Intelligent Data Analysis CS6052

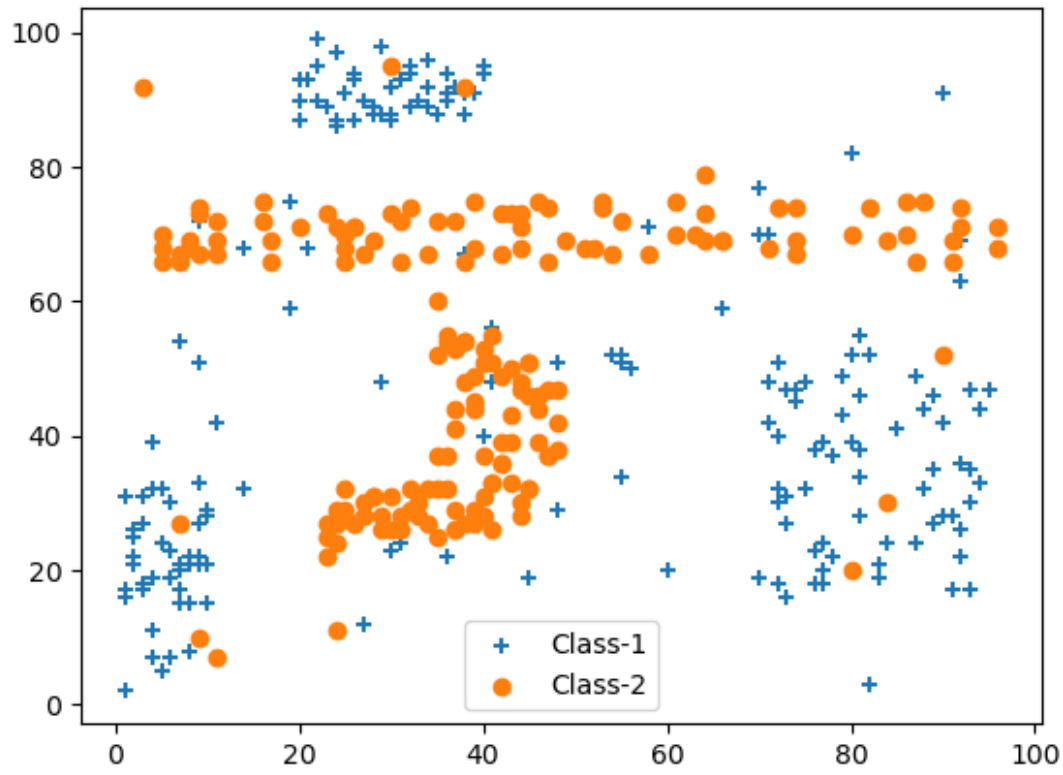
Homework #2

Disha Rao, M13254448

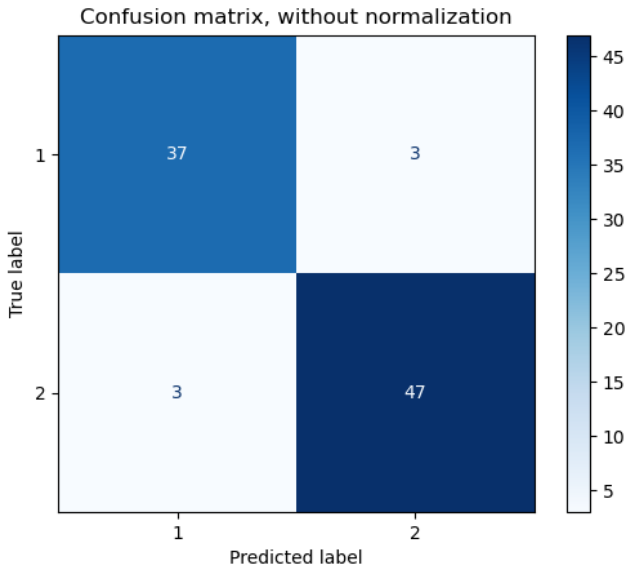
1a) Use non-linear SVM with Radial-basis kernel, to design classifier for the two classes of this dataset. Randomly select 75% of the data points as training data and use the remaining 25% as the test data. Create and test an SVM model for at least 6 different values of the regularization parameter (this parameter is named differently in each toolbox, read the documentation for your toolbox). For each of the six runs, show the chosen parameter value, confusion matrix, and accuracy, and precision values. Plot the accuracy, precision, and recall values for the six parameter values.

