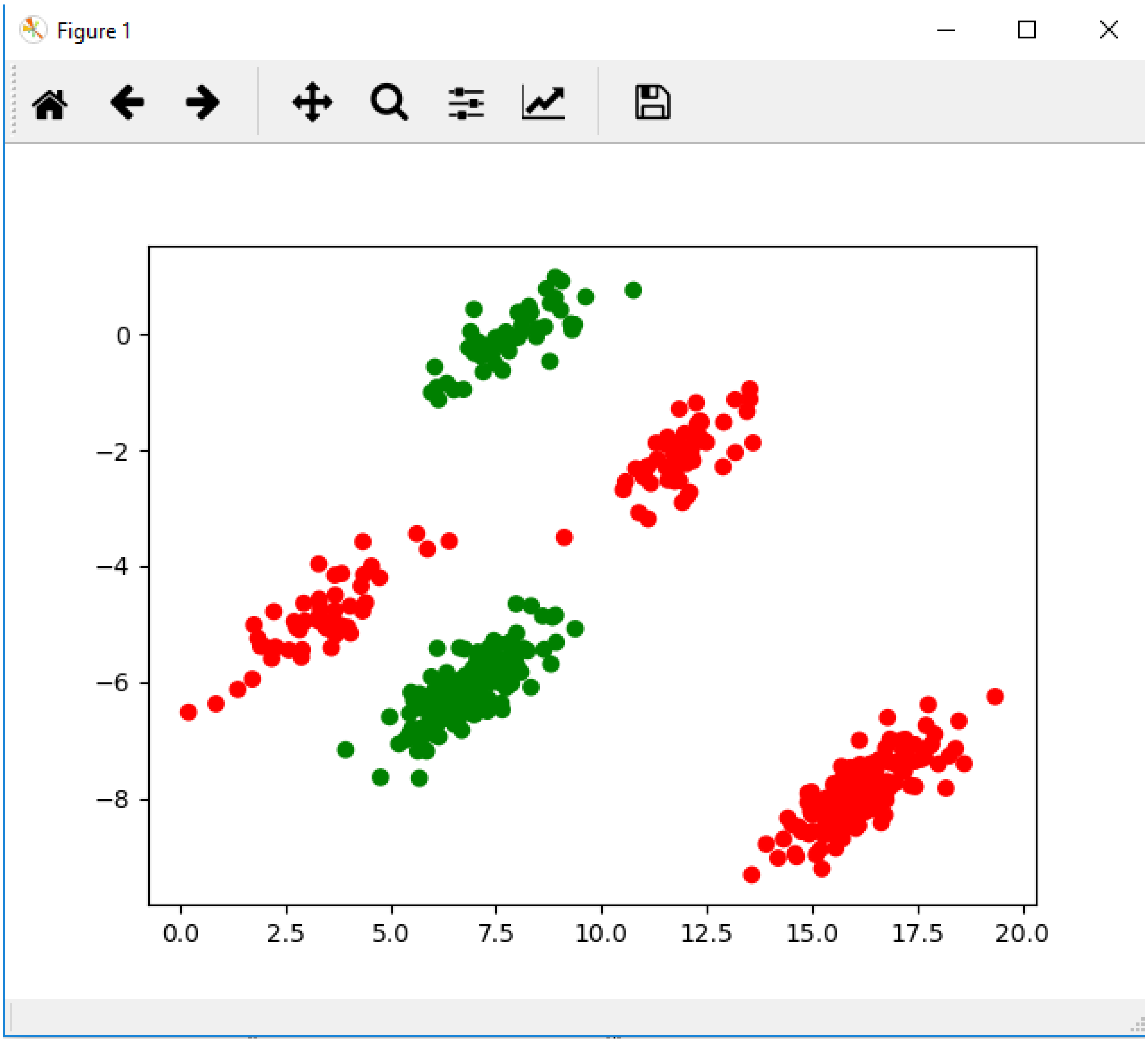
1. Consider the dataset attached with this assignment. It has 450 data points on a 2-D plane. The first two columns contain the X and Y attribute values and the third column contains the class label. Perform the following tasks using this dataset and include the answers/results in your submission. Use Matlab or Scikit toolboxes (Python) to perform all the tasks and include the commands used to achieve each outcome/result.
   a. Display all the data points on a 2-D grid, distinguishing the points belonging to the two classes.

Ans: The points are displayed in 2-D grid as follows:

Green colour: Class label: 1

Red colour: Class label: 0



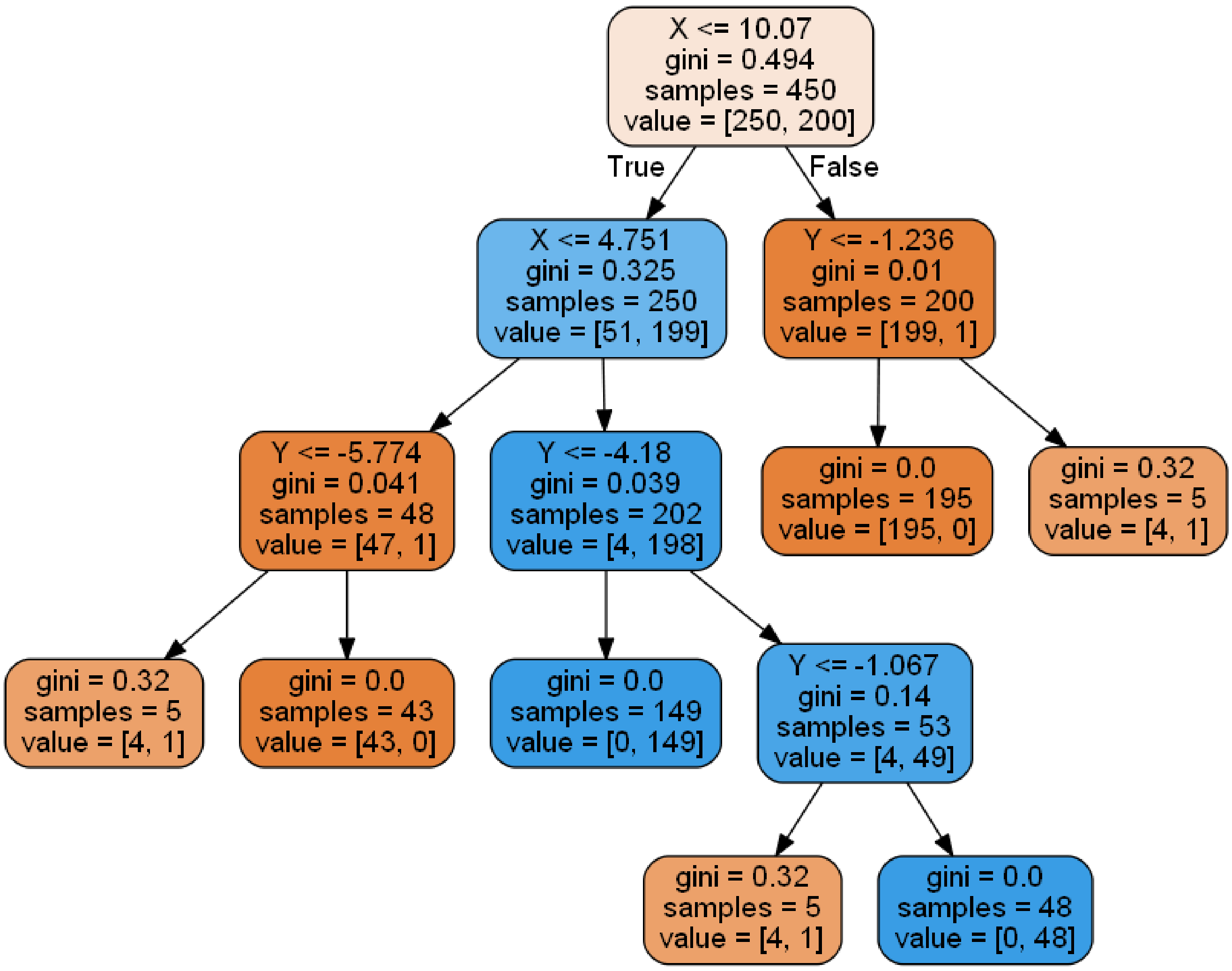
The code used to get the above data graph:

```
frame = download_data('HW2-Synth-Data.xls')
X, Y, labels = get_features_and_labels(frame)
print(labels)
color= ['red' if l == 0 else 'green' for l in labels]
plt.scatter(X, Y, color=color)
plt.show();
```

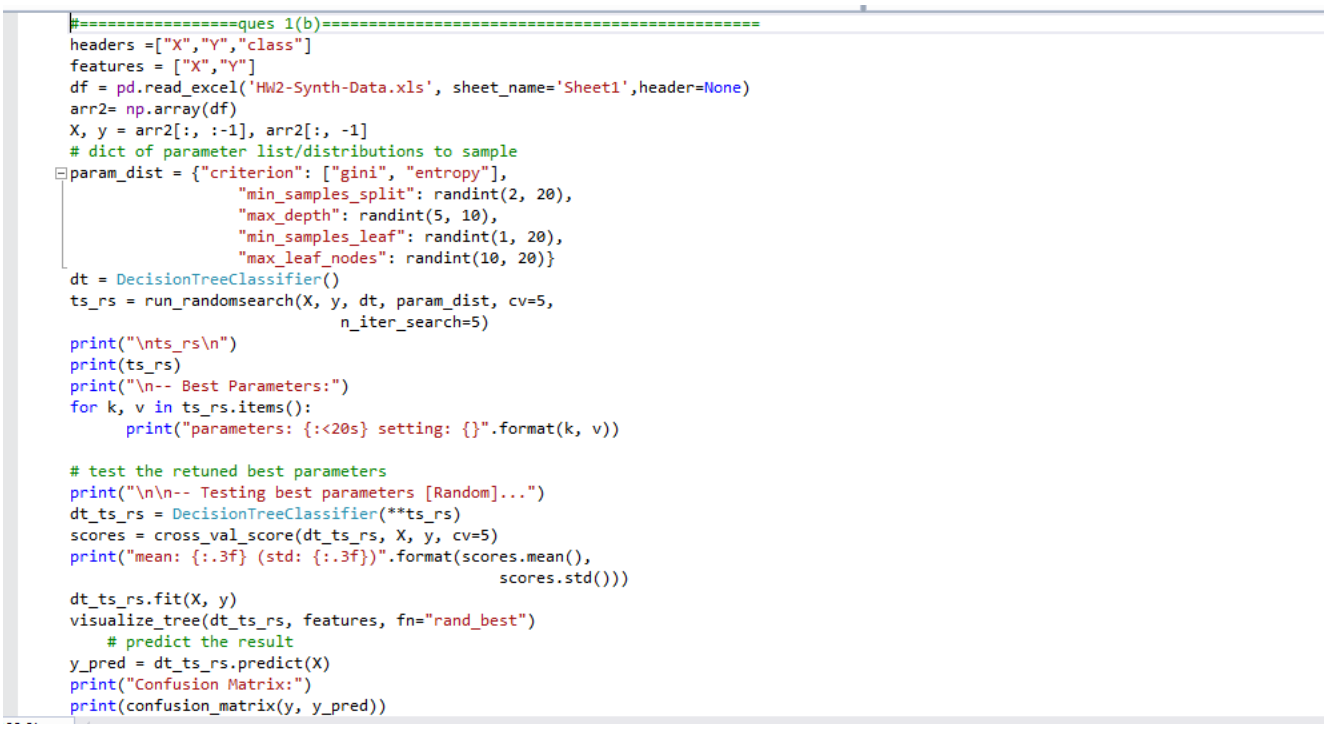
```
#Function for downloading the data
def download_data(fileName):
    frame = pd.read_excel(fileName, sheet_name='Sheet1',header=None)
    return frame
# ====================================================================
#Funtion for splitting and saving the data
def get_features_and_labels(frame):

    arr = np.array(frame)
    # Use the last column as the target value
    X, Y, labels = arr[:, :-2], arr[:, 1], arr[:, -1]
    return X, Y, labels
```

b. Use 5-fold testing to find the best decision tree that you can fit to this dataset. Show all the parameter-value choices you made to get the best performance. You must try to tune the parameters until you get the best possible performance. Use all 450 data points as the test set to compute the confusion matrix, accuracy, and precision and recall values for each class.

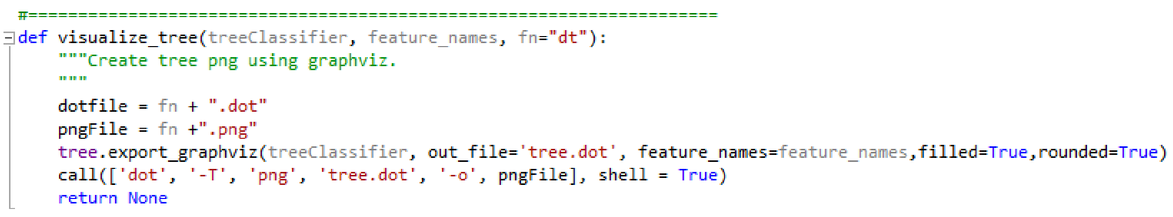
Ans: <u>The best decision tree using 5-fold testing:</u>



Code for creating Decision tree:

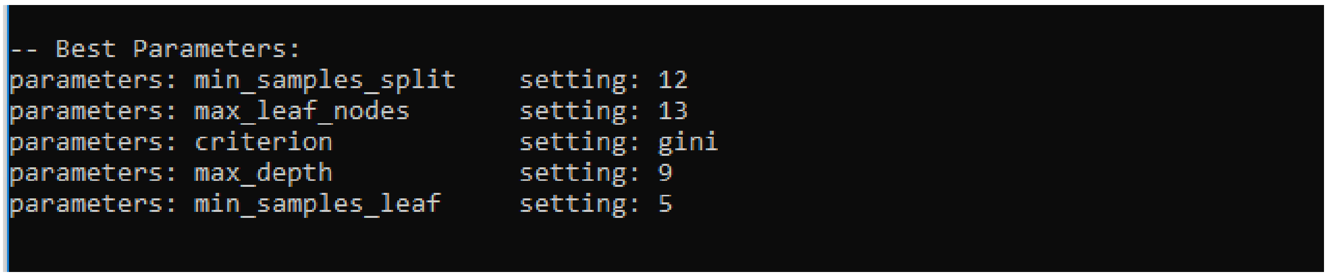


```
#=================ques 1(b)==============================================
headers =["X","Y","class"]
features = ["X","Y"]
df = pd.read_excel('HW2-Synth-Data.xls', sheet_name='Sheet1',header=None)
arr2= np.array(df)
X, y = arr2[:, :-1], arr2[:, -1]
# dict of parameter list/distributions to sample
param_dist = {"criterion": ["gini", "entropy"],
                  "min_samples_split": randint(2, 20),
                  "max_depth": randint(5, 10),
                  "min_samples_leaf": randint(1, 20),
                  "max_leaf_nodes": randint(10, 20)}
dt = DecisionTreeClassifier()
ts_rs = run_randomsearch(X, y, dt, param_dist, cv=5,
                             n_iter_search=5)
print("\nts_rs\n")
print(ts_rs)
print("\n-- Best Parameters:")
for k, v in ts_rs.items():
      print("parameters: {:<20s} setting: {}".format(k, v))

# test the retuned best parameters
print("\n\n-- Testing best parameters [Random]...")
dt_ts_rs = DecisionTreeClassifier(**ts_rs)
scores = cross_val_score(dt_ts_rs, X, y, cv=5)
print("mean: {:.3f} (std: {:.3f})".format(scores.mean(),
                                              scores.std()))
dt_ts_rs.fit(X, y)
visualize_tree(dt_ts_rs, features, fn="rand_best")
    # predict the result
y_pred = dt_ts_rs.predict(X)
print("Confusion Matrix:")
print(confusion_matrix(y, y_pred))
```



