
DS203 : EXERCISE 4

Nirav Bhattad

Last updated September 8, 2024

Pre-requisites

Importing Libraries

I imported the necessary libraries for this assignment, like `pandas` for data manipulation, `numpy` for numerical operations, `matplotlib` for plotting, and `sklearn` for machine learning models and metrics. I also imported `os` to create directories for saving the plots and metrics.

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.model_selection import train_test_split
5 from sklearn.linear_model import LogisticRegression
6 from sklearn.svm import SVC
7 from sklearn.ensemble import RandomForestClassifier
8 from sklearn.neural_network import MLPClassifier
9 from sklearn.metrics import classification_report, accuracy_score, roc_auc_score, roc_curve
10 import os
```

Listing 1: Importing Libraries

Creating Directories

Here, I created two directories, `Images` and `Metrics`, to save the plots and metrics generated in the subsequent steps.

```
1 if not os.path.exists('Images'):
2     os.makedirs('Images')
3
4 if not os.path.exists('Metrics'):
5     os.makedirs('Metrics')
```

Listing 2: Creating Directories

Loading the Data

I loaded the data from the 3 CSV files into `pandas DataFrame`. Each file contains 1440 rows and 3 columns. First 2 columns give the location of the point, and the last column gives the class of the point.

```
1 data_v0 = pd.read_csv('clusters-4-v0.csv')
2 data_v1 = pd.read_csv('clusters-4-v1.csv')
3 data_v2 = pd.read_csv('clusters-4-v2.csv')
```

Listing 3: Loading the Data

Step 1

Step 1

Divide each data set into ‘train’ and ‘test’ datasets, once, and use them for all subsequent steps.

To divide the data into ‘train’ and ‘test’ datasets, I used the `train_test_split` method from the `sklearn.model_selection` module. I divided the data into 80% training and 20% testing datasets. I used the same split for all the subsequent steps.

```

1 x_train_v0, x_test_v0, y_train_v0, y_test_v0 = train_test_split(data_v0[['x1', 'x2']], data_v0['y'],
2   test_size=0.2, random_state=42)
3 x_train_v1, x_test_v1, y_train_v1, y_test_v1 = train_test_split(data_v1[['x1', 'x2']], data_v1['y'],
4   test_size=0.2, random_state=42)
5 x_train_v2, x_test_v2, y_train_v2, y_test_v2 = train_test_split(data_v2[['x1', 'x2']], data_v2['y'],
6   test_size=0.2, random_state=42)
```

Listing 4: Dividing the Data into Train and Test Datasets

Step 2

Step 2

Review the data using appropriate plots and understand the overall structure of the data. Comment on the data and anticipate how well LogisticRegression will perform on the data.

This is how I plotted the 3 datasets

```

1 datasets = [(data_v0, "Dataset 1"),
2   (data_v1, "Dataset 2"),
3   (data_v2, "Dataset 3")]
4
5 for i, (data, label) in enumerate(datasets):
6   plt.figure(figsize=(16, 10))
7
8   plt.scatter(data['x1'], data['x2'], c=data['y'], cmap='viridis', s=10)
9   plt.title(f'Scatter Plot of {label}')
10  plt.xlabel('x1')
11  plt.ylabel('x2')
12
13  plt.tight_layout()
14  plt.savefig(f'Images/dataset-{i+1}-overview.png', dpi=400)
15  plt.show()
16
17  class_counts = data['y'].value_counts()
18  print(f"Class balance in {label}:\n{class_counts}\n")
```

Listing 5: Plotting the Datasets

Remark

All Images generated throughout the report are saved in the `Images` folder.

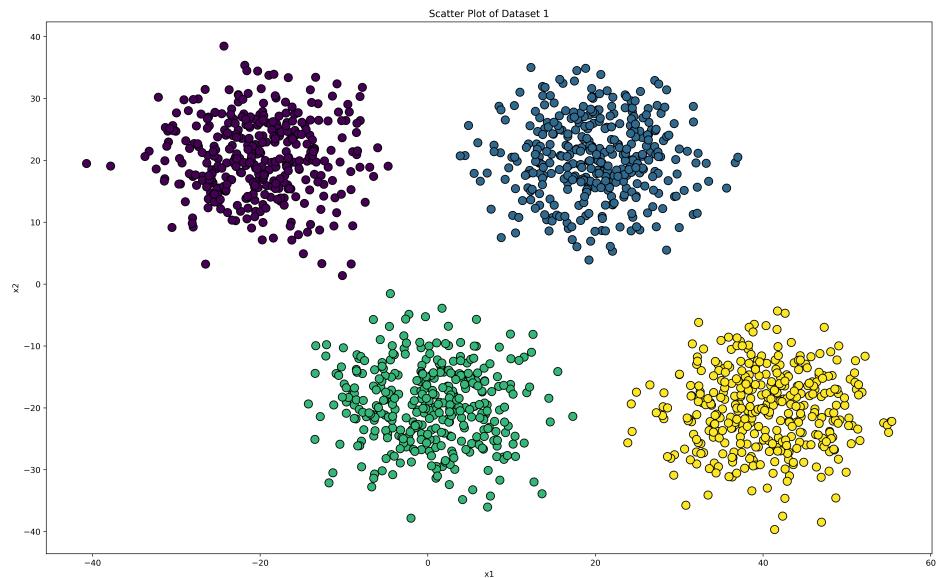


Figure 1: Overview of Data Set 1

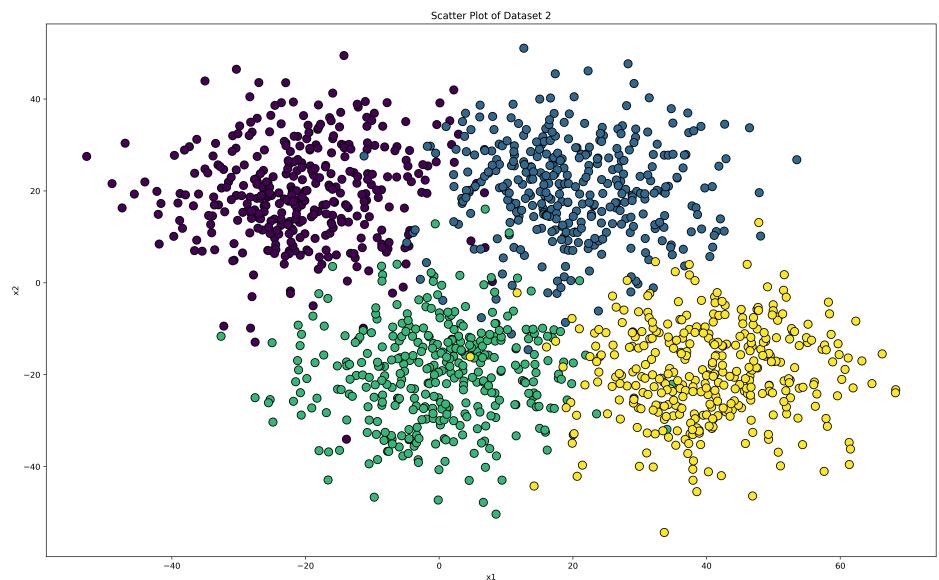


Figure 2: Overview of Data Set 2

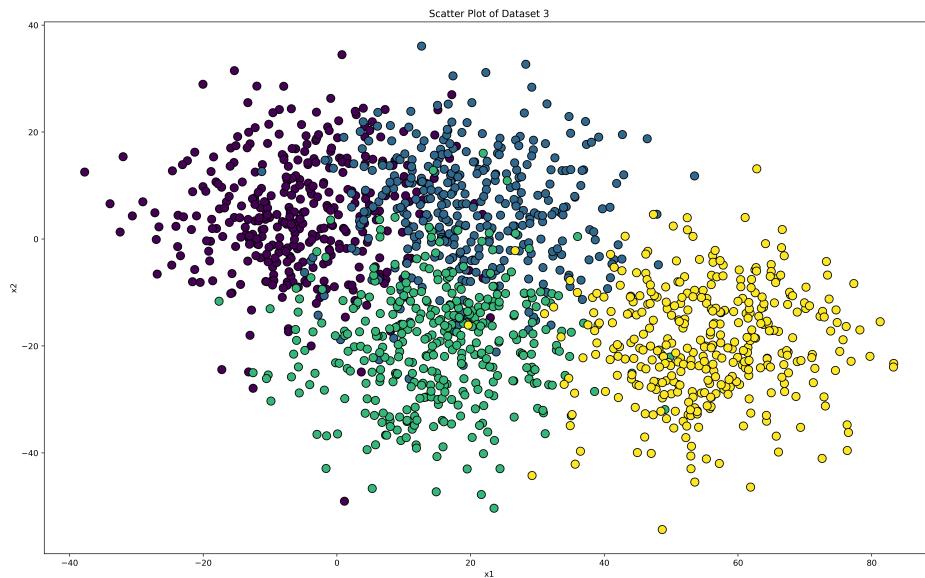


Figure 3: Overview of Data Set 3

We see that in each dataset, there are 360 points. The data is balanced in all the datasets. The data in Dataset 1 is linearly separable, the data in Dataset 2 is not linearly separable, and the data in Dataset 3 is more complex than the data in Dataset 2.

This is how I plotted the training and testing datasets.

