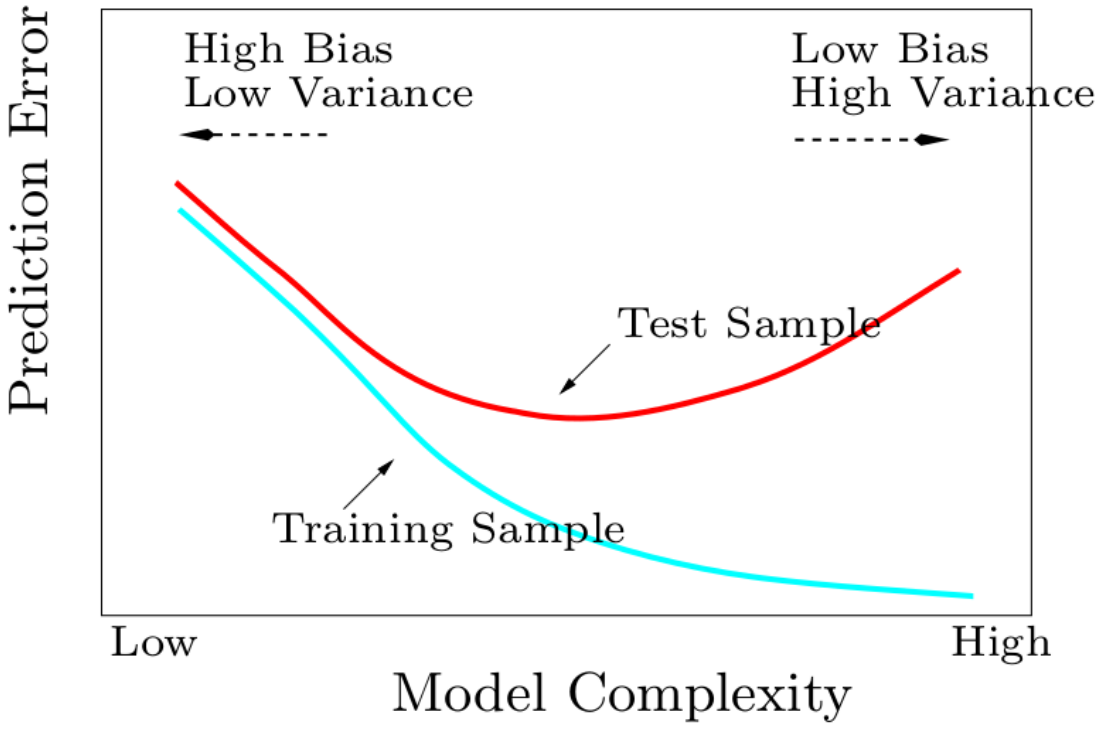
Machine Learning 41204/01 - Midterm

Nathan Matare

2/12/2017

Question 1

- Part 1:
  - A) TRUE - Increasing the number of nearest neighbors (k) will result in a more simple function; thus, as model complexity decreases, bias increases. See below Figure 1
  - B) FALSE - Increasing the number of nearest neighbors (k) will result in a more simple function; thus as model complexity decreases, variance decreases. See below Figure 1



– C) TRUE - Consider a kNN at K = 1 on training data; the model will perfectly fit to its nearest neighbor(itself)–perfectly capturing the relationship; thus as k increases, the model becomes more simple/general and incorporates more neighbores. This results in an increasing misclassification rate on the training set.
– D) UNKNOWN - Dependent on the value of K, the misclassification rate could increase as K increases–the data favoring a more simple model. Or as K increases, the misclassification rate could decrease–the data favoring a more complex model. It depends on the specific value of K, the nature of the data, and whether or not the test data accurately mimics the training data. See Figure 2.