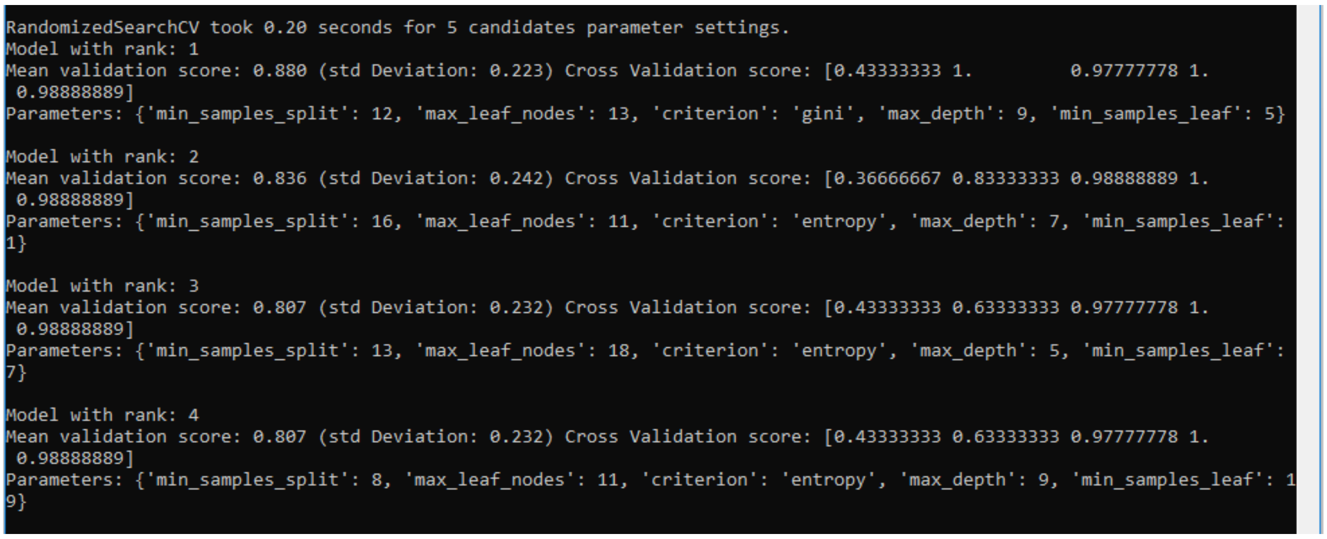
```
#======================================================================
def visualize_tree(treeClassifier, feature_names, fn="dt"):
    """Create tree png using graphviz.
    """
    dotfile = fn + ".dot"
    pngFile = fn +".png"
    tree.export_graphviz(treeClassifier, out_file='tree.dot', feature_names=feature_names,filled=True,rounded=True)
    call(['dot', '-T', 'png', 'tree.dot', '-o', pngFile], shell = True)
    return None
```

Parameter-value choices made:



```
-- Best Parameters:
parameters: min_samples_split    setting: 12
parameters: max_leaf_nodes       setting: 13
parameters: criterion            setting: gini
parameters: max_depth            setting: 9
parameters: min_samples_leaf     setting: 5
```

Models with top rankings:



```
RandomizedSearchCV took 0.20 seconds for 5 candidates parameter settings.
Model with rank: 1
Mean validation score: 0.880 (std Deviation: 0.223) Cross Validation score: [0.43333333 1.         0.97777778 1.
 0.98888889]
Parameters: {'min_samples_split': 12, 'max_leaf_nodes': 13, 'criterion': 'gini', 'max_depth': 9, 'min_samples_leaf': 5}

Model with rank: 2
Mean validation score: 0.836 (std Deviation: 0.242) Cross Validation score: [0.36666667 0.83333333 0.98888889 1.
 0.98888889]
Parameters: {'min_samples_split': 16, 'max_leaf_nodes': 11, 'criterion': 'entropy', 'max_depth': 7, 'min_samples_leaf':
1}

Model with rank: 3
Mean validation score: 0.807 (std Deviation: 0.232) Cross Validation score: [0.43333333 0.63333333 0.97777778 1.
 0.98888889]
Parameters: {'min_samples_split': 13, 'max_leaf_nodes': 18, 'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf':
7}

Model with rank: 4
Mean validation score: 0.807 (std Deviation: 0.232) Cross Validation score: [0.43333333 0.63333333 0.97777778 1.
 0.98888889]
Parameters: {'min_samples_split': 8, 'max_leaf_nodes': 11, 'criterion': 'entropy', 'max_depth': 9, 'min_samples_leaf': 1
9}
```

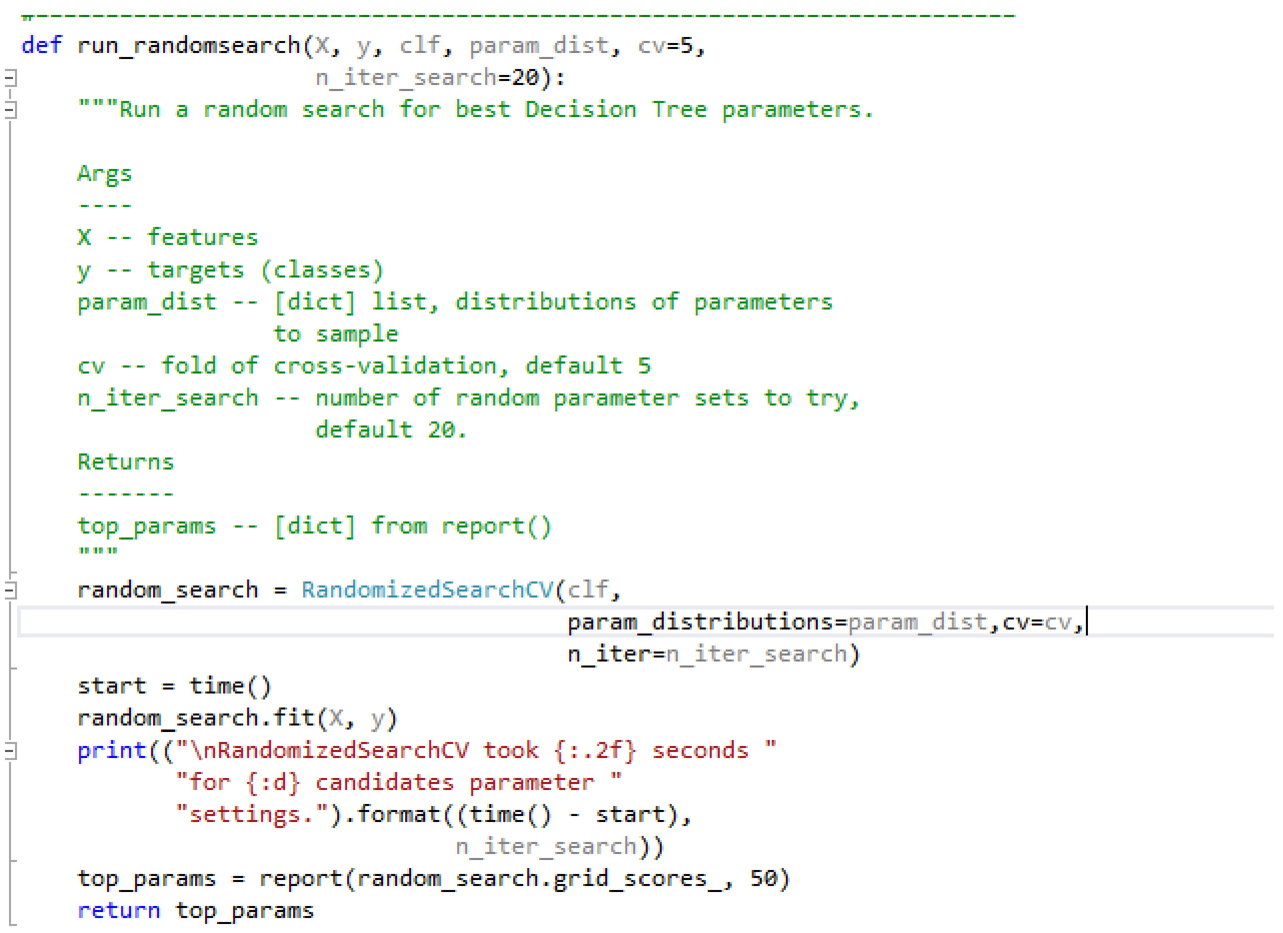
Hence for best parameters of 5-fold testing for decision tree we have chosen the parameter values from the model with Rank: 1.

Used following range for tuning of the parameters:

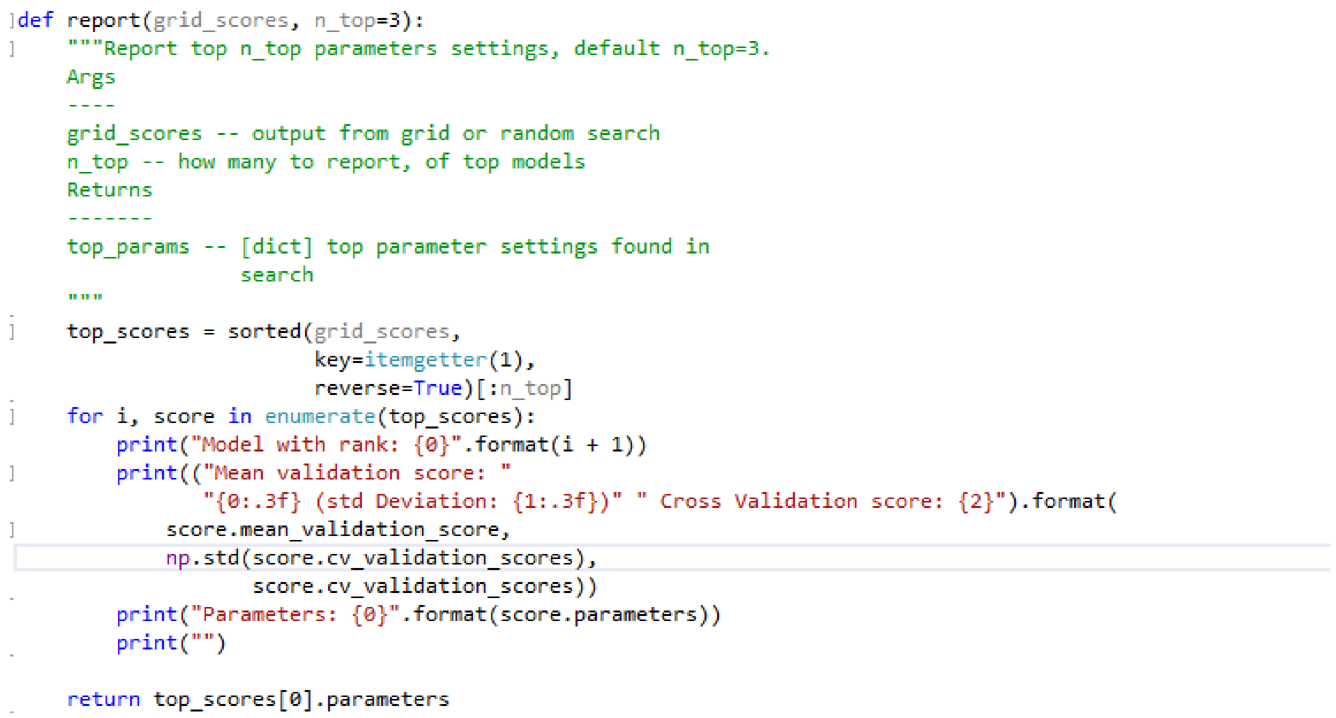
```
{"criterion": ["gini", "entropy"],
    "min_samples_split": randint(2, 20),
    "max_depth": randint(5, 10),
    "min_samples_leaf": randint(1, 20),
    "max_leaf_nodes": randint(10, 20)}
```

Used following function to fine-tune the parameters:

Used RandomizedSearchCV and sorted functions to find out the model with top ranking. Used parameter settings of the model with top most rank.

