Tyler Bryk
CPE-695 Applied Machine Learning
Homework 3

Problem 1:
Please answer the following questions related to Machine Learning concepts:

1. Explain what is the bias-variance trade-off? Describe few techniques to reduce bias and variance respectively.

Before jumping into the trade-off of bias vs. variance, lets first explain what bias and variance are in a machine learning model. Bias is caused when the model pays little attention to the training data and ultimately has an average prediction that is far away from the truth; this is known as under-fitting. On the other hand, variance is caused by the model paying very close attention to the training data and then not being able to generalize the model to other data points; this is known as overfitting. Since a model cannot be both underfit and overfit at the same time, a trade-off is then created. Ultimately, every model should seek a good balance where the model has as few parameters and complexities as possible, but still provides accurate predictions on new data. Hence, being neither underfit, not overfit, but somewhere in the middle.

The easiest way to reduce the bias of a model is to increase the hypothesis space by adding complexity to the model. Doing this essentially helps the model pay closer attention to the training data so that the model is more accurate. One common technique for decreasing variance is resampling. This can be done by using various techniques such as bagging. In the bagging method, such as a random forest tree, the tress can be averaged together which ultimately decreases the variance without increasing the bias. Unfortunately, there are no analytical methods to finding the optimal bias-variance trade-off, only empirical methods can be used which observe the model complexity over time.

2. What is k-fold cross-validation? Why do we need it?

K-fold Cross Validation is a type of cross validation where the dataset is split into 'k' subsets and then used to test a machine learning model. Assuming we pick 'k' folds, then the model will be iterated through 'k' times. On the first iteration, the first of the 'k' number of folds will be used to test the model's accuracy while all of the other folds are used for training. Then on the second iteration, the second fold will be used to test, and the others will be used for training. This process will continue for 'k' iterations, until each of the 'k' folds has the opportunity to be used for testing. One of the main reasons why K-fold CV is used is because it allows each observation to be used for both training and testing. This is especially important in applications where the input data or number of observations are limited because it allows the model an opportunity to learn off of every possible data point. In general, having more data is aways a good thing because it gives more opportunities to train the models, optimize parameters, and gather metrics. Thus, using cross validation is important because it ultimately gives the model more data to test and train off of.

Tyler Bryk

CPE-659 HW3

1.) See text response above.

2.) Compute Precision, Recall, F1-Score

	C1	C2
C1	50	30
C2	40	60

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{50}{50+40} = \boxed{5/9 \text{ or } 0.\overline{555}}$$

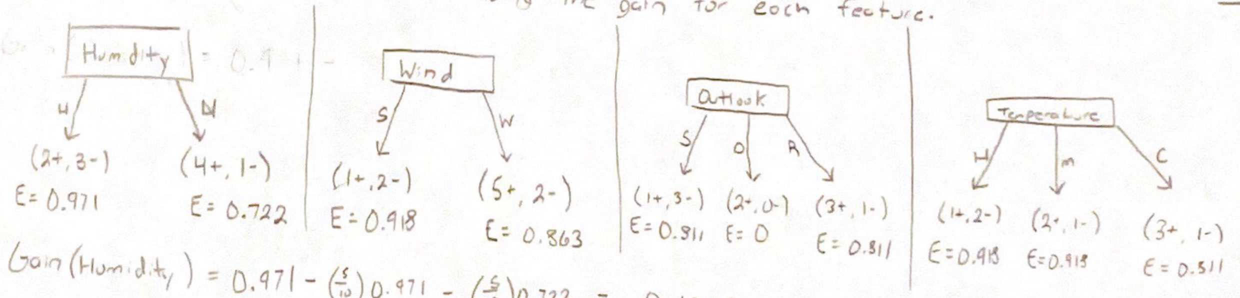
$$\text{Recall} = \frac{TP}{TP+FN} = \frac{50}{50+30} = \boxed{5/8 \text{ or } 0.625}$$

$$F_1\text{-Score} = \frac{2PR}{P+R} = \frac{2(5/9)(5/8)}{(5/9)+(5/8)} = \boxed{23/17 \text{ or } 1.3529}$$

3.) Build a decision tree using Information Gain:

For this dataset, we have 4 features and 2 classes. $E(6+, 4-) = -\frac{6}{10}\text{Log}(\frac{6}{10}) - \frac{4}{10}\text{Log}(\frac{4}{10}) = \underline{0.971}$

First, we determine the root by finding the gain for each feature.



