# Classification, Clustering, & Validation

## k-Nearest Neighbor Model (Cross-Validation)

To find a good classifier for the credit card data set, both a k-Nearest Neighbor and Support Vector Machine model were fit using different validation methods.

### Data Pre-Processing

To begin, the first step in this process was to view the nature of the credit card data set.

```
##        A1               A2               A3               A8
##  Min.   :0.0000   Min.   :13.75   Min.   : 0.000   Min.   : 0.000
##  1st Qu.:0.0000   1st Qu.:22.58   1st Qu.: 1.040   1st Qu.: 0.165
##  Median :1.0000   Median :28.46   Median : 2.855   Median : 1.000
##  Mean   :0.6896   Mean   :31.58   Mean   : 4.831   Mean   : 2.242
##  3rd Qu.:1.0000   3rd Qu.:38.25   3rd Qu.: 7.438   3rd Qu.: 2.615
##  Max.   :1.0000   Max.   :80.25   Max.   :28.000   Max.   :28.500
##        A9              A10              A11              A12
##  Min.   :0.0000   Min.   :0.0000   Min.   : 0.000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.:0.0000
##  Median :1.0000   Median :1.0000   Median : 0.000   Median :1.0000
##  Mean   :0.5352   Mean   :0.5612   Mean   : 2.498   Mean   :0.5382
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.: 3.000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :67.000   Max.   :1.0000
##       A14              A15               R1
##  Min.   :   0.00   Min.   :     0   Min.   :0.0000
##  1st Qu.:  70.75   1st Qu.:     0   1st Qu.:0.0000
##  Median : 160.00   Median :     5   Median :0.0000
##  Mean   : 180.08   Mean   :  1013   Mean   :0.4526
##  3rd Qu.: 271.00   3rd Qu.:   399   3rd Qu.:1.0000
##  Max.   :2000.00   Max.   :100000   Max.   :1.0000
```

From this, it could be seen that not all of the attributes were on the same scale. Because of this, data needed to be scaled prior to fitting a model. Scaling was chosen over standardization because some attributes were already bounded, and therefore bounding all attributes between 0 and 1 was advisable.

```
for (c in seq(1:11)){
  cc.data[c] = (cc.data[c] - min(cc.data[[c]])) / (max(cc.data[[c]]) - min(cc
.data[[c]]))
}
summary(cc.data)
```

```
##        A1               A2               A3                A8
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.0000   1st Qu.:0.1328   1st Qu.:0.03714   1st Qu.:0.00579
```

```
##  Median :1.0000   Median :0.2212   Median :0.10196   Median :0.03509
##  Mean   :0.6896   Mean   :0.2681   Mean   :0.17252   Mean   :0.07866
##  3rd Qu.:1.0000   3rd Qu.:0.3684   3rd Qu.:0.26562   3rd Qu.:0.09175
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :1.00000
##        A9               A10              A11               A12
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.0000
##  Median :1.0000   Median :1.0000   Median :0.00000   Median :1.0000
##  Mean   :0.5352   Mean   :0.5612   Mean   :0.03729   Mean   :0.5382
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:0.04478   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :1.0000
##       A14               A15                R1
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.0000
##  1st Qu.:0.03537   1st Qu.:0.00000   1st Qu.:0.0000
##  Median :0.08000   Median :0.00005   Median :0.0000
##  Mean   :0.09004   Mean   :0.01013   Mean   :0.4526
##  3rd Qu.:0.13550   3rd Qu.:0.00399   3rd Qu.:1.0000
##  Max.   :1.00000   Max.   :1.00000   Max.   :1.0000
```

From this, it can be seen that the data were bounded at 0 and 1. Once the data was clean, analysis using the KNN and SVM models were conducted.

## k-Nearest Neighbor Model (k-Fold Cross-Validation)

To find a good classifier for the credit card data set, first a k-Nearest Neighbor model was used to find a good classifier for the data. For this model, cross-validation was used to ensure any accuracy metrics reported were not too optimistic, and to ensure that important data points appeared in both the training and validation sets.