Solution:

Non-linear SVM with RBF kernel : Initial distribution



1) Regularization parameter, nu = 0.19
   Confusion matrix:



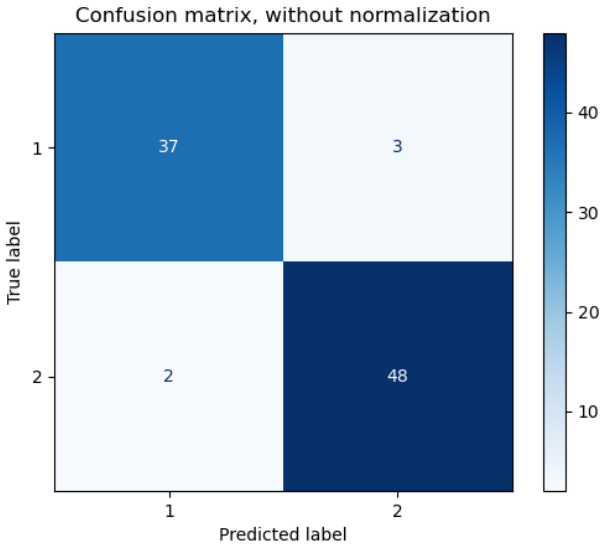
Accuracy: 93.33%

Precision: Class1 : 0.925, Class2: 0.94

---

2) Regularization parameter, nu = 0.23

Confusion matrix:



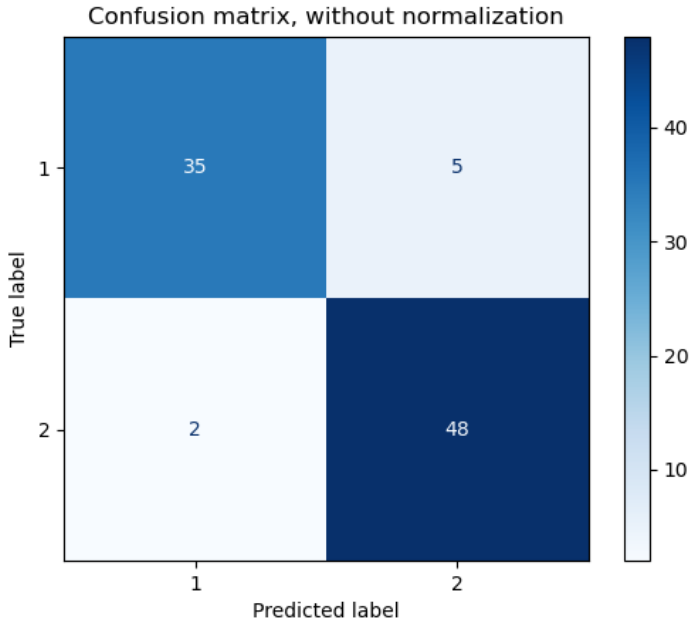
Accuracy: 94.44%

Precision: Class1: 0.948 Class2: 0.941

---

3) Regularization parameter, nu = 0.3

Confusion matrix:



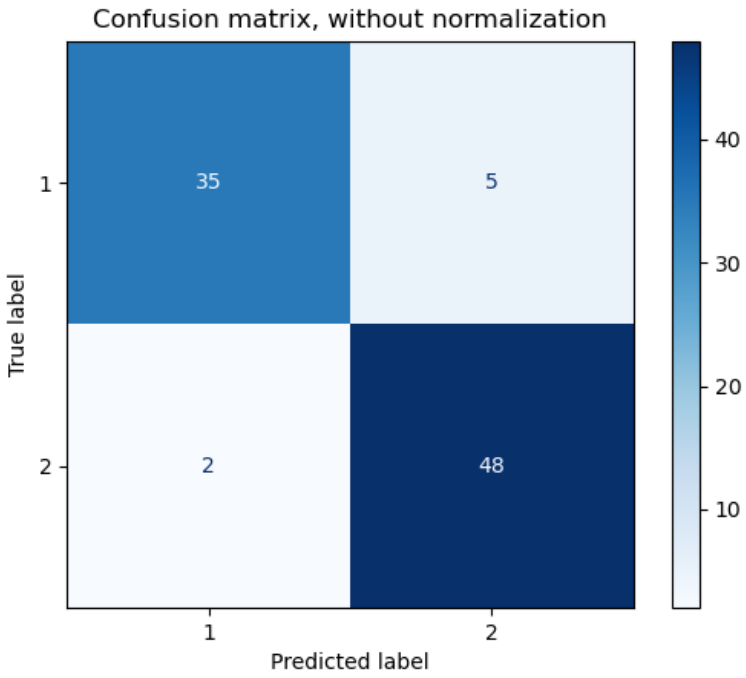
Accuracy: 92.22%

Precision: Class1: 0.945 Class2: 0.905

---

4) Regularization parameter, nu = 0.4

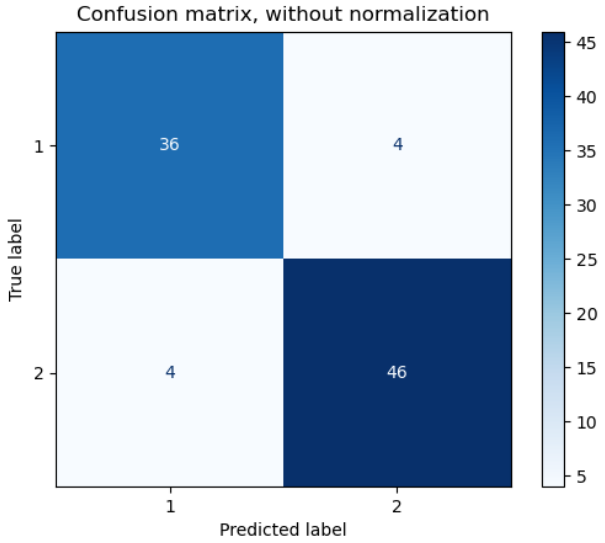
Confusion matrix:



Accuracy: 92.22%
Precision: Class1: 0.945 Class2: 0.905

---

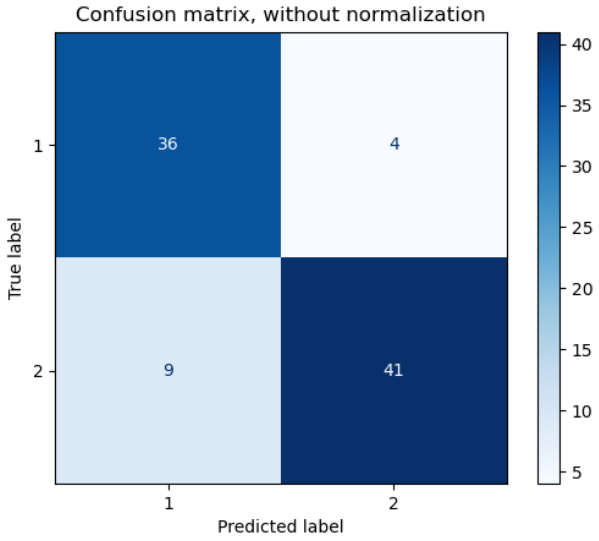
5) Regularization parameter, nu = 0.75
Confusion matrix:



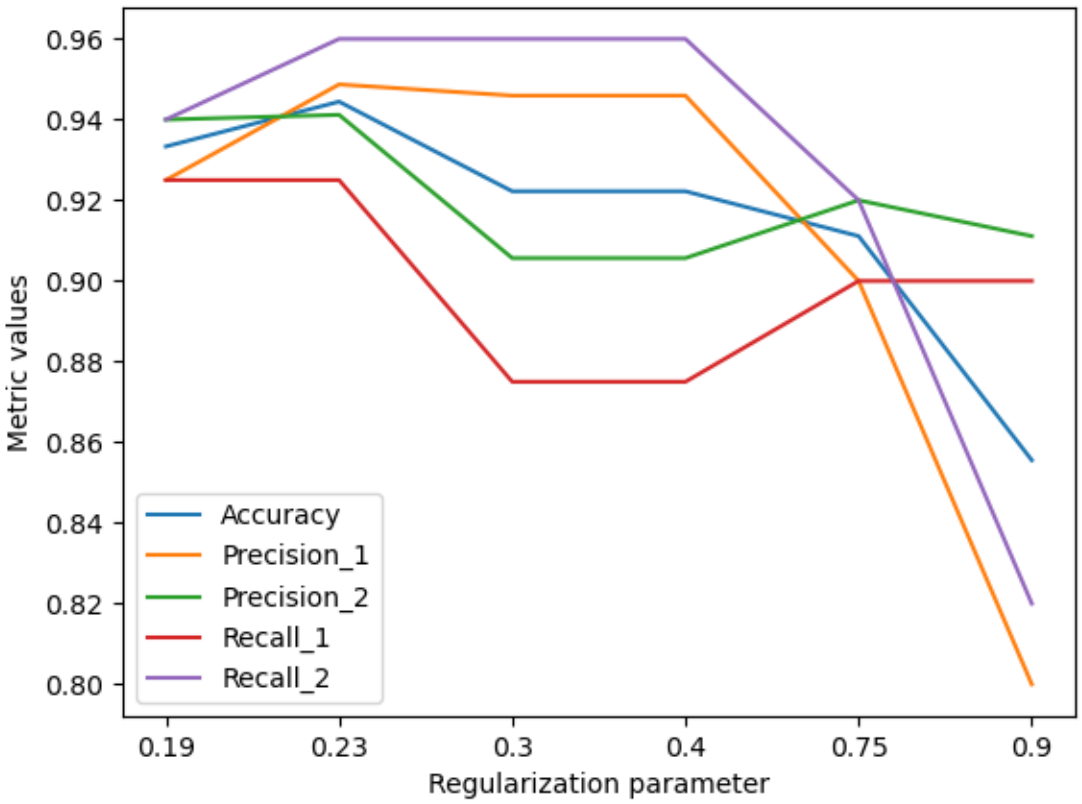
Accuracy: 91.11%
Precision: Class1: 0.9 Class2: 0.92

---

6) Regularization parameter, nu = 0.9
Confusion matrix:

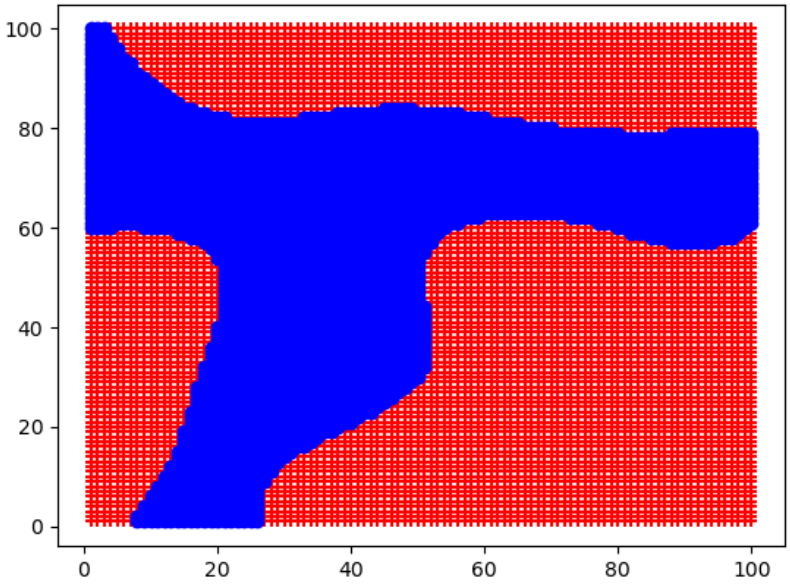


Accuracy: 85.55%
Precision: Class1: 0.8 Class2: 0.911



1b) Select the model having the highest accuracy value from the six models generate above. For this model consider all the points of the 100X100 grid as the test points. Find the class for each of these test points. Use two different colors for denoting the predicted class label for a point and create the plot with corresponding colors for each point of the full grid. Your output will be a grid in which each point will be plotted using one of the two colors.

Solution: The model with the highest accuracy is at regularization parameter = 0.23.



