STOR 565 Spring 2019 Homework 5

Due on 03/21/2018 in Class

Coleman Breen

Remark. Credits for **Theoretical Part** and **Computational Part** are in total *100 pt* (40 pt for theoretical and 60pt for computational) please complete your computational report below in the **RMarkdown** file and submit your printed PDF homework created by it.

Computational Part

You are supposed to finish the Computational Part in a readable report manner for each dataset-based analysis problem. Other than what you are asked to do, you can decide any details on your own discretion. Also goto R demonstrations in the Lecture 6 folder on Sakai. There you will find a working example for LDA, QDA and k-nn for the titanic data as well as a much more extensive demonstartion on k-nn in the folder under classification-knn in the same folder in Sakai.

You may need some of these packages:

```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 3.5.2
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(class)
```

In particular, the MASS package for doing LDA, QDA and the class package for doing K-nn.

1. The following data set is coming from a Kaggle competition that came out on November 12, 2015. Here is the description from the competition:

Time magazine noted Mashable as one of the 25 best blogs in 2009, and described it as a "one stop shop" for social media. As of November 2015, [Mashable] had over 6,000,000 Twitter followers and over 3,200,000 fans on Facebook. In this problem, you'll use data from thousands of articles published by Mashable in a period of two years to see which variables predict the popularity of an article.

Load and read more about the data

- Load the data *OnlineNewsPopularityTraining.csv*, which contains a large portion of the data set from the above competition.
- Read the variable descriptions for the variables at this website: UCI website
- A binary label has been added to the data set `popular`, which specifies whether or not each website is considered a popular website (0 for popular and 1 for not popular).
- `popular` was created by assigning 1 to rows with `shares` values greater than 3300, and zero otherwise.

Prepare the data

- Remove the variables *shares*, *url* and *timedelta* from the dataset.

```
#--> Load
library(tidyverse)
train.raw <- read.csv("OnlineNewsPopularityTraining.csv")
```

```r
train <- select(train.raw, -c(shares, url, timedelta))

test.raw <- read.csv("OnlineNewsPopularityTest.csv")
test <- select(test.raw, -c(shares, url, timedelta))

#--> Display
head(train)
```

```
##   n_tokens_title n_tokens_content n_unique_tokens n_non_stop_words
## 1             12              219       0.6635945                1
## 2              9              255       0.6047431                1
## 3              9              211       0.5751295                1
## 4              9              531       0.5037879                1
## 5             13             1072       0.4156456                1
## 6             10              370       0.5598886                1
##   n_non_stop_unique_tokens num_hrefs num_self_hrefs num_imgs num_videos
## 1                0.8153846         4              2        1          0
## 2                0.7919463         3              1        1          0
## 3                0.6638655         3              1        1          0
## 4                0.6656347         9              0        1          0
## 5                0.5408895        19             19       20          0
## 6                0.6981982         2              2        0          0
##   average_token_length num_keywords data_channel_is_lifestyle
## 1             4.680365            5                         0
## 2             4.913725            4                         0
## 3             4.393365            6                         0
## 4             4.404896            7                         0
## 5             4.682836            7                         0
## 6             4.359459            9                         0
##   data_channel_is_entertainment data_channel_is_bus data_channel_is_socmed
## 1                             1                  0                     0
## 2                             0                  1                     0
## 3                             0                  1                     0
## 4                             1                  0                     0
## 5                             0                  0                     0
## 6                             0                  0                     0
##   data_channel_is_tech data_channel_is_world kw_min_min kw_max_min
## 1                    0                     0         0         0
## 2                    0                     0         0         0
## 3                    0                     0         0         0
## 4                    0                     0         0         0
## 5                    1                     0         0         0
## 6                    1                     0         0         0
##   kw_avg_min kw_min_max kw_max_max kw_avg_max kw_min_avg kw_max_avg
## 1          0          0         0          0          0          0
## 2          0          0         0          0          0          0
## 3          0          0         0          0          0          0
## 4          0          0         0          0          0          0
## 5          0          0         0          0          0          0
## 6          0          0         0          0          0          0
##   kw_avg_avg self_reference_min_shares self_reference_max_shares
## 1          0                      496                      496
## 2          0                        0                        0
## 3          0                      918                      918
```

```
## 4          0                        0                       0
## 5          0                      545                   16000
## 6          0                     8500                    8500
##   self_reference_avg_sharess weekday_is_monday weekday_is_tuesday
## 1                    496.000                 1                  0
## 2                      0.000                 1                  0
## 3                    918.000                 1                  0
## 4                      0.000                 1                  0
## 5                   3151.158                 1                  0
## 6                   8500.000                 1                  0
##   weekday_is_wednesday weekday_is_thursday weekday_is_friday
## 1                    0                   0                 0
## 2                    0                   0                 0
## 3                    0                   0                 0
## 4                    0                   0                 0
## 5                    0                   0                 0
## 6                    0                   0                 0
##   weekday_is_saturday weekday_is_sunday is_weekend     LDA_00     LDA_01
## 1                   0                 0          0 0.50033120 0.37827893
## 2                   0                 0          0 0.79975569 0.05004668
## 3                   0                 0          0 0.21779229 0.03333446
## 4                   0                 0          0 0.02857322 0.41929964
## 5                   0                 0          0 0.02863281 0.02879355
## 6                   0                 0          0 0.02224528 0.30671758
##       LDA_02     LDA_03     LDA_04 global_subjectivity
## 1 0.04000468 0.04126265 0.04012254           0.5216171
## 2 0.05009625 0.05010067 0.05000071           0.3412458
## 3 0.03335142 0.03333354 0.68218829           0.7022222
## 4 0.49465083 0.02890472 0.02857160           0.4298497
## 5 0.02857518 0.02857168 0.88542678           0.5135021
## 6 0.02223128 0.02222429 0.62658158           0.4374086
##   global_sentiment_polarity global_rate_positive_words
## 1                0.09256198                 0.04566210
## 2                0.14894781                 0.04313725
## 3                0.32333333                 0.05687204
## 4                0.10070467                 0.04143126
## 5                0.28100348                 0.07462687
## 6                0.07118419                 0.02972973
##   global_rate_negative_words rate_positive_words rate_negative_words
## 1                0.013698630           0.7692308           0.2307692
## 2                0.015686275           0.7333333           0.2666667
## 3                0.009478673           0.8571429           0.1428571
## 4                0.020715631           0.6666667           0.3333333
## 5                0.012126866           0.8602151           0.1397849
## 6                0.027027027           0.5238095           0.4761905
##   avg_positive_polarity min_positive_polarity max_positive_polarity
## 1             0.3786364            0.10000000                   0.7
## 2             0.2869146            0.03333333                   0.7
## 3             0.4958333            0.10000000                   1.0
## 4             0.3859652            0.13636364                   0.8
## 5             0.4111274            0.03333333                   1.0
## 6             0.3506100            0.13636364                   0.6
##   avg_negative_polarity min_negative_polarity max_negative_polarity
## 1            -0.3500000                -0.600            -0.2000000
```

```
## 2           -0.1187500               -0.125           -0.1000000
## 3           -0.4666667               -0.800           -0.1333333
## 4           -0.3696970               -0.600           -0.1666667
## 5           -0.2201923               -0.500           -0.0500000
## 6           -0.1950000               -0.400           -0.1000000
##   title_subjectivity title_sentiment_polarity abs_title_subjectivity
## 1          0.5000000               -0.1875000              0.00000000
## 2          0.0000000                0.0000000              0.50000000
## 3          0.0000000                0.0000000              0.50000000
## 4          0.0000000                0.0000000              0.50000000
## 5          0.4545455                0.1363636              0.04545455
## 6          0.6428571                0.2142857              0.14285714
##   abs_title_sentiment_polarity popular
## 1                    0.1875000       0
## 2                    0.0000000       0
## 3                    0.0000000       0
## 4                    0.0000000       0
## 5                    0.1363636       0
## 6                    0.2142857       0
```

