training / test for number of folds and
estimate test error """
    # Open list for stats
    stats = list()
    # Subset
    subsetL = len(data) / folds
    for i in range(folds):
      # subset
      testing_cv = data[i*subsetL:][:subsetL]
      training_cv = data[:i*subsetL] + data[(i+1)*subsetL:]
      # train
      (priors, likelihood) = priors_and_likelihood(training_cv,
5000, word_filter = "chi-sq")
      # test set labels
      labels = [line[1] for line in testing_cv]
      # Tokenize
      tokens_test = [tokenize_testset(line) for line in testing_cv]
      # predictions
      pred = [naive_bayes(token, priors, likelihood) for token in
tokens_test]
      # Calculate confusion matrix
      #stats.append(confusion_matrix(labels, pred))
      accuracy = 0
      for i in range(1,len(labels)):
            if(labels[i] == pred[i]):
                 accuracy += 1
            # Add to stats
      stats.append(float(accuracy) / len(labels))
    # Return
    return(stats)

""" TOP-LEVEL """

def main():
    # Arguments
    training_file = sys.argv[1]
    testing_file = sys.argv[2]
    CV = sys.argv[3]
    folds = int(sys.argv[4])
    save_model = sys.argv[5]
    # Read training
    lines_train = read_training_file(training_file)
    # Cross validation
    if(CV == "k-fold") :
      # K-fold CV
      CVres = KfoldCV(lines_train, folds)
      est2 = [str(e) for e in CVres]
      print "Results for K-fold cross-validation with k = 5 folds " 
+ ", ".join(est2) + " with an overall accuracy of " + str(sum(CVres) 
/ len(CVres))
    # Priors and likelihood
    (priors, likelihood) = priors_and_likelihood(lines_train, 15000, 
word_filter = "chi-sq")
    # Read lines
    lines = read_testing_file(testing_file)
    # Tokenize test
    tokens_test = [tokenize_testset(line) for line in lines]
    # Counter
    # test set labels
    labels = [line[1] for line in lines]
    # predictions
    pred = [naive_bayes(token, priors, likelihood) for token in 
tokens_test]
    # Save model if specified
    if(save_model != "FALSE" and save_model.endswith(".txt")):
      # Save labels and pred to txt
      savePredictions(save_model, labels, pred)
    # Calculate confusion matrix
    #stats.append(confusion_matrix(labels, pred))
    accuracy = 0
    for i in range(1,len(labels)):
      if(labels[i] == pred[i]):
           accuracy += 1

    print "Calibration/test: Classified %d correctly out of %d for an 
accuracy of %f"%(accuracy, len(labels), float(accuracy) /
len(labels))

if __name__ == "__main__":
main()
```

Results:

Tokenization Result

```
The president said voters should keep pressure on members of Congress
to reach across party lines and ensure initiatives in his budget are
enacted
```

#Remove Stop Words, punctuations and clean sentence

president said voters keep pressure members congress reach across party lines ensure initiatives budget enacted

# **Stemming**

presid said voter keep pressur member congress reach across parti line ensur initi budget enact

Important If we use POS tags, we don't need to remove stopwords and we need to convert capital letters. Doing so would remove a lot of valuable information . We can filter a sentences of POS tags that are not nouns or verbs and apply stopwords removal after lemmatizing the text to filter out common words like 'be':

```
The-DT president-NN said-VBD voters-NNS should-MD keep-VB pressure-NN on-IN
members-NNS of-IN Congress-NNP to-TO reach-VB across-RP party-NN lines-NNS
and-CC ensure-NN initiatives-NNS in-IN his-PRP$ budget-NN are-VBP enacted-VBN
```

If we then lemmatize this sentence

```
The president say voter should keep pressure on member of Congress to reach
across party line and ensure initiative in his budget be enact
```

After choose to keep only nouns, verbs and remove stopwords.