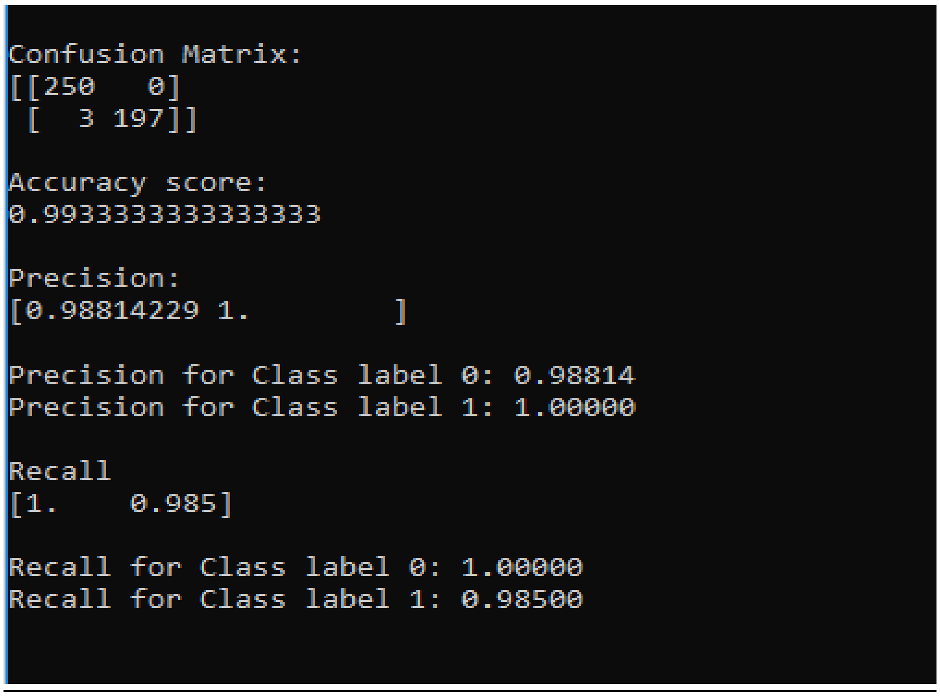


```python
#---------------------------------------------------------------------
def run_randomsearch(X, y, clf, param_dist, cv=5,
                     n_iter_search=20):
    """Run a random search for best Decision Tree parameters.

    Args
    ----
    X -- features
    y -- targets (classes)
    param_dist -- [dict] list, distributions of parameters
                  to sample
    cv -- fold of cross-validation, default 5
    n_iter_search -- number of random parameter sets to try,
                     default 20.
    Returns
    -------
    top_params -- [dict] from report()
    """
    random_search = RandomizedSearchCV(clf,
                                       param_distributions=param_dist,cv=cv,
                                       n_iter=n_iter_search)

    start = time()
    random_search.fit(X, y)
    print(("\nRandomizedSearchCV took {:.2f} seconds "
           "for {:d} candidates parameter "
           "settings.").format((time() - start),
                               n_iter_search))
    top_params = report(random_search.grid_scores_, 50)
    return top_params
```



```python
def report(grid_scores, n_top=3):
    """Report top n_top parameters settings, default n_top=3.
    Args
    ----
    grid_scores -- output from grid or random search
    n_top -- how many to report, of top models
    Returns
    -------
    top_params -- [dict] top parameter settings found in
                  search
    """
    top_scores = sorted(grid_scores,
                        key=itemgetter(1),
                        reverse=True)[:n_top]
    for i, score in enumerate(top_scores):
        print("Model with rank: {0}".format(i + 1))
        print(("Mean validation score: "
              "{0:.3f} (std Deviation: {1:.3f})" " Cross Validation score: {2}").format(
            score.mean_validation_score,
            np.std(score.cv_validation_scores),
                   score.cv_validation_scores))
        print("Parameters: {0}".format(score.parameters))
        print("")

    return top_scores[0].parameters
```

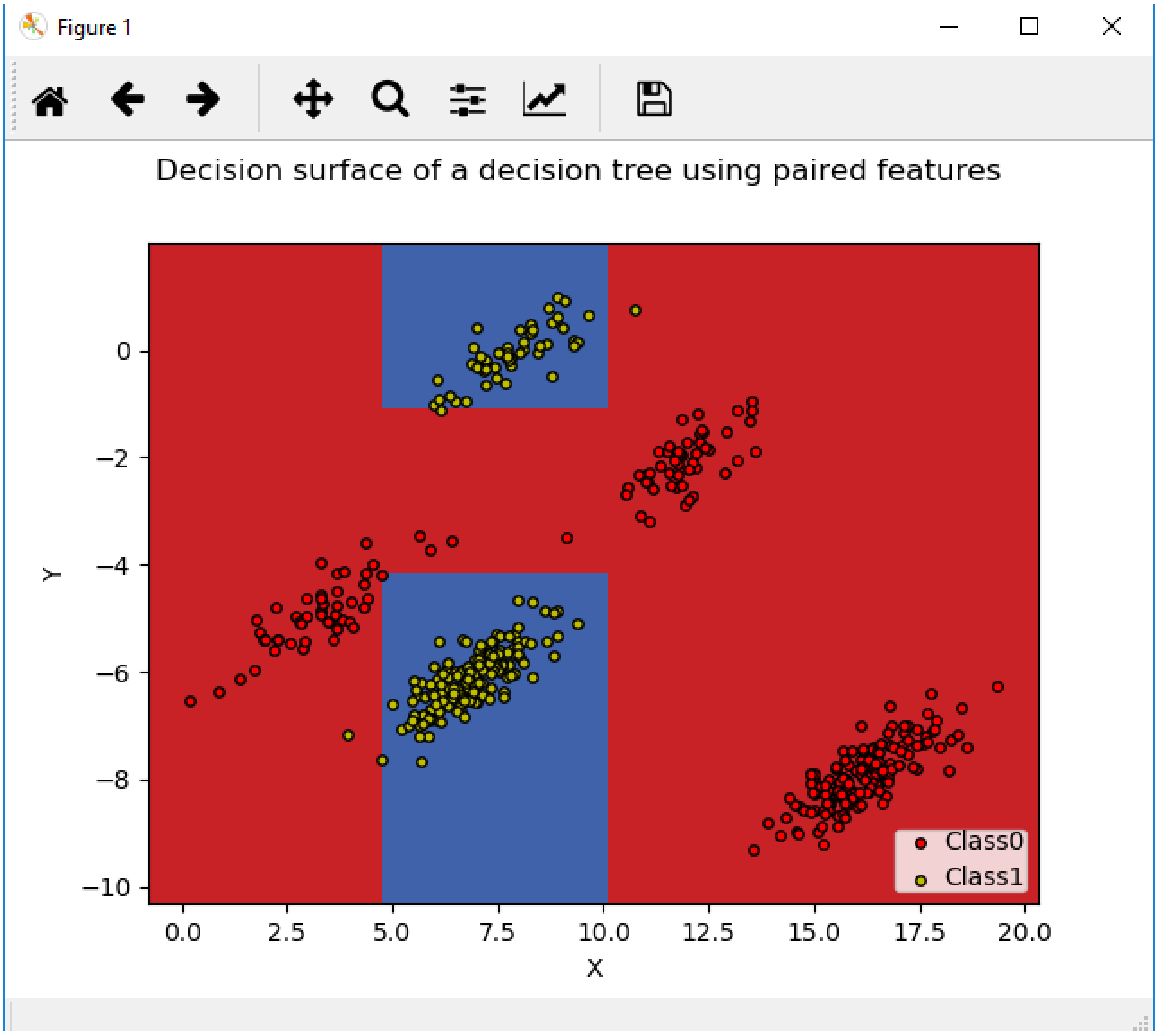
<u>Confusion Matrix, Accuracy, Precision and Recall values:</u>



```
Confusion Matrix:
[[250   0]
 [  3 197]]

Accuracy score:
0.9933333333333333

Precision:
[0.98814229 1.        ]

Precision for Class label 0: 0.98814
Precision for Class label 1: 1.00000

Recall
[1.    0.985]

Recall for Class label 0: 1.00000
Recall for Class label 1: 0.98500
```

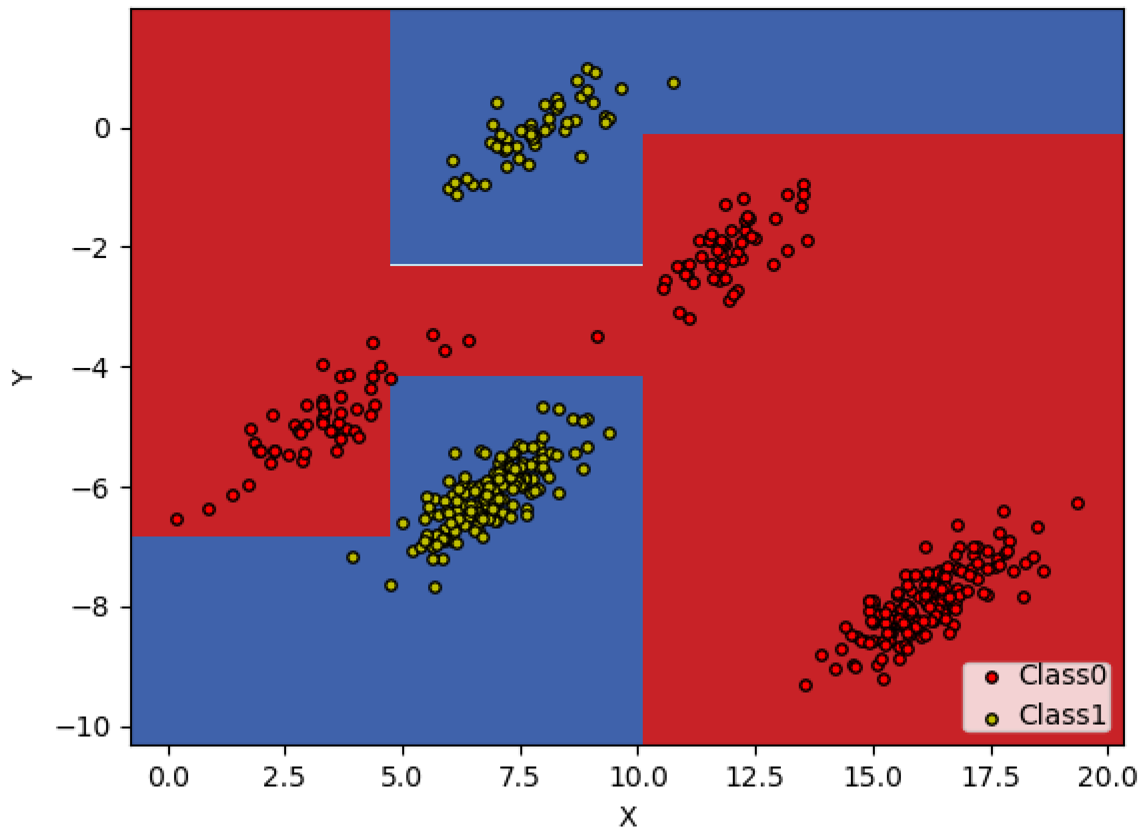
c. Draw the lines on the 2-D plot of the data to show the boundaries that your decision tree learned.

Ans:



d. Draw the best possible decision tree boundaries that you would draw if you were to use your intuition instead of the decision tree induction program. Show the accuracy, precision and recall values for this "ideal" decision tree.

Ans: Decision Tree based on intuition:



Confusion Matrix:

[[250 0]

 [0 200]]

Accuracy:

Accuracy = (TP + TN)/ (TP + TN + FP + FN)

= (250+200)/ (250+200+0+0)

=1.0

Precision:

Precision (class 0) = TP/ (TP+FP)

= 250/ (250 + 0)

=1.0

Precision (class 1) = TN/ (TN+FN)

= 200/ (200+0)

=1.0

Recall:

Recall (class 0) = TP/ (TP +FN)

= 250 / (250 + 0)

=1.0

Recall (class 1) = TN/ (TN +FP)

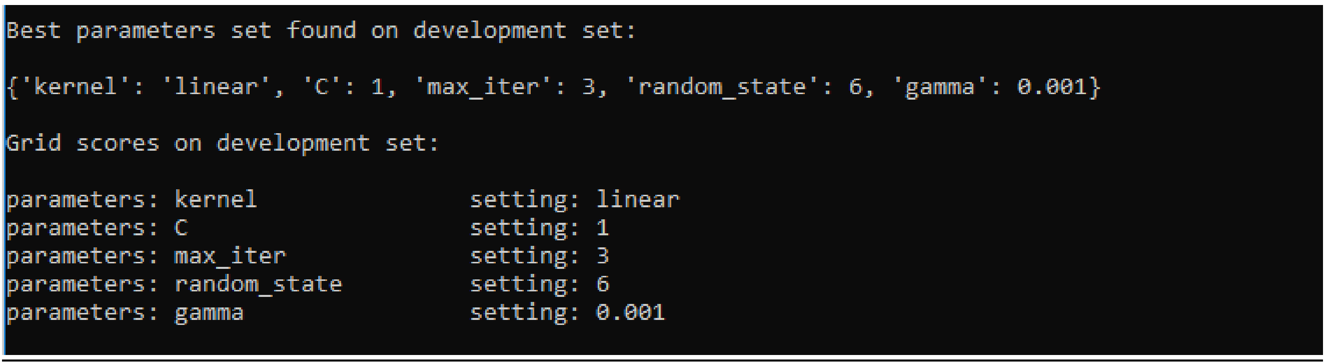
=200/ (200 + 0)

= 1.0

e. Repeat part (b) above for a linear SVM instead of the decision tree. Choose the parameter values that give you the best classification performance in terms of accuracy. State the chosen values of these parameters and give reasons in brief about why you think these values give the best performance. On a 2-D plot of the data show the support vectors learned by your SVM classifier. Using the support vectors draw an estimate of the boundary learned by your program.

Ans:

Best parameter-value choices made based on accuracy:



```
Best parameters set found on development set:

{'kernel': 'linear', 'C': 1, 'max_iter': 3, 'random_state': 6, 'gamma': 0.001}

Grid scores on development set:

parameters: kernel                setting: linear
parameters: C                     setting: 1
parameters: max_iter              setting: 3
parameters: random_state          setting: 6
parameters: gamma                 setting: 0.001
```

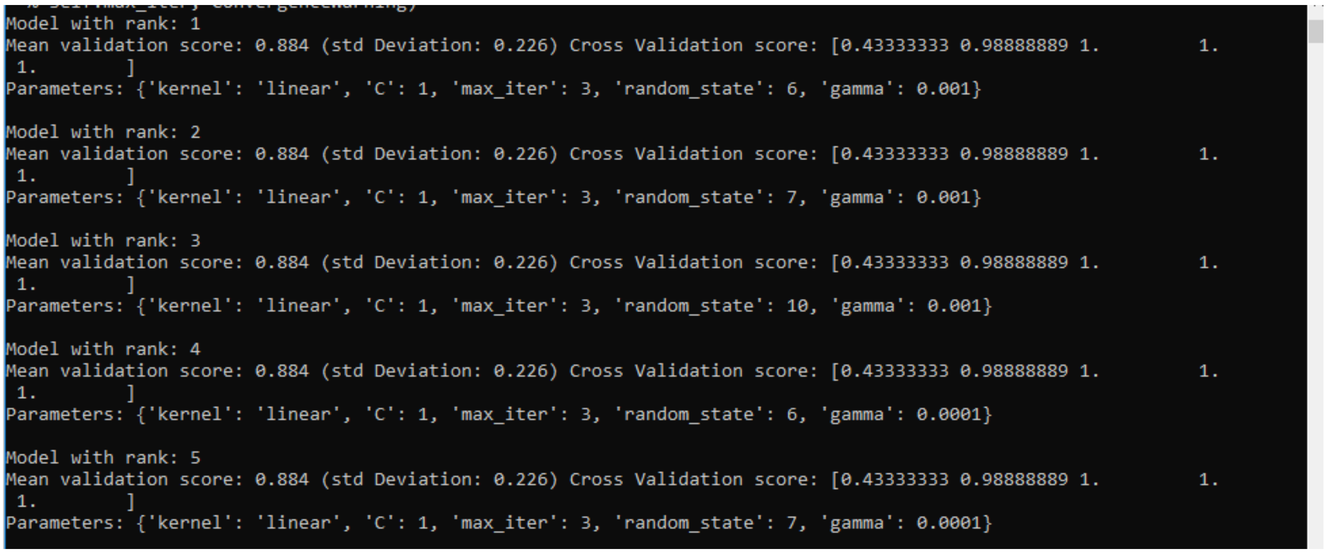
The above values give the best performance according to accuracy and also give best values for Mean Validation score and Cross Validation score. We get these best parameter value pairs by comparing all the models and selecting the model with highest performance rank.

I think in the linear SVM, the parameters which immensely affect the performance are values of C and gamma, which are described as follows:

A high C tries to minimize the misclassification of training data and a low value tries to maintain a smooth classification boundary. Our value of C is neither too high like 10000 nor too low like 0.001. Hence it provides perfect balance in describing decision boundary in linear SVM.

The larger the gamma, the narrow the gaussian "bell" is and hence we chose a lower value for gamma to achieve better performance.