```

1 plt.figure(figsize=(16, 6))
2 plt.subplot(1, 2, 1)
3 plt.scatter(x_train_v0['x1'], x_train_v0['x2'], c=y_train_v0, cmap='viridis', s=10)
4 plt.title('Training data for Dataset 1')
5
6 plt.subplot(1, 2, 2)
7 plt.scatter(x_test_v0['x1'], x_test_v0['x2'], c=y_test_v0, cmap='viridis', s=10)
8 plt.title('Test data for Dataset 1')
9 plt.savefig('Images/dataset-1-train-test.png', dpi=400)
10 plt.show()
11
12 plt.figure(figsize=(16, 6))
13 plt.subplot(1, 2, 1)
14 plt.scatter(x_train_v1['x1'], x_train_v1['x2'], c=y_train_v1, cmap='viridis', s=10)
15 plt.title('Training data for Dataset 2')
16
17 plt.subplot(1, 2, 2)
18 plt.scatter(x_test_v1['x1'], x_test_v1['x2'], c=y_test_v1, cmap='viridis', s=10)
19 plt.title('Test data for Dataset 2')
20 plt.savefig('Images/dataset-2-train-test.png', dpi=400)
21 plt.show()
22
23 plt.figure(figsize=(16, 6))
24 plt.subplot(1, 2, 1)
25 plt.scatter(x_train_v2['x1'], x_train_v2['x2'], c=y_train_v2, cmap='viridis', s=10)
26 plt.title('Training data for Dataset 3')
27
28 plt.subplot(1, 2, 2)
29 plt.scatter(x_test_v2['x1'], x_test_v2['x2'], c=y_test_v2, cmap='viridis', s=10)
30 plt.title('Test data for Dataset 3')
31 plt.savefig('Images/dataset-3-train-test.png', dpi=400)
32 plt.show()
```

Listing 6: Plotting the Training and Testing Datasets

This is how the test and train datasets look for each data set.

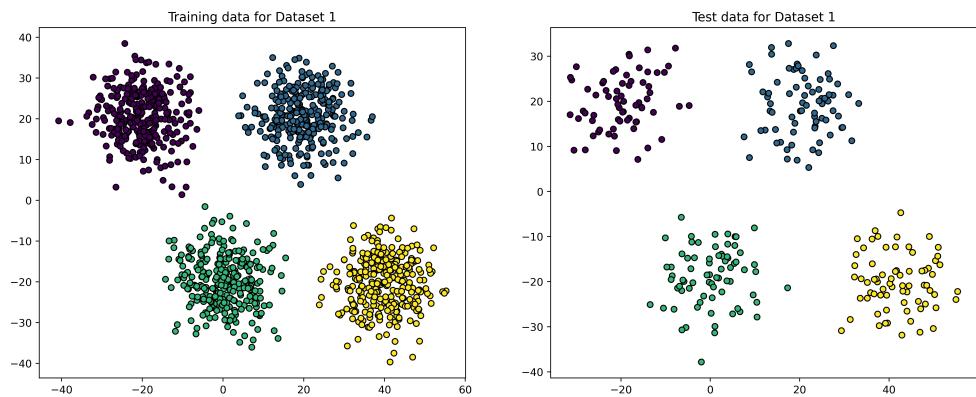


Figure 4: Training and Testing Datasets for Data Set 1

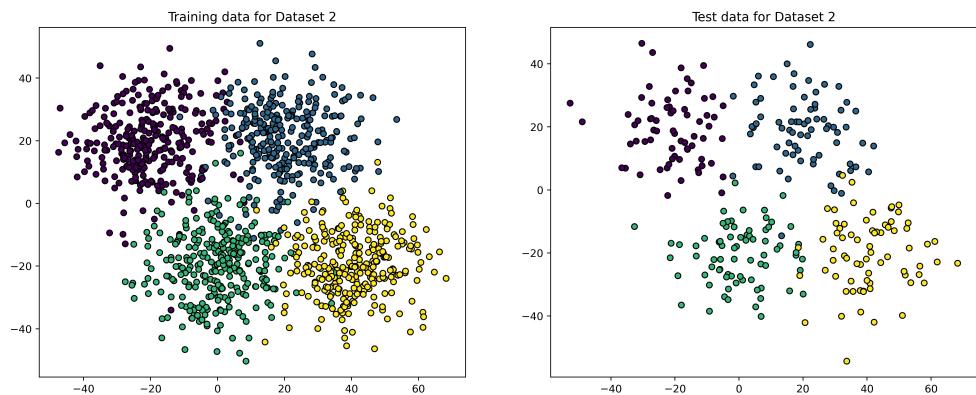


Figure 5: Training and Testing Datasets for Data Set 2

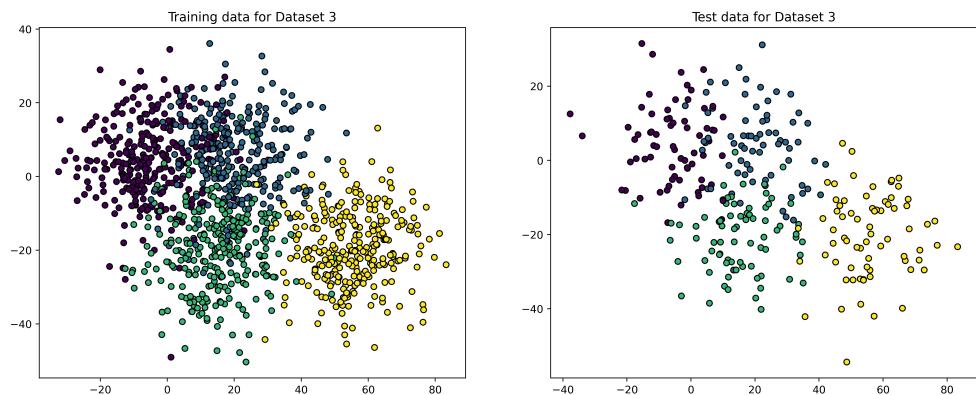


Figure 6: Training and Testing Datasets for Data Set 3

Reviewing the plots above, we can see that the training data and the test data are well balanced in all the 3 situations. The data is somewhat linearly separable in Dataset 1, but not so much in Dataset 2 and Dataset 3. The data in Dataset 3 is more complex than in Dataset 2.

Logistic Regression will perform well on Dataset 1 as it is linearly separable. It will perform poorly on Dataset 2 and Dataset 3 as they are not linearly separable. We can expect that the model will have a high accuracy on Dataset 1 and a low accuracy on Dataset 2 and Dataset 3.

Step 3

Step 3

Use the following algorithms / variants to process the datasets:

- a) Logistic Regression
- b) SVC – with ‘linear’ kernel (what is ‘linear’?)
- c) SVC – with ‘rbf’ kernel (what is ‘rbf’?)
- d) Random Forest Classifier – with `min_samples_leaf=1`
- e) Random Forest Classifier – with `min_samples_leaf=3`
- f) Random Forest Classifier – with `min_samples_leaf=5`
- g) Neural Network Classifier – with `hidden_layer_sizes=(5)`
- h) Neural Network Classifier – with `hidden_layer_sizes=(5,5)`
- i) Neural Network Classifier – with `hidden_layer_sizes=(5,5,5)`
- j) Neural Network Classifier – with `hidden_layer_sizes=(10)`

The list of classifiers is shown in Listing 9. The classifiers are initialized with very few parameters. The classifiers are:

```

1 classifiers = {
2     'Logistic Regression': LogisticRegression(),
3     'SVC Linear': SVC(kernel='linear', probability=True),
4     'SVC RBF': SVC(kernel='rbf', probability=True),
5     'Random Forest (min_samples_leaf=1)': RandomForestClassifier(min_samples_leaf=1),
6     'Random Forest (min_samples_leaf=3)': RandomForestClassifier(min_samples_leaf=3),
7     'Random Forest (min_samples_leaf=5)': RandomForestClassifier(min_samples_leaf=5),
8     'Neural Network (hidden_layer_sizes=(5,))': MLPClassifier(hidden_layer_sizes=(5,)),
9     'Neural Network (hidden_layer_sizes=(5,5))': MLPClassifier(hidden_layer_sizes=(5,5)),
10    'Neural Network (hidden_layer_sizes=(5,5,5))': MLPClassifier(hidden_layer_sizes=(5,5,5)),
11    'Neural Network (hidden_layer_sizes=(10,))': MLPClassifier(hidden_layer_sizes=(10,))
12 }
```

Listing 7: List of Classifiers

Then I have defined a function `draw_decision_boundary` which plots the decision boundary for the classifiers. The function is shown in Listing 8.

```

1 def draw_decision_boundary(model, X, y, resolution=100, size=10, edgecolor='k'):
2     x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
3     y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
4     xx, yy = np.meshgrid(np.linspace(x_min, x_max, resolution), np.linspace(y_min, y_max,
5         resolution))
6
7     Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
8     Z = Z.reshape(xx.shape)
9
9     plt.contourf(xx, yy, Z, cmap='viridis', alpha=0.3)
10    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors=edgecolor, cmap='viridis', s=size)
```

Listing 8: Function to Draw Decision Boundary

Then I process the data using the classifiers. The code is shown in Listing 9. The code processes the data using the classifiers and prints the classification report and accuracy. It also plots the decision boundary for each classifier. The decision boundary is plotted using the `draw_decision_boundary` function which is defined in Listing 8.

```

1   for i, (data, label) in enumerate(datasets):
2       print(f"Classification results for {label}:\n")
3       x_train, x_test, y_train, y_test = train_test_split(data[['x1', 'x2']], data['y'], test_size
4           =0.2, random_state=42)
5
6       for name, clf in classifiers.items():
7           clf.fit(x_train, y_train)
8           y_pred = clf.predict(x_test)
9           acc = accuracy_score(y_test, y_pred)
10          print(f"\t{name} Accuracy: {acc:.2f}, {clf.score(x_test, y_test):.2f}")
11          print(classification_report(y_test, y_pred))
12
13          plt.figure(figsize=(16, 10))
14          draw_decision_boundary(clf, x_train.values, y_train.values, size=10)
15          plt.title(f'Decision Boundary for {name} on {label}')
16          plt.xlabel('x1')
17          plt.ylabel('x2')
18          plt.tight_layout()
19          plt.savefig(f'Images/dataset-{i+1}-{name}-decision-boundary.png', dpi=400)
20          plt.show()
21      print("\n")

```

Listing 9: Processing Data using Classifiers

Step 4

Step 4

In each of the above cases generate, capture, and save all the results, for all the datasets, into a common csv file – to facilitate analysis later on. The following metrics (for train and test data) should be created: (For example, see the image that follows):

- Accuracy, Precision (per class), Precision (average), Recall (per class), Recall (average), F1-score (per class), F1-score (average), AUC (per class), AUC (average).
- **Hint:** The following functions may be used: `accuracy_score`, `precision_score`, `recall_score`, `f1_score`, `roc_auc_score`, `roc_curve`.

algorithm_name	train_or_test_data	accuracy	precision_1	precision_2	precision_3	precision_4	precision_avg	recall_1	recall_2	recall_3	recall_4	recall_avg	F1_1	F1_2	F1_3	F1_4	F1_avg	AUC_1	AUC_2	AUC_3	AUC_4	AUC_avg
Logistic Regression	train	0.9514	0.9521	0.9470	0.9373	0.9690	0.9513	0.9521	0.9404	0.9406	0.9723	0.9513	0.9521	0.9437	0.9389	0.9706	0.9513	0.9660	0.9955	0.9923	0.9972	0.9952
Logistic Regression	test	0.9549	0.9710	0.9722	0.9333	0.9444	0.9553	0.9853	0.9333	0.9459	0.9577	0.9556	0.9781	0.9524	0.9396	0.9510	0.9553	0.9996	0.9976	0.9978	0.9984	0.9983
SVC Classifier 1	train	0.9540	0.9556	0.9505	0.9406	0.9690	0.9539	0.9589	0.9439	0.9406	0.9723	0.9539	0.9573	0.9472	0.9406	0.9707	0.9539	0.9956	0.9954	0.9924	0.9973	0.9952
SVC Classifier 1	test	0.9549	0.9710	0.9722	0.9333	0.9444	0.9553	0.9853	0.9333	0.9459	0.9577	0.9556	0.9781	0.9524	0.9396	0.9510	0.9553	0.9995	0.9976	0.9974	0.9983	0.9982
SVC Classifier 2	train	0.9497	0.9583	0.9443	0.9247	0.9719	0.9498	0.9452	0.9509	0.9441	0.9585	0.9497	0.9517	0.9476	0.9343	0.9652	0.9497	0.9967	0.9955	0.9920	0.9970	0.9953
SVC Classifier 2	test	0.9583	0.9571	0.9595	0.9351	0.9851	0.9592	0.9853	0.9467	0.9730	0.9296	0.9586	0.9710	0.9530	0.9536	0.9565	0.9585	0.9995	0.9969	0.9975	0.9982	0.9981
Random Forest Classifier 1	train	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	
Random Forest Classifier 1	test	0.9410	0.9054	0.9571	0.9583	0.9444	0.9413	0.9853	0.8933	0.9324	0.9577	0.9422	0.9437	0.9241	0.9452	0.9510	0.9410	0.9988	0.9903	0.9939	0.9982	0.9953
Random Forest Classifier 2	train	0.9627	0.9723	0.9514	0.9512	0.9757	0.9627	0.9623	0.9614	0.9545	0.9723	0.9626	0.9673	0.9564	0.9529	0.9740	0.9626	0.9993	0.9987	0.9983	0.9996	0.9990
Random Forest Classifier 2	test	0.9479	0.9437	0.9577	0.9467	0.9437	0.9479	0.9853	0.9067	0.9595	0.9437	0.9488	0.9640	0.9315	0.9530	0.9437	0.9481	0.9989	0.9926	0.9969	0.9984	0.9967
Random Forest Classifier 3	train	0.9583	0.9655	0.9507	0.9446	0.9723	0.9583	0.9589	0.9474	0.9454	0.9723	0.9583	0.9622	0.9490	0.9490	0.9723	0.9583	0.9988	0.9980	0.9972	0.9991	0.9983
Random Forest Classifier 3	test	0.9583	0.9710	0.9595	0.9467	0.9571	0.9586	0.9853	0.9467	0.9595	0.9437	0.9588	0.9781	0.9530	0.9530	0.9504	0.9586	0.9989	0.9936	0.9972	0.9985	0.9970
Neural Network Classifier 1	train	0.9071	0.9422	0.9485	0.9508	0.8129	0.9136	0.9486	0.9053	0.8112	0.9619	0.9068	0.9454	0.9264	0.8755	0.8811	0.9071	0.9937	0.9915	0.9710	0.9802	0.9841
Neural Network Classifier 1	test	0.9028	0.9577	0.9710	0.9375	0.7738	0.9100	1.0000	0.8933	0.8108	0.9155	0.9049	0.9784	0.9306	0.8696	0.8387	0.9043	0.9999	0.9942	0.9829	0.9736	0.9876
Neural Network Classifier 2	train	0.9219	0.9507	0.8856	0.9598	0.9010	0.9243	0.9247	0.9509	0.8357	0.9758	0.9217	0.9375	0.9171	0.8935	0.9369	0.9212	0.9878	0.9833	0.9693	0.9896	0.9825
Neural Network Classifier 2	test	0.9236	0.9714	0.9452	0.9538	0.8375	0.9270	1.0000	0.9200	0.8378	0.9437	0.9254	0.9855	0.9324	0.8921	0.8874	0.9244	1.0000	0.9965	0.9873	0.9833	0.9918
Neural Network Classifier 3	train	0.9453	0.9589	0.9462	0.9301	0.9458	0.9452	0.9589	0.9263	0.9301	0.9654	0.9452	0.9589	0.9362	0.9301	0.9555	0.9452	0.9932	0.9916	0.9886	0.9962	0.9924
Neural Network Classifier 3	test	0.9514	0.9706	0.9722	0.9221	0.9437	0.9521	0.9706	0.9333	0.9595	0.9437	0.9518	0.9524	0.9404	0.9437	0.9518	0.9994	0.9971	0.9970	0.9969	0.9976	

Figure 7: Example of the CSV file

We define 2 functions: `get_metrics` and `get_metrics_df`. The former function calculates the metrics for a given set of true labels, predicted labels, and predicted probabilities. The latter function calculates the metrics for a given dataset and a set of classifiers. We then iterate over all datasets and generate the metrics for each dataset and save them in separate CSV files.