Source code :

```
1.  import pandas as pd
2.  from sklearn.svm import NuSVC
3.  from sklearn import metrics
4.  from sklearn.svm import SVC
5.  import sys, os
6.  import numpy as np
7.  import matplotlib.pyplot as plt
8.  from sklearn import svm
9.  from sklearn.model_selection import train_test_split, GridSearchCV
10. from sklearn.metrics import confusion_matrix,plot_confusion_matrix
11.
12. class_names=["1","2"]
13.
14. def read_data():
15.     data1_df = pd.read_excel("HW2Data.xlsx", usecols = "A,B,D", header = None, names =
    ['X1','X2','Y'])
16.     np_data = data1_df.to_numpy()
17.
18.     x1 = data1_df["X1"]
19.     x2 = data1_df["X2"]
20.
21.     X = np_data[:,[0,1]]
22.     y = np_data[:,2]
23.
24.     x1 = X[:,0]
25.     x2 = X[:,1]
26.
27.     plt.scatter(x1[y == 1],x2[y == 1], marker="+", label="Class-1")
28.     plt.scatter(x1[y == 2],x2[y == 2], marker="o", label="Class-2")
29.     plt.legend()
30.     plt.show()
31.
32.     return X,y
```

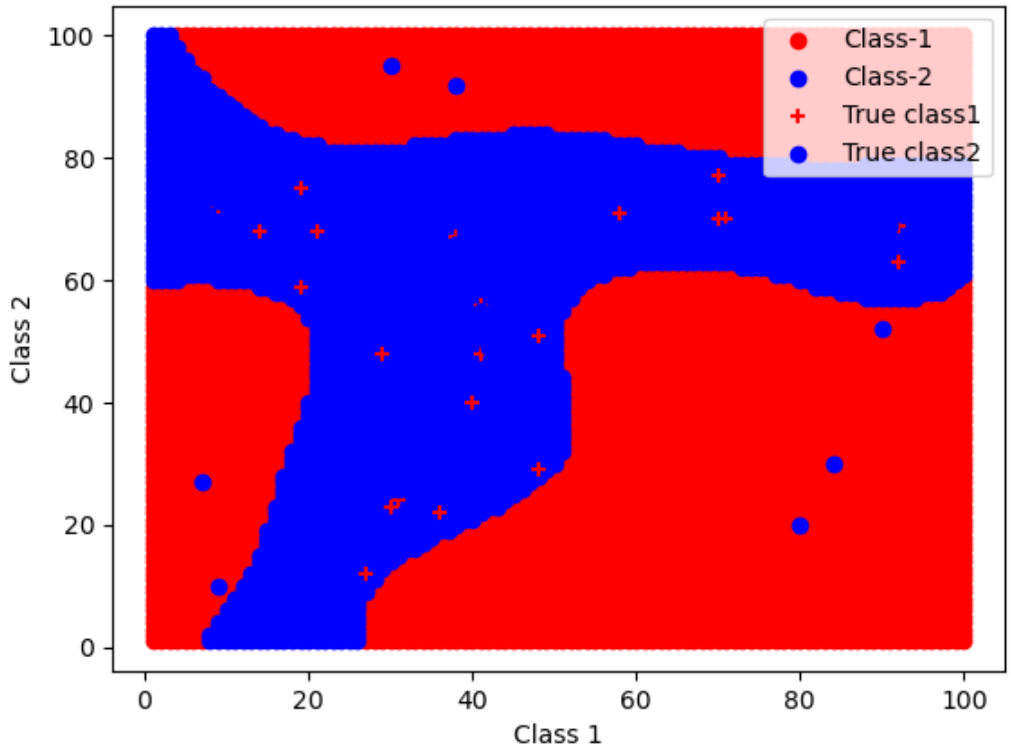
```
33.
34. def non_linear_svm_fit(nu:float):
35.     clf_nsvm = NuSVC(kernel = "rbf", nu=nu)
36.     clf_nsvm.fit(X_train,y_train)
37.
38.     # plot the decision function for each datapoint on the grid
39.     # Z = clf_nsvm.decision_function(np.c_[xx.ravel(), yy.ravel()])
40.
41.     y_pred = clf_nsvm.predict(X_test)
42.
43.     print("Accuracy: {}%".format(clf_nsvm.score(X_test, y_test)))
44.
45.     # Metrics for analysis of the prediction : Accuracy, precision, Recall
46.     metric_dict = metrics.classification_report(y_test, y_pred, target_names=class_name
    s, output_dict=True)
47.     accuracy_list.append(metric_dict["accuracy"])
48.     precision_1.append(metric_dict["1"]["precision"])
49.     precision_2.append(metric_dict["2"]["precision"])
50.     recall_1.append(metric_dict["1"]["recall"])
51.     recall_2.append(metric_dict["2"]["recall"])
52.
53.
54. def plotGraph():
55.     x_axis = ['0.19', '0.23', '0.3', '0.4', '0.75', '0.9']
56.     metric_df = pd.DataFrame(index=x_axis)
57.
58.     metric_df['Accuracy'] = accuracy_list
59.     metric_df['Precision_1'] = precision_1
60.     metric_df['Precision_2'] = precision_2
61.     metric_df['Recall_1'] = recall_1
62.     metric_df['Recall_2'] = recall_2
63.
64.     metric_plot = metric_df.plot()