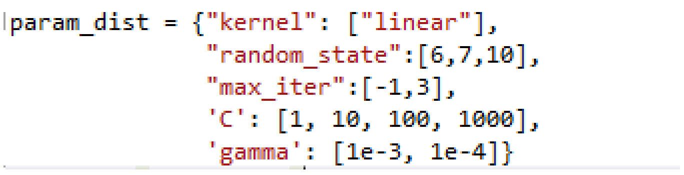
Model with top Rankings:



```
Model with rank: 1
Mean validation score: 0.884 (std Deviation: 0.226) Cross Validation score: [0.43333333 0.98888889 1.         1.
 1.        ]
Parameters: {'kernel': 'linear', 'C': 1, 'max_iter': 3, 'random_state': 6, 'gamma': 0.001}

Model with rank: 2
Mean validation score: 0.884 (std Deviation: 0.226) Cross Validation score: [0.43333333 0.98888889 1.         1.
 1.        ]
Parameters: {'kernel': 'linear', 'C': 1, 'max_iter': 3, 'random_state': 7, 'gamma': 0.001}

Model with rank: 3
Mean validation score: 0.884 (std Deviation: 0.226) Cross Validation score: [0.43333333 0.98888889 1.         1.
 1.        ]
Parameters: {'kernel': 'linear', 'C': 1, 'max_iter': 3, 'random_state': 10, 'gamma': 0.001}

Model with rank: 4
Mean validation score: 0.884 (std Deviation: 0.226) Cross Validation score: [0.43333333 0.98888889 1.         1.
 1.        ]
Parameters: {'kernel': 'linear', 'C': 1, 'max_iter': 3, 'random_state': 6, 'gamma': 0.0001}

Model with rank: 5
Mean validation score: 0.884 (std Deviation: 0.226) Cross Validation score: [0.43333333 0.98888889 1.         1.
 1.        ]
Parameters: {'kernel': 'linear', 'C': 1, 'max_iter': 3, 'random_state': 7, 'gamma': 0.0001}
```

Hence for best parameters of 5-fold testing for linear SVM we have chosen the parameter values from the model with Rank: 1.

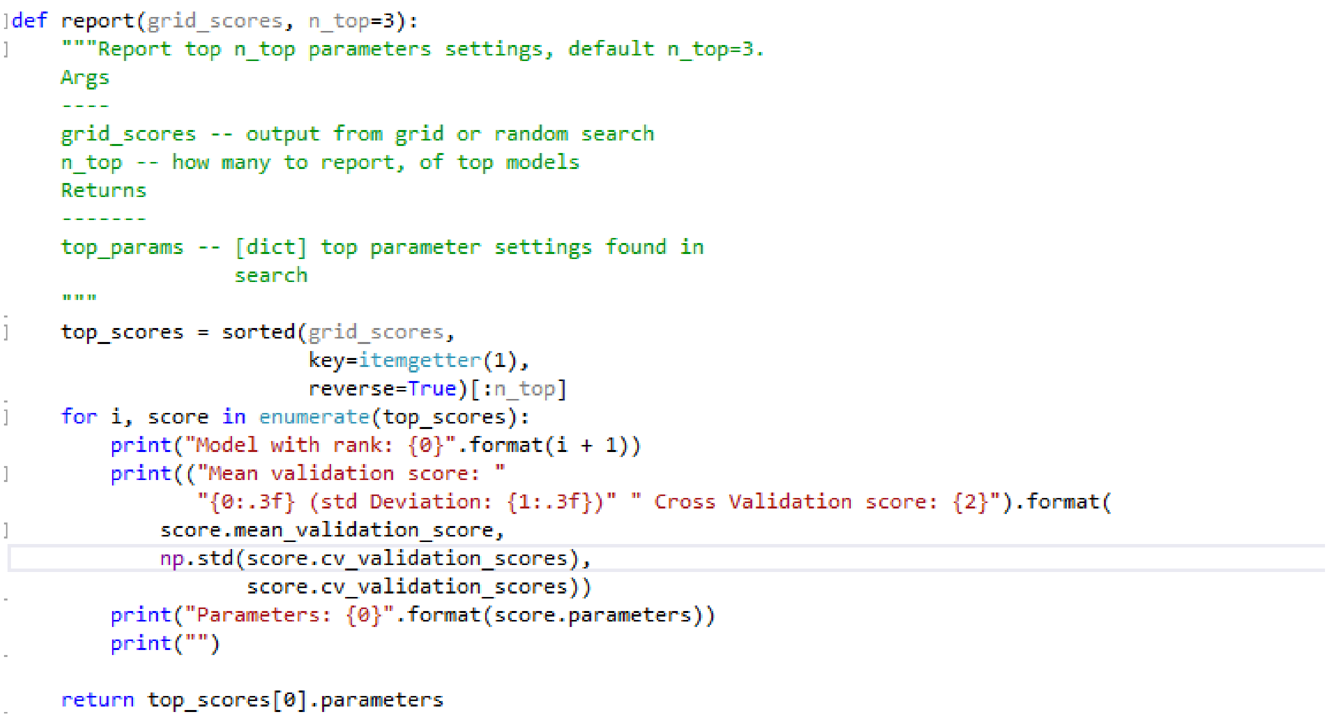
<u>Used following range for tuning of the parameters:</u>



```
param_dist = {"kernel": ["linear"],
              "random_state":[6,7,10],
              "max_iter":[-1,3],
              'C': [1, 10, 100, 1000],
              'gamma': [1e-3, 1e-4]}
```

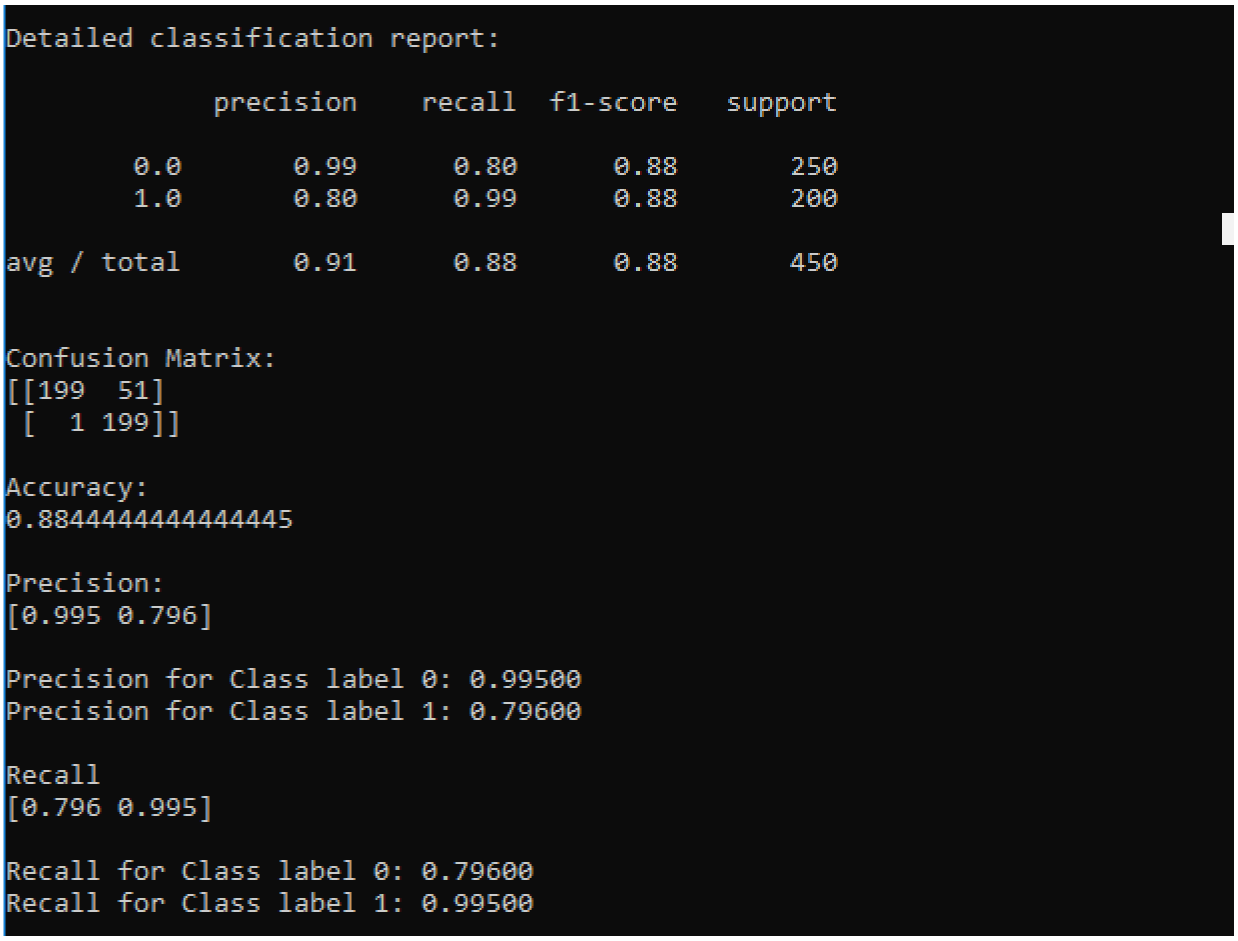
<u>Used following function to fine-tune the parameters:</u>

Used `GridSearchCV` and sorted functions to find out the model with top ranking. Used parameter settings of the model with top most rank.



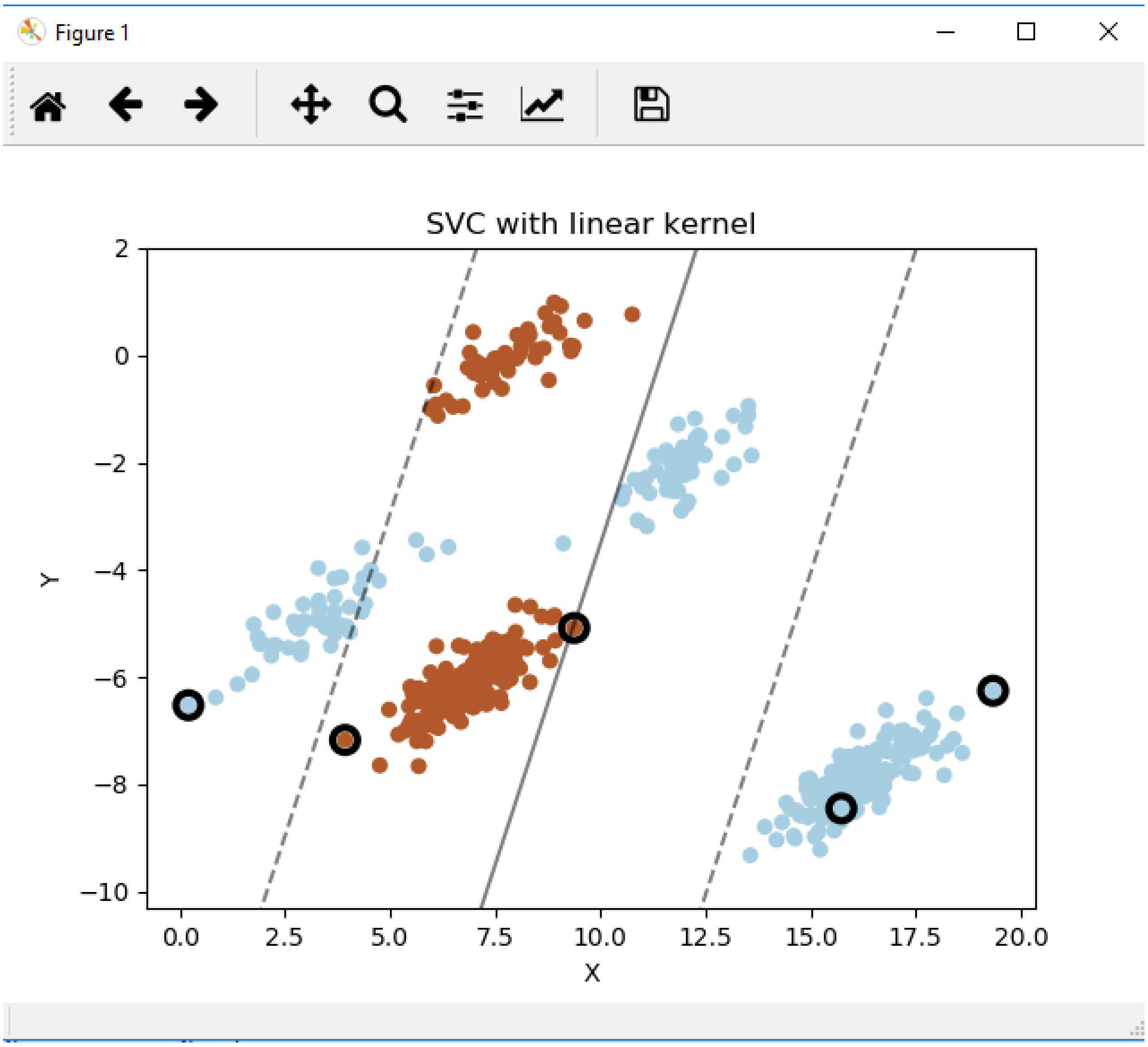
```
def report(grid_scores, n_top=3):
    """Report top n_top parameters settings, default n_top=3.
    Args
    ----
    grid_scores -- output from grid or random search
    n_top -- how many to report, of top models
    Returns
    -------
    top_params -- [dict] top parameter settings found in
                  search
    """
    top_scores = sorted(grid_scores,
                        key=itemgetter(1),
                        reverse=True)[:n_top]
    for i, score in enumerate(top_scores):
        print("Model with rank: {0}".format(i + 1))
        print(("Mean validation score: "
               "{0:.3f} (std Deviation: {1:.3f})" " Cross Validation score: {2}").format(
            score.mean_validation_score,
            np.std(score.cv_validation_scores),
                   score.cv_validation_scores))
        print("Parameters: {0}".format(score.parameters))
        print("")

    return top_scores[0].parameters
```

