Untitled

TA: Leslie Huang

Course: Text as Data

Date: 3/22/2018

Recitation 8: Supervised Learning IV

Setup

```
rm(list = ls())
set.seed(100)
setwd("/Users/Lingyi/TAD/lab/Text-as-Data-Lab-Spr2018/W8_03_22_18/")
```

Questions from last time ...

1 Tuning hyperparameters for SVM? ** Beyond the scope of this class **

```
# install.packages("e1071")
library(e1071)

# Simple example here: https://rischanlab.github.io/SVM.html

# Many other machine learning tools are available in that library as well.

# NB: e1071 uses libsvm(), a Python library. And RTextTools, the package from last time, uses e1071.
# However, if you're using Python, sklearn has a great SVM implementation!


# 2 More SVM

library(RTextTools)
```

```
## Loading required package: SparseM
```

```
##
## Attaching package: 'SparseM'
```

```
## The following object is masked from 'package:base':
##
##     backsolve
```

```
library(tm)
```

```
## Loading required package: NLP
```

```
library(quanteda)
```

```
## quanteda version 1.0.0
```

```
## Using 3 of 4 threads for parallel computing
```

```
##
## Attaching package: 'quanteda'
```

```
## The following objects are masked from 'package:tm':
##
##     as.DocumentTermMatrix, stopwords
```

```
## The following object is masked from 'package:utils':
##
##     View
```

```
library(quanteda.corpora)
```

2.1

```
data("data_corpus_ukmanifestos")
manifesto_corpus <- corpus(data_corpus_ukmanifestos)

# SVM is very difficult with more than 2 classes, so let's subset to Labour and Conservative only
manifesto_corpus <- corpus_subset(manifesto_corpus, Party == "Lab" | Party == "Con")

# Class labels are the party labels
manifesto_labels <- docvars(manifesto_corpus, "Party")

manifesto_dfm <- dfm(manifesto_corpus)

manifesto_mat <- quanteda::convert(manifesto_dfm, "matrix") # convert to matrix format

# Now we're going to do 1-fold cross-validation... "by hand" (without using the cross_validate function

# Create containers
training_break <- floor( 0.9 * length(manifesto_labels) )

training_manifesto_dtm <- manifesto_mat[1:training_break, ]
test_manifesto_dtm <- manifesto_mat[(training_break + 1 ): length(manifesto_labels), ]

train_manifesto_container <- create_container(training_manifesto_dtm,
                                              manifesto_labels[1:training_break],
                                              trainSize = 1:nrow(training_manifesto_dtm),
                                              virgin = FALSE
                                              )

test_manifesto_container <- create_container(test_manifesto_dtm,
                                             manifesto_labels[training_break + 1 : length(manifesto_labe
```

```
                                            trainSize = 1:nrow(test_manifesto_dtm),
                                            virgin = FALSE
                                            )

# Train a model on the training data

manifesto_train_svm <- train_model(train_manifesto_container, "SVM", kernel = "linear")

# Predict the test data
classify_model(test_manifesto_container, manifesto_train_svm)
```

```
##   SVM_LABEL  SVM_PROB
## 1       Con 0.6875774
## 2       Con 0.5690704
## 3       Con 0.6470639
## 4       Con 0.7688652
```

```
# Accuracy?

manifesto_labels[(training_break+1):length(manifesto_labels)]
```

```
## [1] "Con" "Lab" "Con" "Lab"
```

2/4 correct... Not great!