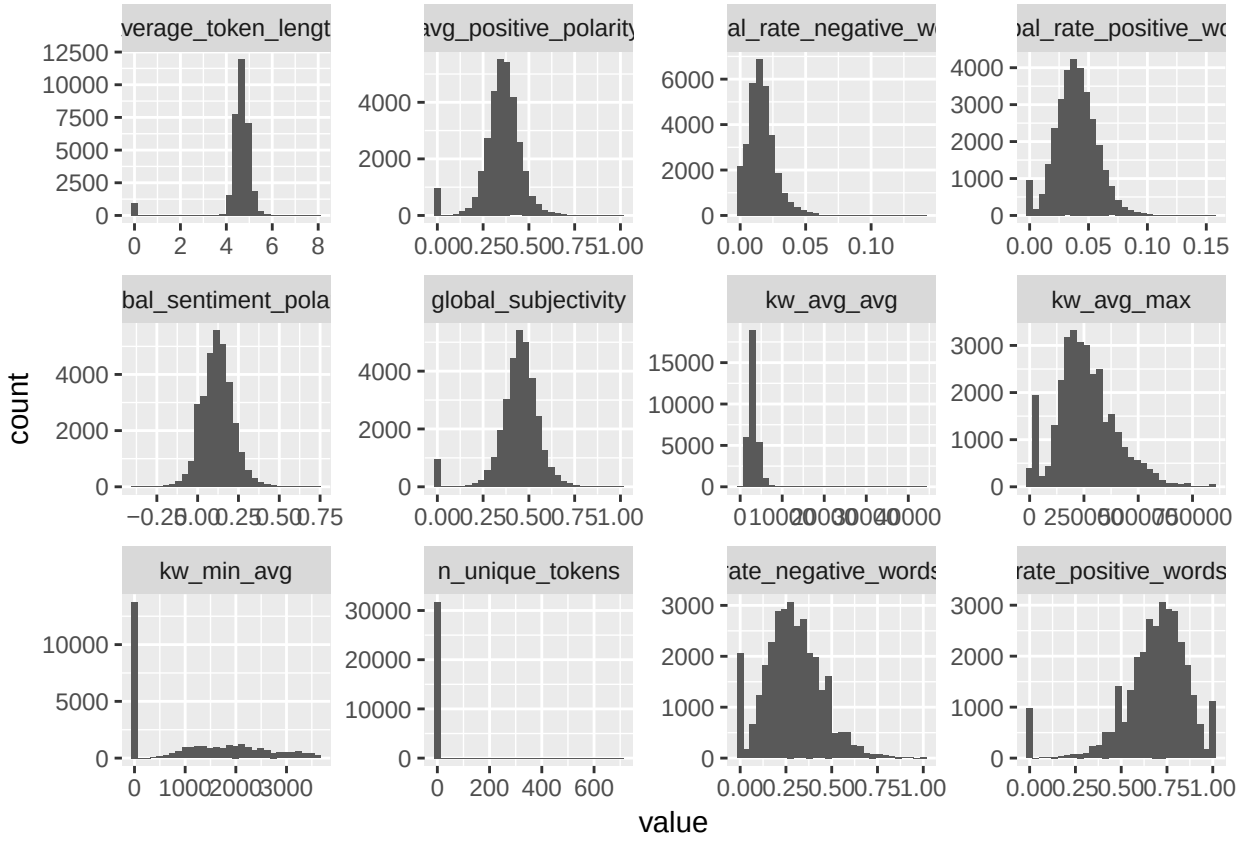
We'll start by removing predictors that don't make sense because they violate the continuous and normality assumptions made by LDA. I am making these decisions based on reading the descriptions of the data on UCI's website. The first group of variables to remove is the `weekday_is_x` because they are binary. The second group to remove is `data_channel_is_x`, again because they are binary.

```
removeDiscrete <- function(input){
  output <- input %>%
    # Days of the week (binary)
    select(-c(weekday_is_monday, weekday_is_tuesday, weekday_is_wednesday,
              weekday_is_thursday, weekday_is_friday, weekday_is_saturday,
              weekday_is_sunday, is_weekend)) %>%
    # Type of article (binary)
    select(-c(data_channel_is_entertainment, data_channel_is_bus,
            data_channel_is_socmed, data_channel_is_tech,
            data_channel_is_world, data_channel_is_lifestyle)) %>%
    # Integer values (not continuous)
    select(-c(n_tokens_title, n_tokens_content,
              n_non_stop_words, n_non_stop_unique_tokens,
              num_hrefs, num_self_hrefs, num_imgs, num_videos,
              num_keywords, kw_min_min, kw_max_min, kw_min_max,
              kw_max_max, kw_avg_min,
              LDA_00, LDA_01, LDA_02, LDA_03, LDA_04,
              self_reference_min_shares, self_reference_max_shares,
              self_reference_avg_sharess)) %>%
    # Don't look normal enough
    select(-c(kw_max_avg, title_sentiment_polarity,
              abs_title_subjectivity, title_subjectivity,
              min_negative_polarity, max_negative_polarity,
              min_positive_polarity, max_positive_polarity,
              abs_title_sentiment_polarity))
  return(output)
}
```

