Supervised Learning

Data Overview

The Mushroom data set I chose came from the UCI Machine Learning Repository and was drawn from the Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf. This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. The definitely poisonous and unknown edibility classifications were combined to create a binary class variable. The data set contains 22 different attributes about the mushrooms, some of which I used in my analysis based on a number of factors.

The data cleansing process I used for the majority of my mushroom models included removing nulls and checking for singularity as these both could affect my ability to create an accurate model. The original mushroom data set contains 8,124 instances, but after removing rows with nulls there still are 5,644 instances that I was able to use in my models. I also removed the column veil.type as there was only one value in the column making it somewhat useless in including it in the analysis. This left me with 21 viable attributes to use in my models.

The UCI Credit Card data set I chose came from Kaggle. It includes a variety of attributes such as default payments, demographic factors, credit data, history of payment, and bill statements. The data set includes a total of 30,000 instances and 24 variables which I used in the creation of the five different models (there were no null values to remove in the original data set). The target variable is whether they have defaulted on their credit card payment.

Why are these data sets interesting?

There is no simple rule for determining the edibility of a mushroom; no rule like “leaflets three, let it be'” for Poisonous Oak and Ivy, so as a result, using this dataset for modeling is actually quite helpful in determining other ways to identify possibly poisonous mushrooms based off the mushrooms attributes.

The UCI Credit Card data set was interesting to me because this is a very common type of data set to use for classification models. It also has a lot of data which I thought would be a good thing to look into given the mushroom data set had below 10,000 rows.

Decision Tree

I began by loading and cleaning the mushroom data for the first decision tree model. This included removing nulls and column veil.type. I then split the data into training and testing sets, with 80% reserved for training and 20% reserved for testing. I followed this with checking the newly split sets to make sure they had an appropriate amount of the two different possible classes ‘e’ for edible and ‘p’ poisonous/possibly poisonous. The results of this are in Table 1 and show that in each split set there is approximately the same number of poisonous mushrooms to analyze at around 40%. I used the prop.table() function in R to verify that the randomization was correct.

	e	p
Train	0.6234773	0.3765227
Test	0.6076174	0.3923826

Table 1

I then plotted the decision tree using the rpart function in R from the rpart library. The resulting tree is shown in Figure 1. It is clear that there are very few variables in this first model that are important in

determining the splitting of the data for prediction. I then used the test set to predict on with the results of that shown in the confusion matrix in Table 2. The prediction results are quite good with only 2 False Negatives and 0 False Positives. The model performed with an accuracy of 0.9982285, which is extremely high.

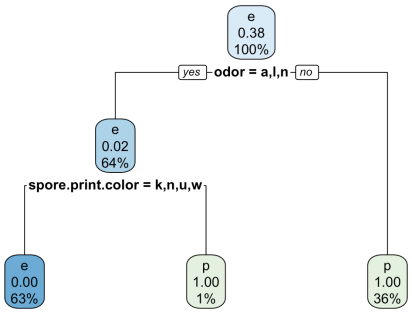

Figure 1

	e	**p**
e	686	0
p	2	441

Table 2

I then looked at the ROC curve and the AUC. The ROC curve in Figure 2 below shows again the high accuracy of the model as it is almost perfectly in the upper left-hand corner. The AUC of the test set was 0.9977427, also extremely high. This could be a possible indication of the model being overfit.