3 SVM with tf-idf or raw term frequencies: which has better performance?

We might expect that tf-idf is better because it upweights terms that discriminate more between documents

3.1 Compare radial/linear SVM over tf/tf-idf inputs

Modified example from https://rpubs.com/bmcole/reuters-text-categorizatio

Another great library!

install.packages("caret")

```
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:NLP':
##
##     annotate
```

3.2 Wrangle in the data

```
r8train <- read.table("r8-train-all-terms.txt", header=FALSE, sep='\t')
r8test <- read.table("r8-test-all-terms.txt", header=FALSE, sep='\t')
# Data credit: https://www.cs.umb.edu/~smimarog/textmining/datasets/


# rename variables
names(r8train) <- c("Class", "docText")
names(r8test) <- c("Class", "docText")

# convert the document text variable to character type
r8train$docText <- as.character(r8train$docText)
r8test$docText <- as.character(r8test$docText)

# create varible to denote if observation is train or test
r8train$train_test <- c("train")
r8test$train_test <- c("test")

# merge the train/test data
merged <- rbind(r8train, r8test)

# remove objects that are no longer needed
remove(r8train, r8test)

# subset to 3 document classes only for sake of computational expense/memory
merged <- merged[which(merged$Class %in% c("grain", "ship", "trade")),]

# drop unused levels in the response variable
merged$Class <- droplevels(merged$Class)

# counts of each class in the train/test sets
table(merged$Class, merged$train_test)
```

```
##
##         test train
##   grain   10    41
##   ship    36   108
##   trade   75   251
```

```
rownames(merged) <- NULL # reset rownames to numbers
```

```
# 3.3 Create corpus, preprocess, DTM
sourceData <- VectorSource(merged$docText)

# create the corpus
corpus <- Corpus(sourceData)

# preprocess/clean the training corpus
```

```r
corpus <- tm_map(corpus, content_transformer(tolower)) # convert to lowercase
corpus <- tm_map(corpus, removeNumbers) # remove digits
corpus <- tm_map(corpus, removePunctuation) # remove punctuation
corpus <- tm_map(corpus, stripWhitespace) # strip extra whitespace
corpus <- tm_map(corpus, removeWords, stopwords('english')) # remove stopwords

# create term document matrix (tdm)
tdm <- DocumentTermMatrix(corpus)

# create tf-idf weighted version of term document matrix
weightedtdm <- weightTfIdf(tdm)
```

3.4 TDM –> DF for test/training

```r
# convert tdm's into data frames
tdm_df <- as.data.frame(as.matrix(tdm))
weightedtdm_df <- as.data.frame(as.matrix(weightedtdm))

# split back into train and test sets
tdmTrain <- tdm_df[which(merged$train_test == "train"), ]
weightedTDMtrain <- weightedtdm_df[which(merged$train_test == "train"), ]

tdmTest <-  tdm_df[which(merged$train_test == "test"), ]
weightedTDMtest <- weightedtdm_df[which(merged$train_test == "test"), ]

# append document labels as last column
tdmTrain$doc.class <- merged$Class[which(merged$train_test == "train")]
tdmTest$doc.class <- merged$Class[which(merged$train_test == "test")]
weightedTDMtrain$doc.class <- merged$Class[which(merged$train_test == "train")]
weightedTDMtest$doc.class  <- merged$Class[which(merged$train_test == "test")]
```

3.5 Linear SVM + tf-idf

```r
# set resampling scheme: 10-fold cross-validation, 3 times
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)

# SVM using the weighted (td-idf) term document matrix
# kernel: linear
# tuning parameters: C

load("svm_knn_workspace.RData") # this will take too long to run here!

#svm.tfidf.linear  <- train(doc.class ~ . , data = weightedTDMtrain, trControl = ctrl, method = "svmLin

# 3.6 Radial SVM + tf-idf

# tuning parameters: sigma, C
#svm.tfidf.radial  <- train(doc.class ~ . , data=weightedTDMtrain, trControl = ctrl, method = "svmRadia
```

```
# predict on test data
#svm.tfidf.linear.predict <- predict(svm.tfidf.linear,newdata = weightedTDMtest)
#svm.tfidf.radial.predict <- predict(svm.tfidf.radial,newdata = weightedTDMtest)
```