We'll perform the first round of culling on the train and test sets and then look at which other variables to remove and correct using plots.

```
#--> First round of cropping
train2 <- removeDiscrete(train)
test2 <- removeDiscrete(test)

#--> Plot
train2[1:12] %>%
  keep(is.numeric) %>%
  gather() %>%
  ggplot(aes(value)) +
    facet_wrap(~ key, scales = "free") +
    geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



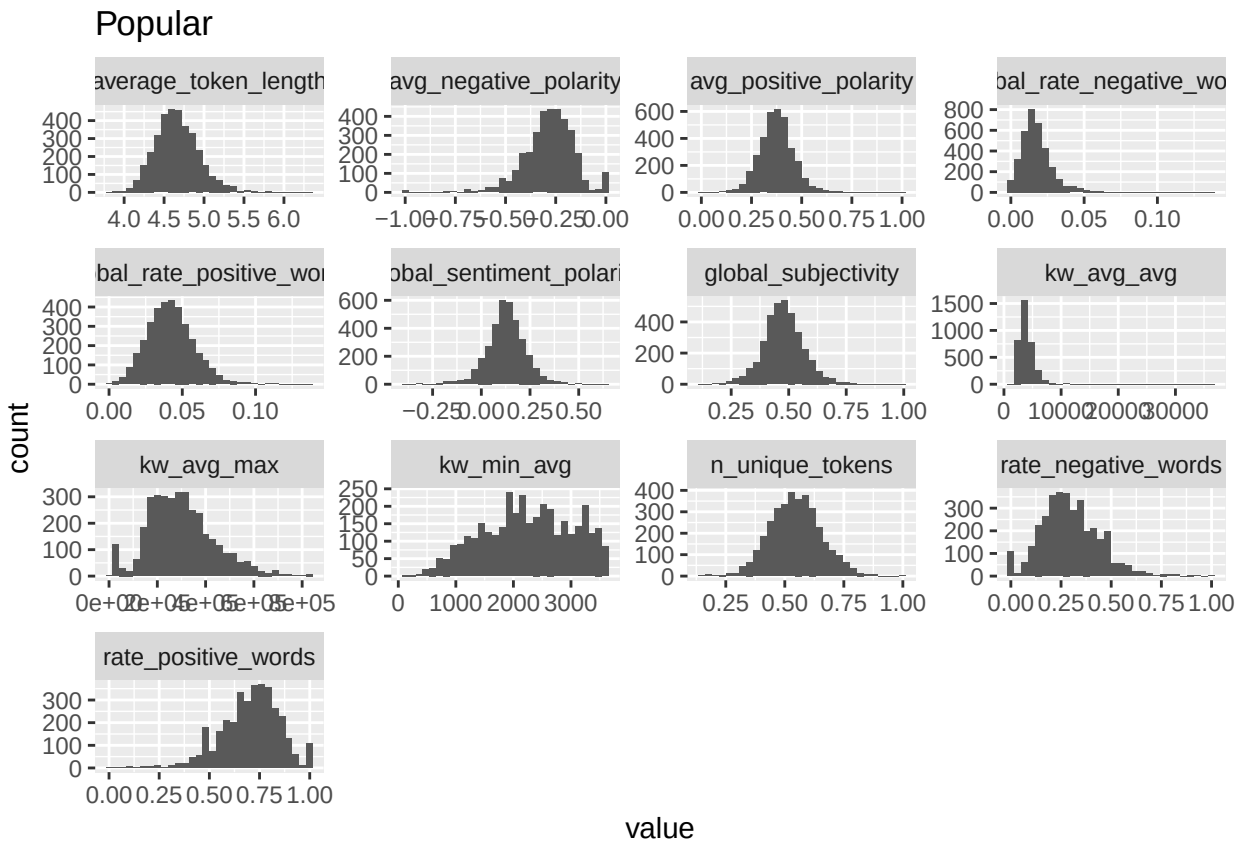
It appears that there are some outliers that are disrupting the normality of some of the metrics. We'll use a function to fix outliers below.