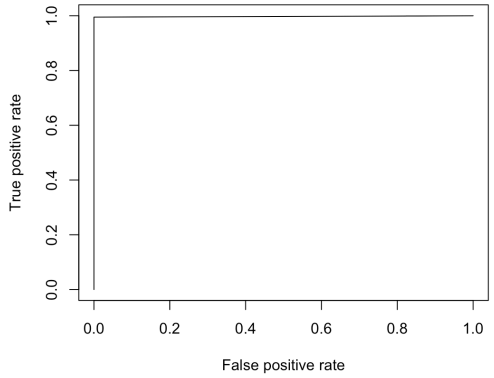

Figure 2

I then began the pruning process by printing the complexity parameter chart of the fitted model shown in Figure 3. The complexity parameter is used to control the decision tree size and select the optimal size. If the cost of adding another variable to the decision tree from the current node is above the value of cp, then the tree size should not change. As Figure 3 shows, the current decision tree model is already minimizing the error as much as possible and thus it does not suggest a change in the current number of splits. This can also be seen in the Figure 4 graph, which shows that pruning the tree back would not necessarily decrease the error.

```
> printcp(fit)

Classification tree:
rpart(formula = class ~ ., data = train, method = "class")

Variables actually used in tree construction:
[1] odor              spore.print.color

Root node error: 1700/4515 = 0.37652

n= 4515

        CP nsplit rel error    xerror      xstd
1 0.958824      0 1.0000000 1.0000000 0.0191508
2 0.035294      1 0.0411765 0.0411765 0.0048832
3 0.010000      2 0.0058824 0.0058824 0.0018581
```

Figure 3

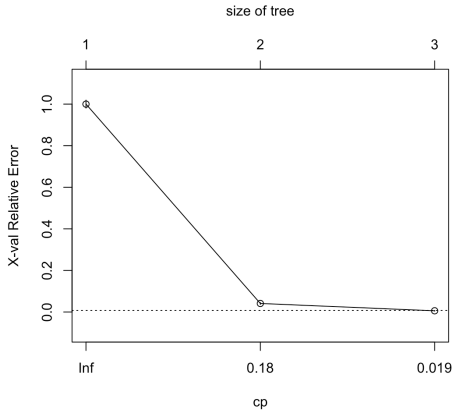

Figure 4

Pruning the tree to two splits to test for increased accuracy showed that indeed it does not improve the accuracy of the model and that three splits is the best tree size. The pruned tree is shown below in Figure 5 and the confusion matrix is in Table 3. The confusion matrix shows that with the pruned tree there are more False Negatives than with the non-pruned tree, another indication that pruning is not helpful.

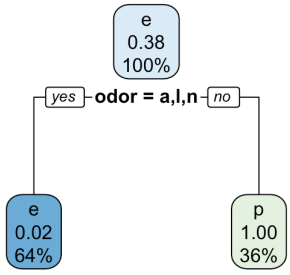

Figure 5

	e	p
e	686	0
p	11	432

Table 3

The accuracy of the pruned tree decreased, though only slightly, to 0.9902569 from the original 0.9982285. The ROC curve and AUC also declined slightly. The new pruned ROC curve is shown below in Figure 6 and can be seen further away from the upper left-hand corner than the non-pruned ROC curve graph. The AUC decreased from 0.9977427 to 0.9875847. These graphs and number would indicate that keep thing original non-pruned tree is the best for the mushroom dataset.

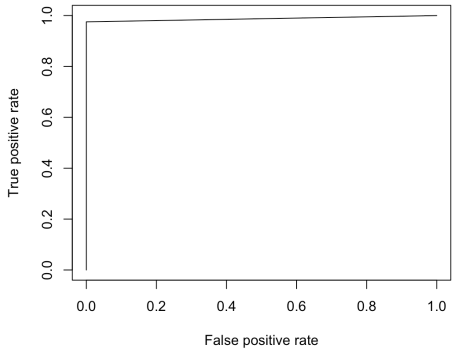

Figure 6

In a similar fashion as the mushroom decision tree, I began by loading and cleaning the UCI credit card data for the second decision tree model. This included searching for nulls or anything that would affect singularity, which I did not find to be the case (there were no nulls or columns with only 1 value). I then split the data into training and testing sets, with 80% reserved for training and 20% reserved for testing. I

followed this with checking the newly split sets to make sure they had an appropriate amount of the two different possible classes ‘0’ no default and ‘1’ for default. The results of this are in Table 4 and show that in each split set there is approximately the same number of defaults to analyze at around 20%. I used the prop.table() function in R to verify that the randomization was correct.