3.7 Linear SVM + unweighted dfm

```
# tuning parameters: C
svm.linear  <- train(doc.class ~ . , data=tdmTrain, trControl = ctrl, method = "svmLinear")
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```r
# 3.8 Radial SVM + unweighted

# tuning parameters: sigma, C
set.seed(100)
svm.radial  <- train(doc.class ~ . , data=tdmTrain, trControl = ctrl, method = "svmRadial")
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```r
# predict on test data
svm.linear.predict <- predict(svm.linear,newdata = tdmTest)
svm.radial.predict <- predict(svm.radial,newdata = tdmTest)
```

3.9 Performance

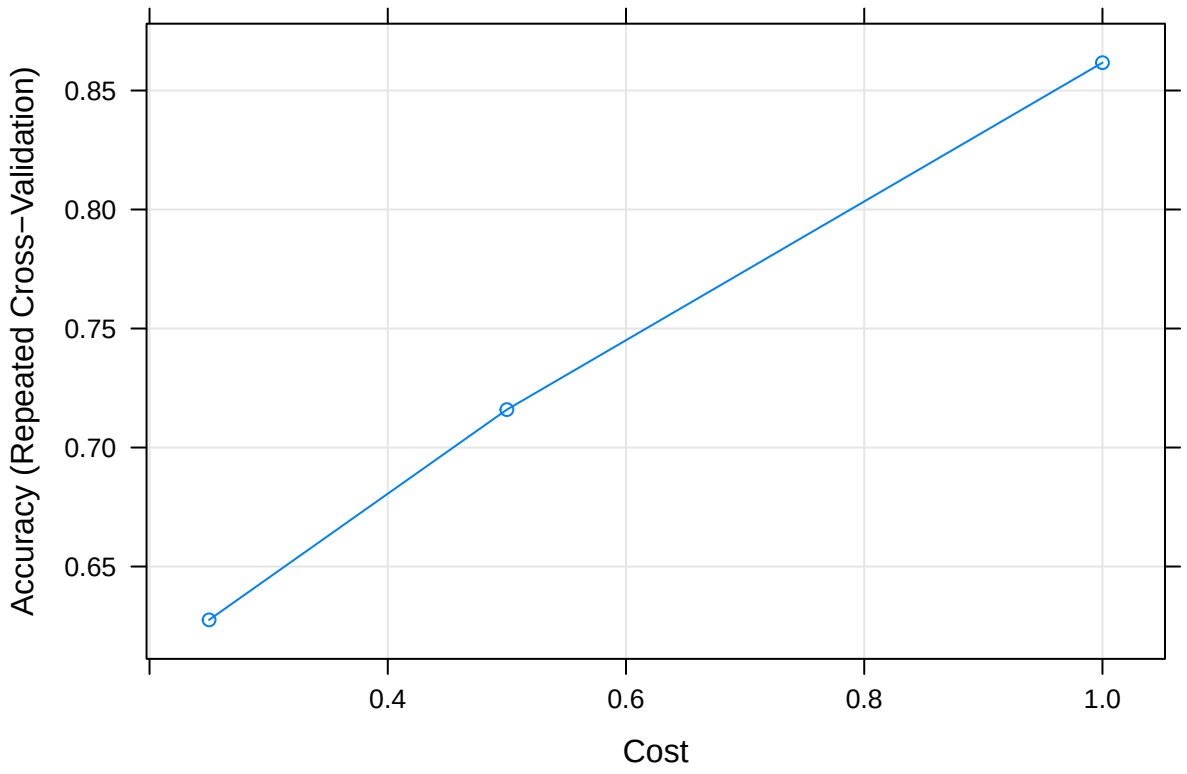
```
# Weighted:

svm.tfidf.linear # linear kernel
```

```
## Support Vector Machines with Linear Kernel
##
##  400 samples
## 7205 predictors
##    3 classes: 'grain', 'ship', 'trade'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 360, 360, 359, 360, 360, 360, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8891661  0.7621611
##
## Tuning parameter 'C' was held constant at a value of 1
```

```
svm.tfidf.radial # radial basis function kernel
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
##  400 samples
## 7205 predictors
##    3 classes: 'grain', 'ship', 'trade'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 360, 360, 360, 360, 361, 361, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.6275954  0.0000000
##   0.50  0.7159303  0.2942553
##   1.00  0.8616901  0.7030959
##
## Tuning parameter 'sigma' was held constant at a value of 1.298288
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 1.298288 and C = 1.
```

```
plot(svm.tfidf.radial)
```




```
# Unweighted:

svm.linear
```