$\text{Gain (Humidity)} = 0.971 - (\frac{5}{10})0.971 - (\frac{5}{10})0.722 = 0.1245$

$\text{Gain (Wind)} = 0.971 - (\frac{3}{10})0.918 - (\frac{7}{10})0.863 = 0.0915$

* $\text{Gain (Outlook)} = 0.971 - (\frac{4}{10})0.811 - (\frac{2}{10})0 - (\frac{4}{10})0.811 = 0.3222$

$\text{Gain (Temp)} = 0.971 - (\frac{3}{10})0.918 - (\frac{3}{10})0.918 - (\frac{4}{10})0.811 = 0.0958$

Here, Outlook gives the greatest information gain, therefore we will select this feature as the root of the tree. We can already select 'Yes' for Outlook = Overcast. We will now repeat to find the best choice for Sunny / Rain.

Outlook == Sunny: $E(1+, 3-) = 0.811$

$\text{Gain}(S_{Sunny}, \text{Humidity}) = 0.811 - \frac{1}{4}0 - \frac{3}{4}0 = 0.811$

$\text{Gain}(S_{Sunny}, \text{Temp}) = 0.811 - \frac{2}{4}0 - \frac{1}{4}0 - \frac{1}{4}0 = 0.811$

$\text{Gain}(S_{Sunny}, \text{Wind}) = 0.811 - \frac{1}{4}(0) - \frac{3}{4}0.918 = 0.1225$

Here, either Humidity or Temp can be selected since their information gains are esual.

Outlook == Rain: $E(3+, 1-) = 0.811$

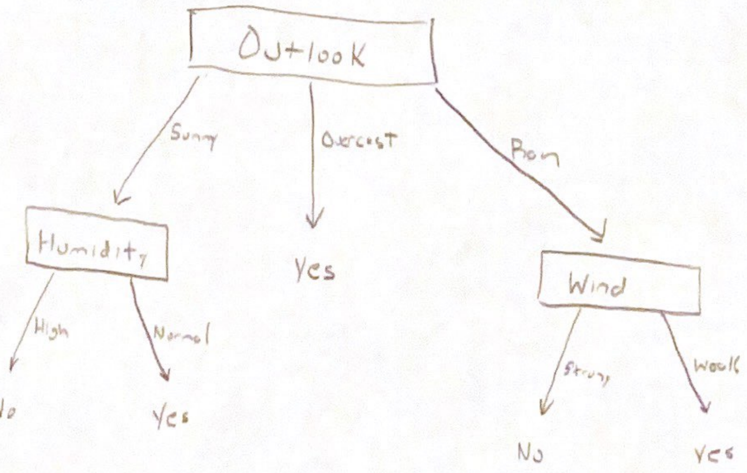
$\text{Gain}(S_{Rain}, \text{Humidity}) = 0.811 - \frac{1}{4}(0) - \frac{3}{4}(0.918) = 0.1225$

$\text{Gain}(S_{Rain}, \text{Temp}) = 0.811 - \frac{2}{4}(0) - \frac{2}{4}(1) = 0.311$

$\text{Gain}(S_{Rain}, \text{Wind}) = 0.811 - \frac{3}{4}(0) - \frac{1}{4}(0) = 0.811$

We will select the wind attribute for Outlook == Rain and the humidity attribute for outlook == Sunny.

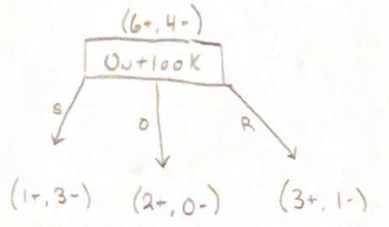
Since every decision can be represented, and the training examples all have the same target attribute value, the tree is completed.



3.)