```
president say voter keep pressure member Congress reach party line ensure
initiative budget enact
```

Classification result

The model for model will be preprocessed by removing the numbers, punctuation and stopwords from the text, after which the porter stemmer is applied to stem the words. We evaluate the performance using k-fold cross validation with and testing the model on the test data.

The result from the k-fold cross validation results in an average accuracy of 95.5%. Likewise, the model result in an overall accuracy of <95.3% on the test set. The high concordance between these two figures indicate that, despite the simplicity of the model, it correctly classifies roughly 95% of the unseen articles in the test set. As such, we are vastly outperforming the base model of 25% and 33%.

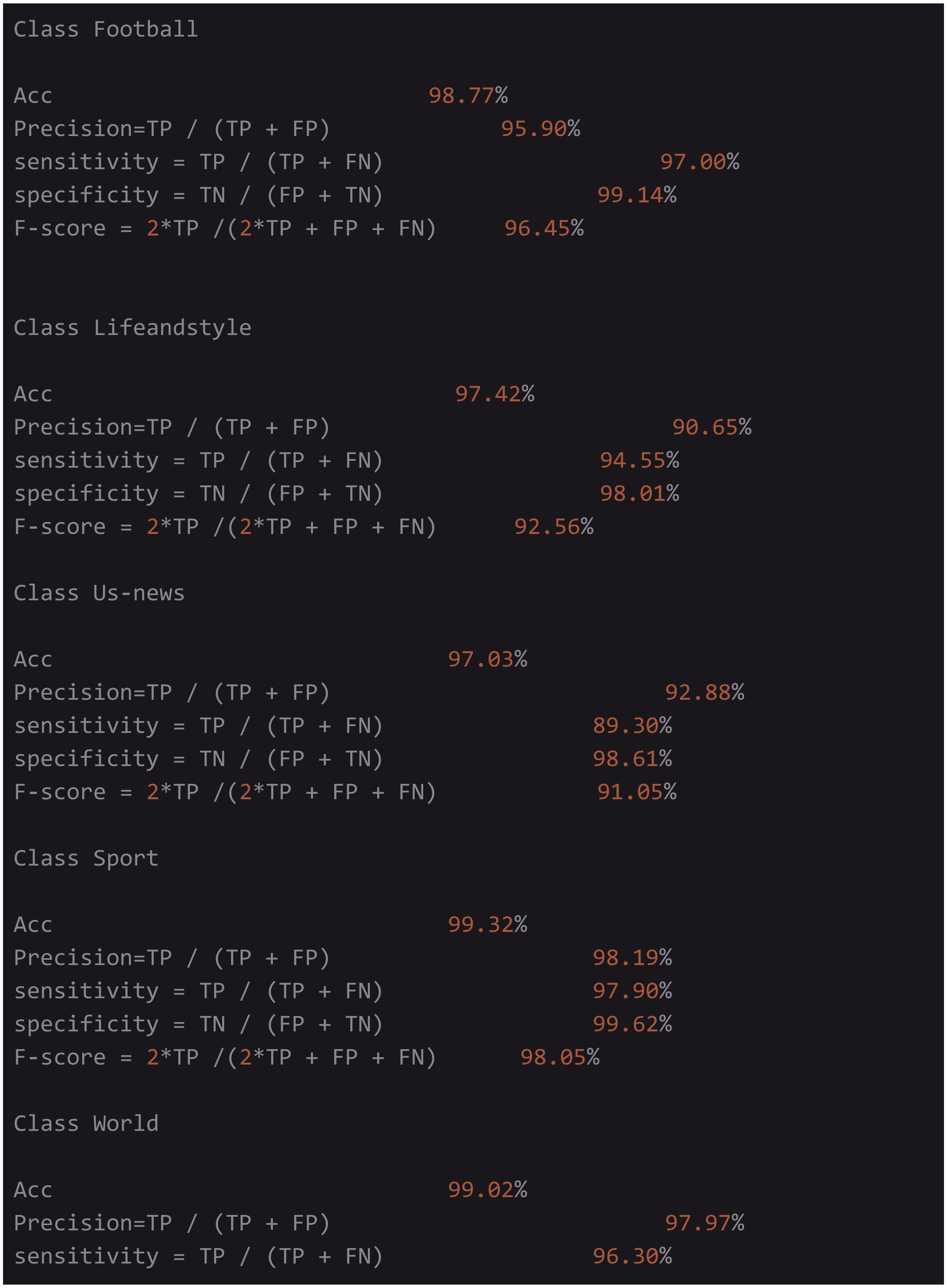
Confusion Matrix

```
Prediction Books   Football Lifeandstyle  Us-news   Sport      World
Books      971     19         1             8         0          1
Football   16      940        2             23        9         10
Lifeandstyle 8     3          891           87        0         11
Us-news      2     25         159           786       16        12
Sport        0     24         4              8       958         6
World       11     12         29            11       11         926
```

For Each Class

Class Books

```
Acc                                 99.43%
precision=TP / (TP + FP)                  98.16%
sensitivity = TP / (TP + FN)                    98.55%
specificity = TN / (FP + TN)                    99.61%
F-score = 2*TP /(2*TP + FP + FN)    98.35%
```



```
Class Football

Acc                            98.77%
Precision=TP / (TP + FP)             95.90%
sensitivity = TP / (TP + FN)                     97.00%
specificity = TN / (FP + TN)                99.14%
F-score = 2*TP /(2*TP + FP + FN)     96.45%


Class Lifeandstyle

Acc                              97.42%
Precision=TP / (TP + FP)                          90.65%
sensitivity = TP / (TP + FN)                98.55%
specificity = TN / (FP + TN)                98.01%
F-score = 2*TP /(2*TP + FP + FN)      92.56%

Class Us-news

Acc                              97.03%
Precision=TP / (TP + FP)                         92.88%
sensitivity = TP / (TP + FN)                89.30%
specificity = TN / (FP + TN)                98.61%
F-score = 2*TP /(2*TP + FP + FN)            91.05%

Class Sport

Acc                              99.32%
Precision=TP / (TP + FP)                    98.19%
sensitivity = TP / (TP + FN)                97.90%
specificity = TN / (FP + TN)                99.62%
F-score = 2*TP /(2*TP + FP + FN)      98.05%

Class World

Acc                              99.02%
Precision=TP / (TP + FP)                         97.97%
sensitivity = TP / (TP + FN)                96.30%
```

```
specificity = TN / (FP + TN)                99.59%
F-score = 2*TP /(2*TP + FP + FN)      97.13%

Average accuracy for all the classes     98.50%
```

Here, its becomes clear that the Football,Books and LifeandStyle classes contains higher false positive than the other classes. Overall, the result make sense, While section like Sport is well separated from other sections, the world section is clearly less bounded, often contains news on international politics.