Problem_Sheet_2

February 4, 2020

[2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import numpy as np
import seaborn as sns
```

[3]:
```
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
```

[64]:
```
from sklearn.model_selection import train_test_split

# Linear Regression, Lasso, Ridge

from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

# SVM, Logistic Regression, Random Forrest
from sklearn import svm
from sklearn import metrics
from sklearn.linear_model import LogisticRegressionCV, LogisticRegression
from sklearn.metrics import roc_curve, auc, roc_auc_score, accuracy_score,␣
↪confusion_matrix
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
```

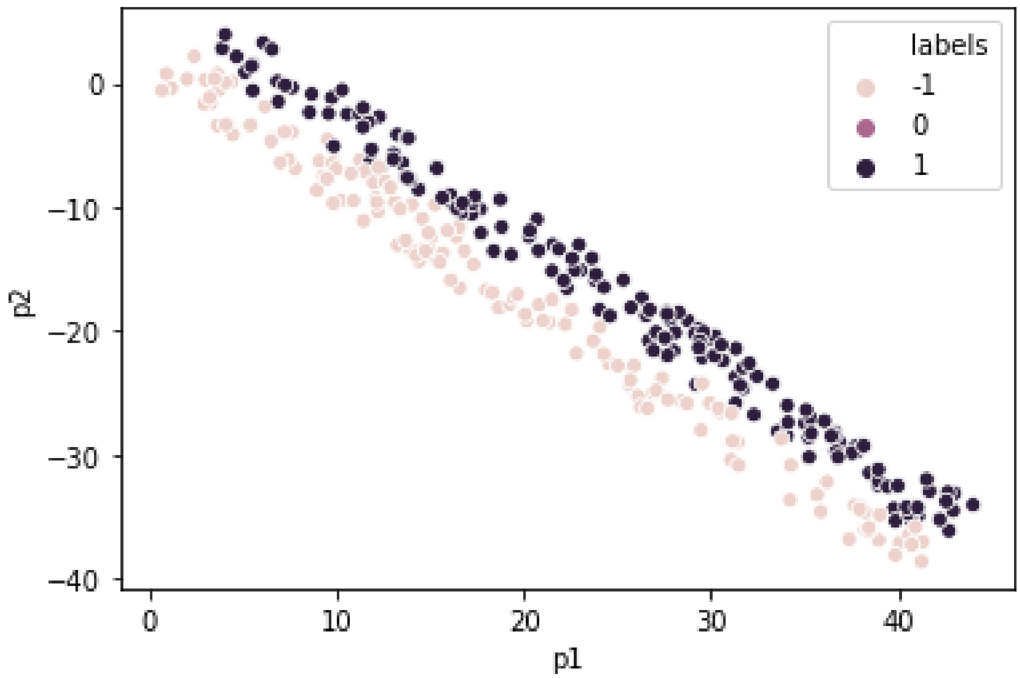
1 Exercise 2

You need to use the Ex2data1.csv to fit multiple models which will be used to classify the target variable in column label.

Use only two features p1, p2 - Plot the points with different colours for the different classes. - Split the data into a training set and a testing set. In each of the following points,the models are to be fit to the training set, the accuracy is to be analysed on both the training set and testing set. - Train the cross-validated logistic regression. Plot the ROC curve, compute the area under the ROC curve for the model. - Train a classification tree on the same dataset. Investigate the accuracy when using the full tree. Compare it to the accuracy attained with the logistic regression. Note: given the particular structure of the data, training the full tree may take a while. - Plot the ROC curve for the full tree (both for the training and learning set.

[39]:
```
Ex2data1 = pd.read_csv("Ex2data1.csv")
Ex2data1.set_index("Unnamed: 0", inplace = True)
```

[40]:
```
#Ex2data1["labels"] = Ex2data1["labels"].astype(str)
#Ex2data1["labels"] = Ex2data1["labels"].astype("category")
ax = sns.scatterplot(x="p1", y="p2", hue="labels", data=Ex2data1)
```



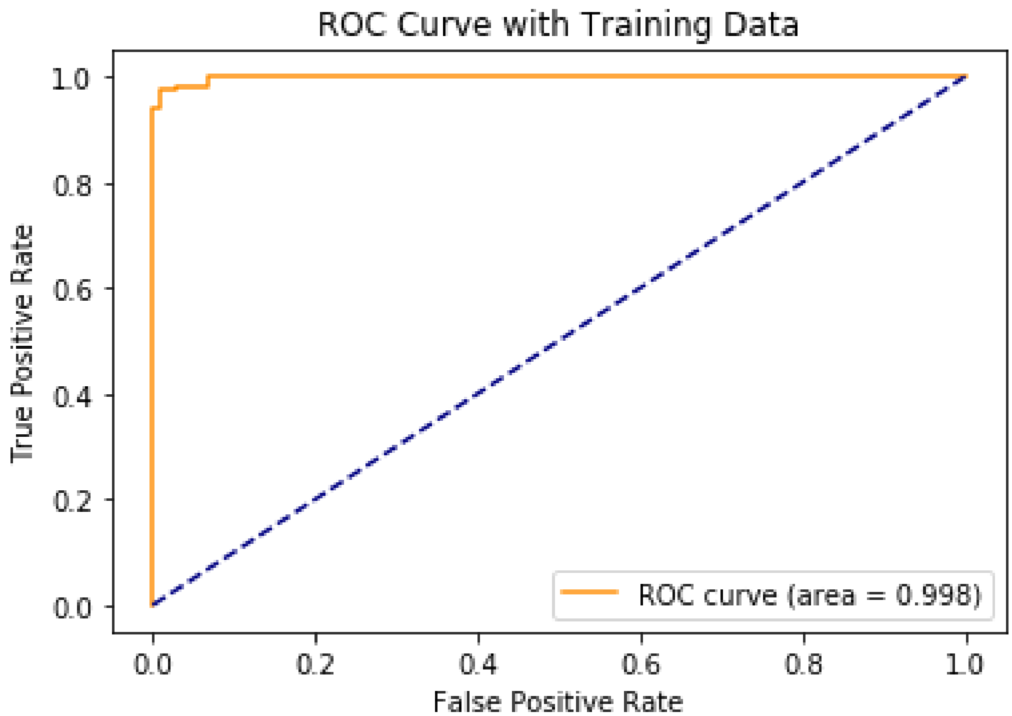
[41]:
```
Ex2data1_tr, Ex2data1_te = train_test_split(Ex2data1, test_size = 0.25,␣
↪random_state = 123)
```

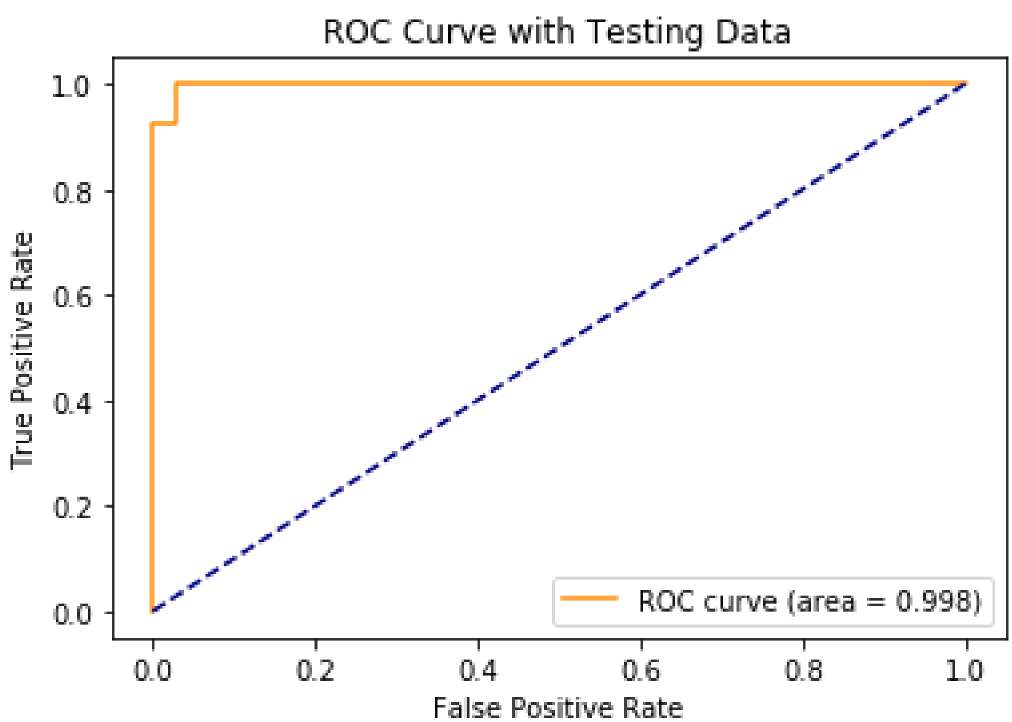
[52]:
```
X = Ex2data1_tr[["p1","p2"]]
y = Ex2data1_tr["labels"]
lr = LogisticRegressionCV(cv=5, random_state=123)
lr.fit(X, y)
```

[52]:
```
LogisticRegressionCV(Cs=10, class_weight=None, cv=5, dual=False,
                     fit_intercept=True, intercept_scaling=1.0, l1_ratios=None,
                     max_iter=100, multi_class='warn', n_jobs=None,
                     penalty='l2', random_state=123, refit=True, scoring=None,
                     solver='lbfgs', tol=0.0001, verbose=0)
```

[94]:
```
predict_tr = lr.predict_proba(X)[:,1] #probability of getting 1 (on the second␣
↪column)
predict_te = lr.predict_proba(Ex2data1_te[["p1","p2"]])[:,1]