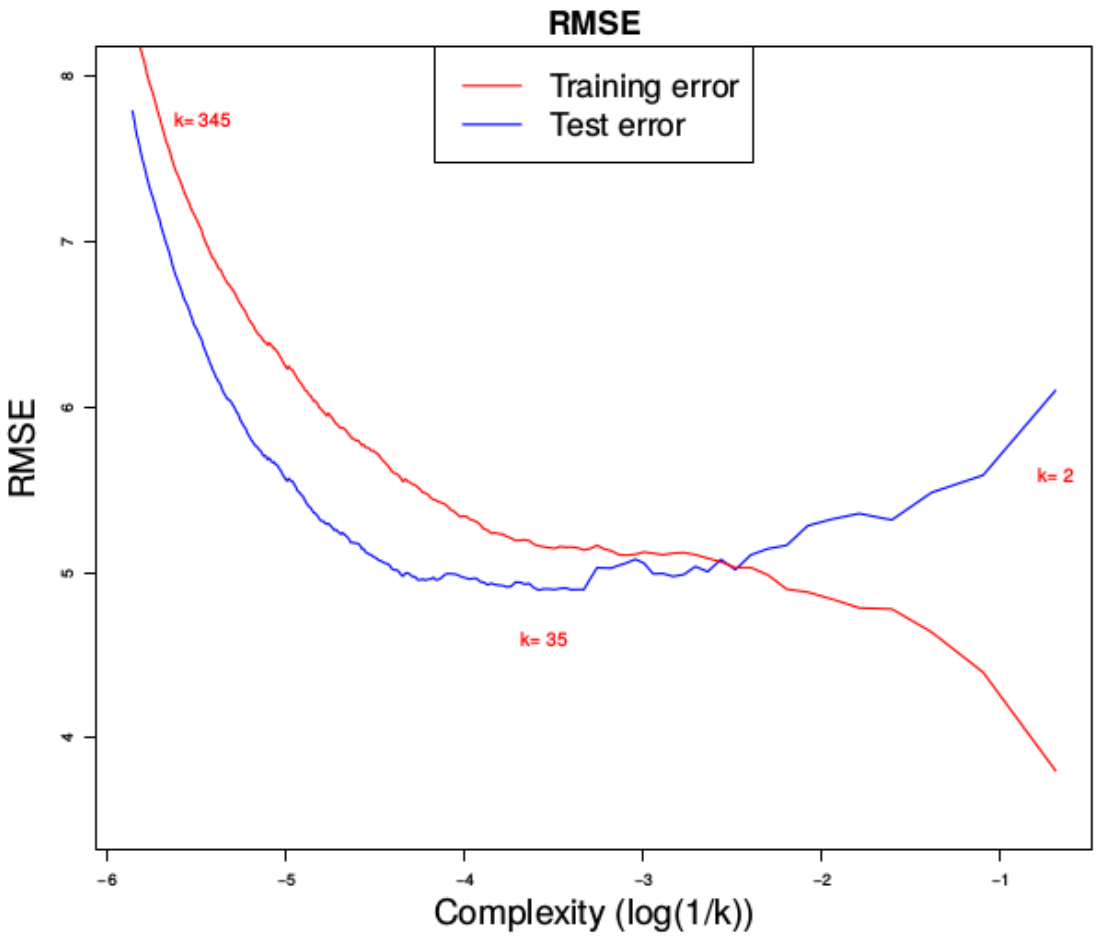


Figure 1: RMSE vs Complexity

- Part 2:
  - A) TRUE - Model 2 is more complex than Model 1; thus as model complexity increases, variance increases. See below Figure 3.
  - B) TRUE - Model 3 is more complex than Model 2; thus as model complexity increases, bias decreases. See below Figure 3.

