Samantha Misurda

smisurda@andrew.cmu.edu

HW2, Part 4

A. For values of k from 1 through 25, the following results were obtained from running get_metrics:

```
########
# Part 4A
########

answer <- data.frame()
for(k in 1:25){
  nn <- ""
  nn <- paste0(k,"nn")
  result <- do_cv_class(wines, 10, nn)
  result.row <- get_metrics(result)
  answer <- rbind(answer, result.row)
}
answer
```

```
##          tpr       fpr       acc precision    recall
## 1  0.8402204 0.2357143 0.8071540 0.8221024 0.8402204
## 2  0.9146006 0.4000000 0.7776050 0.7477477 0.9146006
## 3  0.8622590 0.2392857 0.8180404 0.8236842 0.8622590
## 4  0.9146006 0.3607143 0.7947123 0.7667436 0.9146006
## 5  0.8898072 0.2464286 0.8304821 0.8239796 0.8898072
## 6  0.9063361 0.3392857 0.7993779 0.7759434 0.9063361
## 7  0.8732782 0.2607143 0.8149300 0.8128205 0.8732782
## 8  0.9118457 0.3250000 0.8087092 0.7843602 0.9118457
## 9  0.8677686 0.2821429 0.8024883 0.7994924 0.8677686
## 10 0.9146006 0.3392857 0.8040435 0.7775176 0.9146006
## 11 0.8870523 0.2857143 0.8118196 0.8009950 0.8870523
## 12 0.8953168 0.3214286 0.8009331 0.7831325 0.8953168
## 13 0.8980716 0.3000000 0.8118196 0.7951220 0.8980716
## 14 0.9063361 0.3464286 0.7962675 0.7723005 0.9063361
## 15 0.8953168 0.3142857 0.8040435 0.7869249 0.8953168
## 16 0.9008264 0.3357143 0.7978227 0.7767221 0.9008264
## 17 0.9008264 0.3142857 0.8071540 0.7879518 0.9008264
## 18 0.9146006 0.3357143 0.8055988 0.7793427 0.9146006
## 19 0.9035813 0.3071429 0.8118196 0.7922705 0.9035813
## 20 0.9118457 0.3214286 0.8102644 0.7862233 0.9118457
## 21 0.9090909 0.3107143 0.8133748 0.7913669 0.9090909
## 22 0.9201102 0.3464286 0.8040435 0.7749420 0.9201102
## 23 0.9201102 0.3214286 0.8149300 0.7877358 0.9201102
## 24 0.9201102 0.3464286 0.8040435 0.7749420 0.9201102
## 25 0.9118457 0.3321429 0.8055988 0.7806604 0.9118457
```

For this data set, 5 appears to produce the model with the most accuracy. As the number of neighbors increases, the model is basically “remembering” the training set, as opposed to building a predictive model. As k increases, we will be overfitting. An independent test in the R console for k = 100 reported an accuracy of around 72%. However, as demonstrated in the table, an underfitted model is also produced for low (k < 5) values of k.

B.

For the Logistic Regression, SVM, Naïve Bayes, and Default models, the following metrics were calculated:

```
########
# Part 4B
########

# Calculate logistic regression metrics
logreg.result <- do_cv_class(wines, 10, "logreg")
get_metrics(logreg.result)
```

```
##         tpr       fpr       acc precision    recall
## 1 0.8484848 0.2678571 0.7978227 0.8041775 0.8484848
```

```
# Calculate SVM metrics
svm.result <- do_cv_class(wines, 10, "svm")
get_metrics(svm.result)
```

```
##         tpr       fpr       acc precision    recall
## 1 0.8650138 0.1928571 0.8398134 0.8532609 0.8650138
```

```
# Calculate Naive Bayes metrics
nb.result <- do_cv_class(wines, 10, "nb")
get_metrics(nb.result)
```

```
##         tpr       fpr       acc precision    recall
## 1 0.8787879 0.1964286 0.8460342 0.8529412 0.8787879
```

```
# Calculate Default Classifier
default.result <- get_pred_default(wines, wines)
get_metrics(default.result)
```

```
##   tpr fpr       acc precision recall
## 1   1   1 0.5645412 0.5645412      1
```

From these results, Naïve Bayes appeared to generate the model with the highest accuracy. The default model was accurate about 56% of the time, which is a little better than a coin flip type guess.

C.

When considering only accuracy, Naïve Bayes was the best model built for the Wines dataset. It also however had one of the lowest false positive rates, as well as a comparably high precision and recall.

Naïve Bayes does not suffer from the same performance issues as building an SVM, nor is it affected by a choice of an arbitrary k, as in our nearest neighbors function.