fpr, tpr, thresholds = metrics.roc_curve(y, predict_tr, pos_label=1)
# pos_label - the label that will be predicted if the probability value is high
AUC = roc_auc_score(y, predict_tr)

plt.figure()
plt.plot(fpr,tpr, color='darkorange', label='ROC curve (area = %0.3f)' %AUC)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.title("ROC Curve with Training Data")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")

fpr2, tpr2, thresholds2 = metrics.roc_curve(Ex2data1_te["labels"],predict_te,␣
↪pos_label=1)
# pos_label - the label that will be predicted if the probability value is high
AUC2 = roc_auc_score(Ex2data1_te["labels"], predict_te)

plt.figure()

plt.plot(fpr2,tpr2, color='darkorange', label='ROC curve (area = %0.3f)' %AUC2)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.title("ROC Curve with Testing Data")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
```

[94]:
```
<matplotlib.legend.Legend at 0x1a23fe6b00>
```


ROC Curve with Training Data
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
ROC curve (area = 0.998)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate


ROC Curve with Testing Data
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
ROC curve (area = 0.998)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

[104]:

```
tree = DecisionTreeClassifier(criterion = 'entropy', min_samples_split=2,␣
 ↪random_state=1)
tree.fit(X, y)
tree_predict_tr = tree.predict(X)
tree_predict_te = tree.predict(Ex2data1_te[["p1","p2"]])
lr_predict_tr = lr.predict(X) #probability of getting 1 (on the second column)
lr_predict_te = lr.predict(Ex2data1_te[["p1","p2"]])