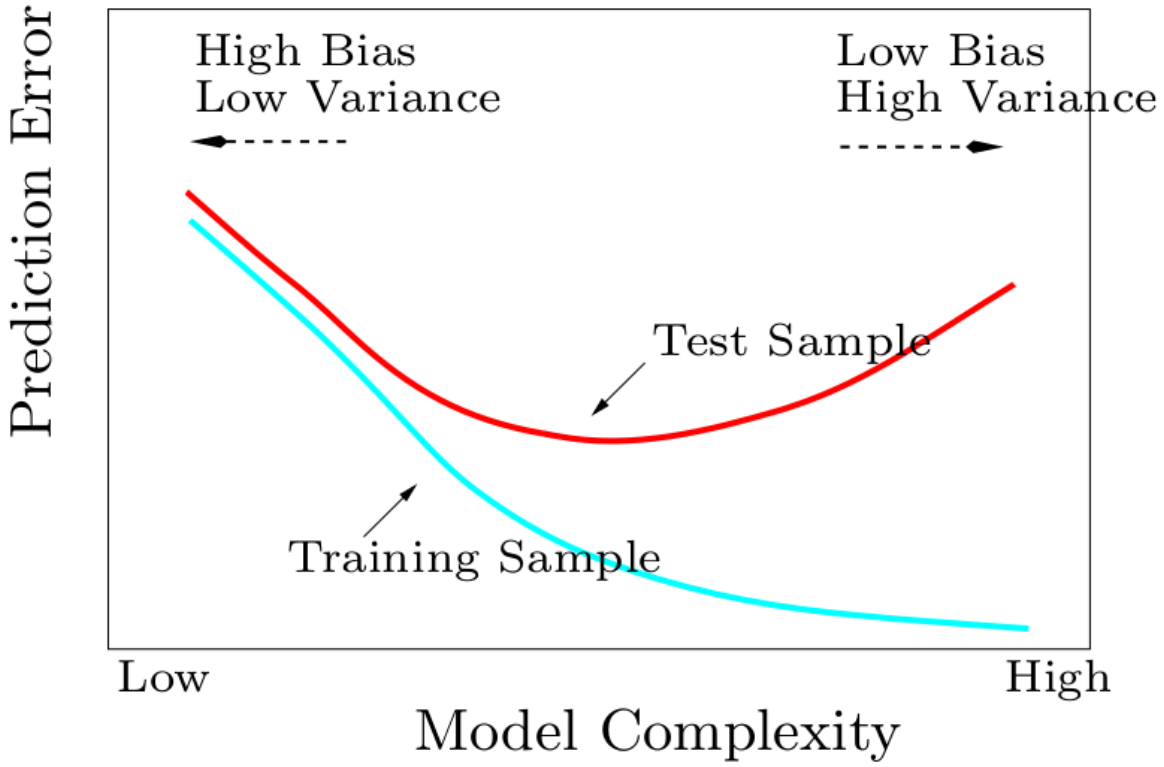


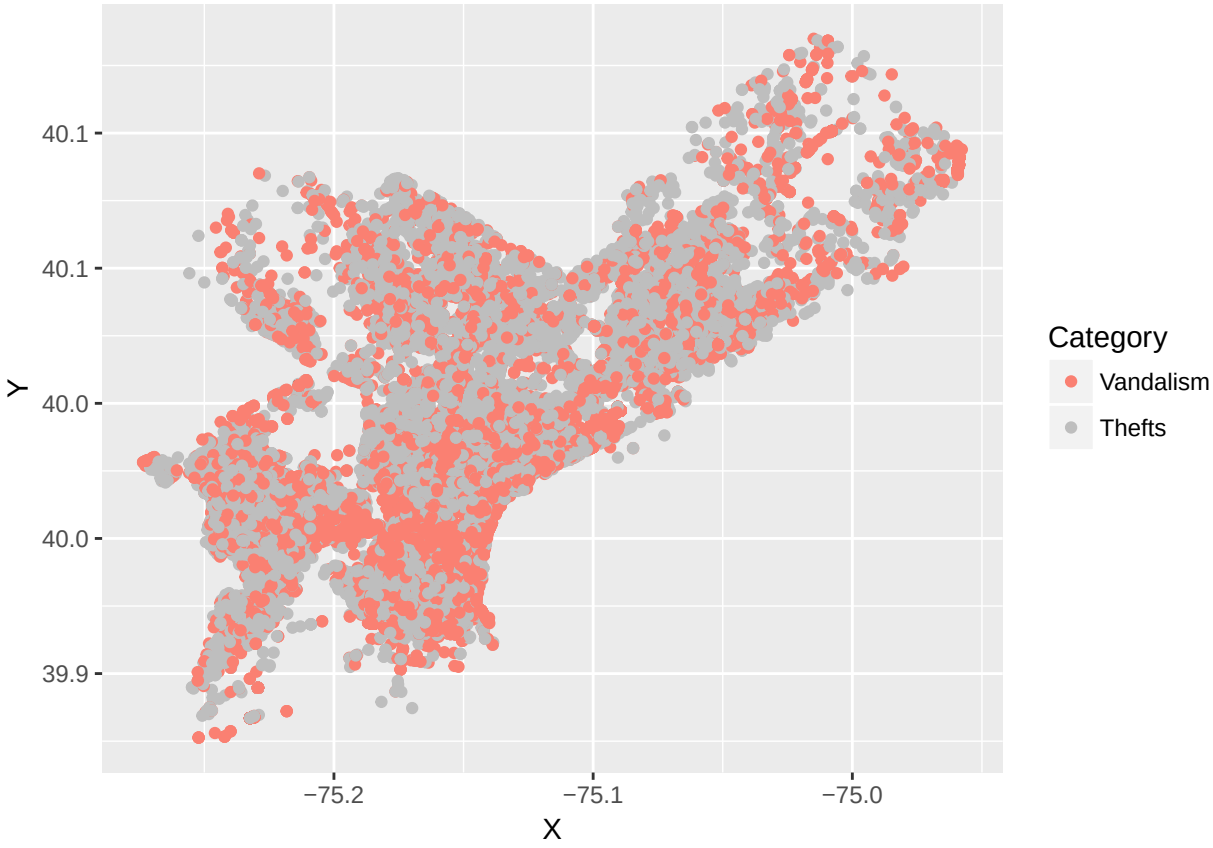
Figure 2: Bias vs Variance

  - C) TRUE - Model 3 is overfit to the training data; however it has the smallest training error due to its low bias.
  - D) FALSE - Model 1 is underfit to the data and does not capture the non-linear relationship well; Model 2 would ostensibly have the smallest test error provided that the testing data mirrors the training data
- Part 3: Usually TRUE - Never is a strong word. When training a model on a training dataset, the aforementioned model will usually fit better to all the data it currently has 'seen'. Thus, provided the model is overfit, the model will subsequently produce greater error on the unseen validation dataset. However, it could be that the model actually evaluates better on the validation dataset (by chance, or peculiarities in the data); this is why we perform multiple random folds of cross-validation.
- Part 4: Slightly TRUE - leave-one-out cross validation is an approximate of an unbiased estimation of the true error; however for theoretical reasons, it is not exactly unbiased[1]. Further, leave-one-out cross validation constrains the heterogeneity of the evaluated data and can produce underestimated predictions. The industry standard is to use 5 or 10 fold cross validation.

[1] http://ai.stanford.edu/~ronnyk/accEst.pdf

Question 2

- Part 1 - I observe a concentration of vandalism around coordinates [40.00, -75.15]; similarly, thefts appear to be more commonplace around the city's main middle region.



- Part 2 - In accordance with question 2, I split the dataset into a 50% training and testing; and fit a kNN model. The function knnMR returns the misclassification rate for kNN given arguments K, a training dataset, and a validation dataset.