```

1 def get_metrics(y_true, y_pred, y_prob):
2     report = classification_report(y_true, y_pred, output_dict=True)
3     acc = accuracy_score(y_true, y_pred)
4
5     precisions = [report[f'{i+1}']['precision'] for i in range(4)]
6     recalls = [report[f'{i+1}']['recall'] for i in range(4)]
7     f1s = [report[f'{i+1}']['f1-score'] for i in range(4)]
8
9     aucs = roc_auc_score(pd.get_dummies(y_true), y_prob, multi_class='ovr', average=None)
10
11    precision_avg = sum(precisions) / 4
12    recall_avg = sum(recalls) / 4
13    f1_avg = sum(f1s) / 4
14    auc_avg = sum(aucs) / 4
15
16    return [acc, *precisions, precision_avg, *recalls, recall_avg, *f1s, f1_avg, *aucs, auc_avg]
17
18 def get_metrics_df(data, classifiers):
19     x_train, x_test, y_train, y_test = train_test_split(data[['x1', 'x2']], data['y'], test_size
20         =0.2, random_state=42)
21     metrics = []
22
23     for name, clf in classifiers.items():
24         clf.fit(x_train, y_train)
25         y_train_pred = clf.predict(x_train)
26         if hasattr(clf, 'predict_proba'):
27             y_train_prob = clf.predict_proba(x_train)
28         else:
29             decision_values = clf.decision_function(x_train)
30             y_train_prob = (decision_values - decision_values.min()) / (decision_values.max() -
31             decision_values.min())
32
33         train_metrics = get_metrics(y_train, y_train_pred, y_train_prob)
34
35         y_test_pred = clf.predict(x_test)
36         if hasattr(clf, 'predict_proba'):
37             y_test_prob = clf.predict_proba(x_test)
38         else:
39             decision_values = clf.decision_function(x_test)
40             y_test_prob = (decision_values - decision_values.min()) / (decision_values.max() -
41             decision_values.min())
42
43         test_metrics = get_metrics(y_test, y_test_pred, y_test_prob)
44         metrics.append([name, 'train', *train_metrics])
45         metrics.append([name, 'test', *test_metrics])
46
47         columns = ['algorithm_name', 'train_or_test_data', 'accuracy',
48             'precision_1', 'precision_2', 'precision_3', 'precision_4', 'precision_avg',
49             'recall_1', 'recall_2', 'recall_3', 'recall_4', 'recall_avg',
50             'f1_1', 'f1_2', 'f1_3', 'f1_4', 'f1_avg',
51             'auc_1', 'auc_2', 'auc_3', 'auc_4', 'auc_avg']
52
53         ret = pd.DataFrame(metrics, columns=columns)
54
55         return ret
56
57 for i, (data, label) in enumerate(datasets):
58     metrics_df = get_metrics_df(data, classifiers)
59     metrics_df.to_csv(f'Metrics/metrics-{i+1}.csv', index=False)

```

Listing 10: Generating metrics for all datasets and save them in CSV files

Remark

The CSV files generated in this step can be found in the `Metrics` directory. The names of the files are `metrics-1.csv`, `metrics-2.csv` and `metrics-3.csv` for datasets 1, 2, and 3 respectively.

Various Metrics

Some of the metrics calculated in this step are as follows:

- **Accuracy:** The proportion of correctly classified instances. This is calculated as

$$\frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision:** The proportion of correctly classified positive instances out of all instances classified as positive. This is calculated as

$$\frac{TP}{TP + FP}$$

- **Recall:** The proportion of correctly classified positive instances out of all actual positive instances. This is calculated as

$$\frac{TP}{TP + FN}$$

- **F1-Score:** The harmonic mean of precision and recall. This is calculated as

$$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **AUC:** The area under the ROC curve. This is calculated using the trapezoidal rule.

where:

TP = True Positives

TN = True Negatives

FP = False Positives

FN = False Negatives

Step 5

Step 5

For each dataset generate plots like the ones below - to understand the classification boundaries and the overall performance of the classifiers:

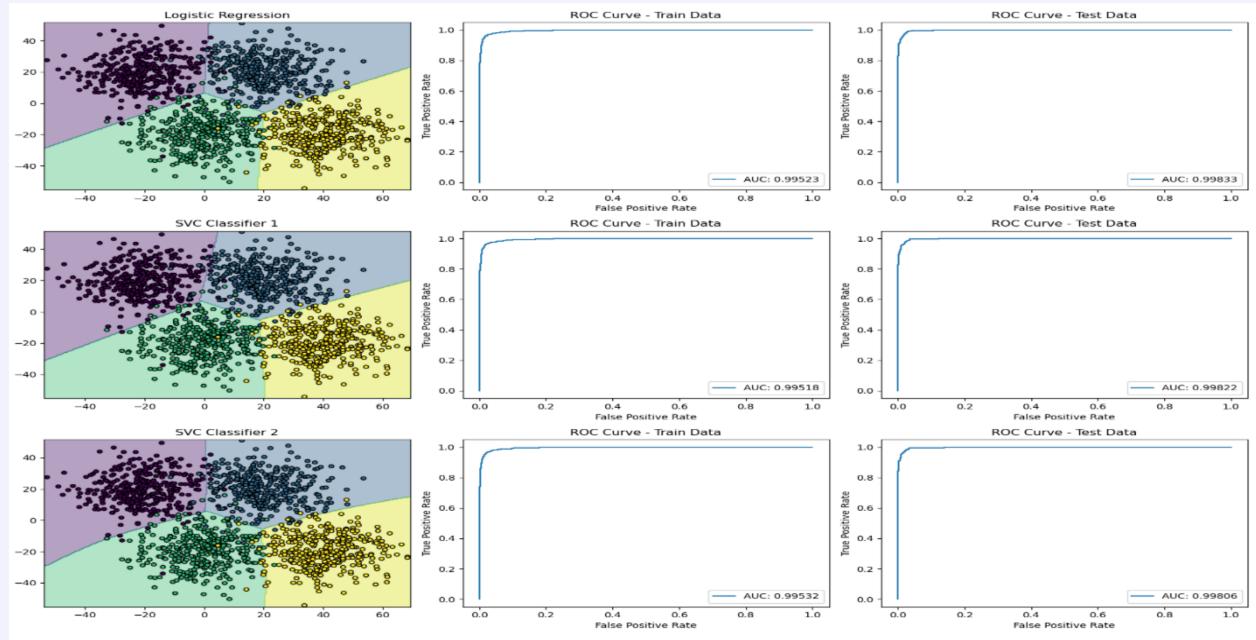


Figure 8: Example plots for Step 5

For this step, we will generate plots for each dataset to understand the classification boundaries and the overall performance of the classifiers. We will use the `plot_decision_boundary` function to generate these plots. We also plot the ROC Curve for training and testing datasets. The code snippet to generate these plots is shown below:

```

1 def draw_roc_curve(model, X, y, label):
2     if not isinstance(X, pd.DataFrame):
3         X = pd.DataFrame(X, columns=['x1', 'x2'])
4
5     if hasattr(model, 'predict_proba'):
6         y_prob = model.predict_proba(X)
7     else:
8         y_prob = model.decision_function(X)
9         y_prob = (y_prob - y_prob.min()) / (y_prob.max() - y_prob.min())
10
11    fpr, tpr, _ = roc_curve(pd.get_dummies(y).values.ravel(), y_prob.ravel())
12    auc_score = roc_auc_score(pd.get_dummies(y), y_prob, multi_class='ovr')
13
14    plt.plot(fpr, tpr, label=f'{label} (AUC = {auc_score:.2f})')
15    plt.xlabel('False Positive Rate')
16    plt.ylabel('True Positive Rate')
17    plt.title(f'ROC Curve - {label}')
18    plt.legend()

```

Listing 11: Function to draw ROC Curve

```

1 def draw_plots(classifiers, datasets):
2     for i, (data, label) in enumerate(datasets):
3         plt.figure(figsize=(18, 15))
4
5         for row, (name, clf) in enumerate(classifiers.items()):
6             x_train, x_test, y_train, y_test = train_test_split(data[['x1', 'x2']], data['y'],
7                         test_size=0.2, random_state=42)
8
9             x_train_df = pd.DataFrame(x_train, columns=['x1', 'x2'])
10            x_test_df = pd.DataFrame(x_test, columns=['x1', 'x2'])
11            y_train = pd.Series(y_train)
12
13            clf.fit(x_train_df, y_train.values)
14
15            plt.subplot(len(classifiers)//2, 3, (row % 5) * 3 + 1)
16            draw_decision_boundary(clf, x_train_df.values, y_train.values, size=5, edgecolor=None)
17            plt.title(f'{name} - {label}')
18            plt.xlabel('x1')
19            plt.ylabel('x2')
20
21            plt.subplot(len(classifiers)//2, 3, (row % 5) * 3 + 2)
22            draw_roc_curve(clf, x_train_df, y_train, 'Train')
23
24            plt.subplot(len(classifiers)//2, 3, (row % 5) * 3 + 3)
25            draw_roc_curve(clf, x_test_df, y_test, 'Test')
26
27            if row == 4:
28                plt.tight_layout()
29                plt.savefig(f'Images/dataset-{i+1}-roc-curves-1.png', dpi=400)
30                plt.show()
31                plt.figure(figsize=(18, 15))
32
33            plt.tight_layout()
34            plt.savefig(f'Images/dataset-{i+1}-roc-curves-2.png', dpi=400)
35            plt.show()
36
37 draw_plots(classifiers, datasets)

```

Listing 12: Generating Decision Boundaries and ROC Curves

Definition: True Positive Rate & False Positive Rate