print("Decision Tree training set accuracy =", accuracy_score(y,tree_predict_tr))
print("Decision Tree testing set accuracy =",␣
 ↪accuracy_score(Ex2data1_te["labels"],tree_predict_te))
print("Logistic Regression training set accuracy =",␣
 ↪accuracy_score(y,lr_predict_tr))
print("Logistic Regression testing set accuracy =",␣
 ↪accuracy_score(Ex2data1_te["labels"],lr_predict_te))
```

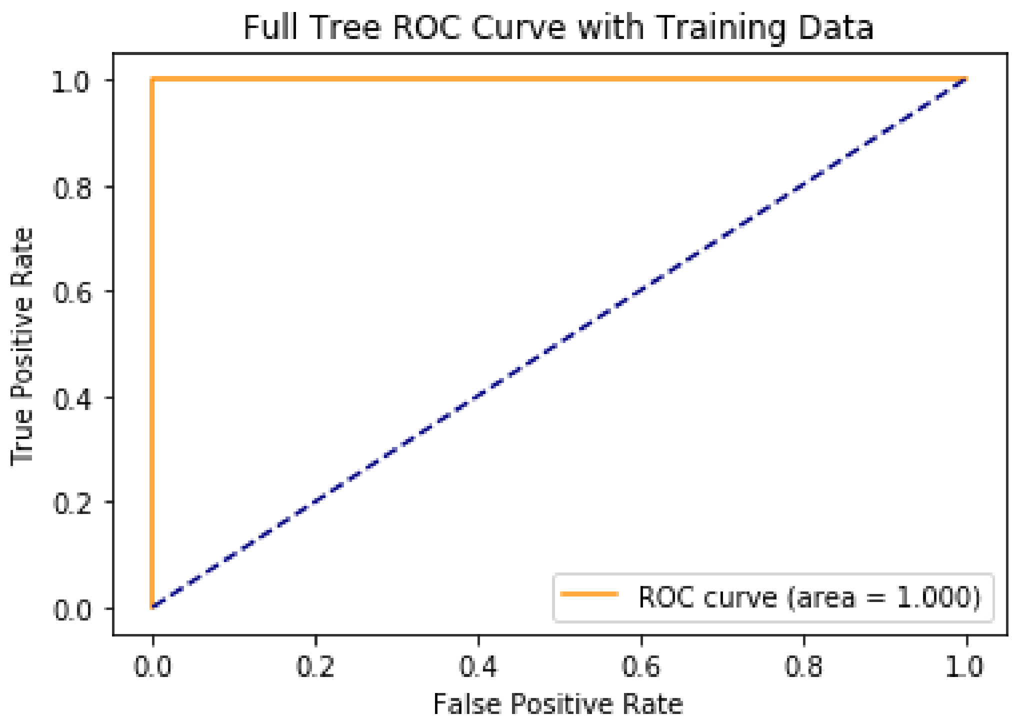
```
Decision Tree training set accuracy = 1.0
Decision Tree testing set accuracy = 0.92
Logistic Regression training set accuracy = 0.9733333333333334
Logistic Regression testing set accuracy = 0.9466666666666667
```

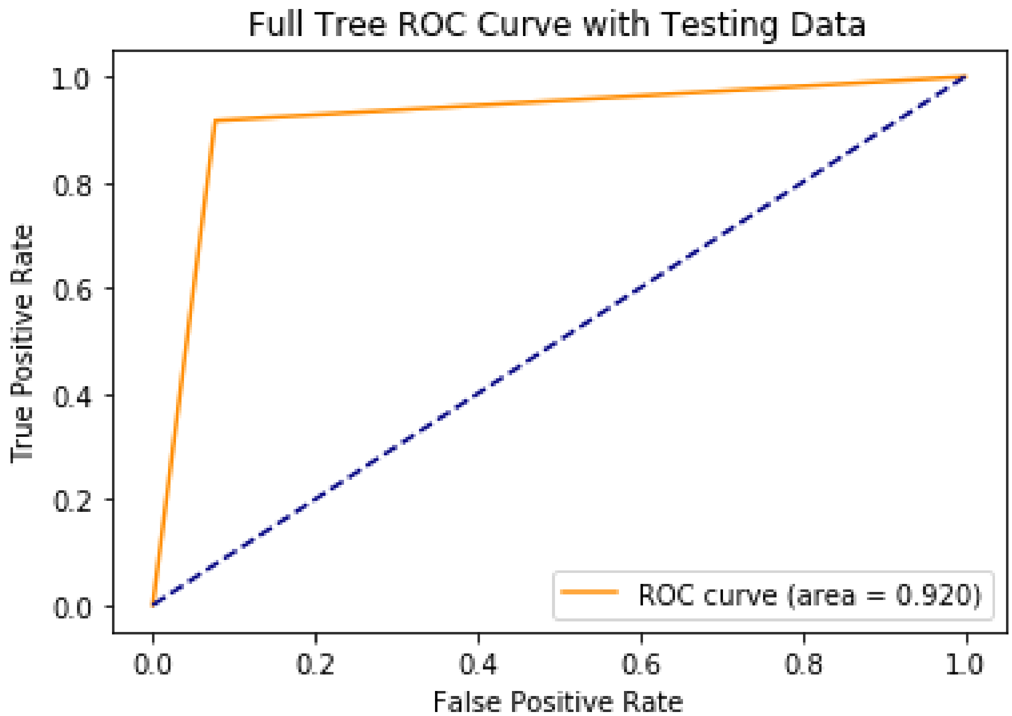
[111]:

```
def ROC(c_matrix):
    TP = c_matrix[0,0]
    FP = c_matrix[1,0]
    TN = c_matrix[1,1]
    FN = c_matrix[0,1]

    TPR = TP/(TP+FN)
    FPR = FP/(FP+TN)

    return (FPR,TPR)

Tree_ROC_curve_tr = ROC(confusion_matrix(y,tree_predict_tr))
Tree_ROC_curve_te = ROC(confusion_matrix(Ex2data1_te["labels"],tree_predict_te))
Tree_AUC_tr = auc([0,Tree_ROC_curve_tr[0],1],[0,Tree_ROC_curve_tr[1],1])
Tree_AUC_te = auc([0,Tree_ROC_curve_te[0],1],[0,Tree_ROC_curve_te[1],1])


plt.figure()
plt.plot([0,Tree_ROC_curve_tr[0],1],[0,Tree_ROC_curve_tr[1],1],␣
 ↪color='darkorange', label='ROC curve (area = %0.3f)' %Tree_AUC_tr)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.title("Full Tree ROC Curve with Training Data")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
```

```
plt.figure()
plt.plot([0,Tree_ROC_curve_te[0],1],[0,Tree_ROC_curve_te[1],1],␣
↪color='darkorange', label='ROC curve (area = %0.3f)' %Tree_AUC_te)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.title("Full Tree ROC Curve with Testing Data")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
```

[111]: <matplotlib.legend.Legend at 0x1a2358f5c0>





Full decision tree seems to overfit the data as it has perfect performance in the training data set but have a worse performance with the testing data set compare to logistic regression.

Use all three features p1, p2, p3 - Train a classification tree on the dataset. Investigate the accuracy on both the full tree, and also on the pruned tree. Compare it to the accuracy and ROC curve. Is there any need for pruning? Explain the outcome. - Train a random forests on the same dataset. Compare it to the performance attained with the classification tree.

[145]:
```
X2_tr = Ex2data1_tr[["p1","p2","p3"]]
y2_tr = Ex2data1_tr["labels"]

X2_te = Ex2data1_te[["p1","p2","p3"]]
y2_te = Ex2data1_te["labels"]

#build full tree with new predictors
tree2 = DecisionTreeClassifier(criterion = 'entropy', min_samples_split=2,␣
 ↪random_state=1)
tree2.fit(X2_tr, y2_tr)
tree2_predict_tr2 = tree2.predict(X2_tr)
tree2_predict_te2 = tree2.predict(X2_te)

#build pruned tree
tree_pruned = DecisionTreeClassifier(criterion = 'entropy', min_samples_leaf=0.
 ↪2, max_depth = 4, max_leaf_nodes = 4, random_state=1)
```

```
tree_pruned.fit(X2_tr, y2_tr)
tree_pruned_predict_tr = tree_pruned.predict(X2_tr)
tree_pruned_predict_te = tree_pruned.predict(X2_te)