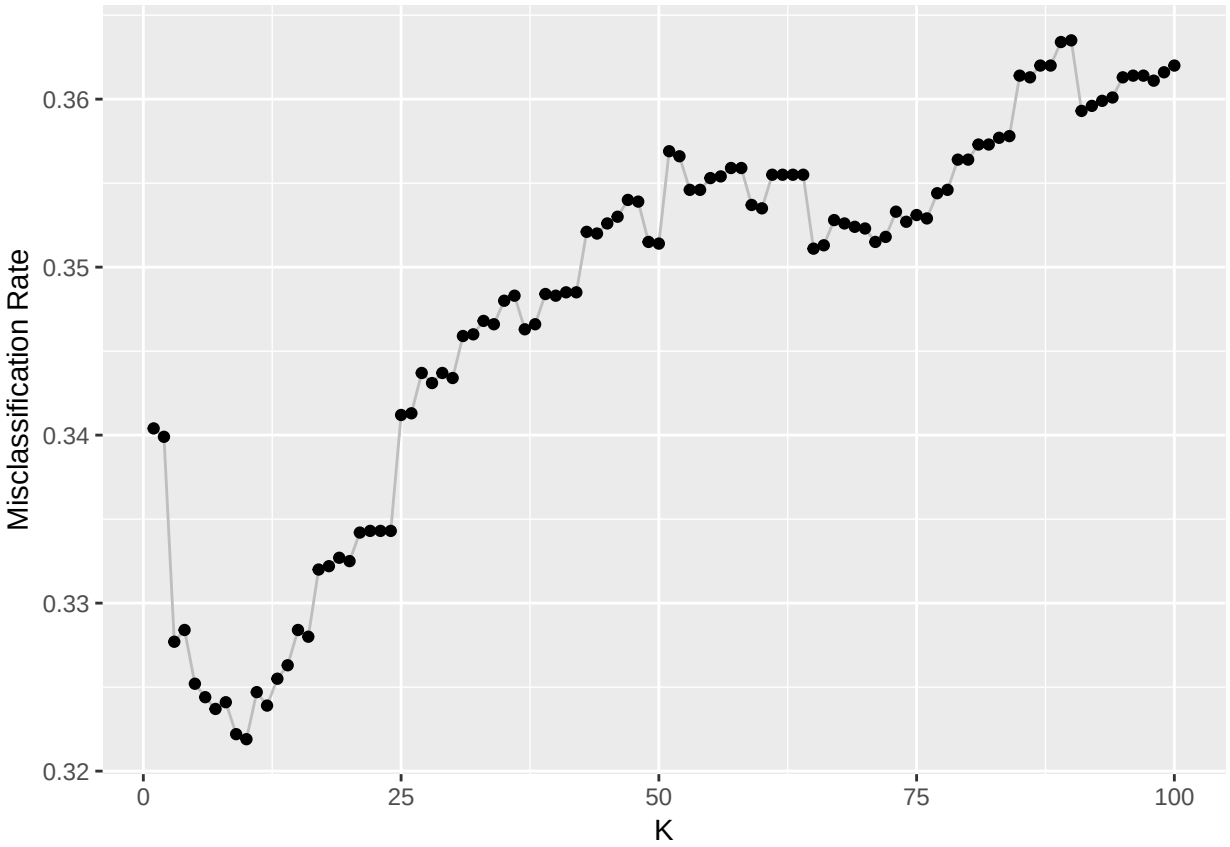
```
data <- data[ ,.(Category, X, Y)]
data[ , ':=' (
    Category = as.factor(Category),
    X        = as.numeric(X),
    Y        = as.numeric(Y)
)]

idx      <- sample(1:NROW(data), NROW(data) * 0.50) # 50% random split
train    <- data[ idx, ]
validate <- data[-idx, ]

knnMR <- function(K, train, validate){
# does KNN for selected K and reports missclassification rate
knn     <- kknn(formula = Category ~ X + Y, train = train,
                test = validate, kernel = "rectangular", k = K)
yhat    <- as.numeric(knn$fitted.values) - 1 # turn into binary; Vandalism = 1, Theft = 0
true_y  <- as.numeric(validate$Category) - 1
MR      <- lossMR(true_y, yhat)
return(MR)
}

result <- as.vector(NULL)
for(k in 1:100) result[k] <- knnMR(k, train, validate)
```

- A) A scatterplot of the misclassification rate on the validation set against the parameter k:



- B) The optimal k selected by the validation-set approach and the minimum misclassification rate on the validation set:

```
which.min(result) # optimal k
```

```
## [1] 10
```

```
result[which.min(result)] # min MR
```

```
## [1] 0.322
```

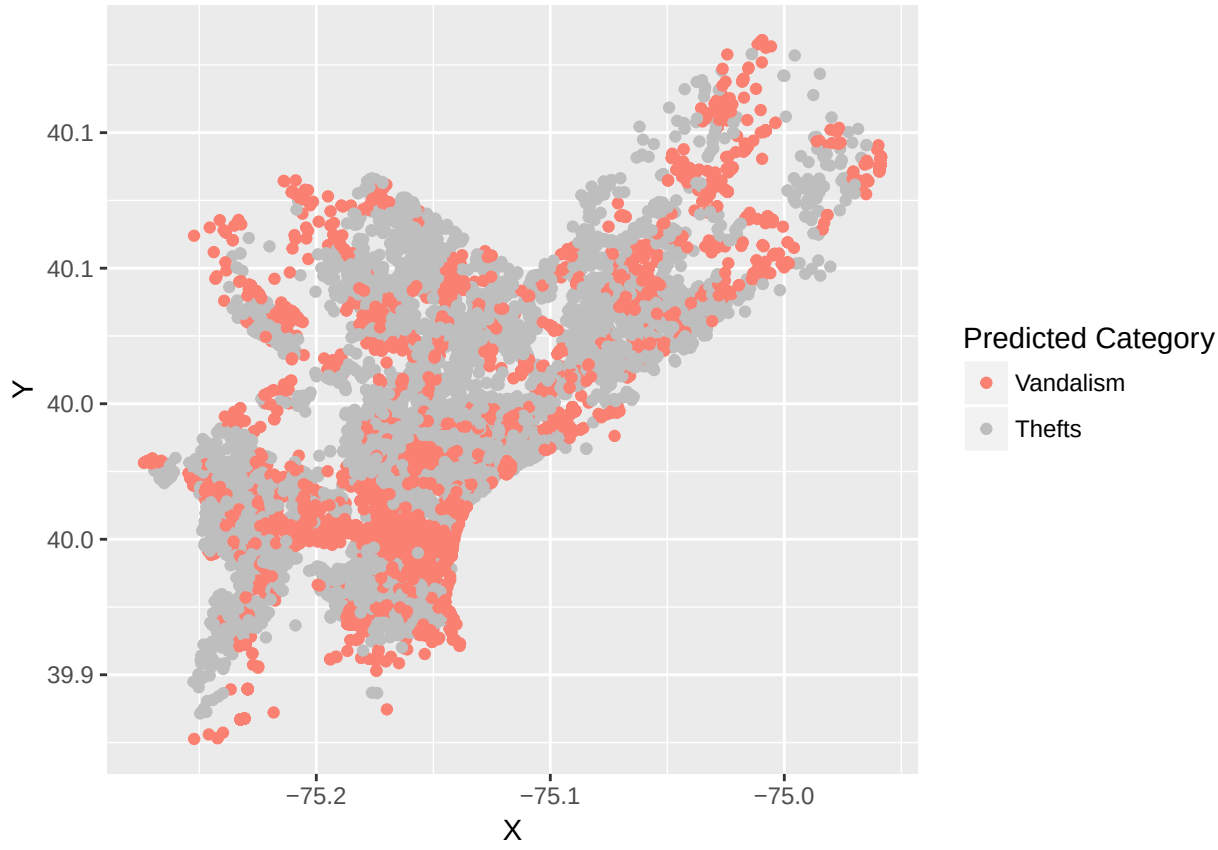
- C) A single scatterplot of the crime incidents using their latitudes (X) and longitudes (Y)

```
plotBestKNN <- function(x, train, validate){

knn     <- kknn(formula = Category ~ X + Y, train = train,
             test = validate, kernel = "rectangular", k = which.min(x)) # at best K
DT      <- cbind(validate, pred_Category = knn$fitted.values) # add the predicted column

p <- ggplot(data = DT, aes(x = X, y = Y, color = pred_Category)) +
            geom_point() +
            scale_color_manual(values = c("salmon", "grey"),
                               name = "Predicted Category",
                               labels = c("Vandalism", "Thefts"))
return(p)
}

plotBestKNN(x = result, train, validate)
```