65.     metric_plot.set_xlabel("Regularization parameter")
66.     metric_plot.set_ylabel("Metric values")
67.     # metric_plot.legend(bbox_to_anchor=(1.2, 0.5))
68.
69.     metric_plot.figure.savefig("question1_graph.png")
70.
71. # Plot non-normalized confusion matrix
72. def plot_confusion_matrix(clf_nsvm):
73.     titles_options = [("Confusion matrix, without normalization", None)]
74.                       # ("Normalized confusion matrix", 'true')]
75.     class_names = ["1","2"]
76.     for title, normalize in titles_options:
77.         disp = plot_confusion_matrix(clf_nsvm, X_test, y_test,
78.                                      display_labels=class_names,
79.                                      cmap=plt.cm.Blues,
80.                                      normalize=normalize)
81.         disp.ax_.set_title(title)
82.
83.         print(title)
84.         print(disp.confusion_matrix)
85.
86.     # plt.show()
87.
88. def question1b():
89.     # Creating mesh grid
90.     x = np.linspace(1, 100, num=100)
91.     y = np.linspace(1, 100, num=100)
92.
93.
94.
```

```
    xv, yv = np.meshgrid(x, y)

    df = pd.DataFrame(np.c_[xv.ravel(), yv.ravel()])

    print(df)

            clf_nsvm = NuSVC(kernel = "rbf", nu=0.23)
            clf_nsvm.fit(X_train,y_train)

            y_grid_pred = clf_nsvm.predict(df)

            x1 = X[:, 0]
            x2 = X[:, 1]

            plt.scatter(df[0][y_grid_pred == 1],df[1][y_grid_pred == 1],label='Class-1',c='r')
            plt.scatter(df[0][y_grid_pred == 2],df[1][y_grid_pred == 2],label='Class-2',c='b')
            plt.scatter(x1[y_copy == 1], x2[y_copy == 1], marker = "+", label = "True class1", c="r")
            plt.scatter(x1[y_copy == 2], x2[y_copy == 2], marker = "o", label = "True class2", c="b")
            plt.xlabel('Class 1')
            plt.ylabel('Class 2')
            plt.legend()
            plt.show()


        X, y = read_data()

        y_copy = y
        X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.25, random_state=0)

        nu_list = [0.19, 0.23, 0.3, 0.4, 0.75, 0.9]

        accuracy_list = list()
        precision_1 = list()
        precision_2 = list()
        recall_1 = list()
        recall_2 = list()

        for nu in nu_list:
            non_linear_svm_fit(nu)

        plotGraph()

        question1b()
```