#build random forest
forest = RandomForestClassifier(criterion='entropy',␣
↪n_estimators=50,min_samples_leaf=0.2, max_depth = 3, max_leaf_nodes = 4,␣
↪random_state=1)
forest.fit(X2_tr, y2_tr)
forest_predict_tr = forest.predict(X2_tr)
forest_predict_te = forest.predict(X2_te)
```

[146]:
```
#compare accurary
print("Full Tree training set accuracy =",␣
↪accuracy_score(y2_tr,tree2_predict_tr2))
print("Full Tree testing set accuracy =",␣
↪accuracy_score(y2_te,tree2_predict_te2))
print("Pruned Tree training set accuracy =",␣
↪accuracy_score(y2_tr,tree_pruned_predict_tr))
print("Pruned Tree testing set accuracy =",␣
↪accuracy_score(y2_te,tree_pruned_predict_te))
print("Random Forest training set accuracy =",␣
↪accuracy_score(y2_tr,forest_predict_tr))
print("Random Forest testing set accuracy =",␣
↪accuracy_score(y2_te,forest_predict_te))
```

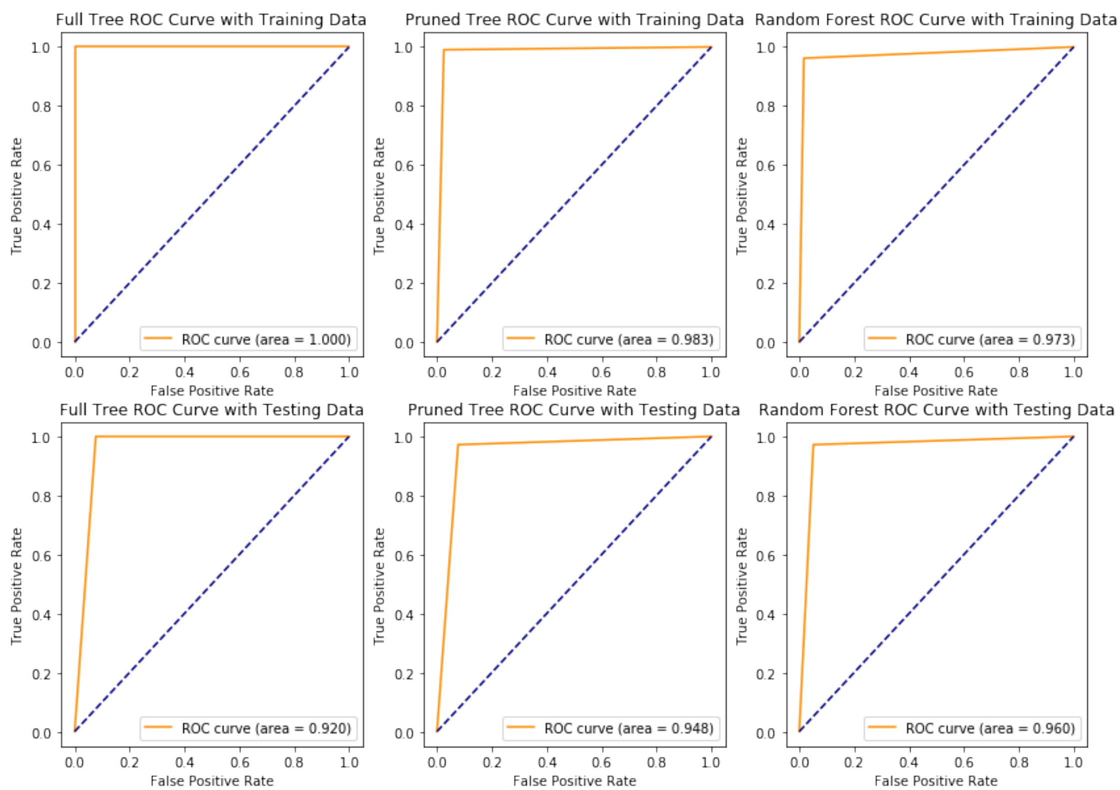
```
Full Tree training set accuracy = 1.0
Full Tree testing set accuracy = 0.96
Pruned Tree training set accuracy = 0.9822222222222222
Pruned Tree testing set accuracy = 0.9466666666666667
Random Forest training set accuracy = 0.9733333333333334
Random Forest testing set accuracy = 0.96
```

[153]:
```
plt.rcParams['figure.figsize'] = [14, 10]

Tree2_ROC_curve_tr = ROC(confusion_matrix(y2_tr,tree2_predict_tr2))
Tree2_ROC_curve_te = ROC(confusion_matrix(y2_te,tree2_predict_te2))
Tree2_AUC_tr = auc([0,Tree_ROC_curve_tr[0],1],[0,Tree_ROC_curve_tr[1],1])
Tree2_AUC_te = auc([0,Tree_ROC_curve_te[0],1],[0,Tree_ROC_curve_te[1],1])

plt.subplot(231)
plt.plot([0,Tree2_ROC_curve_tr[0],1],[0,Tree2_ROC_curve_tr[1],1],␣
↪color='darkorange', label='ROC curve (area = %0.3f)' %Tree2_AUC_tr)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.title("Full Tree ROC Curve with Training Data")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.legend(loc="lower right")

plt.subplot(234)
plt.plot([0,Tree2_ROC_curve_te[0],1],[0,Tree2_ROC_curve_te[1],1], color='darkorange', label='ROC curve (area = %0.3f)' %Tree2_AUC_te)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.title("Full Tree ROC Curve with Testing Data")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")

Pruned_Tree_ROC_curve_tr = ROC(confusion_matrix(y2_tr,tree_pruned_predict_tr))
Pruned_Tree_ROC_curve_te = ROC(confusion_matrix(y2_te,tree_pruned_predict_te))
Pruned_Tree_AUC_tr = auc([0,Pruned_Tree_ROC_curve_tr[0],1],[0,Pruned_Tree_ROC_curve_tr[1],1])
Pruned_Tree_AUC_te = auc([0,Pruned_Tree_ROC_curve_te[0],1],[0,Pruned_Tree_ROC_curve_te[1],1])

plt.subplot(232)
plt.plot([0,Pruned_Tree_ROC_curve_tr[0],1],[0,Pruned_Tree_ROC_curve_tr[1],1], color='darkorange', label='ROC curve (area = %0.3f)' %Pruned_Tree_AUC_tr)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.title("Pruned Tree ROC Curve with Training Data")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")

plt.subplot(235)
plt.plot([0,Pruned_Tree_ROC_curve_te[0],1],[0,Pruned_Tree_ROC_curve_te[1],1], color='darkorange', label='ROC curve (area = %0.3f)' %Pruned_Tree_AUC_te)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.title("Pruned Tree ROC Curve with Testing Data")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")