	0	**1**
Train	0.7784583	0.2215417
Test	0.7681667	0.2318333

Table 4

I then plotted the decision tree using the rpart function in R from the rpart library. The resulting tree is shown in Figure 7. It is clear that there are very few variables in this first model that are important in determining the splitting of the data for prediction. I then used the test set to predict on with the results of that shown in the confusion matrix in Table 5. The prediction results are okay but not great with 926 False Negatives and 188 False Positives. The model performed with an accuracy of 0.8143333, which is also only okay.

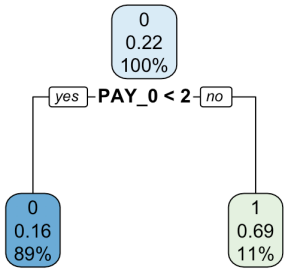

Figure 7

	0	**1**
0	4421	188
1	926	465

Table 5

I then looked at the ROC curve and the AUC. The ROC curve in Figure 8 below shows again the lower accuracy of this model in comparison to the model created with the mushroom data. The AUC of the test set was 0.6467511, also fairly low.

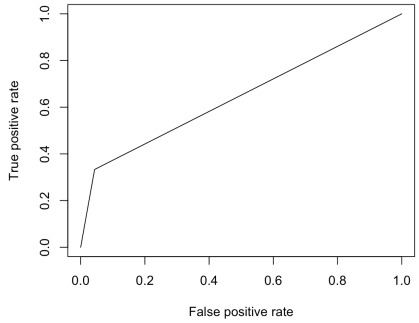

Figure 8

Given the fact that there is only one split in the current decision tree model, the possibility of pruning is not an option. This may indicate that for this particular dataset, decision tree models are not the best method to use for classification.

Neural Networks

The next model I created was the neural net model for the mushroom data set. This required a little more data cleanup than the previous decision tree model, as the data for the mushroom set is initially all in character data type. After the cleanup, I scaled the data using the min-max method, this is particularly

important for a neural network model. Also, since the typical y~. formula is not accepted in the neuralnet() function, I created the formula separately to use in the neural net that included all of the variables left in the mushroom data set after the cleansing process.
I choose to use 10 neurons in my hidden layer as this was something I found online to work well with larger datasets.

The neural network is shown printed in Figure 9 with the black lines showing the connections between each layer and the weights on each connection while the blue lines show the bias term added in each step. Given the difficulty of interpreting this graph, neural networks are often times considered black boxes when it comes to their predictive power.

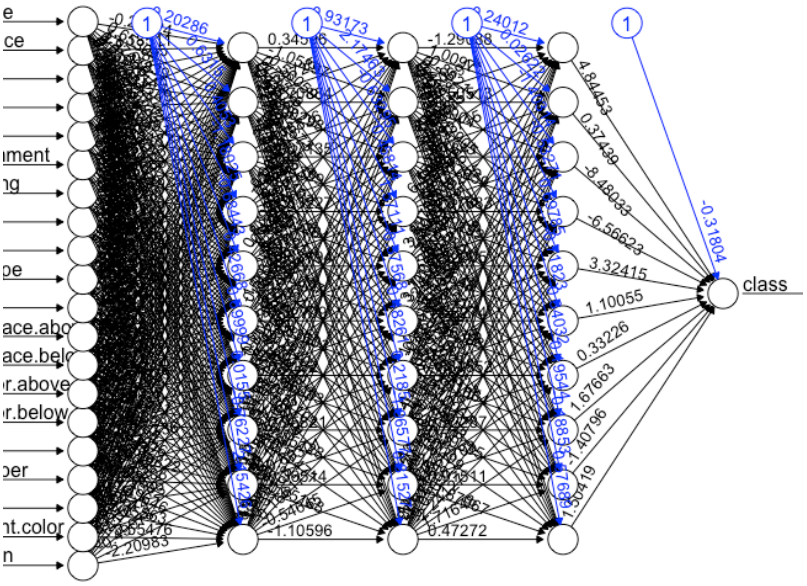

Figure 9

Next, I used the neural network to predict on the test data with the results displayed in the confusion matrix below in Table 6. This table shows great prediction results, with no False Negatives or False Positives.