<u>Confusion matrix, accuracy, precision and recall:</u>

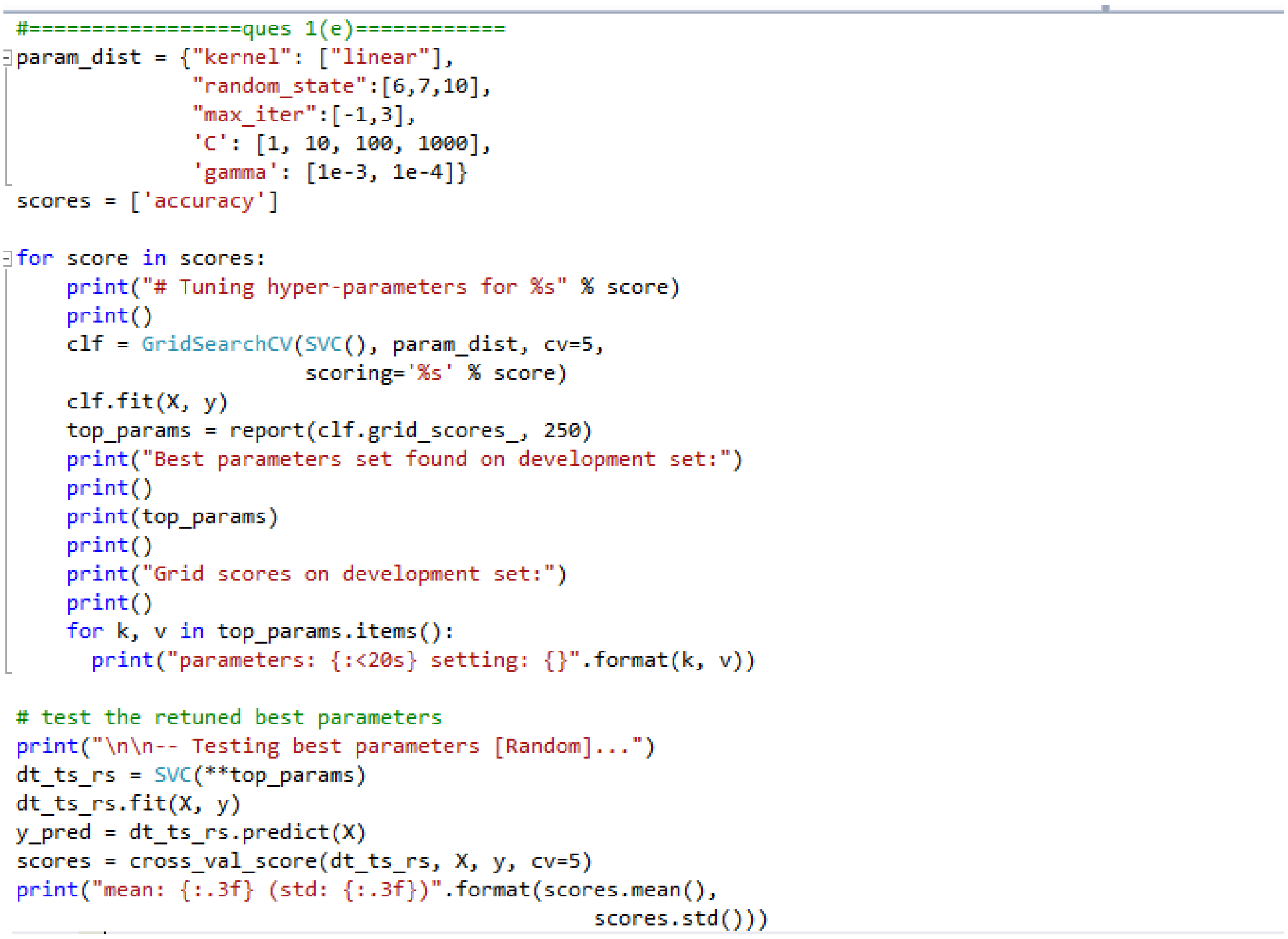


```
Detailed classification report:

             precision    recall  f1-score   support

        0.0       0.99      0.80      0.88       250
        1.0       0.80      0.99      0.88       200

avg / total       0.91      0.88      0.88       450


Confusion Matrix:
[[199  51]
 [  1 199]]

Accuracy:
0.8844444444444445

Precision:
[0.995 0.796]

Precision for Class label 0: 0.99500
Precision for Class label 1: 0.79600

Recall
[0.796 0.995]

Recall for Class label 0: 0.79600
Recall for Class label 1: 0.99500
```

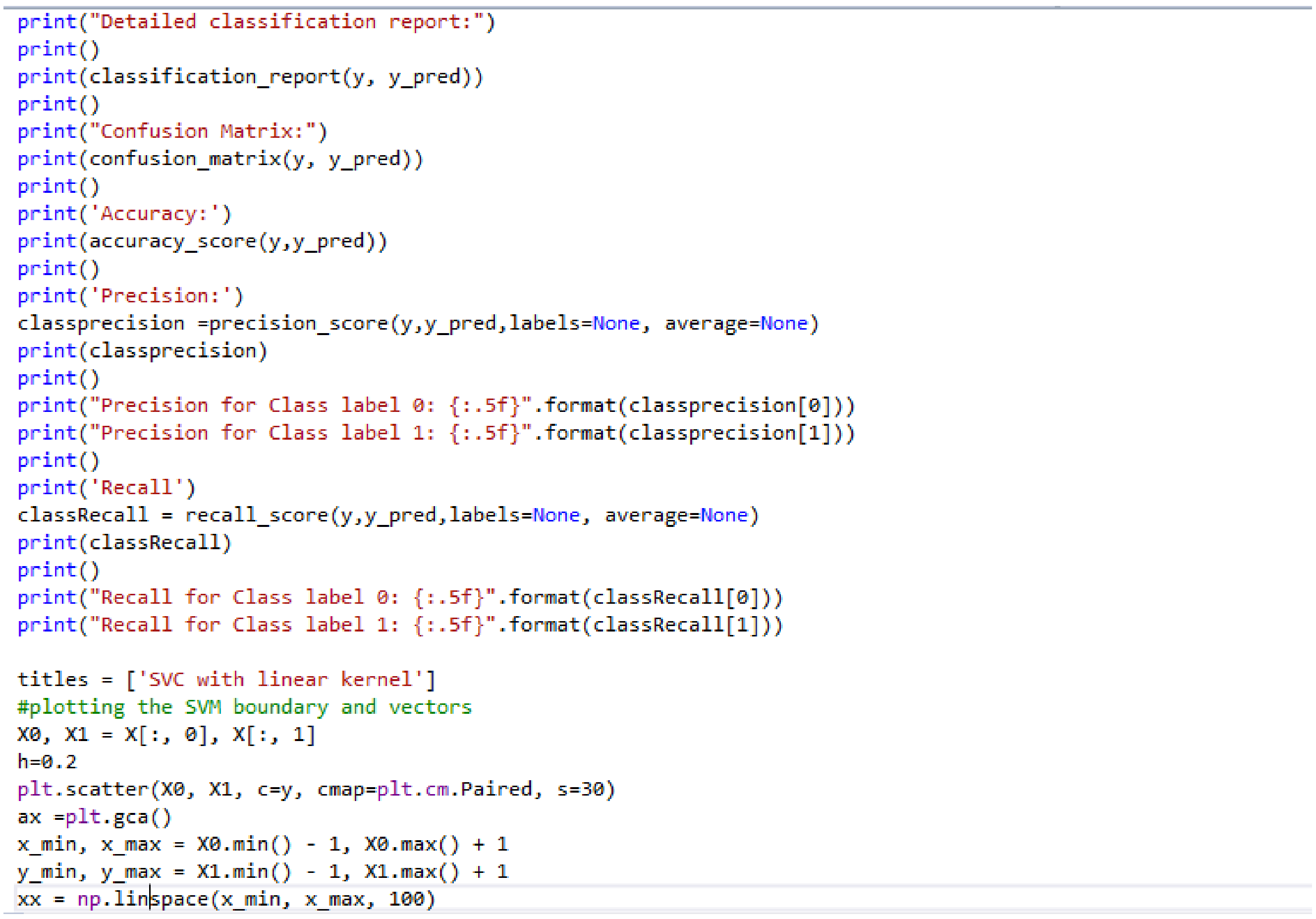
2D data plot for Linear SVM



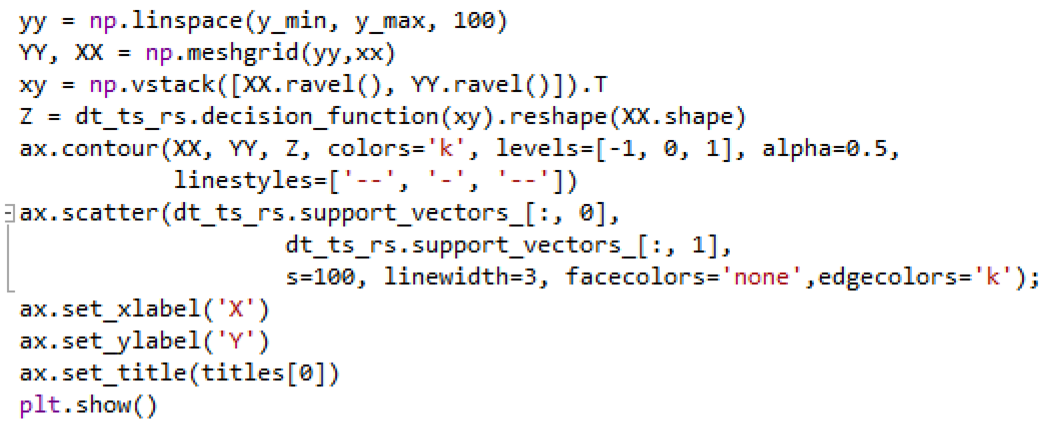
Code used for implementing linear SVM:



```
#=================ques 1(e)===========
param_dist = {"kernel": ["linear"],
              "random_state":[6,7,10],
              "max_iter":[-1,3],
              'C': [1, 10, 100, 1000],
              'gamma': [1e-3, 1e-4]}
scores = ['accuracy']

for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
    print()
    clf = GridSearchCV(SVC(), param_dist, cv=5,
                       scoring='%s' % score)
    clf.fit(X, y)
    top_params = report(clf.grid_scores_, 250)
    print("Best parameters set found on development set:")
    print()
    print(top_params)
    print()
    print("Grid scores on development set:")
    print()
    for k, v in top_params.items():
      print("parameters: {:<20s} setting: {}".format(k, v))

# test the retuned best parameters
print("\n\n-- Testing best parameters [Random]...")
dt_ts_rs = SVC(**top_params)
dt_ts_rs.fit(X, y)
y_pred = dt_ts_rs.predict(X)
scores = cross_val_score(dt_ts_rs, X, y, cv=5)
print("mean: {:.3f} (std: {:.3f})".format(scores.mean(),
                                          scores.std()))
```



```
print("Detailed classification report:")
print()
print(classification_report(y, y_pred))
print()
print("Confusion Matrix:")
print(confusion_matrix(y, y_pred))
print()
print('Accuracy:')
print(accuracy_score(y,y_pred))
print()
print('Precision:')
classprecision =precision_score(y,y_pred,labels=None, average=None)
print(classprecision)
print()
print("Precision for Class label 0: {:.5f}".format(classprecision[0]))
print("Precision for Class label 1: {:.5f}".format(classprecision[1]))
print()
print('Recall')
classRecall = recall_score(y,y_pred,labels=None, average=None)
print(classRecall)
print()
print("Recall for Class label 0: {:.5f}".format(classRecall[0]))
print("Recall for Class label 1: {:.5f}".format(classRecall[1]))

titles = ['SVC with linear kernel']
#plotting the SVM boundary and vectors
X0, X1 = X[:, 0], X[:, 1]
h=0.2
plt.scatter(X0, X1, c=y, cmap=plt.cm.Paired, s=30)
ax =plt.gca()
x_min, x_max = X0.min() - 1, X0.max() + 1
y_min, y_max = X1.min() - 1, X1.max() + 1
xx = np.linspace(x_min, x_max, 100)
```

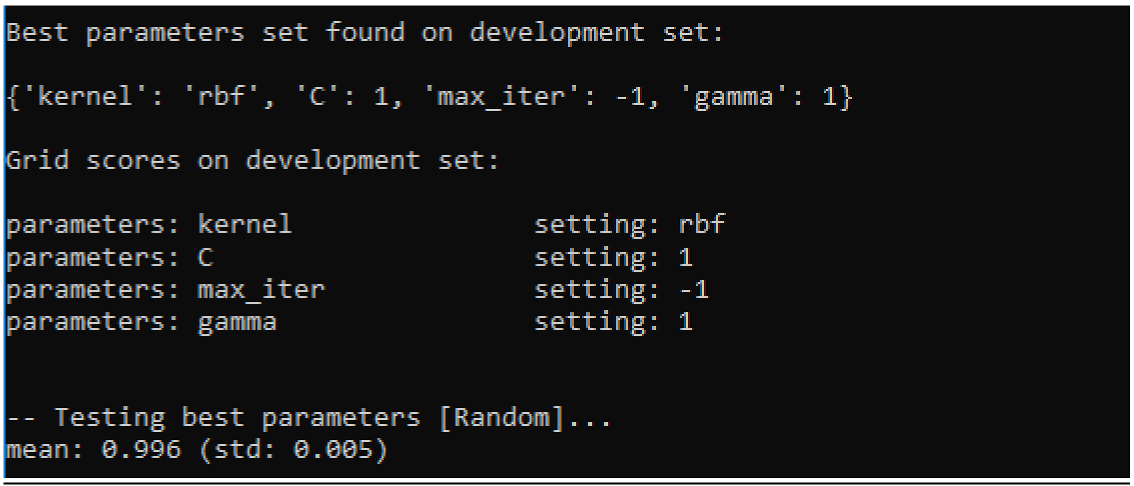


```
yy = np.linspace(y_min, y_max, 100)
YY, XX = np.meshgrid(yy,xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = dt_ts_rs.decision_function(xy).reshape(XX.shape)
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])
ax.scatter(dt_ts_rs.support_vectors_[:, 0],
                   dt_ts_rs.support_vectors_[:, 1],
                   s=100, linewidth=3, facecolors='none',edgecolors='k');
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_title(titles[0])
plt.show()
```

f. Repeat part (e) above using an RBF kernel to learn a non-linear SVM.

Ans:

Best parameter-value choices made based on accuracy:



```
Best parameters set found on development set:

{'kernel': 'rbf', 'C': 1, 'max_iter': -1, 'gamma': 1}

Grid scores on development set:

parameters: kernel                setting: rbf
parameters: C                     setting: 1
parameters: max_iter              setting: -1
parameters: gamma                 setting: 1


-- Testing best parameters [Random]...
mean: 0.996 (std: 0.005)
```

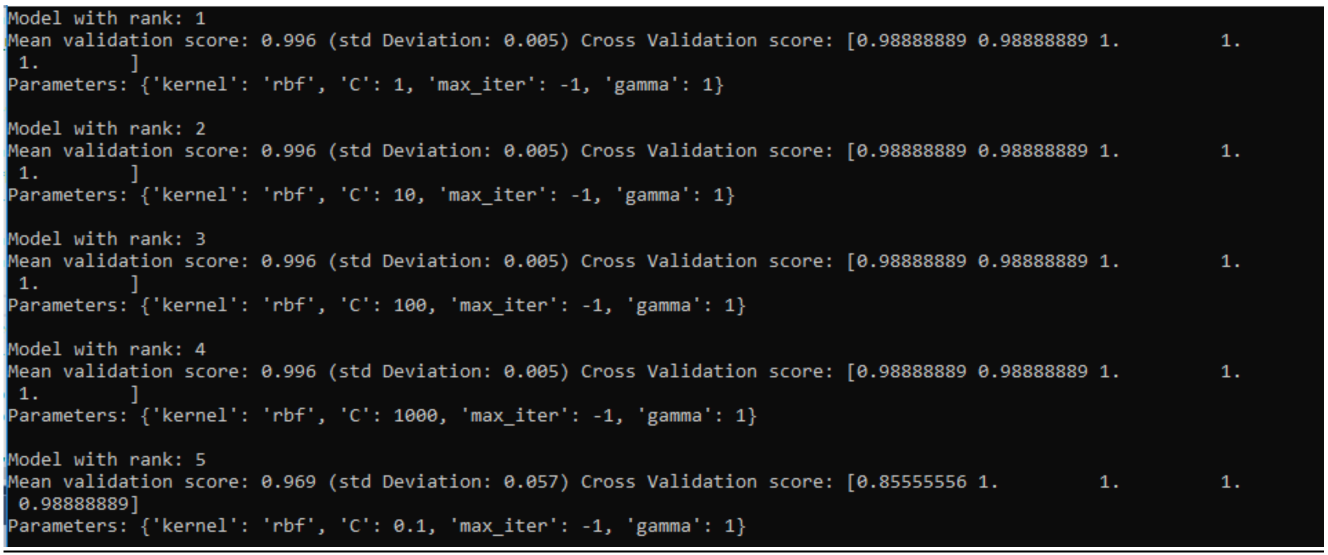
The above values give the best performance according to accuracy and also give best values for Mean Validation score and Cross Validation score. We get these best parameter value pairs by comparing all the models and selecting the model with highest performance rank.