B.) Decide the P-value of the root node using Chi^2 Test

Root Node:



$\hat{P}_1 = \frac{6}{6+4}(4) = \underline{2.4}$ $\quad \hat{P}_2 = \left(\frac{6}{10}\right)(2) = \underline{1.2}$ $\quad \hat{P}_3 = \underline{2.4}$

$\hat{N}_1 = \frac{4}{6+4}(4) = \underline{1.6}$ $\quad \hat{N}_2 = \left(\frac{4}{10}\right)(2) = \underline{0.8}$ $\quad \hat{N}_3 = \underline{1.6}$

$$Q = \sum_{i=1}^{3} \frac{(P_i - \hat{P}_i)^2}{\hat{P}_i} + \frac{(N_i - \hat{N}_i)^2}{\hat{N}_i}$$

$$Q = \frac{(1-2.4)^2}{2.4} + \frac{(3-1.6)^2}{1.6} + \frac{(2-1.2)^2}{1.2} + \frac{(0-0.8)^2}{0.8} + \frac{(3-2.4)^2}{2.4} + \frac{(1-1.6)^2}{1.6} = 3.75$$

Here, the degrees of freedom are $3-1 = 2$

Using the online tool, $\boxed{P_{\text{Chance}} = P(\chi^2 > 3.75) = 0.15 \text{ or } 15\%}$

4.) Use Naive Bayes Fusion method to predict the final decision:

We will use the formula $\mu_j(x) \propto \prod_{i=1}^{L} \hat{P}(\omega_j \mid d_{i,j}(x) = 1)$ to find the probability of voting for either Class 1 or Class 2.

Class 1:

$P(\omega_1 \mid d_{1,1}) = 40/70$

$P(\omega_1 \mid d_{2,1}) = 20/40$

$P(\omega_1 \mid d_{3,2}) = 0/10$

Class 2:

$P(\omega_2 \mid d_{1,1}) = 30/70$

$P(\omega_2 \mid d_{2,1}) = 20/40$

$P(\omega_2 \mid d_{3,2}) = 10/10$

$P(\text{Class } 1) = \left(\frac{40}{70}\right) * \left(\frac{20}{40}\right) * \left(\frac{0}{10}\right) = 0$

$\boxed{P(\text{Class } 2) = \left(\frac{30}{70}\right) * \left(\frac{20}{40}\right) * \left(\frac{10}{10}\right) = 0.21}$

Using Naive Bayes Fusion, this ensemble will choose Class 2 as a final decision because the probability is higher.

CPE695 HW3

By: Tyler Bryk

In this assignment, we will explore the titanic dataset and create a decision tree to predict the survival rate of passengers on board.

Step 1: Read in Titanic.csv and observe a few samples, some features are categorical and others are numerical. Take a random 80% samples for training and the rest 20% for test.

In this step, we load our Titanic.csv dataset into a Pandas DataFrame. First, we convert values in the sex column to be numerical, female->0 and male->1, then we do the same for the pclass column, 1st->1, 2nd->2, 3rd->3. The age column contains several missing values, so we will impute those missing instances with the median value for the age column. Lastly, we will split our dataset into 80% training data, and 20% testing data.

```
# Import Libraries
import pandas as pd
import graphviz
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier, plot_tree, export_graphviz
from sklearn.metrics import confusion_matrix, plot_confusion_matrix, accuracy_score
```

```
# Load Data and Split 80:20
data = pd.read_csv('Titanic.csv', index_col=0)
data['sex'].replace(['female','male'],[0,1], inplace=True)
data['pclass'].replace(['1st','2nd','3rd'],[1,2,3], inplace=True)
data['age'] = data['age'].fillna(data['age'].median())
x = data[['pclass', 'sex', 'age', 'sibsp']]
y = data['survived']
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.2, random_state=1
```

Decision Tree Modeling

Step 2: Fit a decision tree model using independent variables ‘pclass + sex + age + sibsp’ and dependent variable ‘survived’. Then plot the full tree.

In this step, we fit a decision tree classifier with the training data. Our classifier uses the default parameters specified by the sklearn library. Afterwards, the initial tree is plotted and saved to a DT-Initial.png file. The tree is also illustrated below in the notebook. Based on the visualization of the tree, it can be observed that the tree depth is 15, which is very clunky. It is also important to remember that the sex class has been split into numerical values where female->0 and male->1, so for the root node of our tree, females are to the left section, and males are to the right. Despite this tree being very large, it seems to make logical sense, and it also agrees with the example tree that the professor illustrated in the homework assignment. In the next step, we will determine the accuracy of the tree, and then prune the tree to a smaller depth.

```
# Create a Decision Tree with Default Parameters
clf = DecisionTreeClassifier(random_state=10413641).fit(xTrain, yTrain)
```

```
# Plot the Decision Tree using Graphviz
dot_data = export_graphviz(clf, filled=True, feature_names = ['pclass','sex','age',
graph = graphviz.Source(dot_data, format="png")
graph.render("DT-Initial")
```

