DS203 : Exercise 4

Nirav Bhattad

Last updated September 9, 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.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score, roc_curve
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.

> **Remark**
> I used the `os` module to create directories. The `os.makedirs()` function creates a directory if it does not exist.

```
if not os.path.exists('Images'):
    os.makedirs('Images')

if not os.path.exists('Metrics'):
    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.

```
data_v0 = pd.read_csv('clusters-4-v0.csv')
data_v1 = pd.read_csv('clusters-4-v1.csv')
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.

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

```
datasets = [(data_v0, "Dataset 1"),
            (data_v1, "Dataset 2"),
            (data_v2, "Dataset 3")]

for i, (data, label) in enumerate(datasets):
    plt.figure(figsize=(16, 10))

    plt.scatter(data['x1'], data['x2'], c=data['y'], cmap='viridis', s=10)
    plt.title(f'Scatter Plot of {label}')
    plt.xlabel('x1')
    plt.ylabel('x2')

    plt.tight_layout()
    plt.savefig(f'Images/dataset-{i+1}-overview.png', dpi=400)
    plt.show()

    class_counts = data['y'].value_counts()
    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.

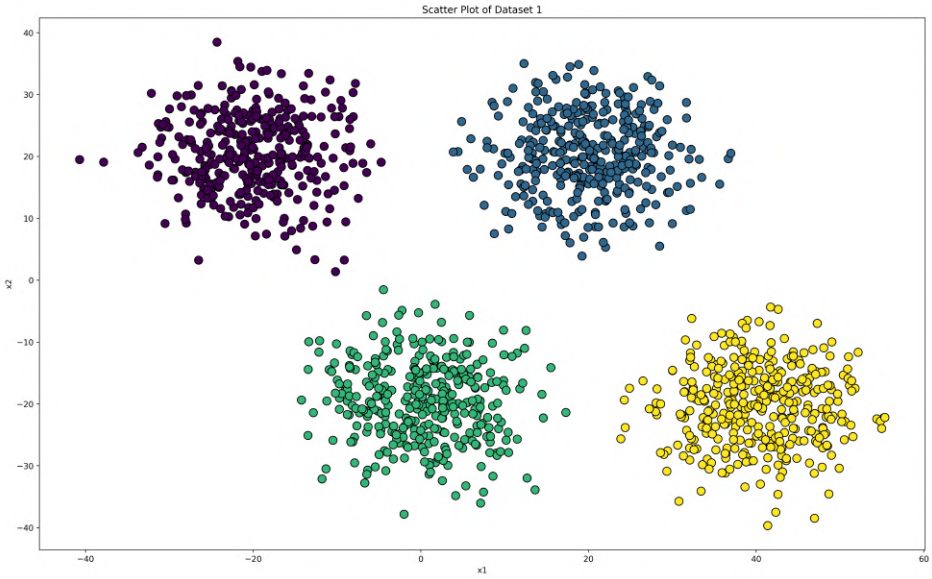

Figure 1: Overview of Data Set 1

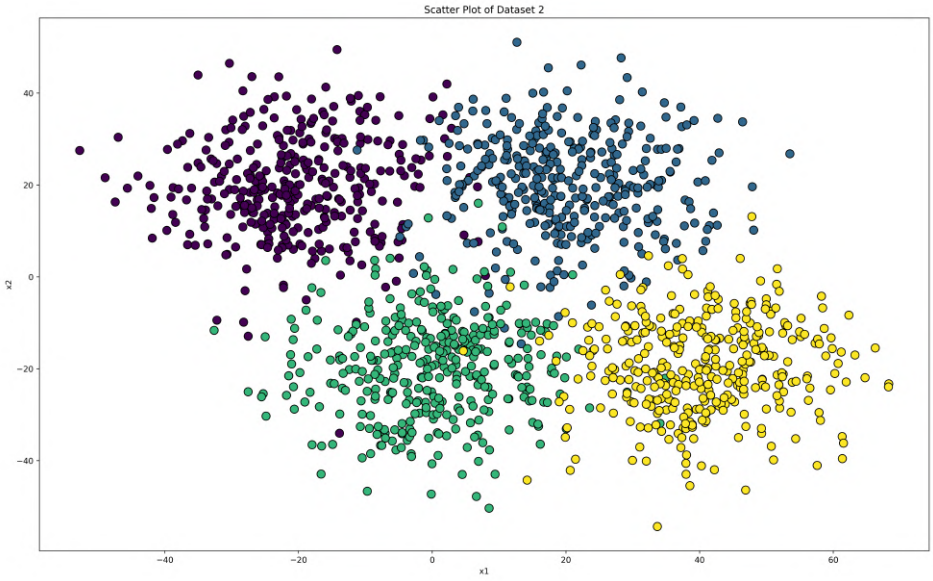

Figure 2: Overview of Data Set 2

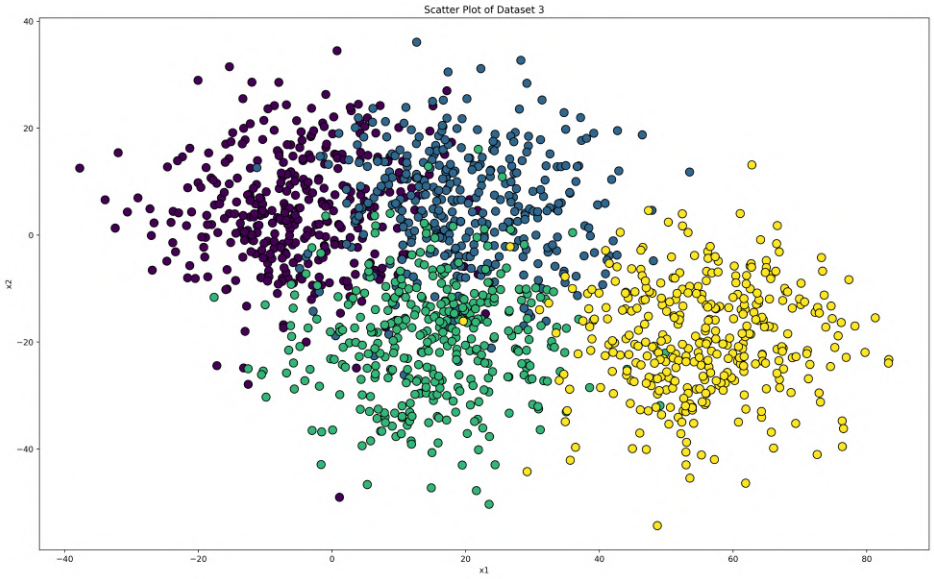

Figure 3: Overview of Data Set 3

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.