In the RBF SVM, the parameters which immensely affect the performance are values of C and gamma, which are described as follows:

A high C tries to minimize the misclassification of training data and a low value tries to maintain a smooth classification boundary. Our value of C is neither too high like 10000 nor too low like 0.001. We had value of C in between 0.1 to 1000 but the best performance we got is when we selected C=1. Thus it provides perfect balance in describing decision boundary in linear SVM.

The larger the gamma, the narrow the gaussian "bell" is. The gamma parameter describes how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. Hence we chose a higher value for gamma so that we can reduce the misclassification error and hence achieved better performance.

Model with top Rankings:



```
Model with rank: 1
Mean validation score: 0.996 (std Deviation: 0.005) Cross Validation score: [0.98888889 0.98888889 1.         1.
 1.        ]
Parameters: {'kernel': 'rbf', 'C': 1, 'max_iter': -1, 'gamma': 1}

Model with rank: 2
Mean validation score: 0.996 (std Deviation: 0.005) Cross Validation score: [0.98888889 0.98888889 1.         1.
 1.        ]
Parameters: {'kernel': 'rbf', 'C': 10, 'max_iter': -1, 'gamma': 1}

Model with rank: 3
Mean validation score: 0.996 (std Deviation: 0.005) Cross Validation score: [0.98888889 0.98888889 1.         1.
 1.        ]
Parameters: {'kernel': 'rbf', 'C': 100, 'max_iter': -1, 'gamma': 1}

Model with rank: 4
Mean validation score: 0.996 (std Deviation: 0.005) Cross Validation score: [0.98888889 0.98888889 1.         1.
 1.        ]
Parameters: {'kernel': 'rbf', 'C': 1000, 'max_iter': -1, 'gamma': 1}

Model with rank: 5
Mean validation score: 0.969 (std Deviation: 0.057) Cross Validation score: [0.85555556 1.         1.         1.
 0.98888889]
Parameters: {'kernel': 'rbf', 'C': 0.1, 'max_iter': -1, 'gamma': 1}
```

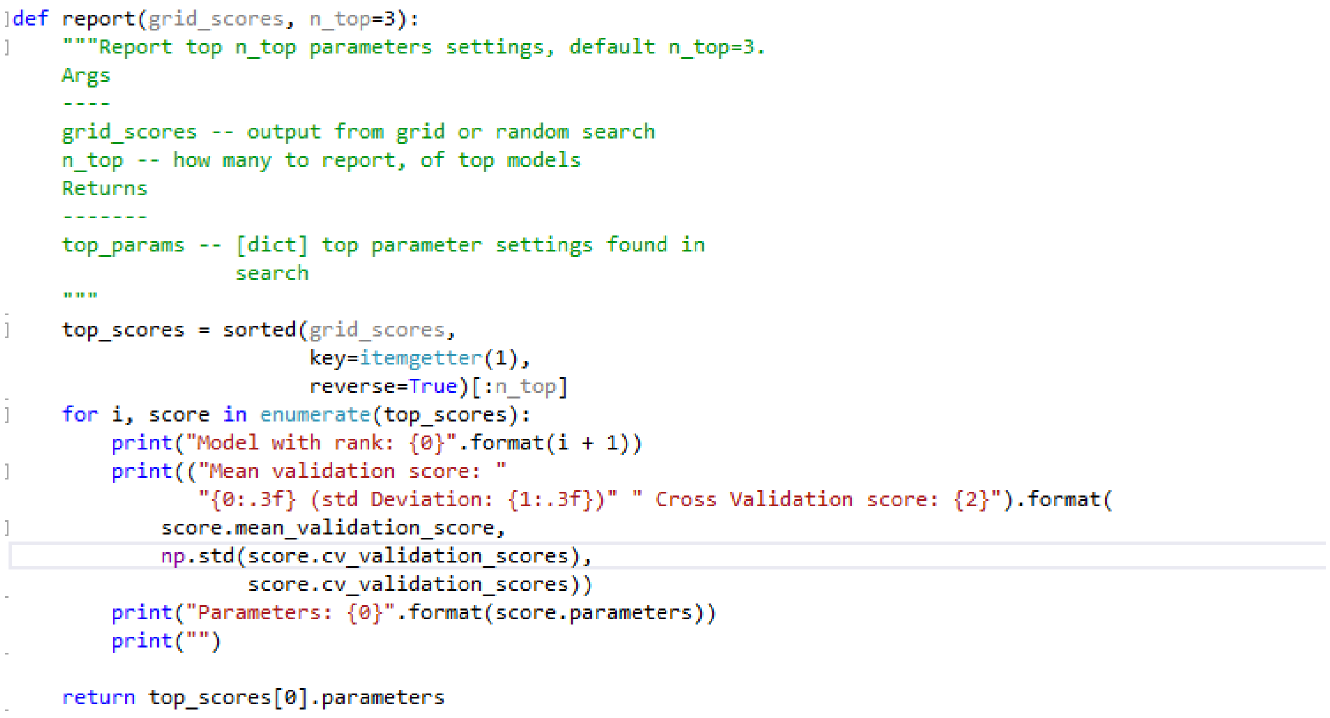
Hence for best parameters of 5-fold testing for rbf SVM we have chosen the parameter values from the model with Rank: 1.

<u>Used following range for tuning of the parameters:</u>

```
param_dist = {"kernel": ["rbf"],
              "max_iter":[-1,3],
              'C': [0.1,1,10,100,1000],
              'gamma': [1,0.1,0.01,0.001,0.0001]}
```

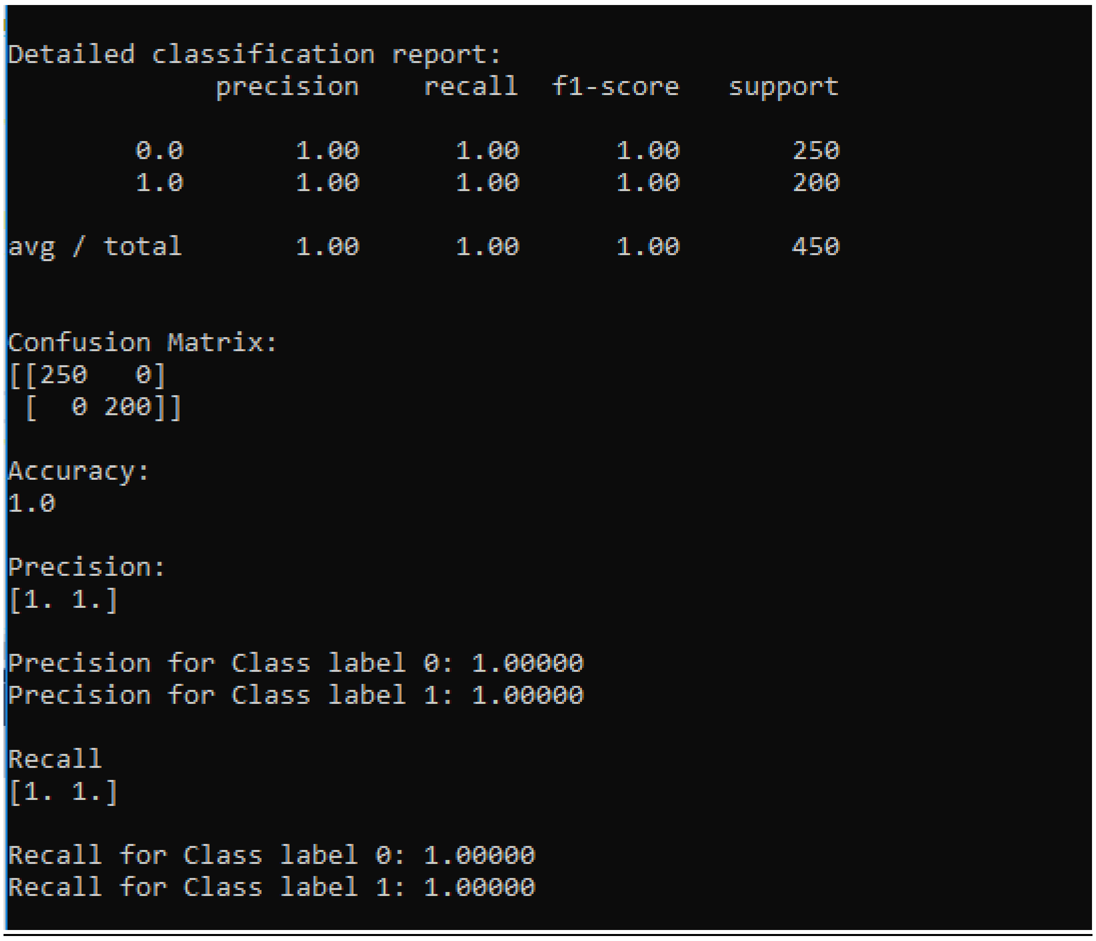
<u>Used following function to fine-tune the parameters:</u>

Used `GridSearchCV` and sorted functions to find out the model with top ranking. Used parameter settings of the model with top most rank.

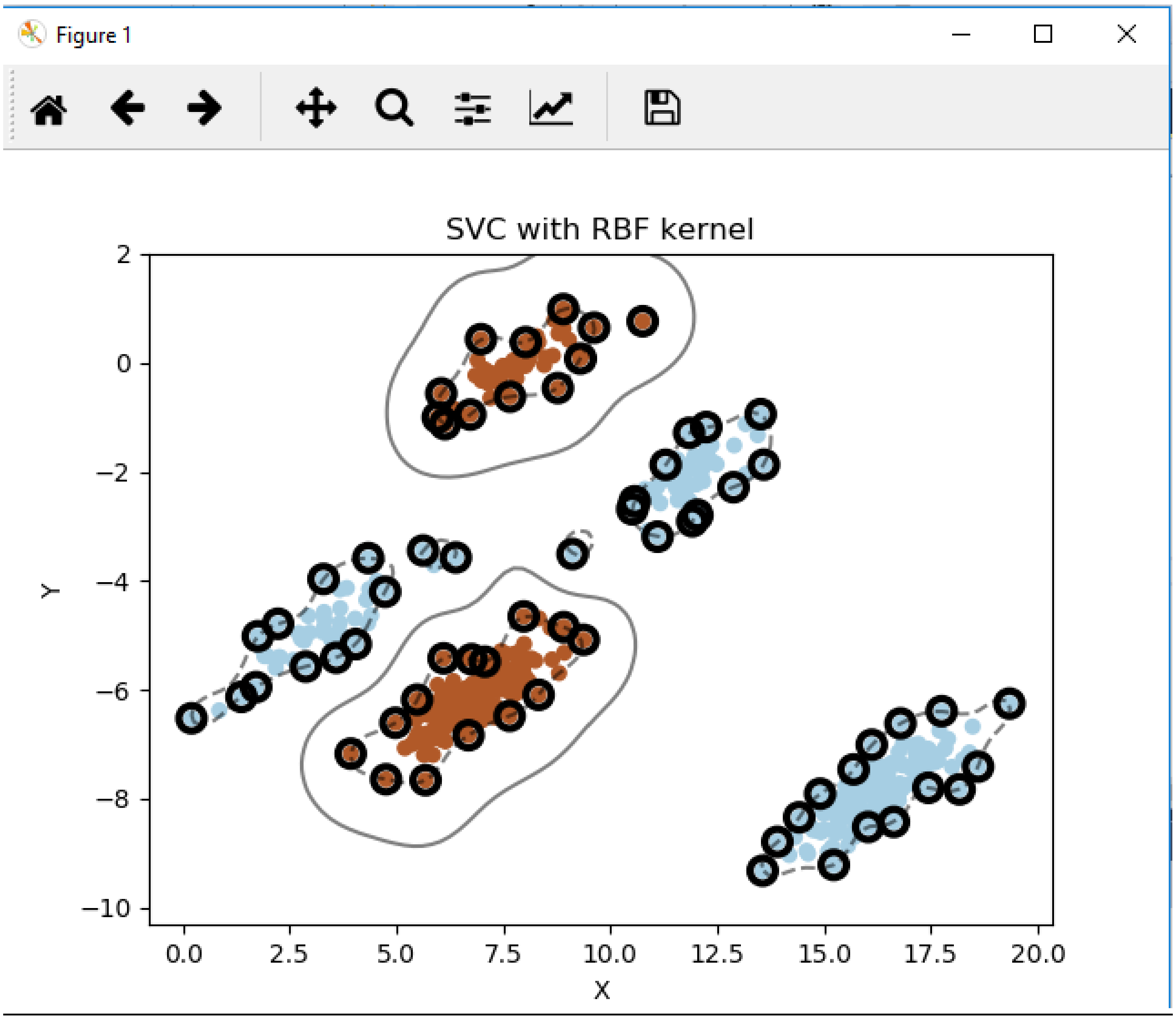


```
def report(grid_scores, n_top=3):
    """Report top n_top parameters settings, default n_top=3.
    Args
    ----
    grid_scores -- output from grid or random search
    n_top -- how many to report, of top models
    Returns
    -------
    top_params -- [dict] top parameter settings found in
                  search
    """
    top_scores = sorted(grid_scores,
                        key=itemgetter(1),
                        reverse=True)[:n_top]
    for i, score in enumerate(top_scores):
        print("Model with rank: {0}".format(i + 1))
        print(("Mean validation score: "
              "{0:.3f} (std Deviation: {1:.3f})" " Cross Validation score: {2}").format(
                score.mean_validation_score,
                np.std(score.cv_validation_scores),
                       score.cv_validation_scores))
        print("Parameters: {0}".format(score.parameters))
        print("")

    return top_scores[0].parameters
```