```
removeOutliers <- function(input){
  output <- input %>%
    filter(n_unique_tokens < max(input$n_unique_tokens)) %>%
    filter(n_unique_tokens != 0) %>%
    filter(kw_min_avg != 0)
  return(output)
}
```

```
#--> Second round of cleaning
train3 <- removeOutliers(train2)
```

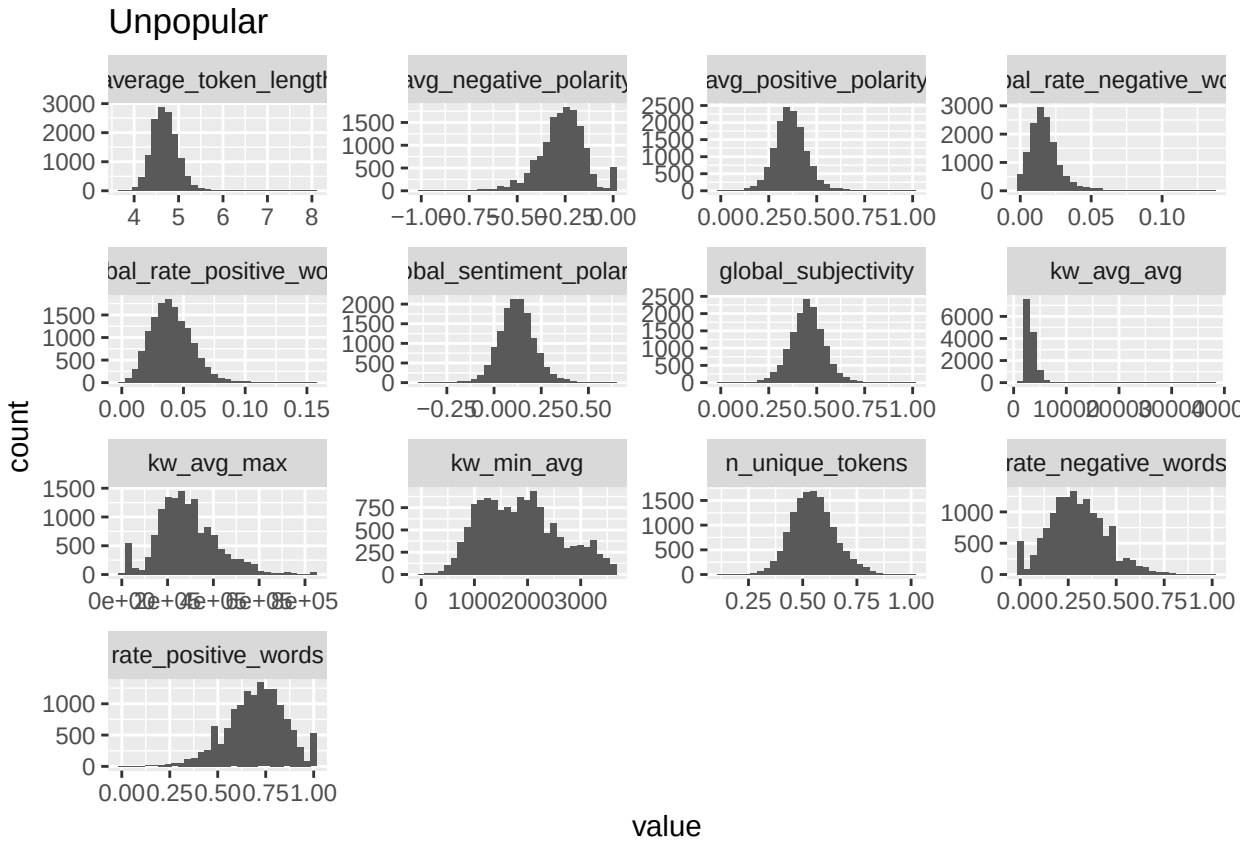
```
#--> Plot
train3 %>%
  filter(popular==1) %>%
  select(-popular) %>%
  keep(is.numeric) %>%
  gather() %>%
  ggplot(aes(value)) +
    facet_wrap(~ key, scales = "free") +
    geom_histogram() +
  ggtitle("Popular")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
train3 %>%
  filter(popular==0) %>%
  select(-popular) %>%
  keep(is.numeric) %>%
  gather() %>%
  ggplot(aes(value)) +
    facet_wrap(~ key, scales = "free") +
    geom_histogram() +
  ggtitle("Unpopular")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