- Part 3 - In accordance with question 2, I repeat the part 2 20 times:

```
registerDoMC(detectCores() - 1) # detect number of cores to split work apart
detectCores() # boothGrid is awesome @ 64 cores!
```

```
## [1] 8
```

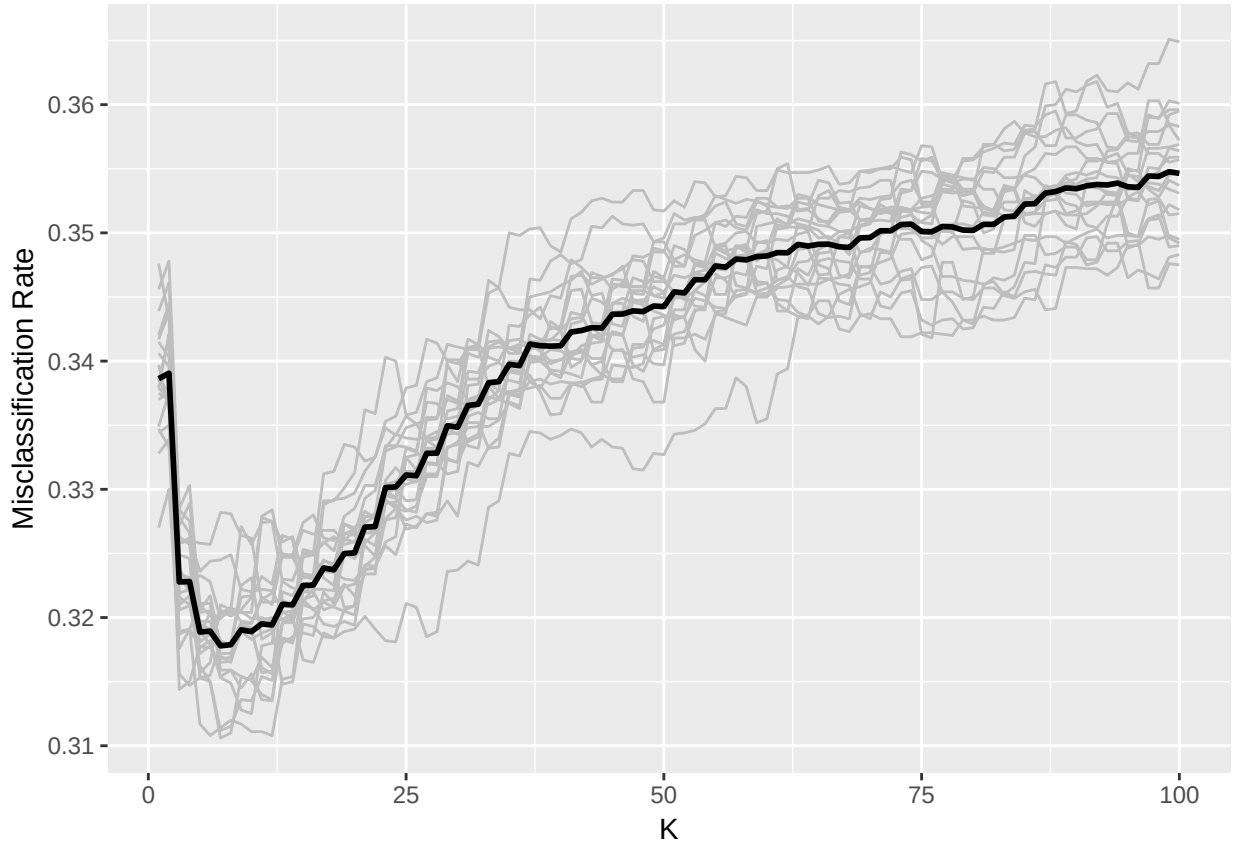
```
idxs     <- replicate(20, sample(1:NROW(data), NROW(data) * 0.50))
# 20, 50% random split; must create outside of foreach
results  <- foreach(i = 1:20) %dopar% {
# resample data 20 times and find optimal k on validation set using devils seed
idx      <- idxs[ ,i]
train    <- data[ idx, ]
validate <- data[-idx, ]

result <- as.vector(NULL)
for(k in 1:100) result[k] <- knnMR(k, train, validate)
return(list(result = result, train = train, validate = validate))
}
```

- A) The 20 optimal k's selected by the validation sets and a scatterplot of the misclassification rate on the validation set against the parameter k.

```
t(matrix(lapply(results, function(x) which.min(x$result)),
         dimnames = list(1:length(results), "K"))) # best K at each N
```

```
##   1  2 3  4  5 6 7  8 9  10 11 12 13 14 15 16 17 18 19 20
## K 12 7 14 12 3 6 13 5 12 5  8  5  5  7  12 14 11 10 7  7
```



- B) The average of the minimum out-of-sample misclassification rates as well as its standard error.