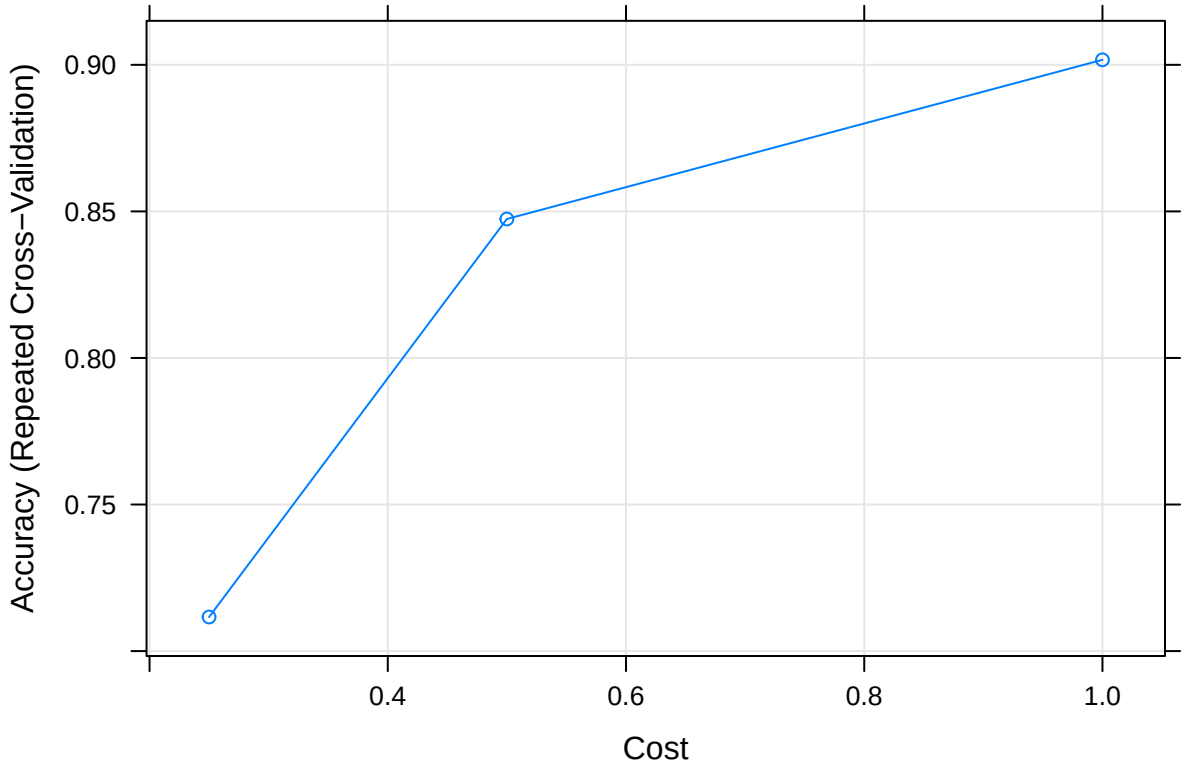
```
## Support Vector Machines with Linear Kernel
##
##  400 samples
## 7205 predictors
##    3 classes: 'grain', 'ship', 'trade'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 361, 360, 359, 360, 359, 361, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.95989   0.9231546
##
## Tuning parameter 'C' was held constant at a value of 1
```

```
svm.radial
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
##  400 samples
## 7205 predictors
##    3 classes: 'grain', 'ship', 'trade'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 360, 361, 360, 360, 360, 360, ...
```

```
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.7116276  0.2818070
##   0.50  0.8474025  0.6660024
##   1.00  0.9017058  0.7936781
##
## Tuning parameter 'sigma' was held constant at a value of 0.003960912
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.003960912 and C = 1.
```

```
plot(svm.radial)
```




```
# confusion matrices allow you to evaluate accuracy and other metrics
confusionMatrix(svm.linear.predict, tdmTest$doc.class)  # linear kernel, unweighted TDM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction grain ship trade
##      grain     9    0     0
##      ship      0   31     2
##      trade     1    5    73
##
## Overall Statistics
##
##                Accuracy : 0.9339
##                  95% CI : (0.8739, 0.971)
##     No Information Rate : 0.6198
##     P-Value [Acc > NIR] : 1.489e-15
```

```
##
##                   Kappa : 0.8699
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: grain Class: ship Class: trade
## Sensitivity               0.90000      0.8611       0.9733
## Specificity               1.00000      0.9765       0.8696
## Pos Pred Value            1.00000      0.9394       0.9241
## Neg Pred Value            0.99107      0.9432       0.9524
## Prevalence                0.08264      0.2975       0.6198
## Detection Rate            0.07438      0.2562       0.6033
## Detection Prevalence      0.07438      0.2727       0.6529
## Balanced Accuracy         0.95000      0.9188       0.9214
```

```
confusionMatrix(svm.radial.predict, tdmTest$doc.class)  # radial kernel, unweighted TDM ** Best perform
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction grain ship trade
##      grain     3    0     0
##      ship      0   31     1
##      trade     7    5    74
##
## Overall Statistics
##
##                Accuracy : 0.8926
##                  95% CI : (0.8233, 0.9415)
##     No Information Rate : 0.6198
##     P-Value [Acc > NIR] : 1.546e-11
##
##                   Kappa : 0.7756
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: grain Class: ship Class: trade
## Sensitivity               0.30000      0.8611       0.9867
## Specificity               1.00000      0.9882       0.7391
## Pos Pred Value            1.00000      0.9688       0.8605
## Neg Pred Value            0.94068      0.9438       0.9714
## Prevalence                0.08264      0.2975       0.6198
## Detection Rate            0.02479      0.2562       0.6116
## Detection Prevalence      0.02479      0.2645       0.7107
## Balanced Accuracy         0.65000      0.9247       0.8629
```