It looks like (besides `popular`) all of our remaining variables are roughly normally distributed conditioned on class. We could perform a transform on `kw_avg_avg` but for the sake of this analysis, our predictors are normal enough.

```
train.clean <- train3
test.clean <- test2
```

```
#--> Confusion matrix function
confusion <- function(yhat, y, quietly = FALSE){

  if(!quietly)
    message("yhat is the vector of predicted outcomes, possibly a factor.\n
          Sensitivity = (first level predicted) / (first level actual) \n
          Specificity = (second level predicted) / (second level actual)")

  if(!is.factor(y) & is.factor(yhat))
    y <- as.factor(y)

  if(!all.equal(levels(yhat), levels(y)))
    stop("Factor levels of yhat and y do not match.")

  confusion_mat <- table(yhat, y, deparse.level = 2)
  stats <- data.frame(sensitivity = confusion_mat[1, 1]/sum(confusion_mat[, 1]),
                                  specificity = confusion_mat[2, 2]/sum(confusion_mat[, 2]))

  return(list(confusion_mat = confusion_mat, stats = stats))
}
```

Questions

(a) (15 points) The aim of this computational exercise is to prepare a classifier to predict whether or not a new website will be popular, i.e. classification by the `popular` variable in the dataset. You will do so using

- LDA
- QDA
- K-nearest neighbors

For each of the methods,

1) carefully describe how you choose any thresholds or tuning parameters.

2) list the predictors you would remove, if any, before fitting your models.

You must justify your answers by specifically naming concepts studied in this course. You also might want to justify your choices with summaries or plots of the data. Please do not print large amounts of data in the output.

I am being intentionally vague here because I want to see how you would handle such a data set in practice. All I ask is that you give proper justification for whatever you are doing. For example: the data contains indicator variables for different days of the week (weekday_is_monday etc). When doing LDA **I would remove these sorts of variables** as LDA inherently assumes that the features are continuous (and have a normal distribution).

(b) (30 points)For **each of the methods** listed in (a):

1) Fit a model to predict `popular` class labels, consistent with your answer in (a).

2) Briefly discuss your results.

You must show summary output of this model, along with plots and other documentation.

LDA

```
#--> Fit model
mlda <- lda(popular ~ ., data = train.clean)
mlda
```

```
## Call:
## lda(popular ~ ., data = train.clean)
##
## Prior probabilities of groups:
##         0         1
## 0.7963689 0.2036311
##
## Group means:
##   n_unique_tokens average_token_length kw_avg_max kw_min_avg kw_avg_avg
## 0       0.5481277             4.691806   287162.5   1868.460   3296.531
## 1       0.5450620             4.667442   299186.8   2253.428   3992.021
##   global_subjectivity global_sentiment_polarity global_rate_positive_words
## 0           0.4523468                 0.1206053                 0.04049151
## 1           0.4778632                 0.1266038                 0.04214764
##   global_rate_negative_words rate_positive_words rate_negative_words
## 0                 0.01719599           0.6992784           0.3005769
## 1                 0.01782619           0.7038998           0.2961002
##   avg_positive_polarity avg_negative_polarity
## 0             0.3642079            -0.2665799
## 1             0.3750295            -0.2836746
##
```

```
## Coefficients of linear discriminants:
##                                       LD1
## n_unique_tokens            -1.180732e+00
## average_token_length       -2.198815e-01
## kw_avg_max                 -1.866882e-06
## kw_min_avg                  7.626532e-04
## kw_avg_avg                  4.411440e-04
## global_subjectivity         2.515480e+00
## global_sentiment_polarity  -6.278107e-01
## global_rate_positive_words -3.580472e-01
## global_rate_negative_words  2.113819e+00
## rate_positive_words         1.542842e+00
## rate_negative_words         9.118538e-01
## avg_positive_polarity      -3.688651e-01
## avg_negative_polarity      -3.703542e-01
```

```
#--> Predict
lda.pred <- predict(mlda, newdata = test.clean)

#--> Confusion matrix
confusion(lda.pred$class, test.clean$popular)
```

```
## yhat is the vector of predicted outcomes, possibly a factor.
##
##           Sensitivity = (first level predicted) / (first level actual)
##
##           Specificity = (second level predicted) / (second level actual)
```

```
## $confusion_mat
##     y
## yhat    0    1
##    0 6238 1610
##    1   47   33
##
## $stats
##   sensitivity specificity
## 1   0.9925219  0.02008521
```

1) See above for model.
2) We used the following variables in our LDA model:

- n_unique_tokens
- average_token_length
- kw_avg_max
- kw_min_avg
- kw_avg_avg
- global_subjectivity
- global_sentiment_polarity
- global_rate_positive_words
- global_rate_negative_words
- rate_positive_words
- rate_negative_words
- avg_positive_polarity

- avg_negative_polarity

These variables are the one that are continuous and normal enough to work as features for LDA. I outlined

the pipeline I used to pick these variables above. Additionally, they are not collinear to the point of setting off any issues in the LDA function so they meet all of the necessary criteria.

The linear discriminants are printed above. Group means are not terribly different between `popular` and `unpopular` articles. The exceptions are `kw_min_avg` and `kw_avg_avg`. Since these predictors have to do with the number of shares based on certain keywords, it makes sense that they'd be good at distinguishing between an article that was shared many times and one that was not.