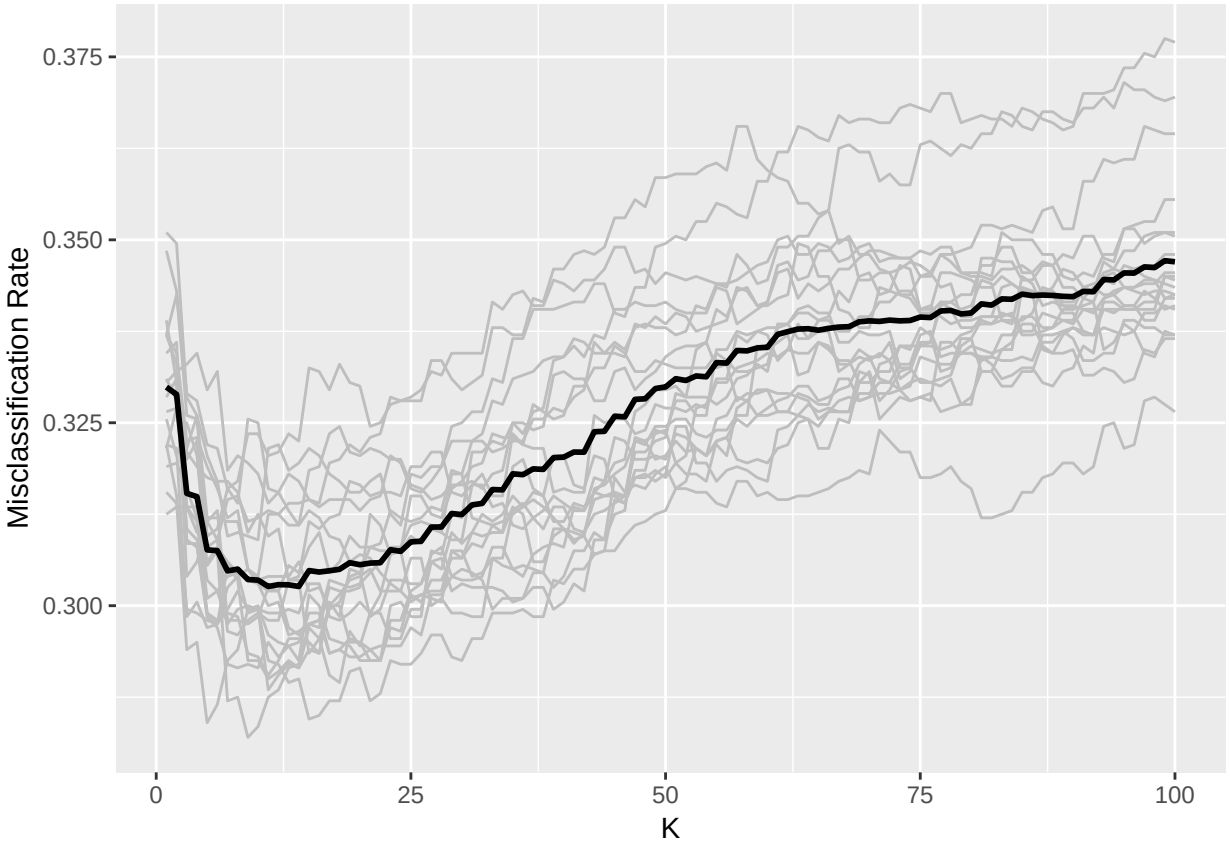
```
mean(unlist(lapply(results, function(x) min(x$result)))) # mean of minimumn OOS classification rate
```

```
## [1] 0.316
```

```
sd(unlist(lapply(results, function(x) min(x$result))))
```

```
## [1] 0.00306
```

- Part 4 - In accordance with question 4, I repeat the part 2 20 times at 90/10 split:
- A) A scatterplot of the misclassification rate on the validation set against the parameter k with the average misclassification rate in bold.



- B) A scatterplot of the misclassification rate on the validation set against the parameter k with the average misclassification rate in bold.
- C) A scatterplot of the misclassification rate on the validation set against the parameter k with the average misclassification rate in bold.

```
t(matrix(lapply(results, function(x) which.min(x$result)),
         dimnames = list(1:length(results), "K"))) # best K at each N
```

```
##   1  2  3 4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
## K 11 20 5 14 31 10 11 14 5 15 8  9  13 10 7  13 14 15 17 9
```

- D) Average minimum out-of-sample misclassification rates and its standard error.

```
mean(unlist(lapply(results, function(x) min(x$result)))) # average min OOS error
```

```
## [1] 0.297
```

```
sd(unlist(lapply(results, function(x) min(x$result))))
```