	0	**1**
0	698	0
1	0	431

Table 6

I then created a neural network model using the UCI Credit Card dataset. After loading and cleaning the data set I had to take a subset of 2,000 rows as larger sample size would cause errors in the neural network model. So, in order to compare it to the mushroom sample, I sampled down to be able to run the model. The neural network is still extremely large as shown in Figure 10 below.

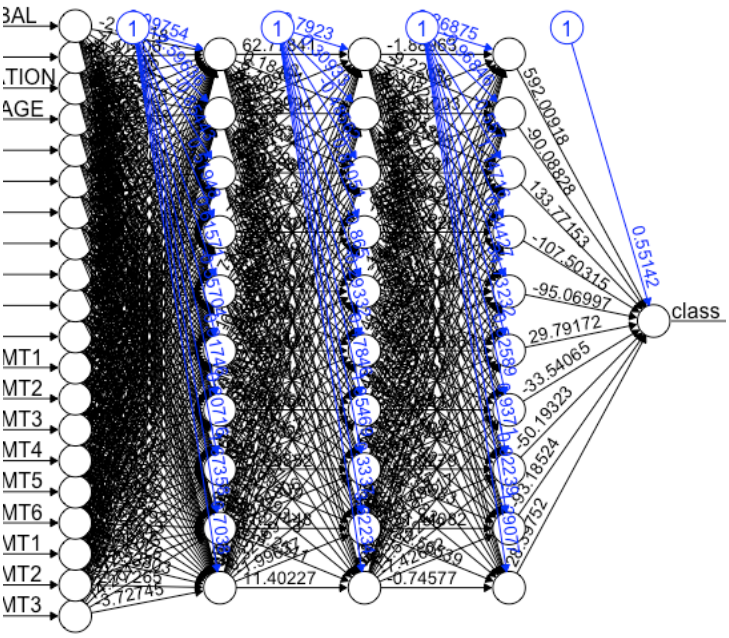

Figure 10

The model with the credit card data performed quite a bit worse than the model with the mushroom data. The overall accuracy was 76%. The confusion matrix is broken out below in Table 7. As shown, there are quite a few False Negatives and False Positives indicating that this neural network is not very good at classification, and definitely not as good as the neural network model for the mushroom data set.

	0	**1**
0	264	43
1	53	40

Table 7

Boosting

The next model I did was the Boosting model in which I used the GBM boosting method. I again started with the mushroom data set which I cleaned by removing nulls and the veil.type column. I then shuffled the data and created training and testing sets. The results of this are in Table 8 and show that in each split set there is approximately the same number of poisonous mushrooms to analyze at around 50%. I used the prop.table() function in R to verify that the randomization was correct.

	e	**p**
Train	0.5234651	0.4765349
Test	0.5341538	0.4658462

Table 8

I then fit the boosting model using the training data and a Bernoulli distribution. Seven of the variables were shown to have relative influence in the model: odor, gill.size, gill.color, spore.print.color, stalk.surface.above.ring, ring.type, and gill.spacing. These can be shown in the graph below (Figure 11).

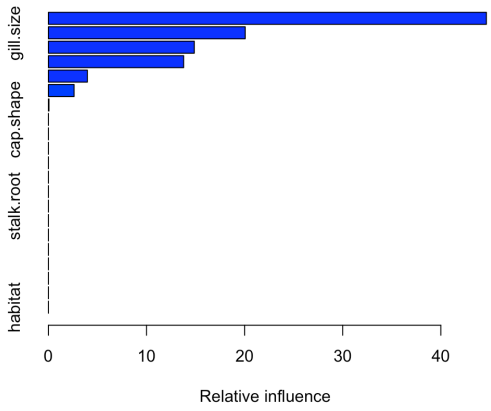

Figure 11

I then used the testing set to make predictions using the created boosting model. I considered a predication value greater than 0.5 to indicate a 1 (‘p’ poisonous) and 0.5 and below to indicate a prediction of 0 (‘e’ edible). The confusion matrix below in Table 9 indicates that the model does a fairly good job at classification of these. The accuracy value of 0.9710769 also indicates this for the model.