```
confusionMatrix(svm.tfidf.linear.predict, weightedTDMtest$doc.class) # linear kernel, weighted TDM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction grain ship trade
##      grain     7    0     0
```

```
##      ship      0   20     0
##      trade     3   16    75
##
## Overall Statistics
##
##                Accuracy : 0.843
##                  95% CI : (0.7657, 0.9027)
##     No Information Rate : 0.6198
##     P-Value [Acc > NIR] : 6.646e-08
##
##                   Kappa : 0.662
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: grain Class: ship Class: trade
## Sensitivity               0.70000      0.5556       1.0000
## Specificity               1.00000      1.0000       0.5870
## Pos Pred Value            1.00000      1.0000       0.7979
## Neg Pred Value            0.97368      0.8416       1.0000
## Prevalence                0.08264      0.2975       0.6198
## Detection Rate            0.05785      0.1653       0.6198
## Detection Prevalence      0.05785      0.1653       0.7769
## Balanced Accuracy         0.85000      0.7778       0.7935
```

```
confusionMatrix(svm.tfidf.radial.predict, weightedTDMtest$doc.class) # radial kernel, weighted TDM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction grain ship trade
##      grain     3    0     0
##      ship      2   20     0
##      trade     5   16    75
##
## Overall Statistics
##
##                Accuracy : 0.8099
##                  95% CI : (0.7286, 0.8755)
##     No Information Rate : 0.6198
##     P-Value [Acc > NIR] : 5.044e-06
##
##                   Kappa : 0.5795
##  Mcnemar's Test P-Value : 4.038e-05
##
## Statistics by Class:
##
##                      Class: grain Class: ship Class: trade
## Sensitivity               0.30000      0.5556       1.0000
## Specificity               1.00000      0.9765       0.5435
## Pos Pred Value            1.00000      0.9091       0.7812
## Neg Pred Value            0.94068      0.8384       1.0000
## Prevalence                0.08264      0.2975       0.6198
## Detection Rate            0.02479      0.1653       0.6198
## Detection Prevalence      0.02479      0.1818       0.7934
```

```
## Balanced Accuracy         0.65000      0.7660       0.7717
```

```
# print various info about parameters, etc. used in the model with highest accuracy
svm.radial$results # error rate and values of tuning parameter
```

```
##         sigma    C  Accuracy     Kappa AccuracySD    KappaSD
## 1 0.003960912 0.25 0.7116276 0.2818070 0.02522633 0.07640852
## 2 0.003960912 0.50 0.8474025 0.6660024 0.03148545 0.07493860
## 3 0.003960912 1.00 0.9017058 0.7936781 0.02823358 0.06447412
```

```
svm.radial$bestTune # final tuning parameter
```

```
##         sigma C
## 3 0.003960912 1
```

```
svm.radial$metric # metric used to select optimal model
```

```
## [1] "Accuracy"
```