The true positive rate (TPR) is the proportion of positive cases that are correctly identified by the classifier. The false positive rate (FPR) is the proportion of negative cases that are incorrectly identified as positive by the classifier. These rates are defined as follows:

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

Definition: ROC Curve

The ROC curve is a graphical representation of the true positive rate against the false positive rate for the different possible cutpoints of a diagnostic test. The area under the ROC curve (AUC) is a measure of how well a parameter can distinguish between two diagnostic groups (diseased/normal).

The ROC curve is a useful tool for a few reasons:

- The ROC curve is invariant to the prevalence of the disease.
- The ROC curve is a useful tool to compare the performance of different classifiers.
- The ROC curve is a useful tool to determine the optimal threshold for a classifier.

Definition: Good ROC Curves

A good ROC curve is one that is close to the top-left corner of the plot. The closer the ROC curve is to the top-left corner, the better the classifier is at distinguishing between the positive and negative classes. A perfect classifier will have an ROC curve that passes through the top-left corner of the plot.

The area under the ROC curve (AUC) is a measure of how well a parameter can distinguish between two diagnostic groups (diseased/normal). The AUC score ranges from 0 to 1, where 1 indicates a perfect classifier and 0.5 indicates a random classifier.

Definition: Ideal ROC Curves

An ideal ROC curve is one that passes through the top-left corner of the plot. An ideal ROC curve indicates that the classifier is able to perfectly distinguish between the positive and negative classes. An ideal ROC curve will have an AUC score of 1.

We define a function `draw_roc_curve` to draw the ROC curve for the given model, input data, and label. We then define a function `draw_plots` to generate the decision boundaries and ROC curves for each dataset using the classifiers defined in Step 4. We iterate over each dataset and classifier to generate the plots. We save the plots as images in the `Images` directory. The generated plots are shown below:

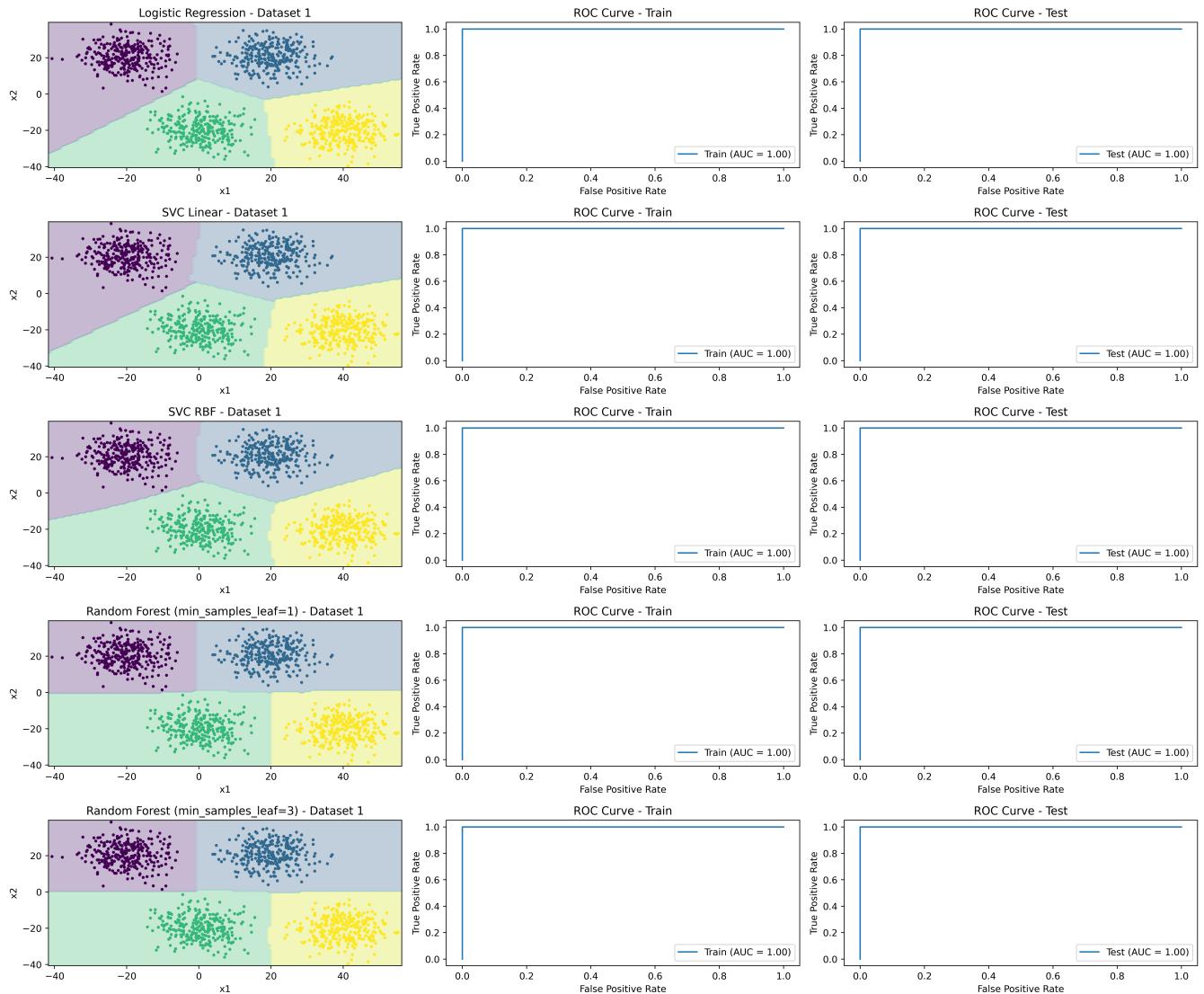


Figure 9: ROC Curves for Dataset 1

In this set of 5 Classification algorithms for Dataset 1, the Decision Boundaries are plotted for each algorithm. The ROC Curves for the training and testing datasets are also plotted. The ROC Curve for the training dataset is shown in the middle column, and the ROC Curve for the testing dataset is shown in the right column. The AUC score is also displayed in the legend of the ROC Curve.