1c(i) . How efficient and effective are these boundaries, given data distribution of the two classes?

Solution:

Below is the visualization for the misclassified points for the grid dataset:



There are 7 datapoints in Class-2 misclassified as Class-1 and 21 datapoints in Class-1 misclassified as Class-2.

To determine if the boundary is efficient, we can rely on the metrics. Since we don't have the true values of all the grid datapoints (9,640), we assume they may be true. Testing against existing true values (360 of them),

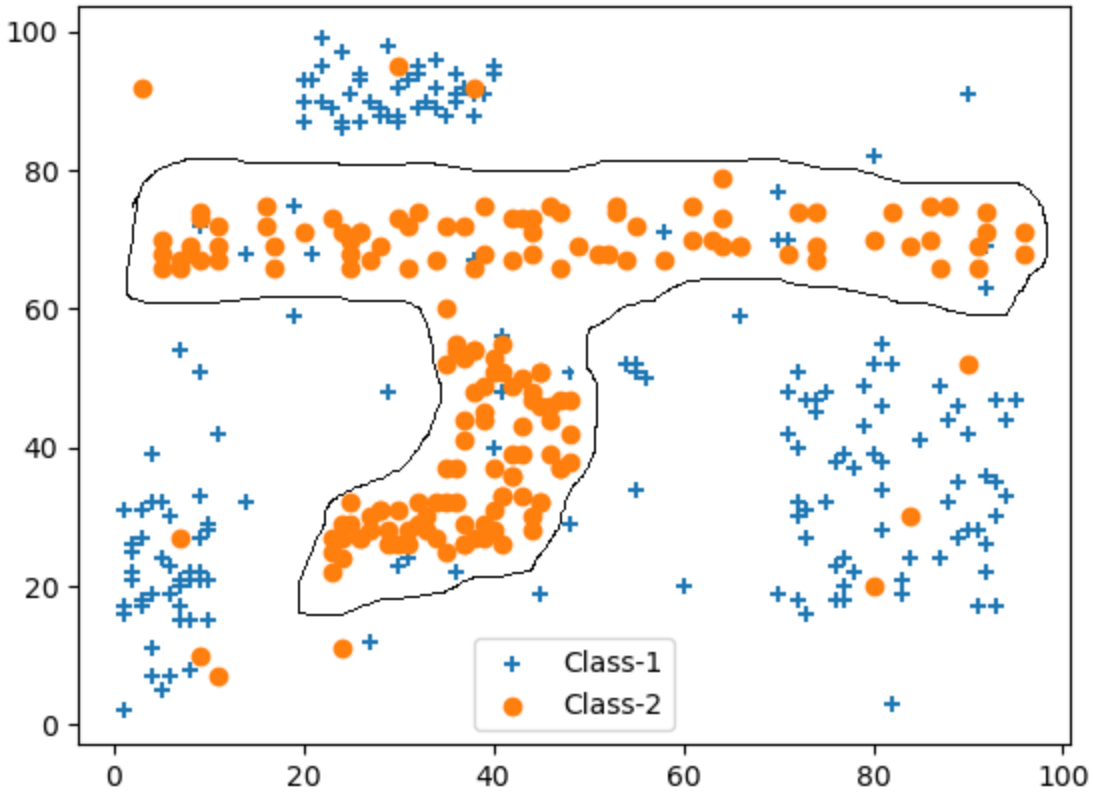
Accuracy : 94.4%
Precision(Class-1): 0.948
Recall (Class-1): 0.925

Precision(Class-2): 0.941
Recall(Class-2): 0.96

These values are reflective of how good the classifier is, and the above values indicate this is a good classifier. However, this isn't taking into account the TRUE ACTUAL values of the 9,640 points.

1c(ii). On a plot of original data points, draw the ideal boundaries that you would like to see from your intuitive point of view. Give reason for your choosing the boundaries that you have drawn.

Solution:



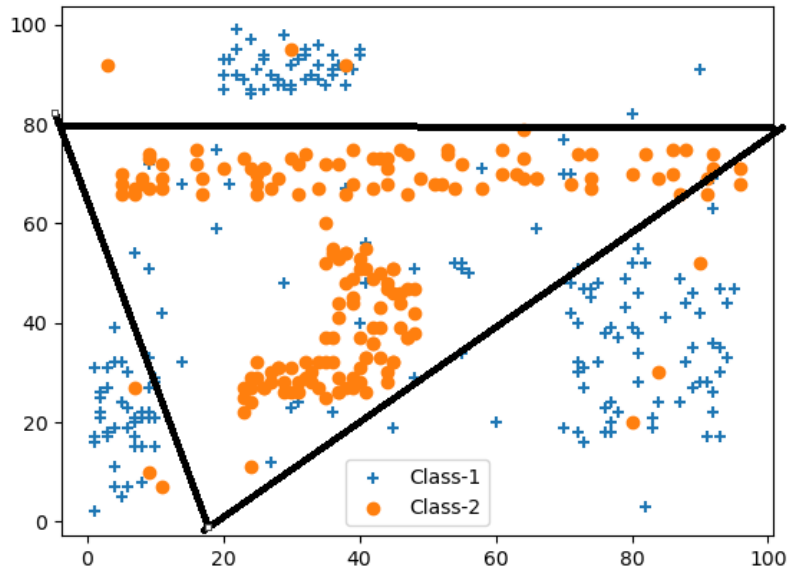
This would be a non-linear boundary. The boundary distinguishes the Class-2 datapoints from Class-1 datapoints. The reason I chose this classification is because of how densely the points are scattered in that specific region. However, there are a few outliers and there will be some classifications.
But, taking them into account will lead to overfitting.