QDA

There is some collinearity that does not throw off LDA but is a problem for QDA. We'll correct that here.

```
x <- dplyr::select(train.clean, -popular)
abs(cov(as.matrix(x)))
```

```
##                            n_unique_tokens average_token_length
## n_unique_tokens               1.072974e-02         1.399914e-03
## average_token_length          1.399914e-03         8.183093e-02
## kw_avg_max                    1.727992e+03         4.041377e+02
## kw_min_avg                    4.675834e+00         1.043635e+01
## kw_avg_avg                    1.413426e+01         5.899650e+00
## global_subjectivity           6.175595e-04         3.458927e-03
## global_sentiment_polarity     4.155060e-04         2.105799e-03
## global_rate_positive_words    8.728668e-05         8.881118e-04
## global_rate_negative_words    6.428586e-06         3.099867e-04
## rate_positive_words           6.866675e-04         2.174579e-03
## rate_negative_words           7.082525e-04         2.198269e-03
## avg_positive_polarity         2.958260e-04         1.798421e-03
## avg_negative_polarity         4.505152e-04         1.339971e-03
##                              kw_avg_max   kw_min_avg   kw_avg_avg
## n_unique_tokens            1.727992e+03 4.675834e+00 1.413426e+01
## average_token_length       4.041377e+02 1.043635e+01 5.899650e+00
## kw_avg_max                 1.777588e+10 3.752803e+07 5.494584e+07
## kw_min_avg                 3.752803e+07 5.911446e+05 4.983008e+05
## kw_avg_avg                 5.494584e+07 4.983008e+05 1.826387e+06
## global_subjectivity        7.330760e+02 1.346664e+01 2.909137e+01
## global_sentiment_polarity  6.475494e+02 4.472218e+00 5.994439e+00
## global_rate_positive_words 7.752864e+01 9.297174e-01 1.208801e+00
## global_rate_negative_words 8.145405e+01 4.592732e-01 1.171037e+00
## rate_positive_words        1.031499e+03 1.647063e+00 3.581465e+00
## rate_negative_words        1.020585e+03 1.644812e+00 3.592799e+00
## avg_positive_polarity      4.233023e+02 8.163781e+00 2.022663e+01
## avg_negative_polarity      1.335980e+03 7.440661e+00 2.272771e+01
##                            global_subjectivity global_sentiment_polarity
## n_unique_tokens                   6.175595e-04              4.155060e-04
## average_token_length              3.458927e-03              2.105799e-03
## kw_avg_max                        7.330760e+02              6.475494e+02
## kw_min_avg                        1.346664e+01              4.472218e+00
## kw_avg_avg                        2.909137e+01              5.994439e+00
## global_subjectivity               8.023436e-03              2.317928e-03
## global_sentiment_polarity         2.317928e-03              9.630383e-03
## global_rate_positive_words        4.344564e-04              8.708926e-04
## global_rate_negative_words        1.015614e-04              6.003270e-04
## rate_positive_words               1.686724e-03              1.165395e-02
## rate_negative_words               1.669330e-03              1.163991e-02
## avg_positive_polarity             3.041414e-03              3.913007e-03
```

```
## avg_negative_polarity             3.340263e-03              4.181795e-03
##                            global_rate_positive_words
## n_unique_tokens                          8.728668e-05
## average_token_length                     8.881118e-04
## kw_avg_max                               7.752864e+01
## kw_min_avg                               9.297174e-01
## kw_avg_avg                               1.208801e+00
## global_subjectivity                      4.344564e-04
## global_sentiment_polarity                8.708926e-04
## global_rate_positive_words               2.754594e-04
## global_rate_negative_words               1.914861e-06
## rate_positive_words                      1.337418e-03
## rate_negative_words                      1.332711e-03
## avg_positive_polarity                    1.649645e-04
## avg_negative_polarity                    2.279081e-05
##                            global_rate_negative_words rate_positive_words
## n_unique_tokens                          6.428586e-06        6.866675e-04
## average_token_length                     3.099867e-04        2.174579e-03
## kw_avg_max                               8.145405e+01        1.031499e+03
## kw_min_avg                               4.592732e-01        1.647063e+00
## kw_avg_avg                               1.171037e+00        3.581465e+00
## global_subjectivity                      1.015614e-04        1.686724e-03
## global_sentiment_polarity                6.003270e-04        1.165395e-02
## global_rate_positive_words               1.914861e-06        1.337418e-03
## global_rate_negative_words               1.168758e-04        1.228572e-03
## rate_positive_words                      1.228572e-03        2.302602e-02
## rate_negative_words                      1.230570e-03        2.294530e-02
## avg_positive_polarity                    3.957857e-05        7.618655e-04
## avg_negative_polarity                    4.044706e-04        5.077015e-03
##                            rate_negative_words avg_positive_polarity
## n_unique_tokens                  7.082525e-04          2.958260e-04
## average_token_length             2.198269e-03          1.798421e-03
## kw_avg_max                       1.020585e+03          4.233023e+02
## kw_min_avg                       1.644812e+00          8.163781e+00
## kw_avg_avg                       3.592799e+00          2.022663e+01
## global_subjectivity              1.669330e-03          3.041414e-03
## global_sentiment_polarity        1.163991e-02          3.913007e-03
## global_rate_positive_words       1.332711e-03          1.649645e-04
## global_rate_negative_words       1.230570e-03          3.957857e-05
## rate_positive_words              2.294530e-02          7.618655e-04
## rate_negative_words              2.297984e-02          7.196254e-04
## avg_positive_polarity            7.196254e-04          7.530247e-03
## avg_negative_polarity            5.108148e-03          9.199520e-04
##                            avg_negative_polarity
## n_unique_tokens                     4.505152e-04
## average_token_length                1.339971e-03
## kw_avg_max                          1.335980e+03
## kw_min_avg                          7.440661e+00
## kw_avg_avg                          2.272771e+01
## global_subjectivity                 3.340263e-03
## global_sentiment_polarity           4.181795e-03
## global_rate_positive_words          2.279081e-05
## global_rate_negative_words          4.044706e-04
## rate_positive_words                 5.077015e-03
```

```
## rate_negative_words                 5.108148e-03
## avg_positive_polarity               9.199520e-04
## avg_negative_polarity               1.501191e-02
```

```
#--> Fit model
mqda <- qda(popular ~ n_unique_tokens+average_token_length+
              global_subjectivity+global_sentiment_polarity+
              global_rate_positive_words+global_rate_negative_words+
              avg_positive_polarity+avg_negative_polarity,
            data = train.clean)
mqda
```

```
## Call:
## qda(popular ~ n_unique_tokens + average_token_length + global_subjectivity +
##     global_sentiment_polarity + global_rate_positive_words +
##     global_rate_negative_words + avg_positive_polarity + avg_negative_polarity,
##     data = train.clean)
##
## Prior probabilities of groups:
##         0         1
## 0.7963689 0.2036311
##
## Group means:
##   n_unique_tokens average_token_length global_subjectivity
## 0       0.5481277             4.691806           0.4523468
## 1       0.5450620             4.667442           0.4778632
##   global_sentiment_polarity global_rate_positive_words
## 0                 0.1206053                 0.04049151
## 1                 0.1266038                 0.04214764
##   global_rate_negative_words avg_positive_polarity avg_negative_polarity
## 0                 0.01719599             0.3642079            -0.2665799
## 1                 0.01782619             0.3750295            -0.2836746
```

```
qda.pred <- predict(mqda, newdata = test.clean)

# Many actual survivors predicted to die
confusion(qda.pred$class, test.clean$popular, quietly = TRUE)
```

```
## $confusion_mat
##      y
## yhat    0    1
##    0 5937 1520
##    1  348  123
##
## $stats
##   sensitivity specificity
## 1   0.9446301  0.07486306
```