Sampling Errors

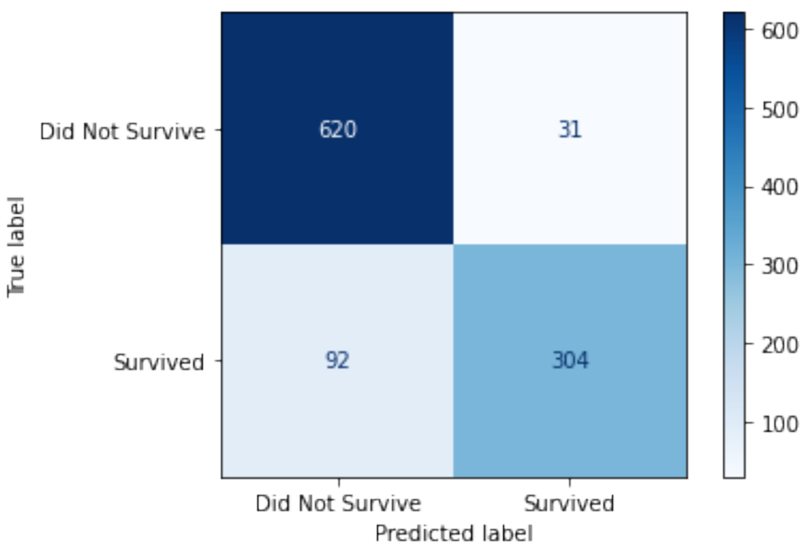
Step 3: Print out the performance measures of the full model:

- In-sample percent survivors correctly predicted (on training set)
- In-sample percent fatalities correctly predicted (on training set)
- Out-of-sample percent survivors correctly predicted (on test set)
- Out-of-sample percent fatalities correctly predicted (on test set)

In this step, we calculate the in and out-of-sample accuracy rates. After training the model on the training data, we then test the model on the training data again (In-sample) and then on the testing data (Out-of-sample). The confusion matrix for each trial is displayed below. The in-sample test gave a very promising overall accuracy of roughly 88% where the positive class was correctly predicted 76% and the negative class was correctly predicted 95% of the time. While the in-sample accuracy is very good, the out-of-sample accuracy was roughly 10% lower across the board, indicating that our decision tree model is overfit. Knowing that our model is overfit, we will now try to prune the tree and find the optimal parameters. We will then test the model on these performance metrics again, and hope that the testing accuracy closer resembles the training accuracy.

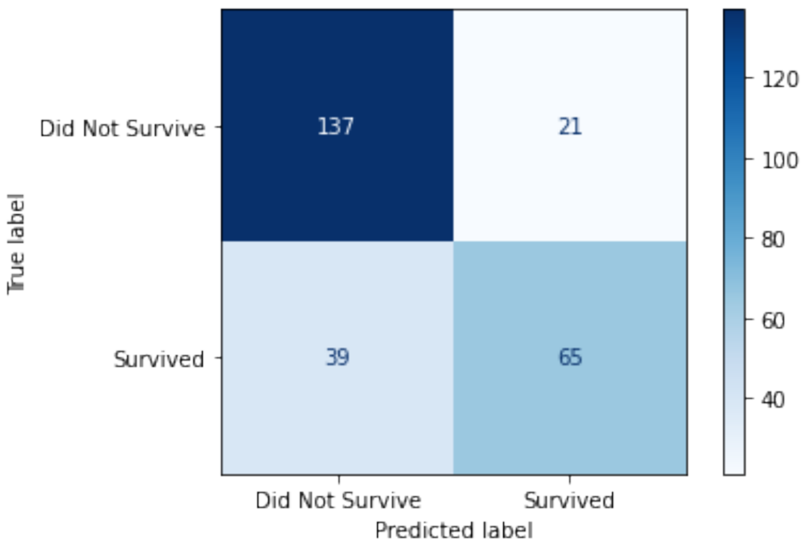
```
# In-Sample Error
insamplePreds = clf.predict(xTrain)
tn, fp, fn, tp = confusion_matrix(yTrain, insamplePreds).ravel()
print('Percent Survivors Correctly Predicted:\t', 100*(tp / (tp + fn)), '%')
print('Percent Fatalities Correctly Predicted:\t', 100*(tn / (tn + fp)), '%')
print('Overall In-Sample Accuracy:\t\t', 100*accuracy_score(insamplePreds, yTrain),
plot_confusion_matrix(clf, xTrain, yTrain, cmap=plt.cm.Blues, display_labels = ['Di
plt.show()
```

```
    Percent Survivors Correctly Predicted:   76.76767676767676 %
    Percent Fatalities Correctly Predicted:  95.23809523809523 %
    Overall In-Sample Accuracy:              88.25214899713467 %
```



```
# Out-of-Sample Error
outsamplePreds = clf.predict(xTest)
tn, fp, fn, tp = confusion_matrix(yTest, outsamplePreds).ravel()
print('Percent Survivors Correctly Predicted:\t', 100*(tp / (tp + fn)), '%')
print('Percent Fatalities Correctly Predicted:\t', 100*(tn / (tn + fp)), '%')
print('Overall Out-of-Sample Accuracy:\t\t', 100*accuracy_score(outsamplePreds, yTe
plot_confusion_matrix(clf, xTest, yTest, cmap=plt.cm.Blues, display_labels = ['Did
plt.show()
```

```
Percent Survivors Correctly Predicted:   62.5 %
Percent Fatalities Correctly Predicted:  86.70886075949366 %
Overall Out-of-Sample Accuracy:          77.09923664122137 %
```