1c(iii). Which classifier, in your view, can learn the boundaries closest to the intuitive ones you have suggested? Give reasons for your answer.

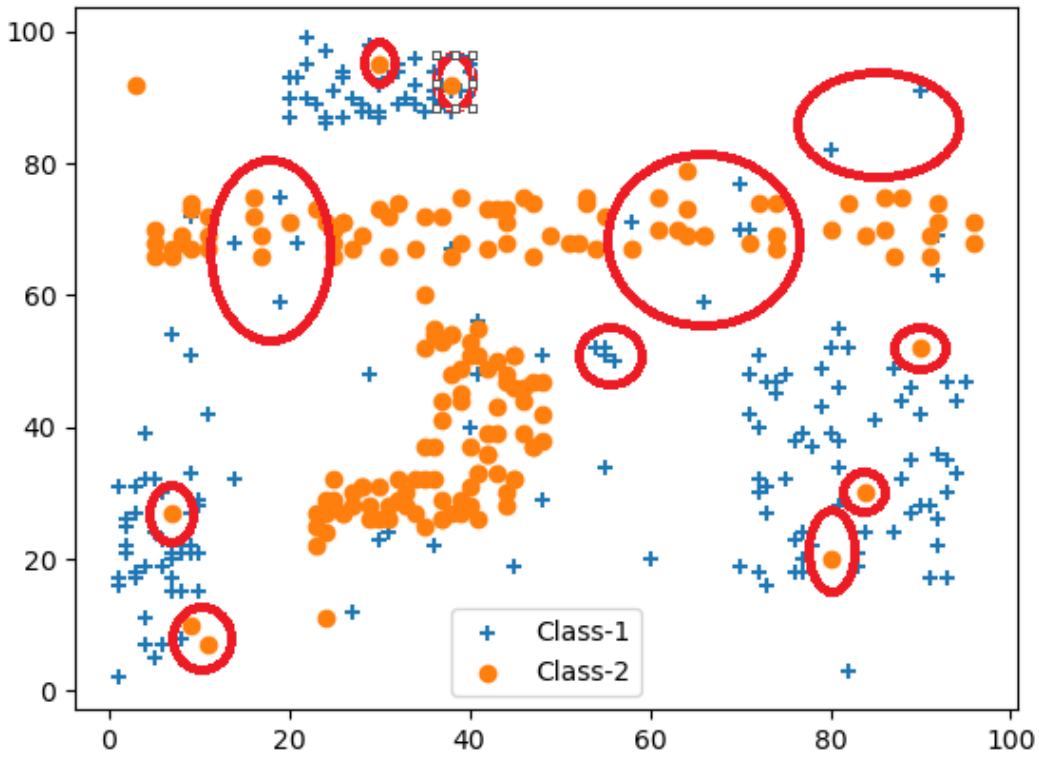
Solution:

Decision trees wouldn't be a good choice as data is scattered and it can lead to overfitting when learning the model.

Perceptrons would be able to classify them considerably okay. But, not the best. There are several outliers and the model learned can lead to wrong classification and underfitting.



K-nearest neighbours would do a fairly similar job when n = 9 (assume). The ones circled are the ones that suffer based on low values of k. But, this ends up memorizing the dataset and can lead to overfitting.



It is clear that a non-linear boundary would be the best to classify this non-linearly separable data. Non-linear SVMs would be very good at classifying this kind of data as they would map this dataset into a higher dimension and classify points. Hence, the best classifier would be non-linear SVMs.

1d) What is the role of the regularization parameter and how does it work?

Solution:

Programmatic perspective:

The parameter 'c' indicates the extent of misclassification that we would like to avoid in learning the model. For higher values of 'c', the margins of the hyperplane would be smaller if it means that no datapoint is left from being properly classified (dependent on the kind of data). For lower values of 'c', the margins chosen are wider as misclassifications are believed to be okay.

Mathematical Perspective:

The objective function in SVMs is to keep the margins as wide as possible, so that the distinction is clear.

$$L(w) = \frac{\|\vec{w}\|^2}{2} + C\left(\sum_{i=1}^{N} \xi_i^k\right)$$

Minimize 2 terms : w^2/2 and the second term, where ξ_i^k refers to the misclassification error and 'C' is the cost of misclassification. If 'C' is high , no misclassifications allowed ; this leads to overfitting inevitably. If C = 0, it allows misclassifications with no penalty and leads to underfitting.