1) See above for model.

2) I included the following variables in QDA (note that I had to eleminate the `kw_x` and `rate_positive/negative_words` variables to reduce covariance; see above for explanation on why I eleminated certain variables):

- `n_unique_tokens`
- `average_token_length`
- `global_subjectivity`

- global_sentiment_polarity
- global_rate_positive_words
- global_rate_negative_words
- avg_positive_polarity
- avg_negative_polarity

Without the aforementioned variables, there are not huge differences in the group means. However, our classifier preformed reasonably well. I will wait until the end to discuss performance.

KNN

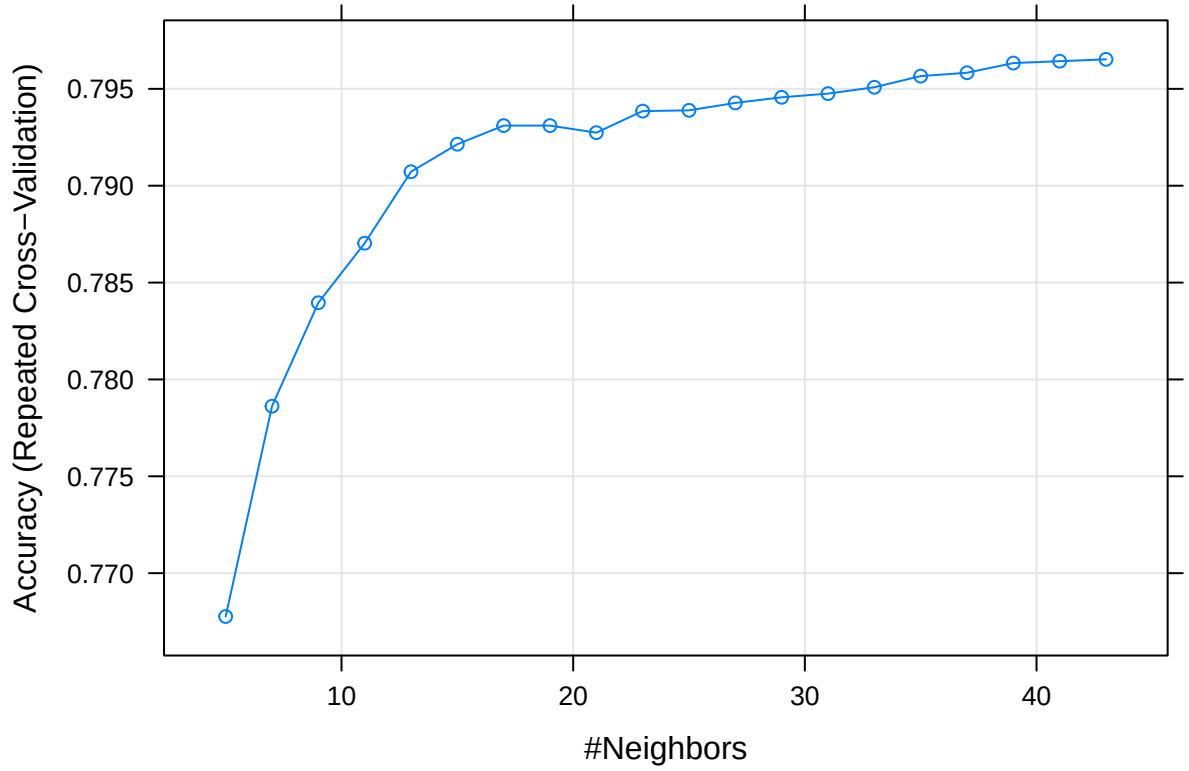
I will start with using cross-validation to pick the best k number of neighbors to use. This pipeline relies heavily on work done by Vijayakumar Jawaharlal.

```
#--> Conver to factor
train.clean$popular <- as.factor(train.clean$popular)
set.seed(919)

#--> 10-fold cross validation
ctrl <- trainControl(method="repeatedcv",repeats = 3)
knnFit <- train(popular ~ ., data = train.clean, method = "knn",
                trControl = ctrl, preProcess = c("center","scale"),
                tuneLength = 20)

#--> Output of kNN fit
knnFit
```

```
## k-Nearest Neighbors
##
## 17350 samples
##    13 predictor
##     2 classes: '0', '1'
##
## Pre-processing: centered (13), scaled (13)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 15614, 15615, 15615, 15615, 15614, 15616, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.7677629  0.09276237
##    7  0.7786176  0.09419155
##    9  0.7839588  0.08859616
##   11  0.7870320  0.08109330
##   13  0.7907209  0.07857562
##   15  0.7921426  0.07186191
##   17  0.7931030  0.06617912
##   19  0.7931032  0.05634339
##   21  0.7927379  0.04796140
##   23  0.7938522  0.04648755
##   25  0.7938908  0.04244812
##   27  0.7942749  0.04076517
##   29  0.7945632  0.03765637
##   31  0.7947553  0.03693973
##   33  0.7950819  0.03398761
##   35  0.7956583  0.03410302
##   37  0.7958314  0.03060684
```

```
##   39  0.7963309  0.03045952
##   41  0.7964269  0.02908111
##   43  0.7965230  0.02843361
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 43.
```

```
#--> Plot
plot(knnFit)
```




```
ktrain <- dplyr::select(train.clean, -popular)
ktest <- dplyr::select(test.clean, -popular)

mknn <- knn(ktrain, ktest, cl = train.clean$popular, k = 43)
confusion(mknn, test.clean$popular)
```

```
## yhat is the vector of predicted outcomes, possibly a factor.
##
##           Sensitivity = (first level predicted) / (first level actual)
##
##           Specificity = (second level predicted) / (second level actual)
```

```
## $confusion_mat
##      y
## yhat    0    1
##    0 6276 1638
##    1    9    5
##
```

```
## $stats
##   sensitivity specificity
## 1    0.998568 0.003043214
```