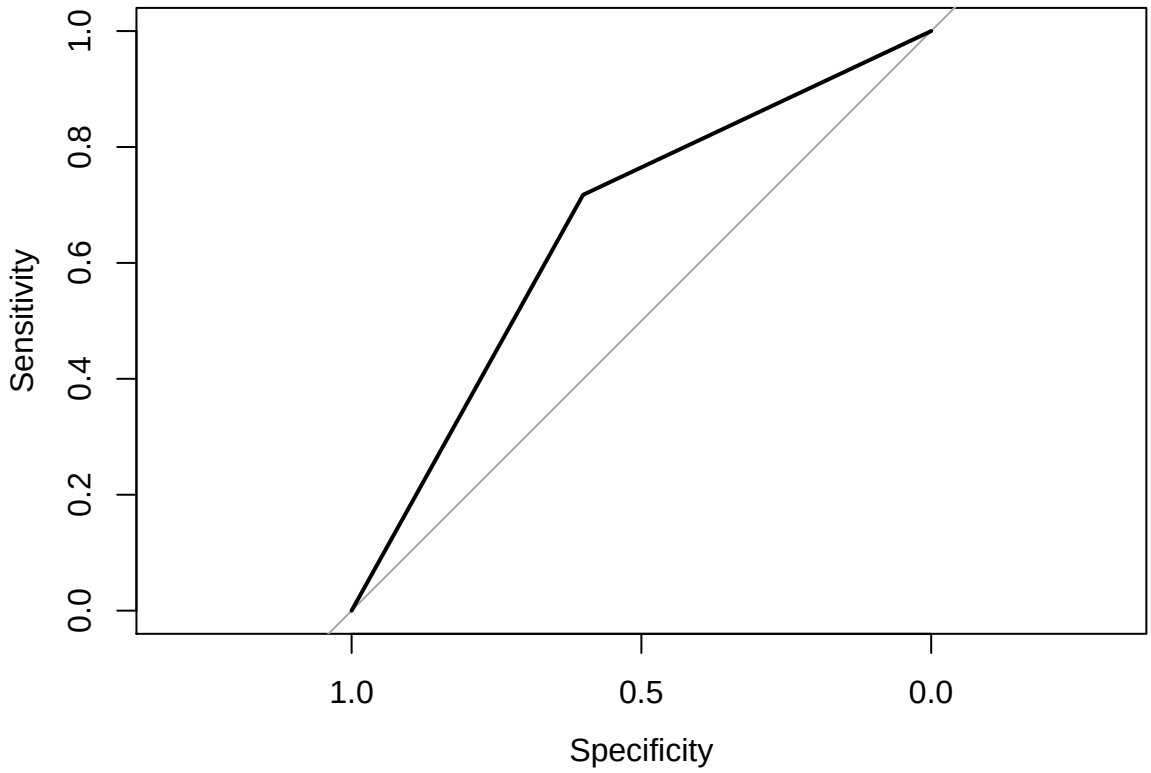
```
## [1] 0.00906
```

- Part 5 - Comment on the difference between the results obtained in (2), (3) and (4).
- Part 6 - I split the data into 50% training/testing and plot the ROC curve:

```
idx      <- sample(1:NROW(data), NROW(data) * 0.50) # 50% random split
train    <- data[ idx, ]
validate <- data[-idx, ]

knn     <- kknn(formula = Category ~ X + Y, train = train, test = validate, kernel = "rectangular", k =
yhat    <- as.numeric(knn$fitted.values) - 1 # turn into binary; Vandalism = 1, Theft = 0
true_y  <- as.numeric(validate$Category) - 1

pROC:::roc(response = true_y, predictor = yhat, plot = TRUE) # ROC curve, AUC is 0.665
```



Question 3

Question 4

- Part 1 - Gross Profit:

```
p <- 5.46 # revenue per catalog
c <- 2.00 # cost per catalog

mr <- p - c # marginal revenue
num <- 180000 * 0.053 # number of customers responding
num * mr # gross profit
```

```
## [1] 33008
```

- Part 2 - I begin by combining the training and validation partitions and removing erroneous columns. I combine the training and validation partitions because I do cross validation automatically in the subsequent packages.

```
# combine training and validation together; remove spending
train_p <- data[Partition %in% c('t', 'v'),
                !which(colnames(data) %in%
                c('Spending', 'sequence_number', 'Partition')),
                with = FALSE]

test_p <- data[Partition %in% c('s'),
               !which(colnames(data) %in%
               c('Spending', 'sequence_number', 'Partition')),
               with = FALSE]
```

First I try a linear model:

```
LASSO <- cv.gamlr(  x = train_p[ ,which(colnames(train_p) != 'Purchase'), with = FALSE],
                    y = train_p[ ,Purchase],
                    family = 'binomial', verb = FALSE, lambda.start = 0.1, nfold = 10)
```

```
## Warning in gamlr(x, y, ...): numerically perfect fit for some observations.
```

```
yhat <- predict(LASSO, newdata = train_p[ ,which(colnames(train_p) != 'Purchase'),
                                          with = FALSE], type = 'response')

lossMR(train_p[ ,Purchase], yhat) # LASSO prediction error
```

```
## [1] 0.205
```

Second, I try a random forest:

```
RF <- ranger(   as.factor(Purchase) ~.,     data = train_p,
                probability = TRUE, classification = TRUE, num.trees = 5000,
                write.forest = TRUE, num.threads = detectCores() - 1,
                importance = 'impurity', verbose = TRUE
)

RF$prediction.error * 100 # OOB prediction error
```

```
## [1] 12.2
```

Finally, I employ boosted trees:

```
params <- list(max_depth = 4, booster = "gbtree", objective = "binary:logistic")
XGBST.cv <- xgb.cv( params = params,
                    data = as.matrix(train_p[ ,which(colnames(train_p) != 'Purchase'), with = FALSE]),
```

```
                    label = as.vector(train_p[ ,Purchase]),
                        nthread = detectCores() - 1, verbose = 1, nfold = 10, nrounds = 100)
```

```
tail(XGBST.cv$evaluation$test_error_mean, 1) # last cv.fold missclassification error
```