To fit the KNN model, first the number of partitions for the k-fold cross-validation were set (k=10).

```
# set number of partitions for k-fold cross-validation (k=10)
cross_val = trainControl(method  = "cv",
                         number  = 10)
```

Next, the KNN model was fit using k-fold cross-validation. Additionally, the best value for k (k-nearest neighbor) was selected (k: 1-25)

```
# fit knn model, find best k (1-25), using k-fold cross-validation
knn.best.fit = train(x = cc.data[,1:10],
                     y = as.factor(cc.data[,11]),
                     method = "knn",
                     tuneGrid = expand.grid(k = 1:25),
                     trControl = cross_val)
# best k & accuracy
knn.best.fit$results[which.max(knn.best.fit$results$Accuracy),]

##     k Accuracy     Kappa AccuracySD    KappaSD
## 11 11 0.856317 0.7136513 0.04890542 0.09691206
```

From this analysis, it was found that the most accurate KNN model was when k = 11, and this model had an accuracy of 85.63%.

## Support Vector Machine Model (Training, Validation, & Test Sets)

Next, to find a good classifier for the credit card data set, a support vector machine model was used. A model was fit using a training set, the best one was selected using a validation set, and an accuracy metric was calculated using a testing set. This approach was taken to ensure that any accuracy metrics used throughout this process were not too optimistic.

First, the credit card data set was split into training, testing, and validation sets. These sets were selected at random to ensure bias was not introduced into the training, validation, or testing process.

```
# create data set splits
splits = sample(c(rep(0,0.7*nrow(cc.matrix)),
         rep(1,0.15*nrow(cc.matrix)),
         rep(2,0.15*nrow(cc.matrix))))
# create training, validation, and testing data
train = cc.matrix[splits == 0,]
valid = cc.matrix[splits == 1,]
test = cc.matrix[splits == 2,]
```

After the data was split into training, validation, and testing sets, an initial model was fit with lambda = $10^{-10}$. Last time this data set was analyzed, it was found that the accuracy of SVM models using different values of lambda did not change drastically unless lambda changed by multiple orders of magnitude. To account for this, models would be fit where lambda ranged from $10^{-10}$ to $10^{5}$, and the most accurate model would be selected. The radial basis function kernel was used for these SVM models, as it provided the most accurate classifier when analyzing this data set previously. SVM models were fit with the training set, each model's accuracy was calculated with the validation set to choose the best model, and the best model's accuracy was reported using the testing set.

```
# fit initial ksvm model on training set, make prediction
# find accuracy on validation set (lambda = 10^-10)
best.rbf.fit = ksvm(x = train[,1:10],
                    y = train[,11],
                    type = 'C-svc',
                    kernel = 'rbfdot',
                    C = 10^-10)
best.rbf.pred = predict(best.rbf.fit,valid[,1:10])
best.rbf.acc = sum(best.rbf.pred == valid[,11]) / nrow(valid)

# find the best ksvm model on training set
# maximizing accuracy on validation set, lambda: 10^-10 - 10^5
for(x in seq(-9,5)){
  rbf.fit = ksvm(x = train[,1:10],
                 y = train[,11],
                 type = 'C-svc',
```

```
                  kernel = 'rbfdot',
                  C = 10^x)
  rbf.pred = predict(rbf.fit,valid[,1:10])
  rbf.acc = sum(rbf.pred == valid[,11]) / nrow(valid)
  if(rbf.acc > best.rbf.acc){
    best.rbf.fit = rbf.fit
    best.rbf.pred = rbf.pred
    best.rbf.acc = rbf.acc
  }
}
## best lambda
best.rbf.fit@param$C

## [1] 10

# best model accuracy using testing set
best.rbf.pred = predict(best.rbf.fit,test[,1:10])
best.rbf.acc = sum(best.rbf.pred == test[,11]) / nrow(test)
best.rbf.acc

## [1] 0.8367347
```