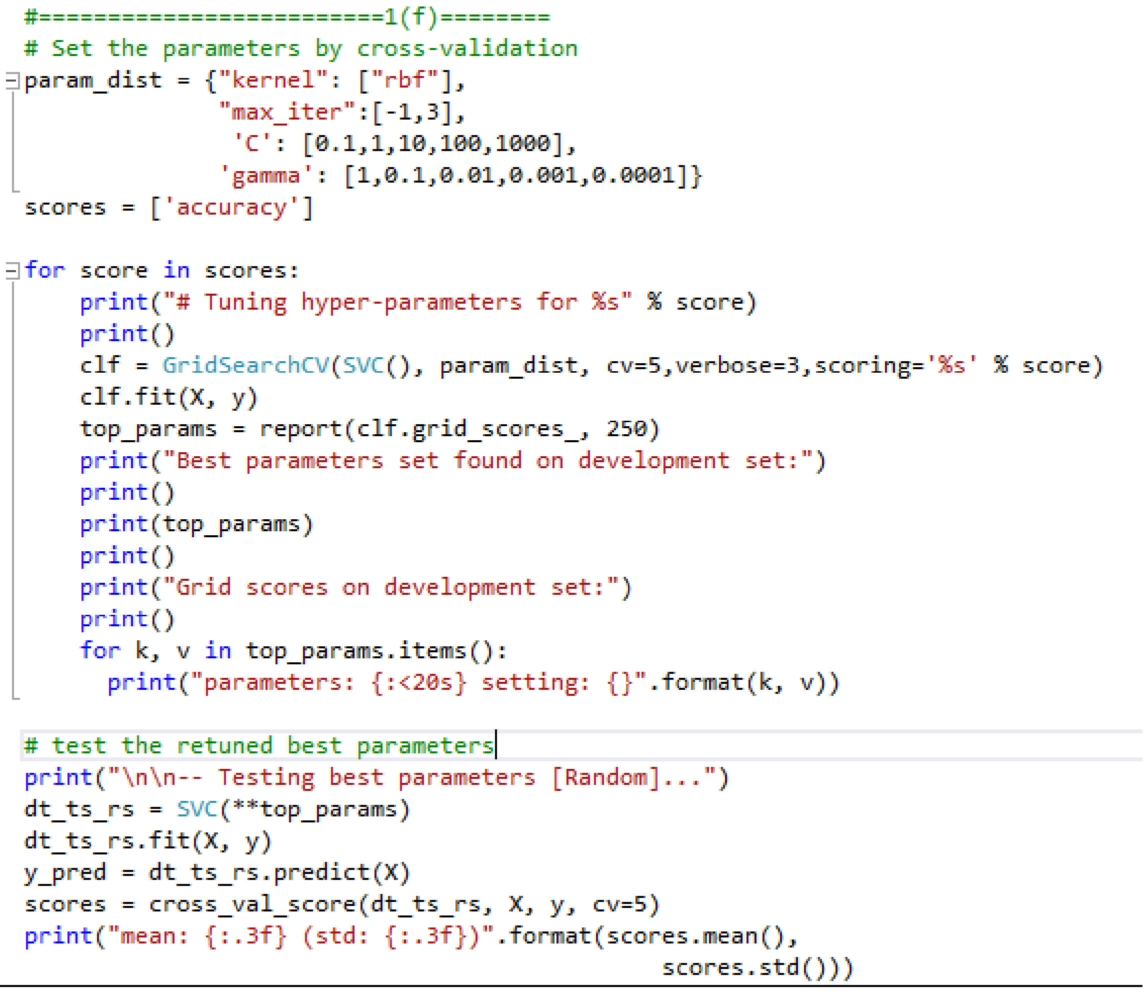
<u>Confusion matrix, accuracy, precision and recall:</u>


```
Detailed classification report:
             precision    recall  f1-score   support

        0.0       1.00      1.00      1.00       250
        1.0       1.00      1.00      1.00       200

avg / total       1.00      1.00      1.00       450


Confusion Matrix:
[[250   0]
 [  0 200]]

Accuracy:
1.0

Precision:
[1. 1.]

Precision for Class label 0: 1.00000
Precision for Class label 1: 1.00000

Recall
[1. 1.]

Recall for Class label 0: 1.00000
Recall for Class label 1: 1.00000
```

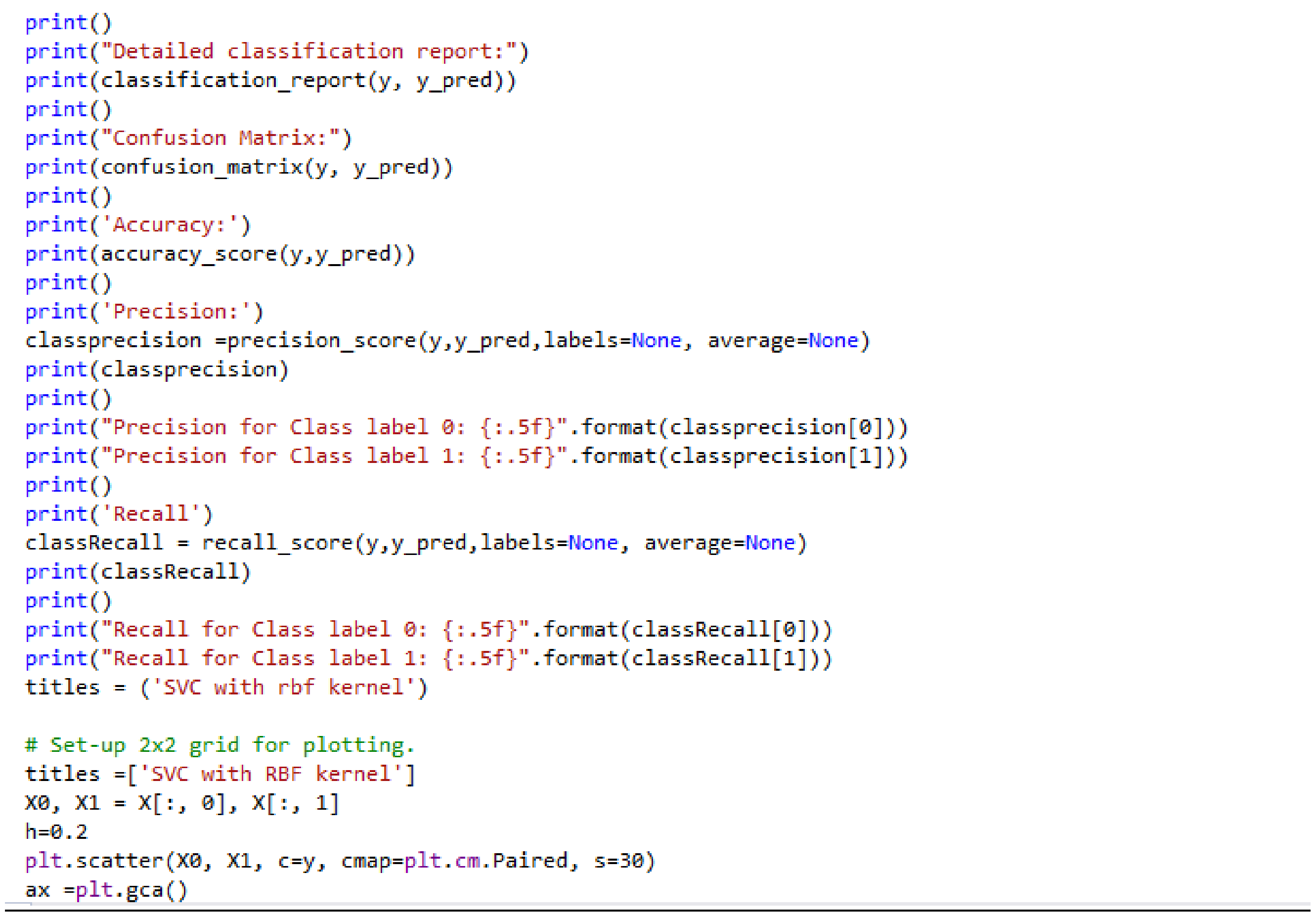
2D data plot for RBF kernel SVM:



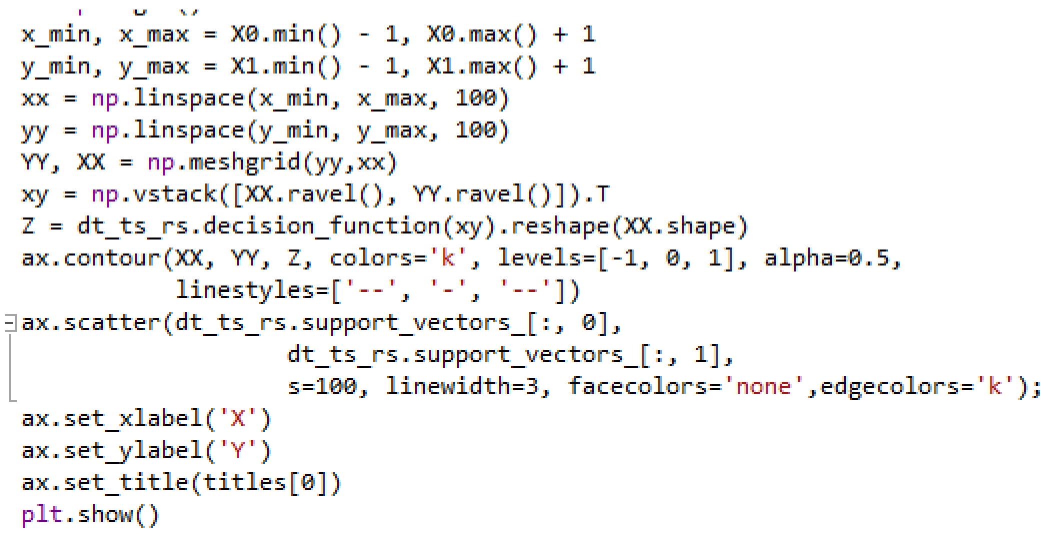
<u>Code used for RBF SVM:</u>



```
#=========================1(f)========
# Set the parameters by cross-validation
param_dist = {"kernel": ["rbf"],
              "max_iter":[-1,3],
               'C': [0.1,1,10,100,1000],
              'gamma': [1,0.1,0.01,0.001,0.0001]}
scores = ['accuracy']

for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
    print()
    clf = GridSearchCV(SVC(), param_dist, cv=5,verbose=3,scoring='%s' % score)
    clf.fit(X, y)
    top_params = report(clf.grid_scores_, 250)
    print("Best parameters set found on development set:")
    print()
    print(top_params)
    print()
    print("Grid scores on development set:")
    print()
    for k, v in top_params.items():
      print("parameters: {:<20s} setting: {}".format(k, v))

# test the retuned best parameters
print("\n\n-- Testing best parameters [Random]...")
dt_ts_rs = SVC(**top_params)
dt_ts_rs.fit(X, y)
y_pred = dt_ts_rs.predict(X)
scores = cross_val_score(dt_ts_rs, X, y, cv=5)
print("mean: {:.3f} (std: {:.3f})".format(scores.mean(),
                                              scores.std()))
```



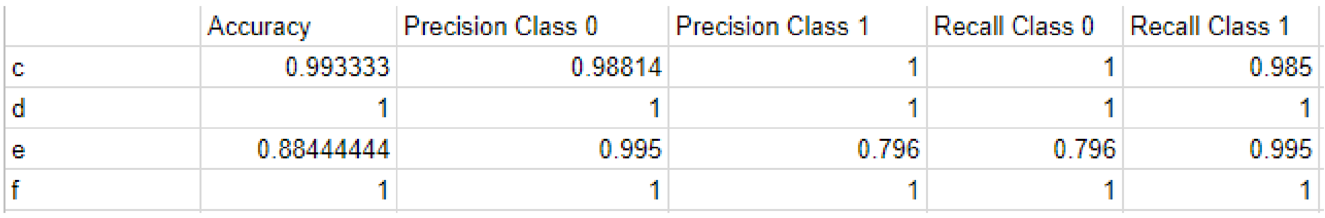
```
print()
print("Detailed classification report:")
print(classification_report(y, y_pred))
print()
print("Confusion Matrix:")
print(confusion_matrix(y, y_pred))
print()
print('Accuracy:')
print(accuracy_score(y,y_pred))
print()
print('Precision:')
classprecision =precision_score(y,y_pred,labels=None, average=None)
print(classprecision)
print()
print("Precision for Class label 0: {:.5f}".format(classprecision[0]))
print("Precision for Class label 1: {:.5f}".format(classprecision[1]))
print()
print('Recall')
classRecall = recall_score(y,y_pred,labels=None, average=None)
print(classRecall)
print()
print("Recall for Class label 0: {:.5f}".format(classRecall[0]))
print("Recall for Class label 1: {:.5f}".format(classRecall[1]))
titles = ('SVC with rbf kernel')

# Set-up 2x2 grid for plotting.
titles =['SVC with RBF kernel']
X0, X1 = X[:, 0], X[:, 1]
h=0.2
plt.scatter(X0, X1, c=y, cmap=plt.cm.Paired, s=30)
ax =plt.gca()
```



```
x_min, x_max = X0.min() - 1, X0.max() + 1
y_min, y_max = X1.min() - 1, X1.max() + 1
xx = np.linspace(x_min, x_max, 100)
yy = np.linspace(y_min, y_max, 100)
YY, XX = np.meshgrid(yy,xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = dt_ts_rs.decision_function(xy).reshape(XX.shape)
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])
ax.scatter(dt_ts_rs.support_vectors_[:, 0],
                     dt_ts_rs.support_vectors_[:, 1],
                     s=100, linewidth=3, facecolors='none',edgecolors='k');
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_title(titles[0])
plt.show()
```

g. Compare and contrast the performance and decision boundaries arrived at in parts (c), (d), (e) and (f) above.

Ans:



	Accuracy	Precision Class 0	Precision Class 1	Recall Class 0	Recall Class 1
c	0.993333	0.98814	1	1	0.985
d	1	1	1	1	1
e	0.88444444	0.995	0.796	0.796	0.995
f	1	1	1	1	1

We would choose part c i.e. Decision tree with 5 fold as the best decision boundary model.