Cross Validation

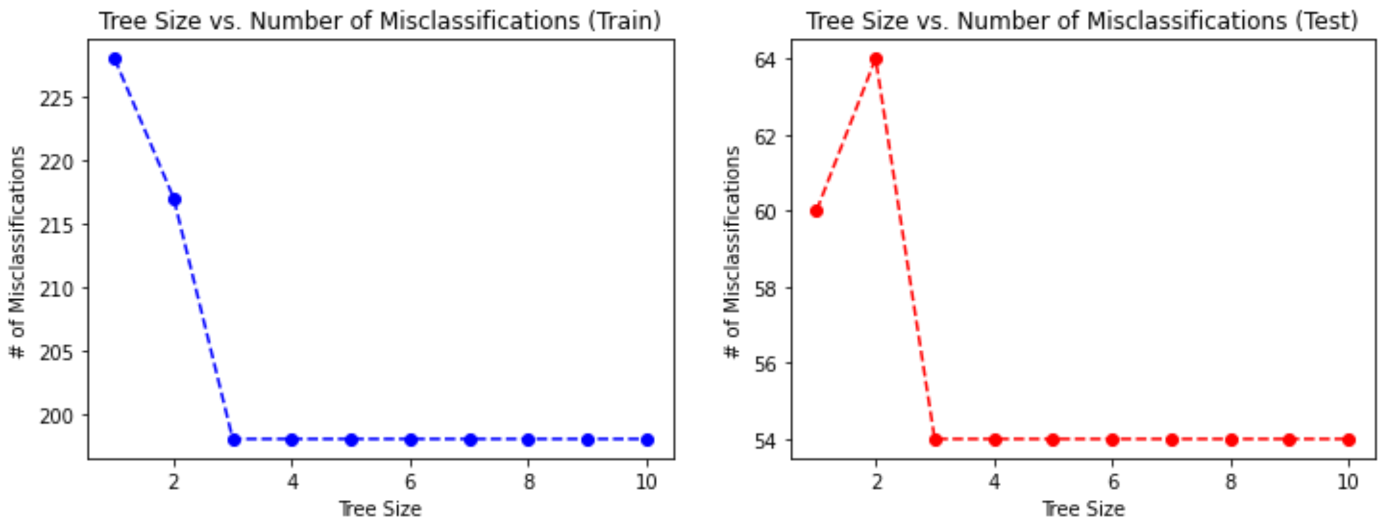
Step 4: Use cross-validation to find the best parameter to prune the tree. You should be able to plot a graph with the 'tree size' as the x-axis and 'number of misclassification' as the Y-axis. Find the minimum number of misclassification and choose the corresponding tree size to prune the tree.