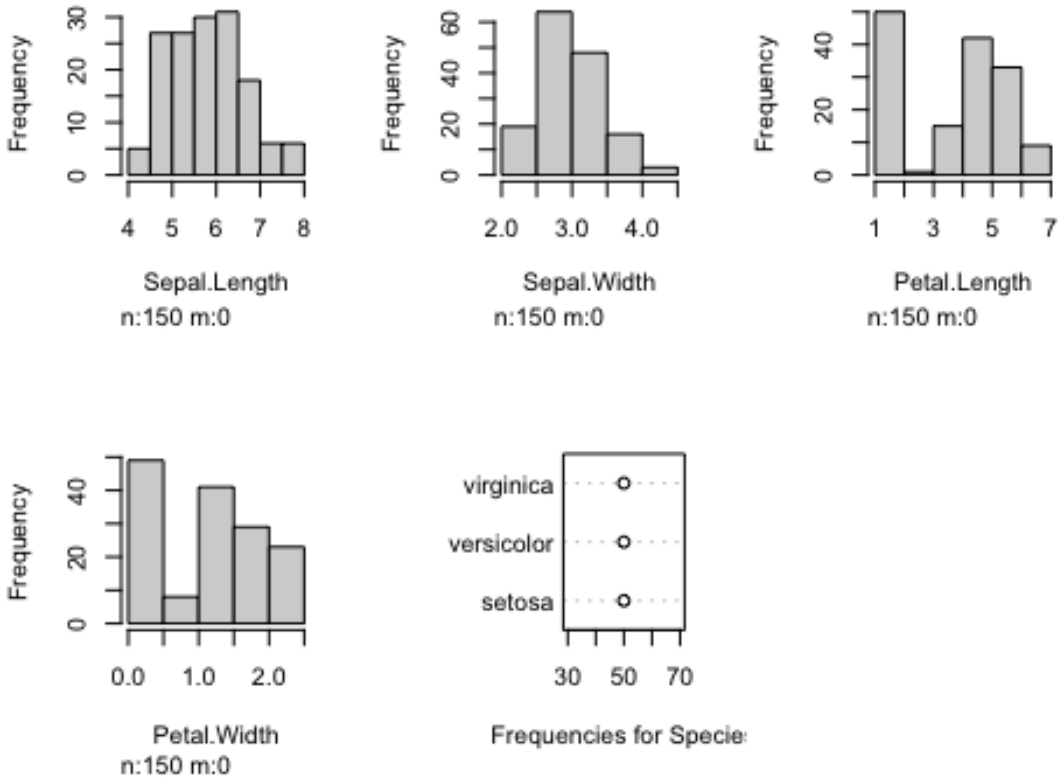
From this analysis, it was found that the most accurate SVM model was when lambda = 10, and this model had an accuracy of 83.67%. Notably the KNN model fit previously was more accurate than the SVM model.

## Clustering Model (k-Means Algorithm)

To cluster the points of the *iris* data set, the k-Means Algorithm was using to find the best combination of predictors.

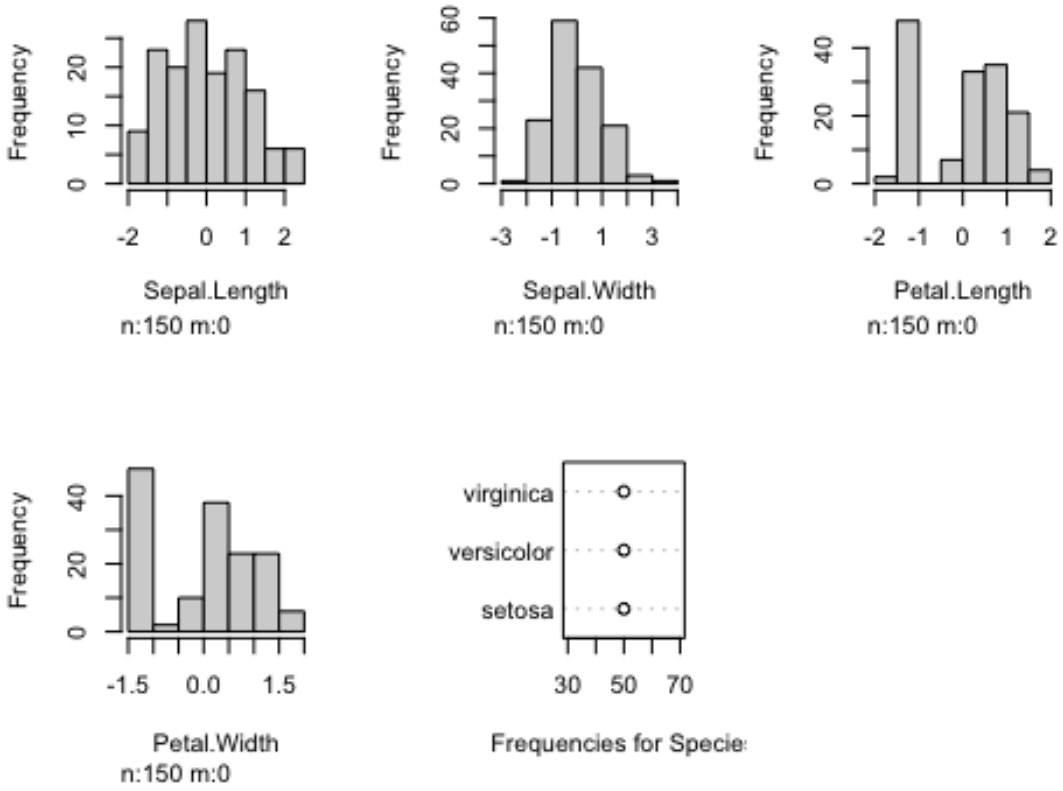
### Data Pre-Processing

To begin, the first step in this process was to view the nature of the *iris* data set.