4 KNN – also from https://rpubs.com/bmcole/reuters-text-categorization

```
# set resampling scheme
ctrl_knn <- trainControl(method = "repeatedcv", number = 10, repeats = 3)

# 4.1 fit a kNN model using the weighted (td-idf) term document matrix

# tuning parameter: K
knn.tfidf <- train(doc.class ~ ., data = weightedTDMtrain, method = "knn", trControl = ctrl_knn)

# predict on test data
knn.tfidf.predict <- predict(knn.tfidf, newdata = weightedTDMtest)

# 4.2 fit a kNN model using the unweighted TDM
# tuning parameter: K

knn <- train(doc.class ~ ., data = tdmTrain, method = "knn", trControl = ctrl_knn)

# predict on test data
knn.predict <- predict(knn, newdata = tdmTest)

# 4.3 Performance

knn.tfidf
```

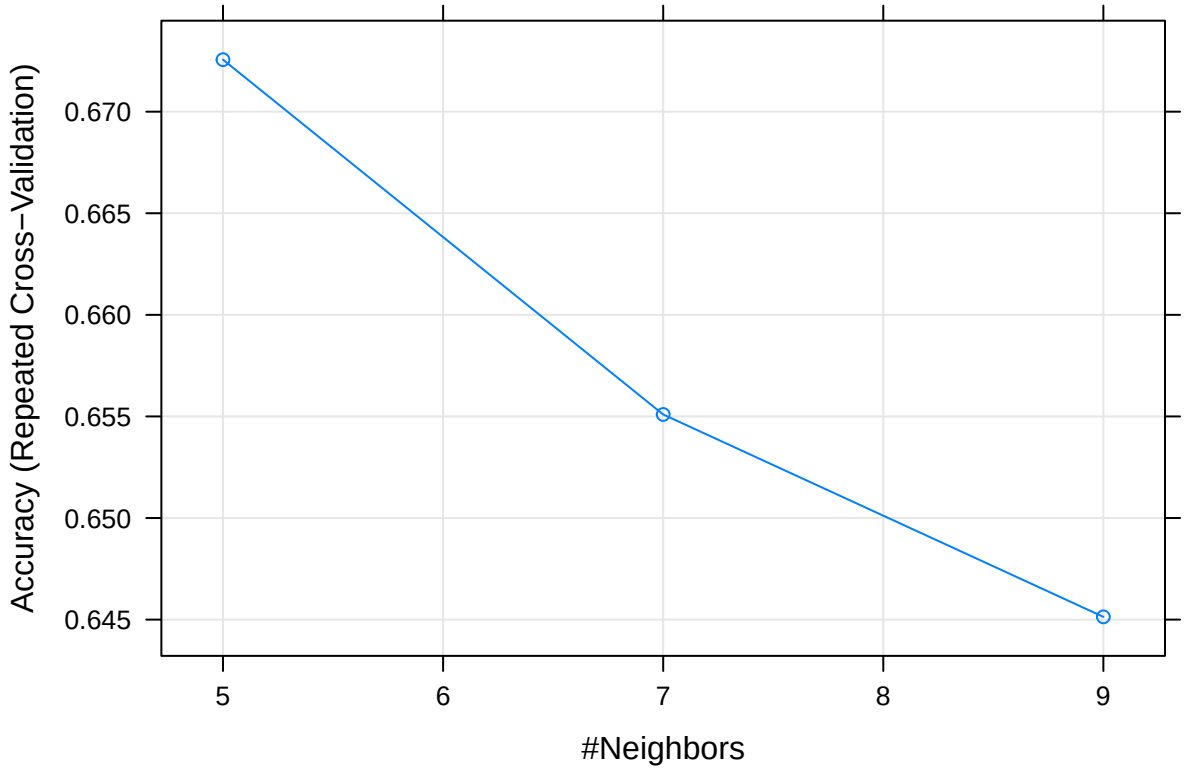
```
## k-Nearest Neighbors
##
##  400 samples
## 7205 predictors
##    3 classes: 'grain', 'ship', 'trade'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 360, 360, 360, 361, 360, 360, ...
## Resampling results across tuning parameters:
##
```

```
##   k  Accuracy   Kappa
##   5  0.6725583  0.15702685
##   7  0.6550959  0.09740623
##   9  0.6451356  0.06214621
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