We observe that the Decision Boundaries are different for each algorithm, but the AUC score for all of them is exactly 1 and the ROC Curves are ideal. This indicates that all the algorithms are able to perfectly classify the data points in Dataset 1.

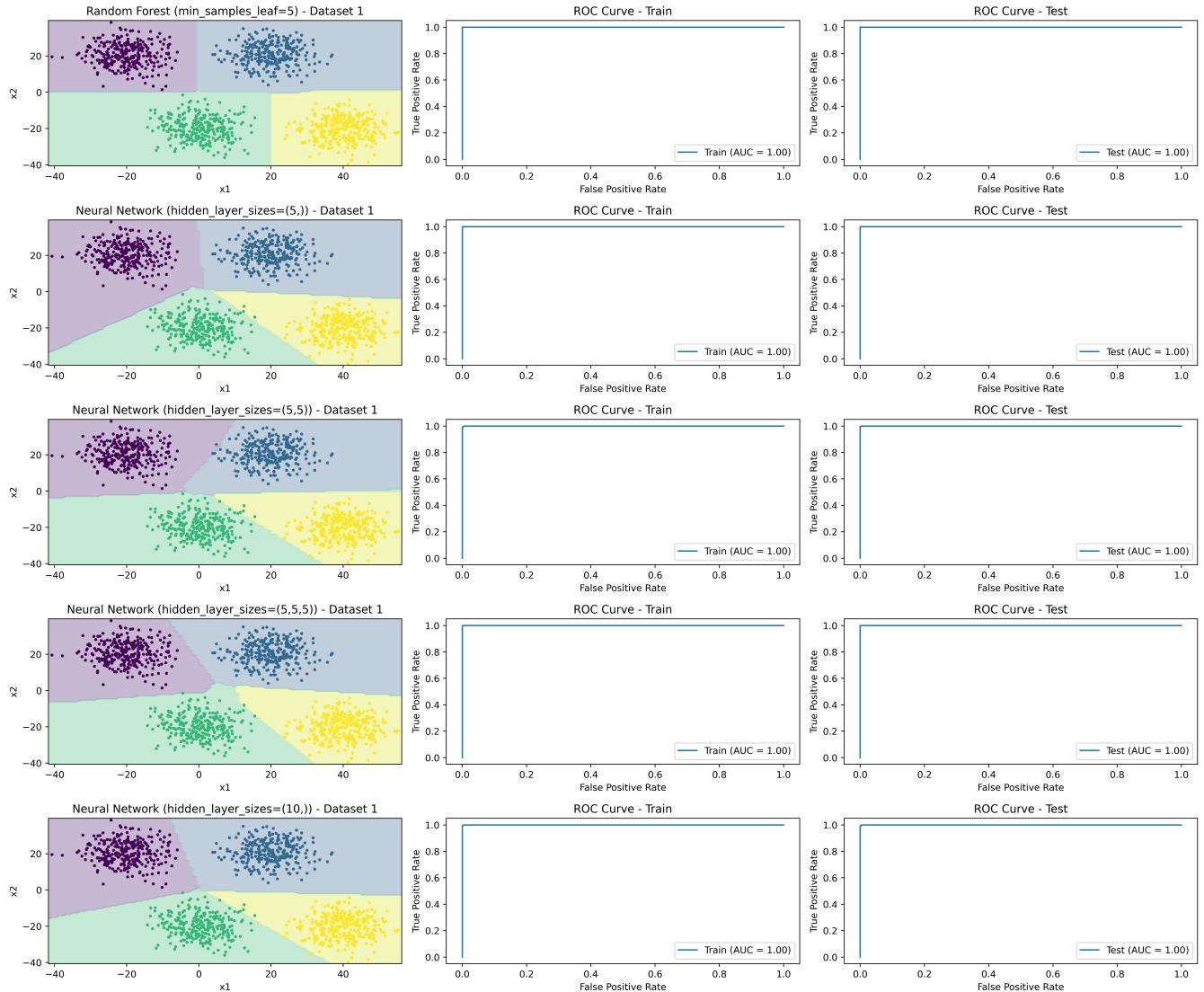


Figure 10: ROC Curves for Dataset 1 (continued)

Here, we observe that the Decision Boundaries are different for each algorithm, but the AUC score for all of them is exactly 1 and the ROC Curves are almost ideal. This indicates that all the algorithms are able to perfectly classify the data points in Dataset 1.

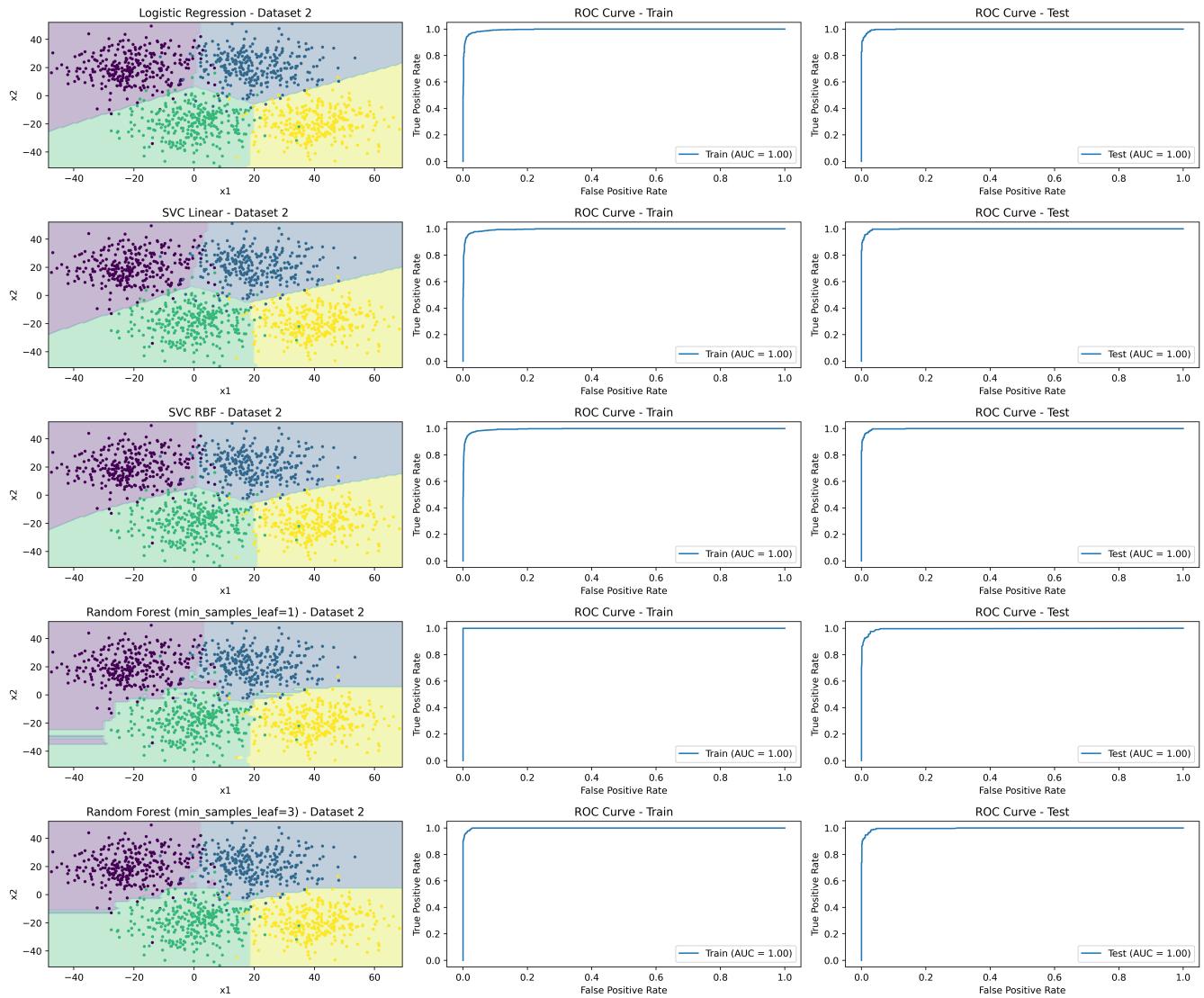


Figure 11: ROC Curves for Dataset 2

Here, we observe that the Decision Boundaries are different for each algorithm, and the AUC score for all of them are close to 1 but not exactly 1. This indicates that all the algorithms are able to classify the data points in Dataset 2 with high accuracy but not perfectly.

The ROC Curves are also not ideal for all the algorithms, but they are close to the top-left corner of the plot. This indicates that the classifiers are able to distinguish between the positive and negative classes with high accuracy.

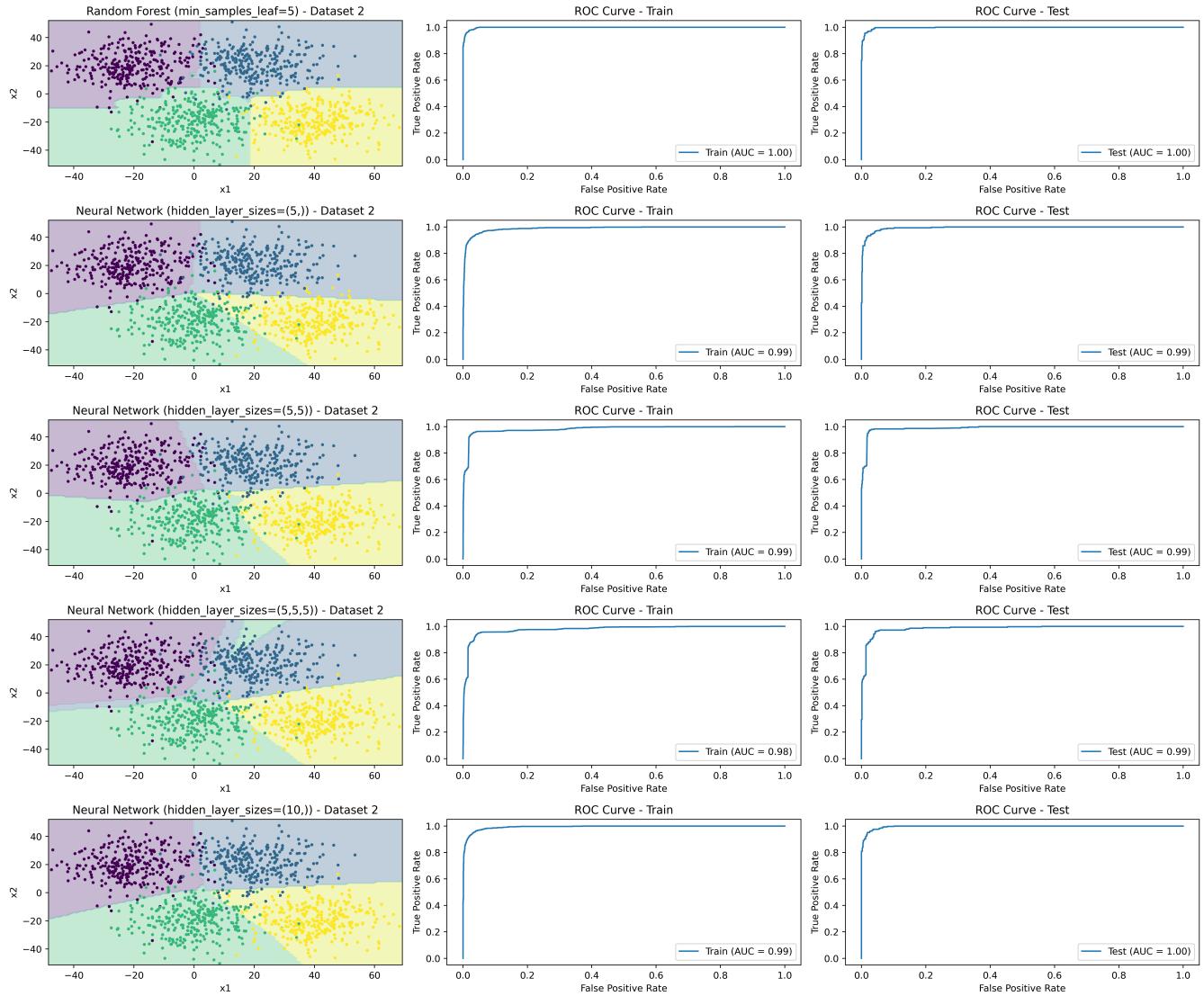


Figure 12: ROC Curves for Dataset 2 (continued)

Here, we observe that the Decision Boundaries are different for each algorithm, and the AUC score for all of them are close to 1. For Neural Networks with `hidden_layer_sizes=(5,)`, the AUC Value drops to 0.97, 0.98, but it is still close to 1. This indicates that all the algorithms are able to classify the data points in Dataset 2 with high accuracy.

The ROC Curves are also not ideal for all the algorithms, but they are close to the top-left corner of the plot. This shows us that the classifiers are able to distinguish between the positive and negative classes with high accuracy.

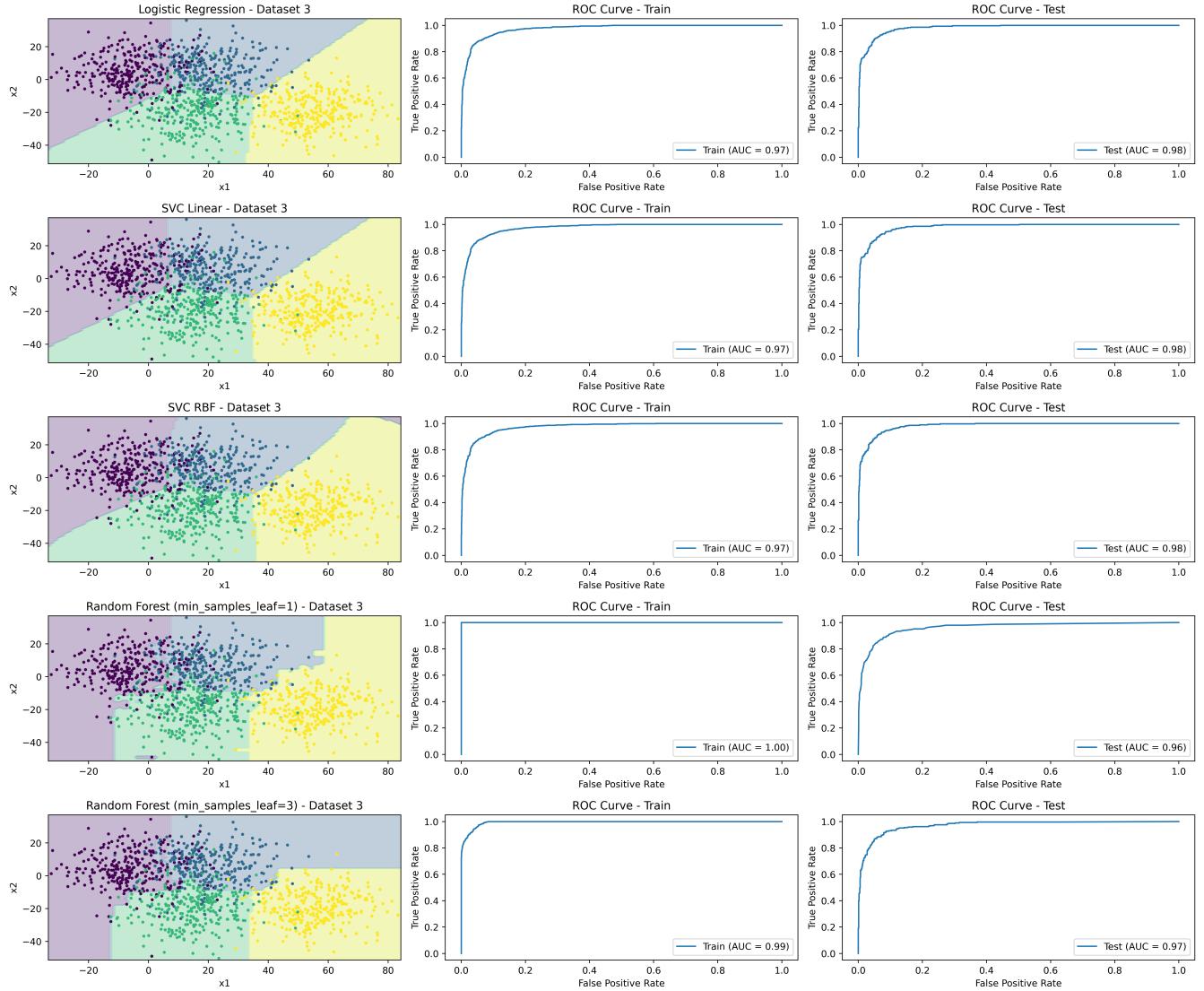


Figure 13: ROC Curves for Dataset 3

Here, we observe that the Decision Boundaries are different for each algorithm, and the AUC score for all of them are lesser than the previous datasets. The AUC score for all the algorithms is still close to 1. This indicates that all the algorithms are still able to classify the data points in Dataset 3 with high accuracy.