From this, it could be seen that all the attributes in the data set were distributed normally. Because of this, data needed to be standardized before analysis could take place. Standardization was chosen over scaling the data since it is better for clustering models.

```
for (c in seq(1:4)){
  iris[c] = (iris[c] - mean(iris[[c]])) / sd(iris[[c]])
}
hist.data.frame(iris)
```




From this, it can be seen that the data are no closer to being normally distributed after standardization. Because of this, analysis using the k-Means Algorithm would take place with the unstandardized data (the iris data set was re-initialized as the original, unstandardized version).

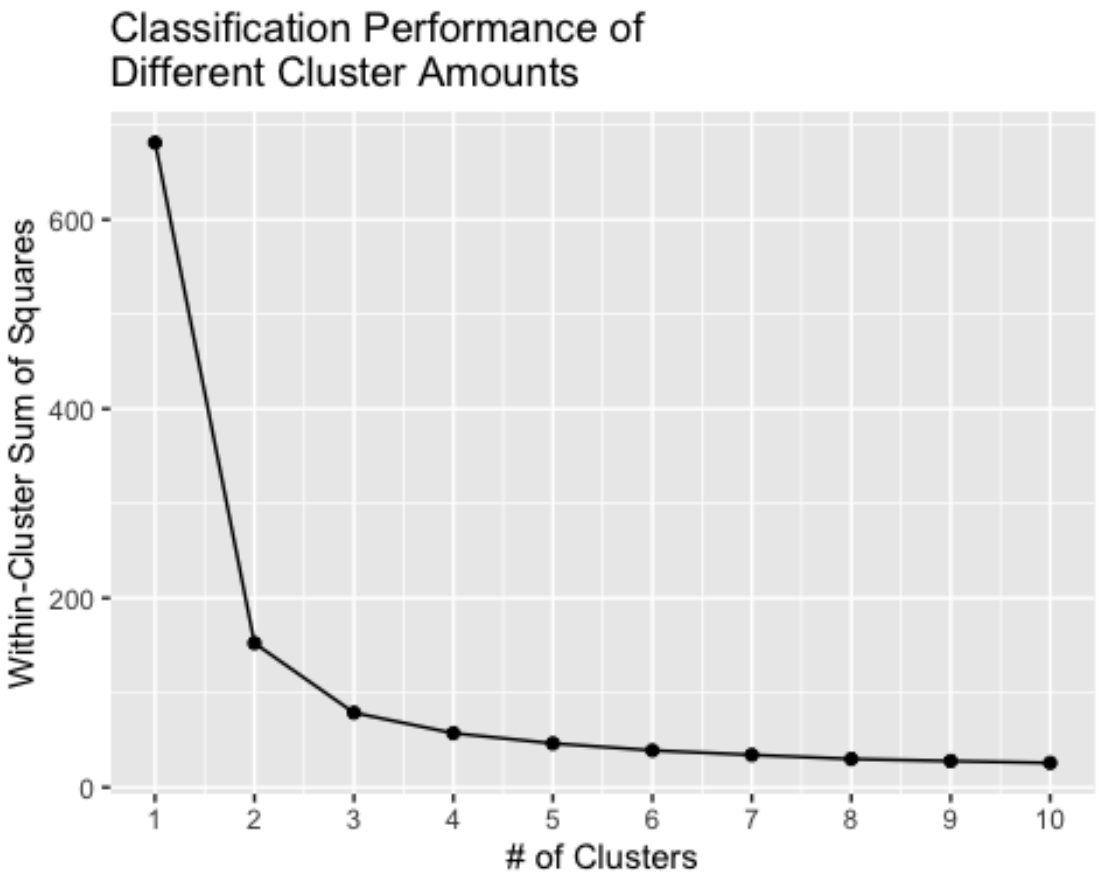
## k-Means Model

To fit a k-Means Model which clusters the data well, it needed to be determined how many clusters provided the most accurate fit. To do so, a k-Means Model was fit with clusters ranging from 1-10. Initially, all four predictors were used. Additionally, 50 initial configurations of the clusters were attempted and the best one was used, and, the maximum iterations of the algorithm was set at 1000 iterations. Since the k-Means Model is heuristic, both of these parameters was specified to increase the likelihood of finding the best possible model. Furthermore, the Hartigan-Wong algorithm was selected since it updates centroids and assigns data points with more sophistication than other algorithms, at the cost of more computational time. The within-cluster sum of squares was calculated for each model to be examined after when choosing the best model.

```
# create empty list to store accuracies calculated in for loop
kmeans.acc = data.frame(k=seq(1,10),
                        wcss=rep(0,10))
# iterate through different numbers of clusters (k: 1-10)
```

```
for(k in seq(1,10)){
  # create fit using k-means algorithm
  kmeans.fit = kmeans(x = iris[1:4],
                      centers = k,
                      nstart = 50,
                      iter.max = 1000,
                      algorithm = "Hartigan-Wong")
  kmeans.acc[k,2] = kmeans.fit$tot.withinss
}
```

From the plot below, it can be seen using the elbow rule that k = 3 is the correct number of clusters, as the within-cluster sum of squares begins decreasing at a much slower rate with the addition of more clusters.




Once the best number of clusters was determined, next the best combinations of the four possible predictors was determined. To do so, all combinations of the attributes were created (4 choose 1, 2, 3, and 4)

```
# find all combinations of 4 iris attributes
one = combn(c(1,2,3,4),1, simplify = F)
two = combn(c(1,2,3,4),2, simplify = F)
three = combn(c(1,2,3,4),3, simplify = F)
```

```
four = combn(c(1,2,3,4),4, simplify = F)
combinations = list(one,two,three,four)
```