Random_Forest_ROC_curve_tr = ROC(confusion_matrix(y2_tr,forest_predict_tr))
Random_Forest_ROC_curve_te = ROC(confusion_matrix(y2_te,forest_predict_te))
Random_Forest_AUC_tr = auc([0,Random_Forest_ROC_curve_tr[0],1],[0,Random_Forest_ROC_curve_tr[1],1])
Random_Forest_AUC_te = auc([0,Random_Forest_ROC_curve_te[0],1],[0,Random_Forest_ROC_curve_te[1],1])

plt.subplot(233)
```

```
plt.
 ↪plot([0,Random_Forest_ROC_curve_tr[0],1],[0,Random_Forest_ROC_curve_tr[1],1],␣
 ↪color='darkorange', label='ROC curve (area = %0.3f)' %Random_Forest_AUC_tr)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.title("Random Forest ROC Curve with Training Data")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")

plt.subplot(236)
plt.
 ↪plot([0,Random_Forest_ROC_curve_te[0],1],[0,Random_Forest_ROC_curve_te[1],1],␣
 ↪color='darkorange', label='ROC curve (area = %0.3f)' %Random_Forest_AUC_te)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.title("Random Forest ROC Curve with Testing Data")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
```

[153]: <matplotlib.legend.Legend at 0x1a25bf97b8>



The pruned tree definitely have an improved performance in its testing dataset compare to the full tree, with a drop in performance in with the training data set.

The (pruned) random forest seems to have the best testing dataset performance and the least difference in performance between training and testing dataset. This suggest it best address the overfitting problem for these decision tree models.

This makes sense as random forest chooses not all predictors to use for deciding the optimal split of each nodes and it also create a lot of "randomize" decision trees to aggregates its prediction results, these methods should contribute to the reduction in the overfitting of decision trees model, but introducing randomness to the training process to make model perform better for out of sample prediction.

2 Exercise 3

The Ex2data2.csv is a dataset containing sales of child car seats at 400 different stores. You need to develop a few classification models to predict the stores with high sales. - Create a new feature (column) High based on Sales. Hint: Sales which are greater than 6, can be classifed as High. - Split the dataset into a training set and a test set. - Fit a classification tree to the training set. Plot the tree, calculate the confusion matrix and interpret the results. What test accuracy do you obtain? - Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test accuracy? - Use random forests to analyze this data. What test accuracy do you obtain? Determine which features are most important.

[7]:
```
Ex2data2 = pd.read_csv("Ex2data2.csv")

Ex2data2.set_index("Unnamed: 0", inplace = True)
```

[52]:
```
Ex2data2["Sales Level"] = Ex2data2["Sales"]
Ex2data2.loc[Ex2data2['Sales'] > 6, 'Sales Level'] = 1
Ex2data2.loc[Ex2data2['Sales'] <= 6, 'Sales Level'] = 0
Ex2data2["Sales Level"] = Ex2data2["Sales Level"].astype("category")


Ex2data2.loc[Ex2data2['ShelveLoc'] == "Good" , 'ShelveLoc'] = 1
Ex2data2.loc[Ex2data2['ShelveLoc'] == "Medium" , 'ShelveLoc'] = 0
Ex2data2.loc[Ex2data2['ShelveLoc'] == "Bad" , 'ShelveLoc'] = -1

Ex2data2['ShelveLoc'].astype('int64')

Ex2data2.loc[Ex2data2['Urban'] == "Yes" , 'Urban'] = 1
Ex2data2.loc[Ex2data2['Urban'] == "No" , 'Urban'] = 0

Ex2data2['Urban'].astype('int64')

Ex2data2.loc[Ex2data2['US'] == "Yes" , 'US'] = 1
Ex2data2.loc[Ex2data2['US'] == "No" , 'US'] = 0
```

```
Ex2data2['US'].astype('int64')
```

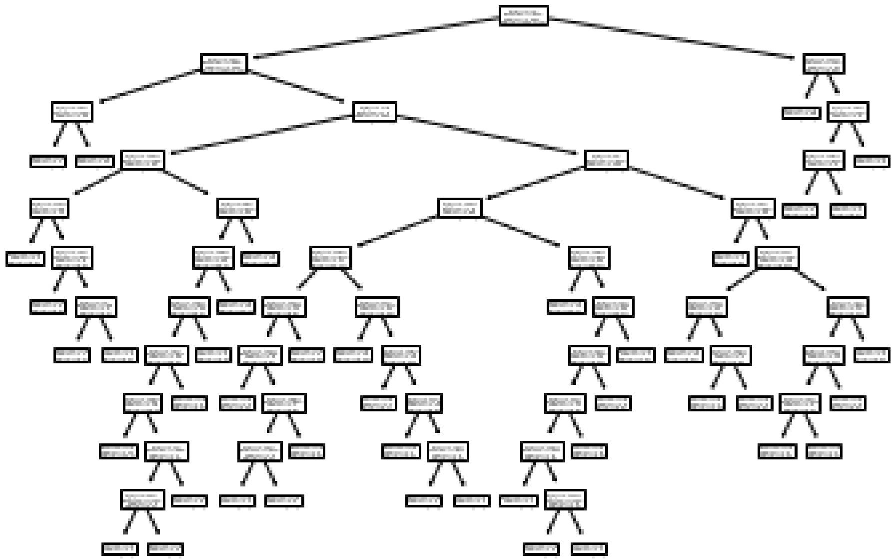
```
/Users/Dominic/anaconda3/lib/python3.7/site-
packages/pandas/core/ops/__init__.py:1115: FutureWarning: elementwise comparison
failed; returning scalar instead, but in the future will perform elementwise
comparison
  result = method(y)
```

[52]:
```
Unnamed: 0
1      1
2      1
3      1
4      1
5      0
      ..
396    1
397    1
398    1
399    1
400    1
Name: US, Length: 400, dtype: int64
```

[78]:
```
Ex2data2_tr, Ex2data2_te = train_test_split(Ex2data2, test_size = 0.25,␣
 ↪random_state = 123)
X_train = Ex2data2_tr.drop(['Sales Level','Sales'], axis=1)
y_train = Ex2data2_tr["Sales Level"]
X_test = Ex2data2_te.drop(['Sales Level','Sales'], axis=1)
y_test = Ex2data2_te["Sales Level"]
```

[81]:
```
tree_2 = DecisionTreeClassifier(criterion = 'entropy', min_samples_leaf=1,␣
 ↪random_state=1)
tree_2.fit(X_train, y_train)
tree_predict_tr_2 = tree_2.predict(X_train)
tree_predict_te_2 = tree_2.predict(X_test)
```

[90]:
```
plot_tree(tree_2);
```



[127]:
```
print(confusion_matrix(y_train,tree_predict_tr_2))
print(confusion_matrix(y_test,tree_predict_te_2))