	0	**1**
0	850	18
1	29	728

Table 9

I then looked into fitting the boosting model to the UCI Credit Card data. I loaded and cleaned the data again. I then split the data into training and testing sets, with 80% reserved for training and 20% reserved for testing. I followed this with checking the newly split sets to make sure they had an appropriate amount of the two different possible classes ‘0’ no default and ‘1’ for default. The results of this were similar to the previous models with about 77% in the ‘0’ field and 22% in the ‘1’ field for testing and the same for training. After this, I fit the same boosting GBM model as I did for the mushroom data with the summary results below in Figures 12 and 13. All of the variables seemed to have some influence on the model’s predictive capabilities.

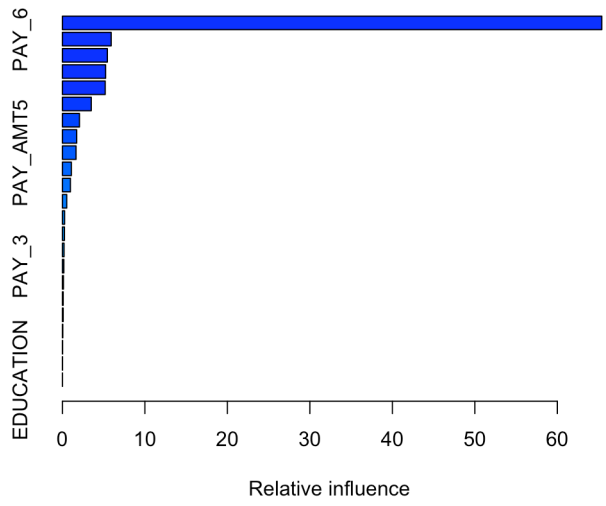

Figure 12

```
> summary(boost)
                var      rel.inf
PAY_0         PAY_0 65.390801016
PAY_6         PAY_6  5.911020287
PAY_2         PAY_2  5.448685806
PAY_5         PAY_5  5.239424306
PAY_4         PAY_4  5.188543168
PAY_AMT3   PAY_AMT3  3.496974107
PAY_AMT1   PAY_AMT1  2.069332307
PAY_AMT6   PAY_AMT6  1.727323841
PAY_AMT5   PAY_AMT5  1.650783169
PAY_AMT4   PAY_AMT4  1.085993787
LIMIT_BAL LIMIT_BAL  0.972642147
PAY_AMT2   PAY_AMT2  0.530480341
BILL_AMT4 BILL_AMT4  0.270776450
BILL_AMT2 BILL_AMT2  0.247049762
BILL_AMT5 BILL_AMT5  0.199634241
PAY_3         PAY_3  0.170882039
BILL_AMT1 BILL_AMT1  0.126050258
SEX             SEX  0.092633446
AGE             AGE  0.090306118
BILL_AMT6 BILL_AMT6  0.051180422
BILL_AMT3 BILL_AMT3  0.024159674
MARRIAGE   MARRIAGE  0.012190991
EDUCATION EDUCATION  0.003132315
```

Figure 13

I then used the testing set to make predictions using the created boosting model. I considered a predication value greater than 0.5 to indicate a 1 (default) and 0.5 and below to indicate a prediction of 0 (no default). The confusion matrix below in Table 10 indicates that the model does a fairly good job at classification of these. The accuracy value of 0.8116667 is also decent, though not as good as the accuracy for the mushroom data set.