```
knn
```

```
## k-Nearest Neighbors
##
##  400 samples
## 7205 predictors
##    3 classes: 'grain', 'ship', 'trade'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 360, 360, 360, 360, 361, 360, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.7309688  0.3616490
##   7  0.6978163  0.2426985
##   9  0.6778132  0.1759111
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

```
# plot accuracy vs. number of neighbors

plot(knn.tfidf)
```




```
# confusion matrices allow you to evaluate accuracy and other metrics
confusionMatrix(knn.predict, tdmTest$doc.class)  # unweighted TDM
```

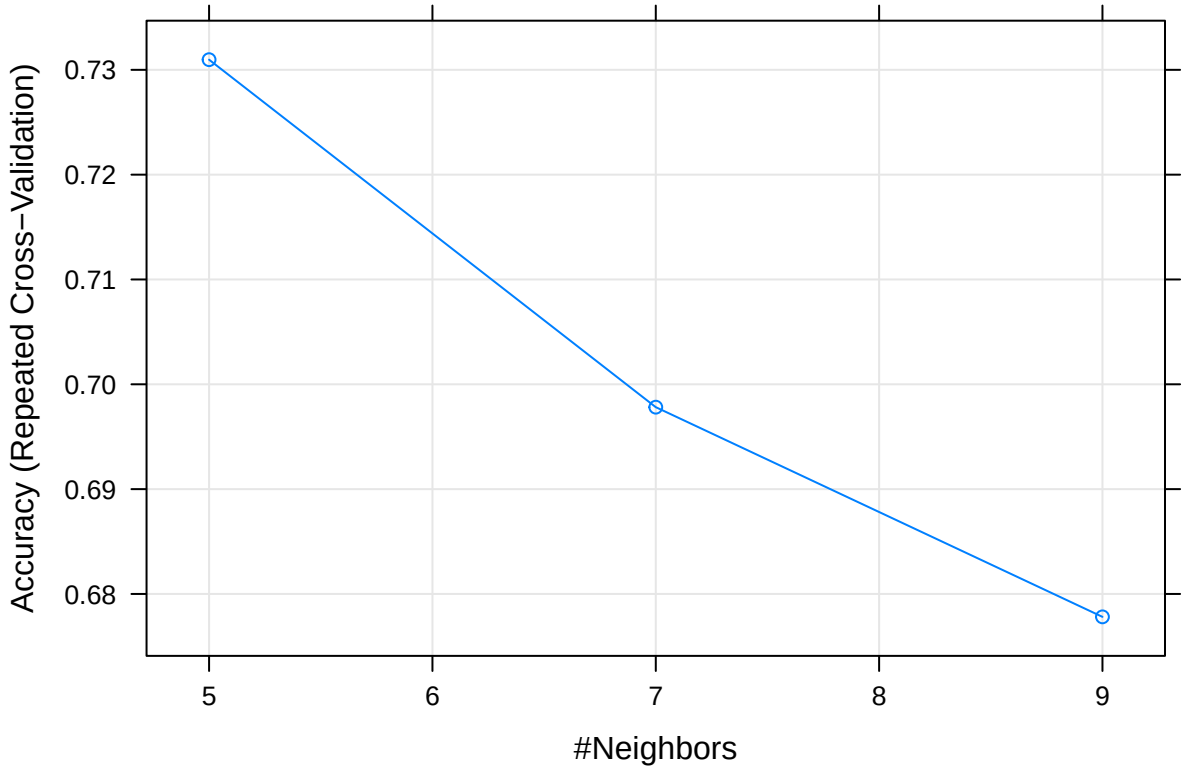
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction grain ship trade
##      grain     8    1     0
##      ship      0    4     0
##      trade     2   31    75
##
## Overall Statistics
##
##                Accuracy : 0.719
##                  95% CI : (0.6301, 0.7969)
##     No Information Rate : 0.6198
##     P-Value [Acc > NIR] : 0.01432
##
##                   Kappa : 0.3477
##  Mcnemar's Test P-Value : 1.981e-07
##
## Statistics by Class:
##
##                      Class: grain Class: ship Class: trade
## Sensitivity               0.80000     0.11111       1.0000
## Specificity               0.99099     1.00000       0.2826
## Pos Pred Value            0.88889     1.00000       0.6944
## Neg Pred Value            0.98214     0.72650       1.0000
## Prevalence                0.08264     0.29752       0.6198
```

```
## Detection Rate            0.06612     0.03306       0.6198
## Detection Prevalence      0.07438     0.03306       0.8926
## Balanced Accuracy         0.89550     0.55556       0.6413
```