print("Full Tree training set accuracy =",␣
 ↪accuracy_score(y_train,tree_pruned_predict_tr_2))
print("Full Tree testing set accuracy =",␣
 ↪accuracy_score(y_test,tree_pruned_predict_te_2))
```

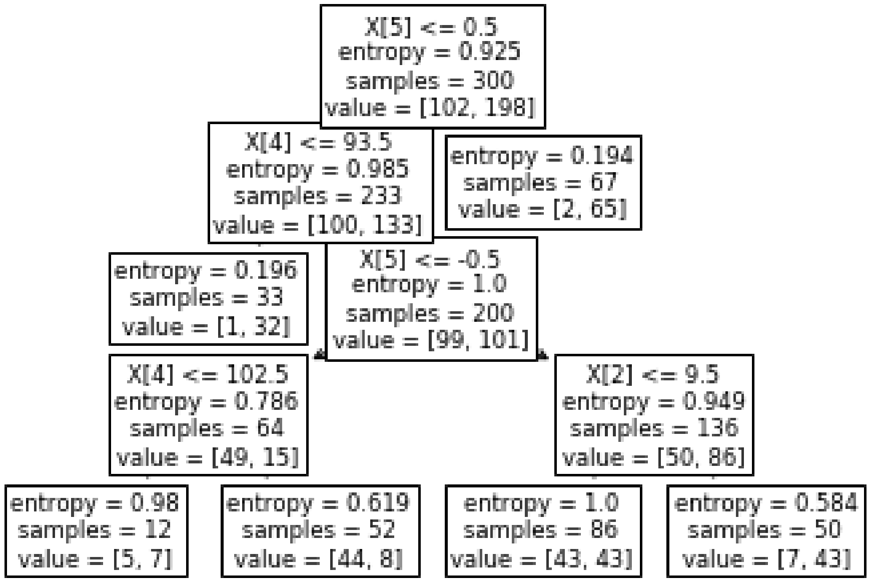
```
[[102   0]
 [  0 198]]
[[16 12]
 [11 61]]
Full Tree training set accuracy = 1.0
Full Tree testing set accuracy = 0.77
```

parameters = {'min_samples_leaf':np.linspace(0.02,0.3,10), 'max_depth':[1,2,3,4,5,6,6,7,8,9], 'max_leaf_nodes':[2,3,4,5,6,7,8]} Pruned_tree = DecisionTreeClassifier() Pruned_tree_CV = GridSearchCV(Pruned_tree, parameters) Pruned_tree_CV.fit(X_train, y_train)

[119]:
```
print(Pruned_tree_CV.best_params_)
Pruned_tree_Best_CV = DecisionTreeClassifier(criterion = 'entropy',max_depth=␣
 ↪4, max_leaf_nodes= 6, min_samples_leaf= 0.02)
Pruned_tree = Pruned_tree_Best_CV.fit(X_train, y_train)
Pruned_tree_predict_tr = Pruned_tree.predict(X_train)
Pruned_tree_predict_te = Pruned_tree.predict(X_test)
```

```
{'max_depth': 4, 'max_leaf_nodes': 6, 'min_samples_leaf': 0.02}
```

[120]:
```
plot_tree(Pruned_tree_Best_CV);
```



[121]:
```
print(confusion_matrix(y_train,Pruned_tree_predict_tr))
print(confusion_matrix(y_test,Pruned_tree_predict_te))

print("Pruned Tree training set accuracy =",␣
 ↪accuracy_score(y_train,Pruned_tree_predict_tr))
print("Pruned Tree testing set accuracy =",␣
 ↪accuracy_score(y_test,Pruned_tree_predict_te))
```

```
[[ 87  15]
 [ 51 147]]
[[20  8]
 [25 47]]
Pruned Tree training set accuracy = 0.78
Pruned Tree testing set accuracy = 0.67
```

Pruning tree does not improve test accuracy, but decrease the difference between the performance of the test set and the training set. Reducing overfitting.

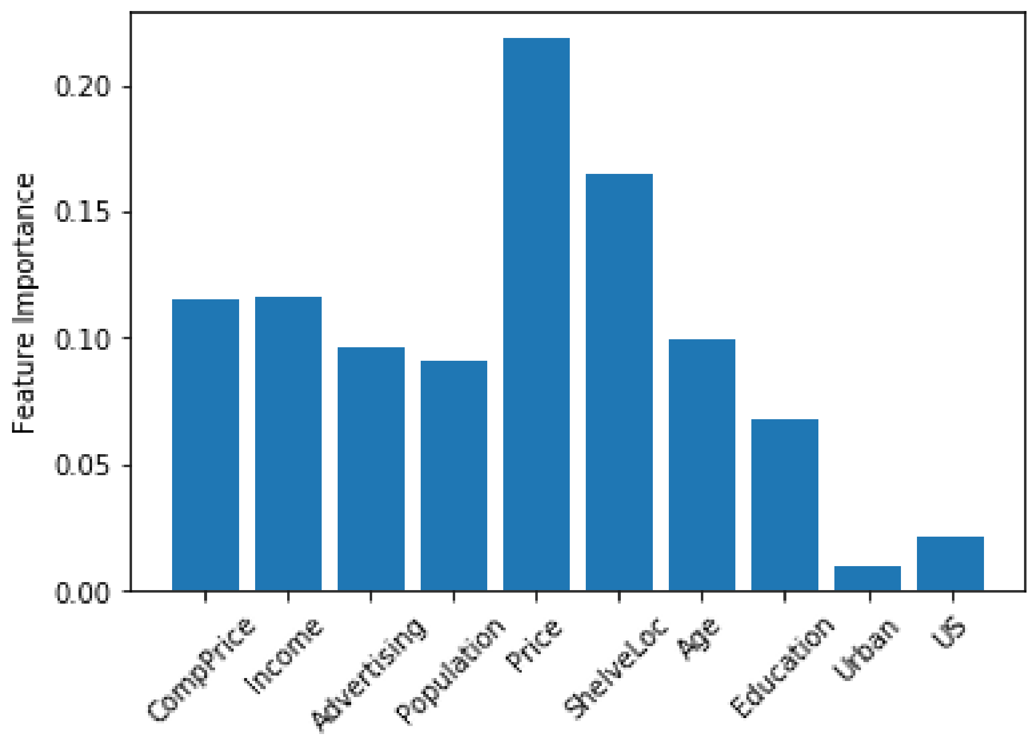
[125]:
```
forest_2 = RandomForestClassifier(criterion='entropy', n_estimators=50,␣
 ↪random_state=1)
forest_2.fit(X_train, y_train)
forest_predict_tr_2 = forest_2.predict(X_train)
forest_predict_te_2 = forest_2.predict(X_test)
```

[128]:
```
print(confusion_matrix(y_train,forest_predict_tr_2))
print(confusion_matrix(y_test,forest_predict_te_2))