	0	**1**
0	894	46
1	180	80

Table 10

Support Vector Machines

The next model I created was the SVM model for the mushroom data set. As before, I cleaned the data and got it ready for the model input. I then created a simple svm model that had the following timing outputs (Table 11):

User	**System**	**Elapsed**
0.122	0.001	0.124

Table 11

I then created predictions on the dataset where if the predicted value was greater than 0.5 it would be a 1 (‘p’ poisonous) and 0.5 and below to indicate a prediction of 0 (‘e’ edible). Oddly the prediction accuracy for this model was a perfect 1 with the categorizations having no False Positives or False Negatives as shown in the confusion matrix below as Table 12.

	0	1
0	3488	0
1	0	2156

Table 12

I then looked into tuning the mushroom svm model. This showed that the best cost was 10 and the best gamma value was 0.5 (shown in Figure 14). I re-ran the model with these parameters and the radial kernel to see if performance would improve at all.

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
   10   0.5

- best performance: 0.002919951
```

Figure 14

The time outputs definitely increased when I used these new parameters as shown in Table 13. However, the confusion matrix did not change at all leading me to believe that the ideal parameters were already being used in the original model. The accuracy also did not change from 100%.

User	System	Elapsed
0.724	0.007	0.763

Table 13

I then created an SVM model for the UCI Credit Card data set after loading and cleaning it. This produced the following timing outputs (Table 14):

User	System	Elapsed
0.884	0.008	0.920

Table 14

I then made predictions using the created svm model. I considered a predication value greater than 0.5 to indicate a 1 (default) and 0.5 and below to indicate a prediction of 0 (no default). The confusion matrix below in Table 15 indicates that the model does an okay job at classification of these. The accuracy value of 0.8251667 is also decent, though not as good as the accuracy for the mushroom data set, which seems to be the case for all of the models created thus far.

	0	1
0	4472	194
1	855	479

Table 15

After this, I looked into the tuning of the SVM model using the credit card data. This also took quite a while to run, as did the mushroom data set tuning. This showed that the best cost was 1 and the best gamma value was 0.5 (shown in Figure 15).

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    1   0.5

- best performance: 0.1593001
```

Figure 15

The system times of the new model run with the new parameters increased as shown in Table 16.

User	System	Elapsed
1.307	0.010	1.334

Table 16

I then created predictions using this newly tuned model on the credit card data set which are shown below in Table 17. This model is definitely superior to the first one as the number of False Positives and False Negatives decreased to give an overall accuracy of 90.433%, which is a pretty big improvement from 82.5% pre-tuning. From this example, tuning can be shown to be something to use with SVM that can increase accuracy of your models.

	0	1
0	4616	50
1	524	810

Table 17

k-nearest neighbors

The final model I created was the k-nearest neighbors model. I began with the mushroom data set which I loaded, cleaned and shuffled before changing the variable types to numeric for the purposes of this model. An important part in knn modeling is normalizing numeric data types, which I did using a normalize function that I then applied to all the variables (except the target class variable). I then split the data into the usual 80/20 training/testing split I have been using and created the two different datasets. I then created a knn model initially using a k value of 10. I chose this given the rather large nature of the data set. It ended up creating a "perfect" model with 100% accuracy, making me question whether the model is overfitted or something else is affecting the results. Using the gmodels library I evaluated the model as shown in Figure 16 below. This represents the confusion matrix for the model, which shows 0 False Negative and 0 False Positives for an accuracy of 100%.

```
Total Observations in Table:  1129


             | data_test_pred
 test_labels |         e |         p | Row Total |
-------------|-----------|-----------|-----------|
           e |       672 |         0 |       672 |
             |     1.000 |     0.000 |     0.595 |
             |     1.000 |     0.000 |           |
             |     0.595 |     0.000 |           |
-------------|-----------|-----------|-----------|
           p |         0 |       457 |       457 |
             |     0.000 |     1.000 |     0.405 |
             |     0.000 |     1.000 |           |
             |     0.000 |     0.405 |           |
-------------|-----------|-----------|-----------|
Column Total |       672 |       457 |      1129 |
             |     0.595 |     0.405 |           |
-------------|-----------|-----------|-----------|
```