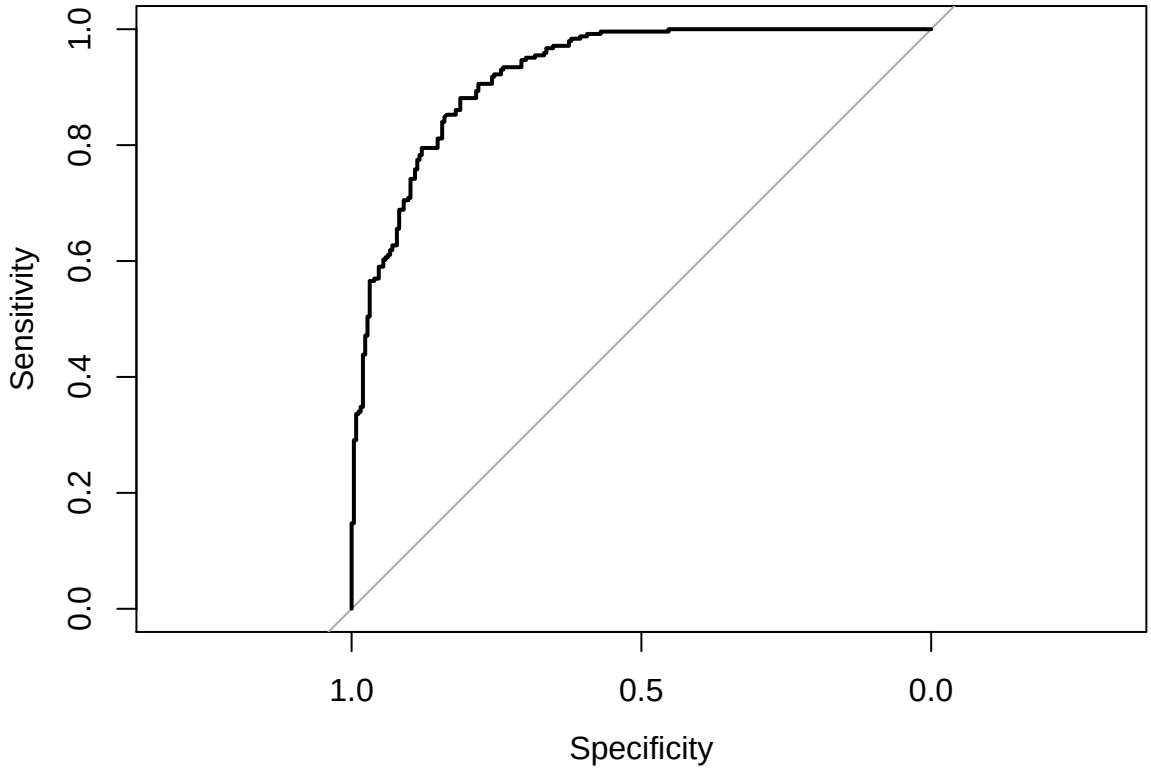
```
## [1] 0.187
```

Thus, I choose bagged trees (Random Forest) because it fits the data the best – it has the lowest misclassification rate at 12.2%

- Part 3 - A ROC curve for the chosen model in the previous step on the test data.

```
phat <- ranger:::predict.ranger(RF,
                                test_p[ ,which(colnames(test_p) != 'Purchase'),
                                           with = FALSE])$predictions[ ,'TRUE']

# ROC curve, AUC is 0.923
pROC:::roc(response = test_p[ ,Purchase], predictor = phat, plot = TRUE)
```



- Part 4 - I again combine the training and validation partitions and remove erroneous columns. I combine the training and validation partitions because I do cross validation automatically in the subsequent packages. I only include the observations where users purchased a catalog.

```
# combine training and validation together; remove spending
train_s <- data[Partition %in% c('t', 'v') & Purchase == TRUE,
                !which(colnames(data) %in%
                c('sequence_number', 'Partition', 'Purchase')), with = FALSE]

test_s <- data[Partition %in% c('s') & Purchase == TRUE,
               !which(colnames(data) %in%
               c('sequence_number', 'Partition', 'Purchase')), with = FALSE]
```

First I try a linear model:

```
LASSO <- cv.gamlr(  x = train_s[ ,which(colnames(train_s) != 'Spending'), with = FALSE],
                    y = train_s[ ,Spending],
                    family = 'gaussian', verb = FALSE, lambda.start = Inf, nfold = 10)

sqrt(LASSO$cvm[which.min(LASSO$cvm)]) # RMSE
```

```
## seg75
##   166
```

Second, I try a random forest:

```
RF <- ranger(   Spending ~.,    data = train_s,
                probability = FALSE, classification = FALSE, num.trees = 5000,
                write.forest = TRUE, num.threads = detectCores() - 1,
                importance = 'impurity', verbose = TRUE
)

sqrt(RF$prediction.error) # RMSE
```

```
## [1] 167
```

Finally, I employ boosted trees:

```
params <- list(max_depth = 4, booster = "gbtree", objective = "reg:linear")
XGBST.cv <- xgb.cv( params = params,
                    data = as.matrix(train_s[ ,which(colnames(train_s) != 'Spending'), with = FALSE]),
                    label = as.vector(train_s[ ,Spending]),
                        nthread = detectCores() - 1, verbose = 1, nfold = 10, nrounds = 100)
```

```
tail(XGBST.cv$evaluation$test_rmse_mean, 1) # last cv.fold missclassification error; best RMSE
```

```
## [1] 167
```

Thus, I choose boosted trees because they fit the data the best – the model has the lowest RMSE at 167

- Part 5 - Using the best parameters found in cross validation, I fit boosted trees to the training/validation data and predict on the testing partition:

```
XGBST <- xgboost(   params = params,
                  data = as.matrix(train_s[ ,which(colnames(train_s) != 'Spending'), with = FALSE]),
                           label = as.vector(train_s[ ,Spending]),
                           nthread = detectCores() - 1, verbose = 1, nrounds = 500)

# predicted spending given best model
yhat <- predict(XGBST, as.matrix(test_s[ ,which(colnames(test_s) != 'Spending'), with = FALSE]))
```

Next, I place the probability or responding into the testing subset and record the predicted spending for those who purchased:

```
test_p[ ,phat := phat] # add the phats to purchase data.table
test_p[Purchase == TRUE, yhat := yhat] # add yhats to those that purchased
```

Finally, I plot the expected spending:

```
expected_spending <- test_p[ ,cumsum(sort(yhat * phat * 0.107)) / num * mr]
ggplot(data = data.frame(expected_spending),
       aes(x = 1:NROW(expected_spending), y = expected_spending)) +
      geom_line() #xlab("K") + ylab("Misclassification Rate")
```