In this step, we use cross validation to prune the tree, and find the optimal model parameters. We will use sklearn's gridsearchCV method to opitimize our model. A k-fold CV method with 5 folds was selected because it yielded the highest overall accuracy across samples where 5, 10, 15 folds were sampled. The parameters being tested for sklearn's decision tree classifier are displayed in output below, along with each of their optimal values. For our tree, it appears that the optimal tree depth is between 3 and 4 nodes, and the max number of leaf nodes should be 8. The tree depth results vary, the gridsearch suggests 4, however the graph of tree size vs. error suggest a depth of 3. We will try both depths on our final pruned model, and we will select whichever depth gives the highest sample accuracy. Looking ahead, a max depth of four was the ideal size.

```
# Use 5-Fold Cross Validation to Tune Model Parameters

params = [{ 'max_depth':[2,3,4,5,6],
            'max_leaf_nodes':[None,2,3,4,5,6,7,8,9,10]  }]

gridsearchDT05 = GridSearchCV(clf, params, cv= 5).fit(xTrain,yTrain)
gridsearchDT10 = GridSearchCV(clf, params, cv=10).fit(xTrain,yTrain)
gridsearchDT15 = GridSearchCV(clf, params, cv=15).fit(xTrain,yTrain)

print("Accuracy for  5-folds: {}".format(gridsearchDT05.score(xTest,yTest)))
print("Accuracy for 10-folds: {}".format(gridsearchDT10.score(xTest,yTest)))
print("Accuracy for 15-folds: {}".format(gridsearchDT15.score(xTest,yTest)))

print("Optimal Parameters with 5-fold: {}".format(gridsearchDT05.best_params_))
```

```
    Accuracy for  5-folds: 0.7938931297709924
    Accuracy for 10-folds: 0.7633587786259542
    Accuracy for 15-folds: 0.7633587786259542
    Optimal Parameters with 5-fold: {'max_depth': 4, 'max_leaf_nodes': 8}
```

```
# Plot Tree Size vs. Misclassification Rate
misclassTr = []
misclassTe = []
for node in range(10):
  c = DecisionTreeClassifier(max_depth=(node+1), max_leaf_nodes=8).fit(xTrain,yTrai
  predsTr = c.predict(xTrain)
  predsTe = c.predict(xTest)
  tnr, fpr, fnr, tpr = confusion_matrix(yTrain, predsTr).ravel()
  tne, fpe, fne, tpe = confusion_matrix(yTest,  predsTe).ravel()
  misclassTr.append(fpr+fnr)
  misclassTe.append(fpe+fne)
plt.figure(figsize=(12,4))
plt.subplot(121)
plt.xlabel("Tree Size")
plt.ylabel("# of Misclassifications")
plt.title("Tree Size vs. Number of Misclassifications (Train)")
plt.plot(range(1,11),misclassTr, linestyle='--', marker='o', color='b')
plt.subplot(122)
plt.xlabel("Tree Size")
plt.ylabel("# of Misclassifications")
plt.title("Tree Size vs. Number of Misclassifications (Test)")
plt.plot(range(1,11),misclassTe, linestyle='--', marker='o', color='r')
plt.show()
```



Pruning the Tree

Step 5: Prune the tree with the optimal tree size and plot the pruned tree.

```
# Create a Pruned Decision Tree with Max Depth 4 and Max Leaf Nodes 8
pclf = DecisionTreeClassifier(max_depth=4, max_leaf_nodes=8, random_state=10413641)


# Plot the Pruned Decision Tree using Graphviz
dot_data = export_graphviz(pclf, filled=True, feature_names = ['pclass','sex','age'
graph = graphviz.Source(dot_data, format="png")
graph.render("DT-Pruned")
```

Pruned Sampling Errors

Step 6: For the final pruned tree, report its in-sample and out-of-sample accuracy, defined as:

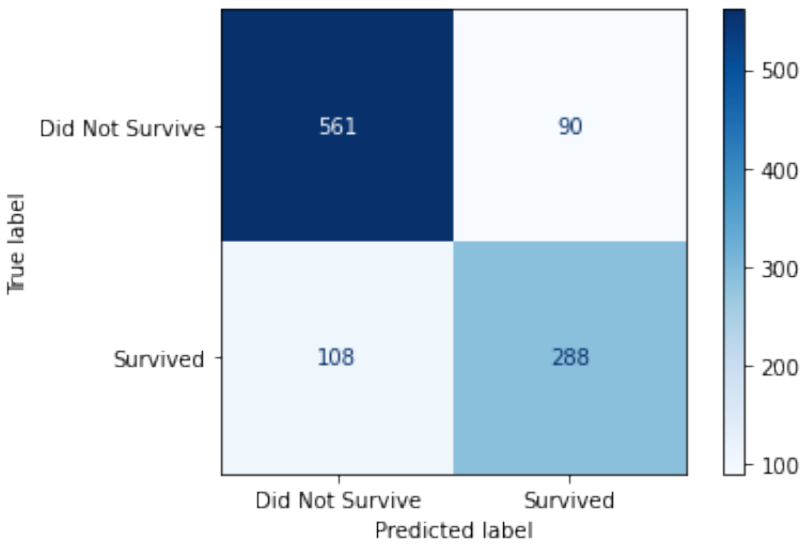
- In-sample percent survivors correctly predicted (on training set)
- In-sample percent fatalities correctly predicted (on training set)
- Out-of-sample percent survivors correctly predicted (on test set)
- Out-of-sample percent fatalities correctly predicted (on test set)

Check whether there is improvement in out-of-sample for the full tree (bigger model) and the pruned tree (smaller model).