print("Random Forest training set accuracy =",
 ↪accuracy_score(y_train,forest_predict_tr_2))
print("Random Forest testing set accuracy =",
 ↪accuracy_score(y_test,forest_predict_te_2))
```

```
[[102   0]
 [  0 198]]
[[18 10]
 [ 3 69]]
Random Forest training set accuracy = 1.0
Random Forest testing set accuracy = 0.87
```

Best performance in terms of testing and training set. Although there is still signs of overfitting with perfect fit for random forest but lower performance with testing set.

[138]:
```
plt.bar(X_train.columns,forest_2.feature_importances_)
plt.ylabel("Feature Importance")
plt.xticks(rotation=45);
```



The feature important scores are calculated by the average decrease in entropy when the variable is used in the splitting of nodes, so from this graph we see that the ShelveLoc and Price variable seems to be the more important feature for this decision tree.

[131]:

```

```

[131]:

```
             CompPrice  Income  Advertising  Population  Price  ShelveLoc  Age  \
Unnamed: 0
171                128      39           12         356    118          0   71
298                118      83           13         276    104         -1   75
149                110     119            0         384     97          0   72
145                132      68            0         264    123          1   34
13                 122      35            2         393    136          0   62
…                    …       …            …           …      …        …  …
231                115      60            0         119    114         -1   38
99                 122      77           24         382    127          1   36
323                140      50           10         300    139          1   60
383                121      28           19         315    121          0   66
366                154      30            0         122    162          0   57

             Education  Urban  US
Unnamed: 0
171                 10      1   1
298                 10      1   1
149                 14      0   1
145                 11      0   0
13                  18      1   0
…                    …      …  ..
231                 14      0   0
99                  16      0   1
323                 15      1   1
383                 14      1   1
366                 17      0   0

[300 rows x 10 columns]
```

3 Exercise 5

You need to use the Ex2data3.csv to fit SVM models which will be able to classified target variable in column V1. - Split the data-set into training set and testing set. - Run SVM using the following kernels: linear, rbf, sigmoid. Use the default value for C. Generate the contour plot of classification, evaluate the accuracy of the classification on the testing set. How would you make a choice between the kernels? - Pick one of the kernels, and run SVM using cross-validation. Use the different value for C. Generate the contour plot of classification, evaluate the accuracy of the classification on the testing set. How would you make a choice of different value for C?

[141]:

```python
Ex2data3 = pd.read_csv("Ex2data3.csv")
```

```
Ex2data3.set_index("Unnamed: 0", inplace = True)
```

[152]:
```
Ex2data3_tr, Ex2data3_te = train_test_split(Ex2data3, test_size = 0.25,␣
 ↪random_state = 123)
X_train = Ex2data3_tr.drop(['V1'], axis=1)
y_train = Ex2data3_tr["V1"]
X_test = Ex2data3_te.drop(['V1'], axis=1)
y_test = Ex2data3_te["V1"]
```

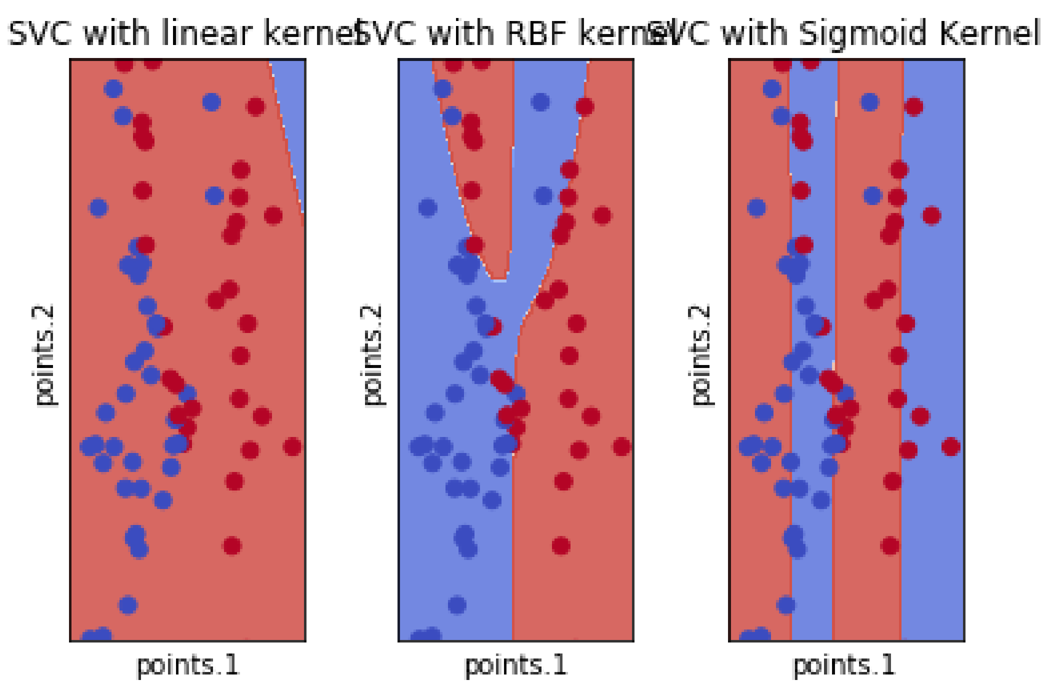
[198]:
```
SVM_linear = svm.SVC(kernel="linear",gamma = "scale")
SVM_linear.fit(X_train,y_train)
SVM_rbf = svm.SVC(C=1, kernel="rbf",gamma = "scale")
SVM_rbf.fit(X_train,y_train)
SVM_sigmoid = svm.SVC(kernel="sigmoid",gamma = "scale")
SVM_sigmoid.fit(X_train,y_train)