Comparisons:

As per the above performance comparison, at first glance it seems that the intuitive decision tree (d part) and RBF kernel SVM (f part) gives best performance with highest Accuracy, Precision and Recall values but there might be following risks with these two:

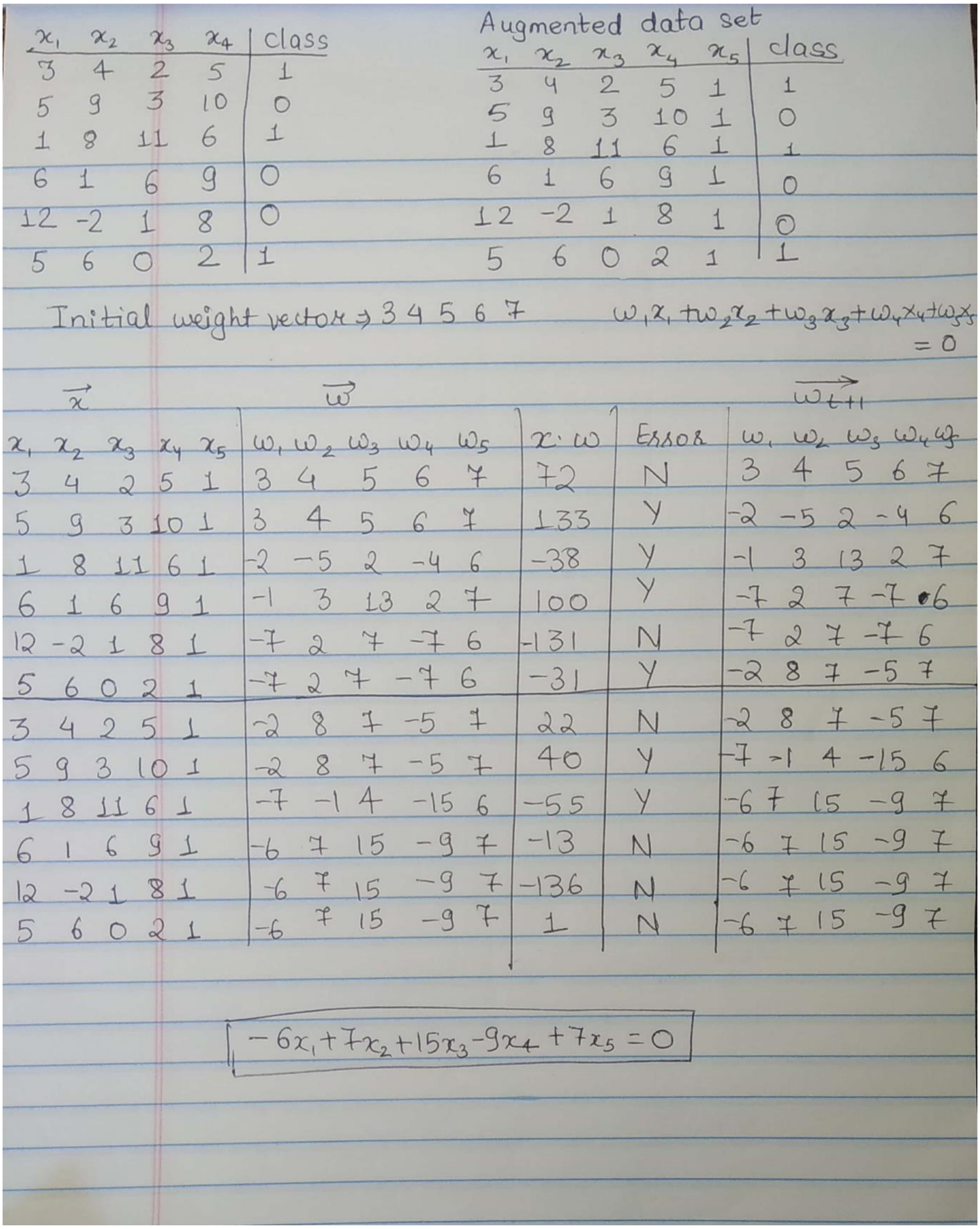
a) In this decision tree of part d there is a risk of overfitting because for accommodating two points of class 1, it creates two additional linear boundary. If those two points turn out to be noise points then there is a risk of misclassifying data points from class 0 to class 1.
b) In the RBF kernel approach the SVM tries to attain highest accuracy through classifying every data points and hence it also considers 4 scattered data points of class 0 as well. However there could be a slight chance that these points are outliers or noise points and hence this might affect future predictions of data points. This model does not create smooth decision boundaries.

Among the remaining parts i.e. c and e, we get higher accuracy, precision and recall values in c part and also get finely defined decision boundaries. Thus from the point of view of performance as well as better decision boundaries, the decision tree from part c gives best result.

Hence, we would choose the decision tree that we achieved in part(c) of the question because it ignores the noise points and gives better accuracy than Linear SVM.

2. Consider a dataset containing six 4-D points followed by their class labels, given as follows: (3 4 2 5; C1), (5 9 3 10; C0), (1 8 11 6; C1), (6 1 6 9: C0), (12, -2 1 8; C0), (5 6 0 2; C1). Show two full epochs of the perceptron training algorithm with these data points. Use (3 4 5 6 7) as the initial weight vector.

Ans:



x_1	x_2	x_3	x_4	class
3	4	2	5	1
5	9	3	10	0
1	8	11	6	1
6	1	6	9	0
12	-2	1	8	0
5	6	0	2	1

Augmented data set

x_1	x_2	x_3	x_4	x_5	class
3	4	2	5	1	1
5	9	3	10	1	0
1	8	11	6	1	1
6	1	6	9	1	0
12	-2	1	8	1	0
5	6	0	2	1	1

Initial weight vector ⇒ 3 4 5 6 7

$w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 = 0$

$\vec{x}$					$\vec{w}$							$\overrightarrow{w_{t+1}}$				
x_1	x_2	x_3	x_4	x_5	w_1	w_2	w_3	w_4	w_5	$x \cdot w$	Error	w_1	w_2	w_3	w_4	w_5
3	4	2	5	1	3	4	5	6	7	72	N	3	4	5	6	7
5	9	3	10	1	3	4	5	6	7	133	Y	-2	-5	2	-4	6
1	8	11	6	1	-2	-5	2	-4	6	-38	Y	-1	3	13	2	7
6	1	6	9	1	-1	3	13	2	7	100	Y	-7	2	7	-7	6
12	-2	1	8	1	-7	2	7	-7	6	-131	N	-7	2	7	-7	6
5	6	0	2	1	-7	2	7	-7	6	-31	Y	-2	8	7	-5	7
3	4	2	5	1	-2	8	7	-5	7	22	N	-2	8	7	-5	7
5	9	3	10	1	-2	8	7	-5	7	40	Y	-7	-1	4	-15	6
1	8	11	6	1	-7	-1	4	-15	6	-55	Y	-6	7	15	-9	7
6	1	6	9	1	-6	7	15	-9	7	-13	N	-6	7	15	-9	7
12	-2	1	8	1	-6	7	15	-9	7	-136	N	-6	7	15	-9	7
5	6	0	2	1	-6	7	15	-9	7	1	N	-6	7	15	-9	7

$$\boxed{-6x_1 + 7x_2 + 15x_3 - 9x_4 + 7x_5 = 0}$$

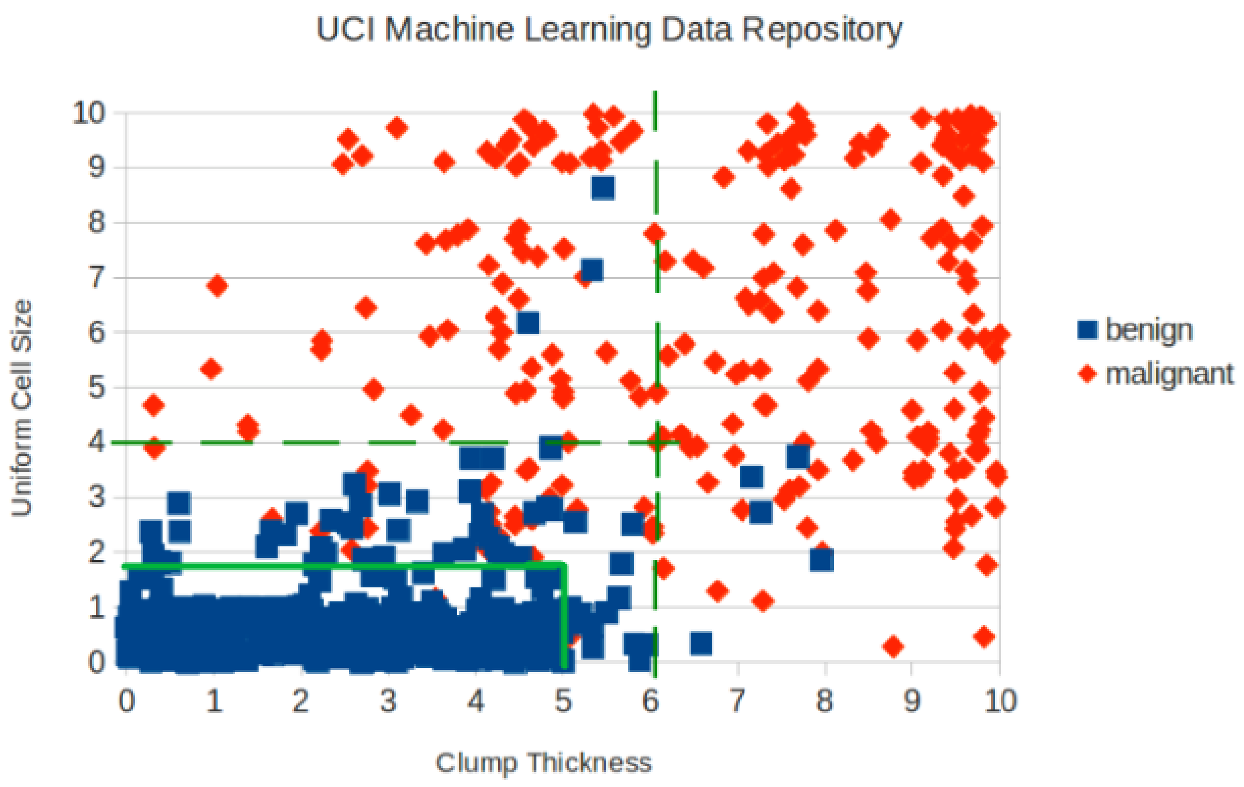
3. Consider the attached image for a real Breast-Cancer dataset. It shows a decision tree fitted to the data to separate malignant cases from the benign cases. The decision tree rule for classifying the benign cases is included in the image. This decision tree minimizes the number of misclassifications (maximum accuracy). Answer the following questions in the context of this dataset/image.
   a. (8) The cost of announcing a malignant case as benign is fifty times the cost of announcing a benign case as malignant. How would you adjust the boundaries of this

decision tree? Show by drawing the new lines on data plot image and writing the new rule for classifying a case as benign. Explain briefly why you chose the new boundaries.

b. (8) Show how you will change the boundaries of the original decision tree in the image to increase the precision of the benign class. Draw the new boundary lines and briefly explain why this causes the desired effect. How does this change affect the precision of the malignant class?

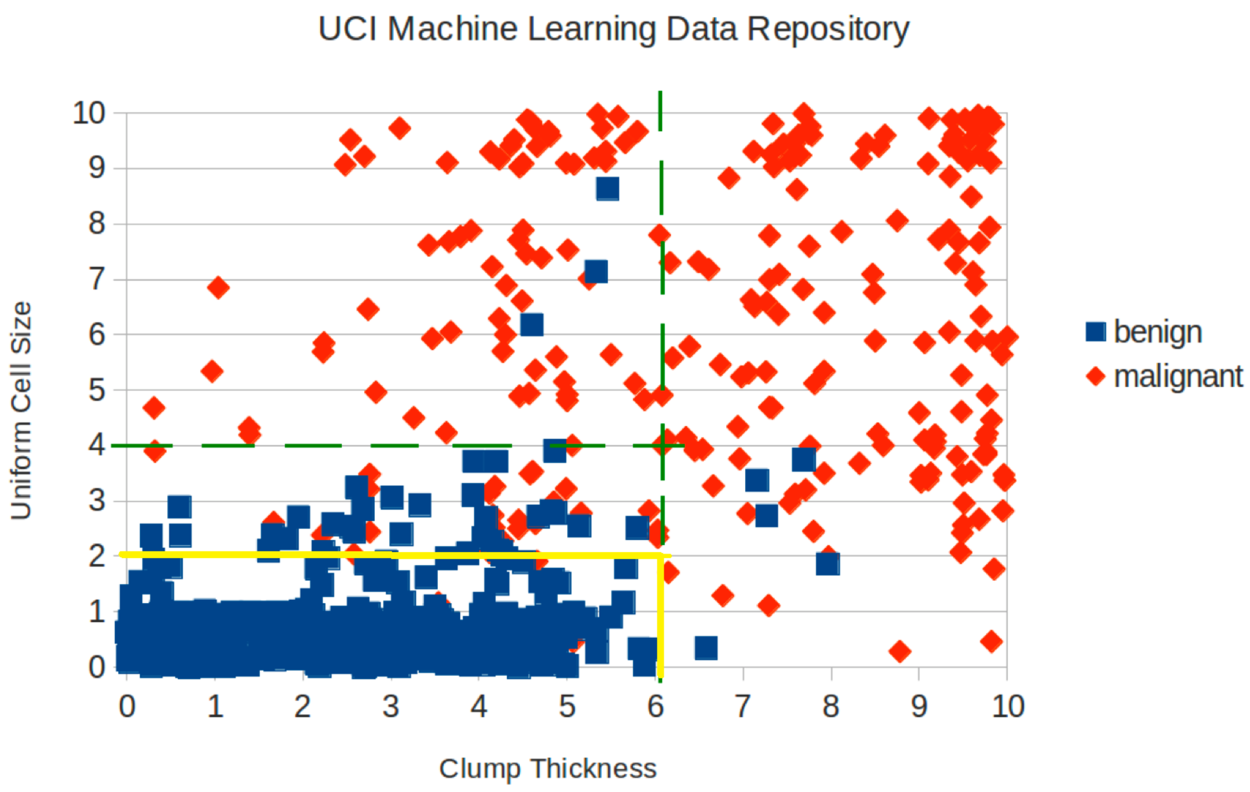
Ans:

a) New rule:
If "Clump thickness"< 5 and "Uniform Cell Size"<1.8 then sample is of the Benign class.
Decision tree with updated boundaries:



I have chosen these decision boundaries because the cost of declaring a malignant class as benign is 50 times as compared to declaring benign case as malignant. So, we need to give more importance to correctly capturing all the malignant cases. In the previous decision boundary a lot of points which belonged to malignant class were misclassified as benign because they were residing in region "Clump thickness" <= 6 and "Uniform cell size"<4. Since now the misclassification cost of Malignant class is much higher than Benign class so we need to correctly classify these points.
In "Clump thickness"< 5 and "Uniform cell size"<1.8, there is only one hidden point of malignant class is seen, we ignore this because considering this would make our decision tree overfitting and also we would mostly misclassify much more than 50 points of benign class so cost wise also we would not get advantage. Hence we have chosen our decision boundary as depicted in the above image and Rule.

b) Precision = TP/(TP+FP)
Hence to increase the precision we should try to minimize FP. Thus we need to find the boundary through which we can reduce the probability of announcing a malignant case as benign.
Updated decision boundary :



In the updated decision boundary, the region (2<Uniform Cell Size<4 and Clump Thickness< 6) reduces the False announcement of Malignant case as Benign class because this region contains considerable amount of data points from Malignant class, we assign this region to malignant class and hence reduce the False positive declaration of benign class.

The above decision boundary reduces the precision of Malignant class because it has increased benign class's points in malignant class region which would be falsely declared as malignant thereby increasing FP portion of Precision formula and hence decreasing the value of precision of malignant class.