```
confusionMatrix(knn.tfidf.predict, weightedTDMtest$doc.class)  # weighted TDM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction grain ship trade
##      grain     3    0     0
##      ship      0    0     0
##      trade     7   36    75
##
## Overall Statistics
##
##                Accuracy : 0.6446
##                  95% CI : (0.5525, 0.7295)
##     No Information Rate : 0.6198
##     P-Value [Acc > NIR] : 0.322
##
##                   Kappa : 0.0969
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: grain Class: ship Class: trade
## Sensitivity               0.30000      0.0000      1.00000
## Specificity               1.00000      1.0000      0.06522
## Pos Pred Value            1.00000         NaN      0.63559
## Neg Pred Value            0.94068      0.7025      1.00000
## Prevalence                0.08264      0.2975      0.61983
## Detection Rate            0.02479      0.0000      0.61983
## Detection Prevalence      0.02479      0.0000      0.97521
## Balanced Accuracy         0.65000      0.5000      0.53261
```

```
plot(knn)
```




```
# print info about parameters, etc. used in the model with highest accuracy
knn$results # error rate and values of tuning parameter
```

```
##   k  Accuracy     Kappa AccuracySD   KappaSD
## 1 5 0.7309688 0.3616490 0.04293426 0.1288363
## 2 7 0.6978163 0.2426985 0.04420650 0.1367342
## 3 9 0.6778132 0.1759111 0.04405583 0.1397122
```

```
knn$bestTune # final tuning parameter
```

```
##   k
## 1 5
```

```
save.image("svm_knn_workspace.RData")
```