1) See above for model.
2) We used the following variables in our model (the same ones from LDA; see above for explanation on why I eleminated certain variables):

- n_unique_tokens
- average_token_length
- kw_avg_max
- kw_min_avg
- kw_avg_avg
- global_subjectivity
- global_sentiment_polarity
- global_rate_positive_words
- global_rate_negative_words
- rate_positive_words
- rate_negative_words
- avg_positive_polarity
- avg_negative_polarity

Using 10-fold cross validation on our training data, I found there to be an (almost) monotonic increase in the model accuracy as we increase the number of k neighbors. For tractability, I picked 43, which is the max number that our cross validation tested. If this were a model for a company, I would increase k until we found a global maximum in accuracy. However, to let this file run in a reasonable amount of time, I am selecting k = 43. I will discuss model accuracy below.

(c) (15 points) Download the test data *OnlineNewsPopularityTest.csv*. Predict `popular` class labels using each of the models in (b). Then:

1) Discuss the performance of each method using assessment measures such as MSPE, sensitivity, and specificity (see slide 68-69 for definitions of these objects; here popularity (class label 1) counts as "positives" and not popularity (class label 0) counts as negatives).

I predicted in the same cells that I built the model. I will show a summary of the sensisitivity and specificity of each of our three models below.

```
print("LDA")
```

```
## [1] "LDA"
```

```
confusion(lda.pred$class, test.clean$popular, quietly = TRUE)
```

```
## $confusion_mat
##     y
## yhat    0    1
##    0 6238 1610
##    1   47   33
##
## $stats
##   sensitivity specificity
## 1   0.9925219  0.02008521
```

```
print("QDA")
```

```
## [1] "QDA"
```

```
confusion(qda.pred$class, test.clean$popular, quietly = TRUE)
```

```
## $confusion_mat
##     y
## yhat    0    1
##    0 5937 1520
##    1  348  123
##
## $stats
##   sensitivity specificity
## 1   0.9446301  0.07486306
```

```
print("KNN")
```

```
## [1] "KNN"
```

```
confusion(mknn, test.clean$popular, quietly = TRUE)
```

```
## $confusion_mat
##     y
## yhat    0    1
##    0 6276 1638
##    1    9    5
##
## $stats
##   sensitivity specificity
## 1    0.998568 0.003043214
```

2) Discuss which classifier you prefer and why.

Looking at the metrics above, it is clear that QDA performs the worst for this data set. It has the lowest sensitivity and highest sensitivity. Part of this may have to do with the fact that we had to cull so many variables to meet the strict independence assumption. Ths, we had less "good" data going into our QDA model.

The other two–LDA and KNN–are both strong classifiers. KNN has marginally better sensitivity than LDA (.9986 vs .9925). KNN's specificity is also an order of magnitude better than LDA (.003 vs .02). I would recommend using KNN for this publisher, but with a higher k (as discussed previously).