Overall, pruning the decision tree helped imporve the out-of-sample accuracy of the model. If we look at the accuracies for the initial full tree: 88% In-sample, 77% Out-of-sample, we noticed that the training accuracy was faily high, and the model was clearly overfit because of the large gap between training and testing accuracy. Thankfully, this was no longer the case in the pruned model. Looking at the new performance: 81% In-sample, 79% Out-of-sample, we first notice that the testing accuracy increased by a modest 2%, but more importantly the difference between training and testing accuracy is very close, which indicates that the model is no longer overfit.

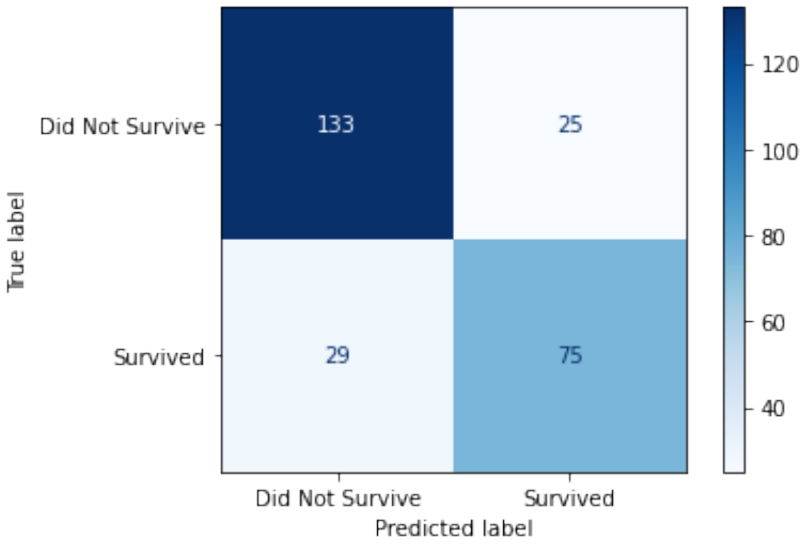
```
# In-Sample Error
insamplePreds = pclf.predict(xTrain)
tn, fp, fn, tp = confusion_matrix(yTrain, insamplePreds).ravel()
print('Percent Survivors Correctly Predicted:\t', 100*(tp / (tp + fn)), '%')
print('Percent Fatalities Correctly Predicted:\t', 100*(tn / (tn + fp)), '%')
print('Overall In-Sample Accuracy:\t\t', 100*accuracy_score(insamplePreds, yTrain),
plot_confusion_matrix(pclf, xTrain, yTrain, cmap=plt.cm.Blues, display_labels = ['D
plt.show()
```

```
Percent Survivors Correctly Predicted:   72.72727272727273 %
Percent Fatalities Correctly Predicted:  86.17511520737328 %
Overall In-Sample Accuracy:              81.08882521489971 %
```



```
# Out-of-Sample Error
outsamplePreds = pclf.predict(xTest)
tn, fp, fn, tp = confusion_matrix(yTest, outsamplePreds).ravel()
print('Percent Survivors Correctly Predicted:\t', 100*(tp / (tp + fn)), '%')
print('Percent Fatalities Correctly Predicted:\t', 100*(tn / (tn + fp)), '%')
print('Overall Out-of-Sample Accuracy:\t\t', 100*accuracy_score(outsamplePreds, yTe
plot_confusion_matrix(pclf, xTest, yTest, cmap=plt.cm.Blues, display_labels = ['Dic
plt.show()
```

```
Percent Survivors Correctly Predicted:   72.11538461538461 %
Percent Fatalities Correctly Predicted:  84.17721518987342 %
Overall Out-of-Sample Accuracy:          79.38931297709924 %
```


Did Not Survive
Survived
True label
Did Not Survive
Survived
Predicted label