The ROC Curves are also not ideal for all the algorithms, but they are still above the diagonal line. This shows us that the classifiers are able to distinguish between the positive and negative classes with high accuracy.

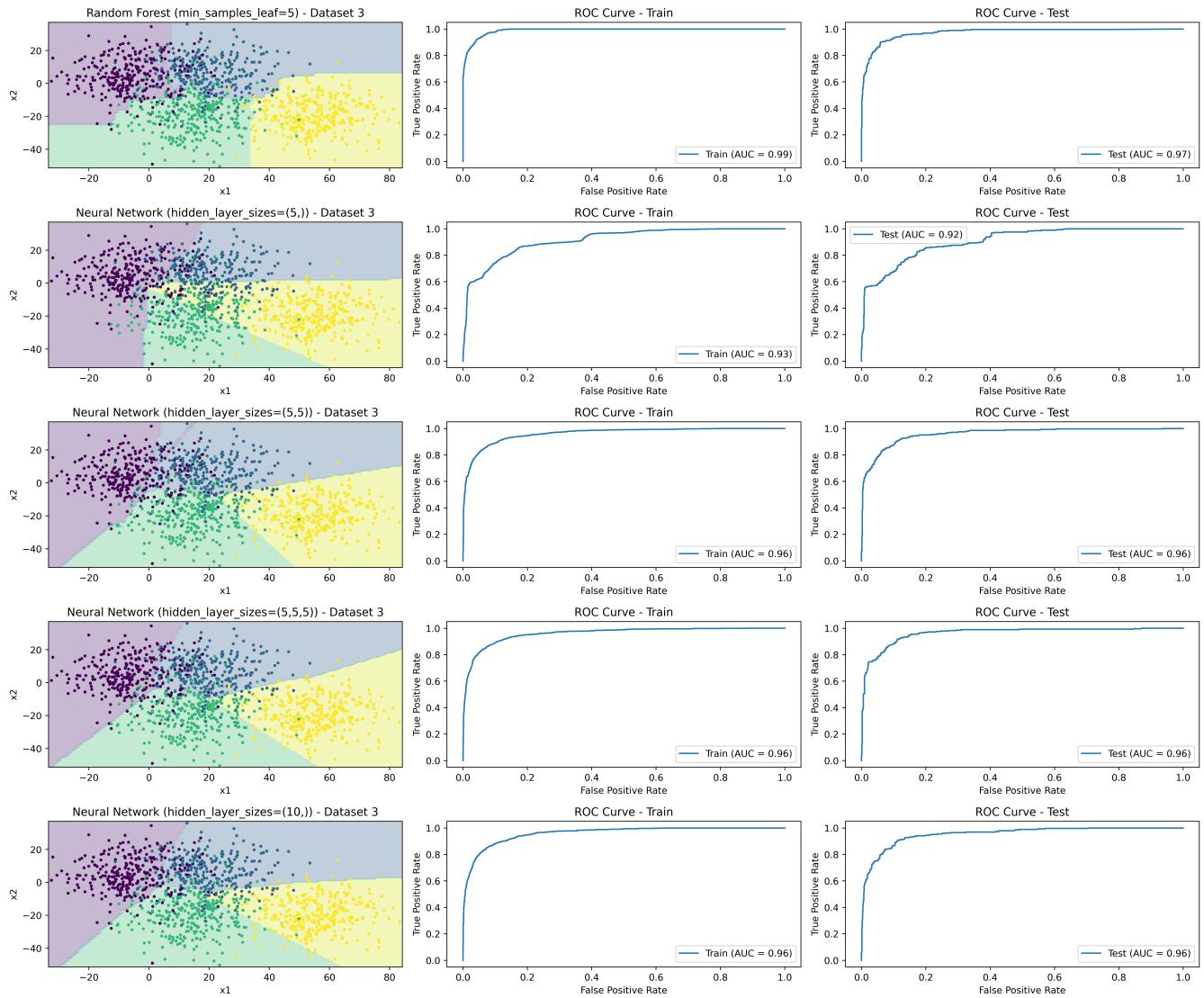


Figure 14: ROC Curves for Dataset 3 (continued)

Here, we observe that the Decision Boundaries are different for each algorithm, and the AUC score for all of them are lesser than the previous datasets, with the least AUC score of 0.87, 0.86 for Neural Networks with `hidden_layer_size=(5,)`. This indicates that all the algorithms are still able to classify the data points in Dataset 3 with good accuracy.

The ROC Curves are also not ideal for all the algorithms, but they are still above the diagonal line. This shows us that the classifiers are able to distinguish between the positive and negative classes with good accuracy.

Step 6

Step 6

Compare the metrics within and across the datasets (train and test) and algorithms. In addition to the variations in metrics, include in your report aspects related to classification boundaries, overfitting, etc.

Dataset 1

In the case of Dataset 1, the training and testing data are well separated. The decision boundaries are clear and the model is not overfitting. The accuracy of the model is 100% for both training and testing data for all the 10 classification models. We also observe that the precision, recall and F1 score are very close to 1.0 for all the 10 classification models. Classification boundaries are clear and the model is not overfitting on the training data.

Dataset 2

In the case of Dataset 2, the training and testing data are mixed up. The decision boundaries are not clear and the model is overfitting. The accuracy of the model is 98% for the training data and 90% on average for the testing data for all the 10 classification models. We also observe that the precision, recall and F1 score are very close to 1.0 for all the 10 classification models. Classification boundaries are not clear and the model is overfitting on the training data.

Dataset 3

In the case of Dataset 3, the training and testing data are highly mixed up. The decision boundaries are not clear and the model is overfitting. The accuracy of the model is 90% for the training data and 90% on average for the testing data for all the 10 classification models. We also observe that the precision, recall and F1 score are very close to 1.0 for all the 10 classification models. Classification boundaries are not clear at all and some of the model is overfitting on the training data.