Next, each combination was iterated through, a k-Means model was fit with that combination of attributes, and the within-cluster sum of squares was calculated. The best model was chosen by finding the model with the lowest within-cluster sum of squares. k = 3 was used as it was determined to be the best number of clusters, and the best combination of parameters was reported.

```
# create empty variable to check within-cluster sum of squares
best.wcss = Inf
# iterate through all combiantions of predictors,
for(i in seq(1:4)){
  for(n in seq(length(combinations[i]))){
    kmeans.fit = kmeans(x = iris[combinations[[i]][[n]]],
                        centers = 3,
                        nstart = 50,
                        iter.max = 1000,
                        algorithm = "Hartigan-Wong")
    model.wcss = kmeans.fit$tot.withinss
    if (model.wcss < best.wcss){
      best.param = combinations[[i]][[n]]
    }
  }
}
best.param

## [1] 1 2 3 4
```

From this, it can be seen that all four predictors being used provided the model with the lowest within-cluster sum of squares.

Now that the best predictors and number of clusters had been determined, the best model was fit on the data, predictions of classifications were created, and the predictions were tested against the actual classifications to see how well the model performed.

```
# fit best k-means model
best.kmeans.fit = kmeans(x = iris[1:4],
                         centers = 3,
                         nstart = 50,
                         iter.max = 1000,
                         algorithm = "Hartigan-Wong")
# predict the classificaitons using the model
best.kmeans.pred = fitted(best.kmeans.fit,
                          method = 'classes')
# test the accuracy of the prediction on the actual classifications
sum(best.kmeans.pred == as.numeric(iris$Species)) / length(iris$Species)

## [1] 0.8933333
```

From this, it could be seen the the k-Means model (k = 3) accurately predicted 89.33% of the flower types in the *iris* data set.

## Conclusions

From fitting the SVM and KNN models on the credit card data set, it was interesting to see how the use of validation sets impacted the final results of the models selected, and their accuracies. While the SVM model did not find a different value for lambda compared to not using a validation set, the final accuracy reported from the testing set was different. Additionally, the KNN model did find a better value for k compared to not using a cross-validation method, and the KNN model ended up being the more accurate model. This was also different than when no validation sets were used, though it is worth noting that the accuracy of each model did not differ much from each other.

Furthermore, the k-Means model's best cluster predicted the flower type from the *iris* data set relatively accurately. When three clusters and all four predictors were used the k-Means model provided the most accurate fit, though it must be noted that using this algorithm the absolute best fit for clusters cannot be ensured.