SVM_linear_predict_tr = SVM_linear_model.predict(X_train)
SVM_rbf_predict_tr = SVM_rbf_model.predict(X_train)
SVM_sigmoid_predict_tr = SVM_sigmoid_model.predict(X_train)

SVM_linear_predict_te = SVM_linear_model.predict(X_test)
SVM_rbf_predict_te = SVM_rbf_model.predict(X_test)
SVM_sigmoid_predict_te = SVM_sigmoid_model.predict(X_test)
```

[199]:
```
print("Linear SVM testing set accuracy =",␣
 ↪accuracy_score(y_test,SVM_linear_predict_te))
print("RBF SVM testing set accuracy =",␣
 ↪accuracy_score(y_test,SVM_rbf_predict_te))
print("Sigmoid SVM testing set accuracy =",␣
 ↪accuracy_score(y_test,SVM_sigmoid_predict_te))
```

```
Linear SVM testing set accuracy = 0.57
RBF SVM testing set accuracy = 0.9
Sigmoid SVM testing set accuracy = 0.51
```

[200]:
```
h = .02  # step size in the mesh

# create a mesh to plot in
x_min, x_max = X_test.iloc[:, 0].min() - 1, X_test.iloc[:, 0].max() + 1
y_min, y_max = y_test.min() - 1, y_test.max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# title for the plots
titles = ['SVC with linear kernel',
          'SVC with RBF kernel',
          'SVC with Sigmoid Kernel']
```

```
for i, model in enumerate((SVM_linear_model, SVM_rbf_model, SVM_sigmoid_model)):
    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    plt.subplot(1, 3
                , i + 1)
    plt.subplots_adjust(wspace=0.4, hspace=0.4)

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

    # Plot also the training points
    plt.scatter(X_test.iloc[:, 0], X_test.iloc[:, 1], c=y_test, cmap=plt.cm.
 ↪coolwarm)
    plt.xlabel('points.1')
    plt.ylabel('points.2')
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks(())
    plt.yticks(())
    plt.title(titles[i])

plt.show()
```



RBF Seems to perform the best given the ways the data points is grouped according to points.1 and points.2.

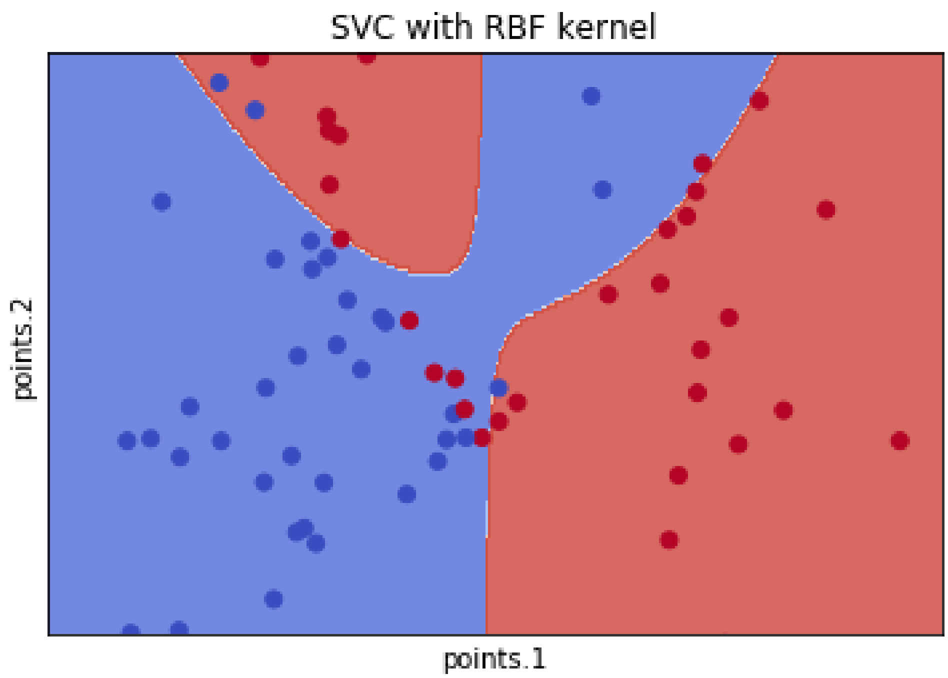
[201]:
```
parameters = {'C':np.linspace(0.01,3,100)}
SVM_rbf = svm.SVC(kernel="rbf",gamma = "scale")
SVM_rbf_CV = GridSearchCV(SVM_rbf, parameters)
SVM_rbf_CV.fit(X_train, y_train);
print(SVM_rbf_CV.best_params_)

SVM_rbf_CV_predict_tr = SVM_rbf_CV.predict(X_train)
SVM_rbf_CV_predict_te = SVM_rbf_CV.predict(X_test)
```

```
/Users/Dominic/anaconda3/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default
value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to
silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

```
{'C': 0.9160606060606061}
```

[202]:
```
h = .02  # step size in the mesh

# create a mesh to plot in
x_min, x_max = X_test.iloc[:, 0].min() - 1, X_test.iloc[:, 0].max() + 1
y_min, y_max = y_test.min() - 1, y_test.max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# title for the plots
titles = ['SVC with RBF kernel']


for i, model in enumerate((SVM_rbf_CV,)):
    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    plt.subplot(1, 1, i + 1)
    plt.subplots_adjust(wspace=0.4, hspace=0.4)

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

    # Plot also the training points
```

```
    plt.scatter(X_test.iloc[:, 0], X_test.iloc[:, 1], c=y_test, cmap=plt.cm.
 ↪coolwarm)
    plt.xlabel('points.1')
    plt.ylabel('points.2')
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks(())
    plt.yticks(())
    plt.title(titles[i])

plt.show()
```



[203]:
```
print("Cross Validated RBF SVM training set accuracy =",␣
 ↪accuracy_score(y_train,SVM_rbf_CV_predict_tr))
print("Cross Validated RBF SVM testing set accuracy =",␣
 ↪accuracy_score(y_test,SVM_rbf_CV_predict_te))
```

```
Cross Validated RBF SVM training set accuracy = 0.91
Cross Validated RBF SVM testing set accuracy = 0.9
```

Although the testing set accuracy did not change, the difference between the accuracy of training and testing set decreases by a little, showing that the cross validation parameter tuning help reduce the overfitting of SVM models.

In general, a larger C should be used when there is a chance that the model is under defined/specified, ie when situation like perfect fitting happens.