Figure 16

I tested several different k values including 3, 5, 20, and 30. It seemed that anything below 10 would keep the same results of a perfectly accurate model. However, the higher numbers did produce some False Negatives and False Positives. For example, k = 30 had 2 False Negative and 2 False Positives, as shown in Figure 17, for an overall accuracy of 99.6%, still very high. This indicates to me that knn may not be the best type of model for this data.

```
Total Observations in Table:  1129


             | data_test_pred3
 test_labels |         e |         p | Row Total |
-------------|-----------|-----------|-----------|
           e |       670 |         2 |       672 |
             |     0.997 |     0.003 |     0.595 |
             |     0.997 |     0.004 |           |
             |     0.593 |     0.002 |           |
-------------|-----------|-----------|-----------|
           p |         2 |       455 |       457 |
             |     0.004 |     0.996 |     0.405 |
             |     0.003 |     0.996 |           |
             |     0.002 |     0.403 |           |
-------------|-----------|-----------|-----------|
Column Total |       672 |       457 |      1129 |
             |     0.595 |     0.405 |           |
-------------|-----------|-----------|-----------|
```

Figure 17

I then created a knn model for the UCI Credit Card data set. I took a sample of the data set to make it easier to work with and then changed the necessary factors to numeric for the purposes of the knn model. I then normalized these factors and created testing and training sets based on a 80/20 split. From here I created the first knn model with k = 10 as I did with the mushroom data set. The resulting model gave me an accuracy of about 79.666%, which is not terrible, but is definitely worse than the performance of the mushroom data set. The output of the model's predictions can be seen in Figure 18.

```
Total Observations in Table:  1200


             | data_test_pred
 test_labels |         0 |         1 | Row Total |
-------------|-----------|-----------|-----------|
           0 |       889 |        47 |       936 |
             |     0.950 |     0.050 |     0.780 |
             |     0.819 |     0.412 |           |
             |     0.741 |     0.039 |           |
-------------|-----------|-----------|-----------|
           1 |       197 |        67 |       264 |
             |     0.746 |     0.254 |     0.220 |
             |     0.181 |     0.588 |           |
             |     0.164 |     0.056 |           |
-------------|-----------|-----------|-----------|
Column Total |      1086 |       114 |      1200 |
             |     0.905 |     0.095 |           |
-------------|-----------|-----------|-----------|
```

Figure 18

From here I recreated the model using different values of k to try to see if I could improve the accuracy at all. As with the mushroom data set, I tested 3, 5, 20, and 30 as values for k to see if the results would improve at all. I was able to improve the accuracy by a small amount when I increased the values of k. IN Figure 19 below, you can see the output for k=30 which gave an accuracy of about 80.25%.

```
Total Observations in Table:  1200


             | data_test_pred3
 test_labels |         0 |         1 | Row Total |
-------------|-----------|-----------|-----------|
           0 |       903 |        33 |       936 |
             |     0.965 |     0.035 |     0.780 |
             |     0.817 |     0.347 |           |
             |     0.752 |     0.028 |           |
-------------|-----------|-----------|-----------|
           1 |       202 |        62 |       264 |
             |     0.765 |     0.235 |     0.220 |
             |     0.183 |     0.653 |           |
             |     0.168 |     0.052 |           |
-------------|-----------|-----------|-----------|
Column Total |      1105 |        95 |      1200 |
             |     0.921 |     0.079 |           |
-------------|-----------|-----------|-----------|
```

Figure 19

Conclusion

Overall, the mushroom data set consistently performed better than the UCI Credit Card data set for every model. The mushroom data set seems to commonly be used online as a solid example data set for modeling, which leads me to believe the data lends itself to modeling and prediction. The UCI Credit Card data on the other hand was something I found on Kaggle, a site known for modeling competitions. Thus, this data set may be more of a challenging set to work with in order for modelers to develop more complex algorithms to fit the data. It would seem that the best method for the Credit Card data set would be to use the tuned SVM model which gave a 90% accuracy rate. The models for the mushroom data set all performed almost equal well.