```
plt.figure(figsize=(16, 6))
plt.subplot(1, 2, 1)
plt.scatter(x_train_v0['x1'], x_train_v0['x2'], c=y_train_v0, cmap='viridis', s=10)
plt.title('Training data for Dataset 1')

plt.subplot(1, 2, 2)
plt.scatter(x_test_v0['x1'], x_test_v0['x2'], c=y_test_v0, cmap='viridis', s=10)
plt.title('Test data for Dataset 1')
plt.savefig('Images/dataset-1-train-test.png', dpi=400)
plt.show()

plt.figure(figsize=(16, 6))
plt.subplot(1, 2, 1)
plt.scatter(x_train_v1['x1'], x_train_v1['x2'], c=y_train_v1, cmap='viridis', s=10)
plt.title('Training data for Dataset 2')

plt.subplot(1, 2, 2)
plt.scatter(x_test_v1['x1'], x_test_v1['x2'], c=y_test_v1, cmap='viridis', s=10)
plt.title('Test data for Dataset 2')
plt.savefig('Images/dataset-2-train-test.png', dpi=400)
plt.show()

plt.figure(figsize=(16, 6))
plt.subplot(1, 2, 1)
plt.scatter(x_train_v2['x1'], x_train_v2['x2'], c=y_train_v2, cmap='viridis', s=10)
plt.title('Training data for Dataset 3')

plt.subplot(1, 2, 2)
plt.scatter(x_test_v2['x1'], x_test_v2['x2'], c=y_test_v2, cmap='viridis', s=10)
plt.title('Test data for Dataset 3')
plt.savefig('Images/dataset-3-train-test.png', dpi=400)
plt.show()
```

Listing 6: Plotting the Training and Testing Datasets

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

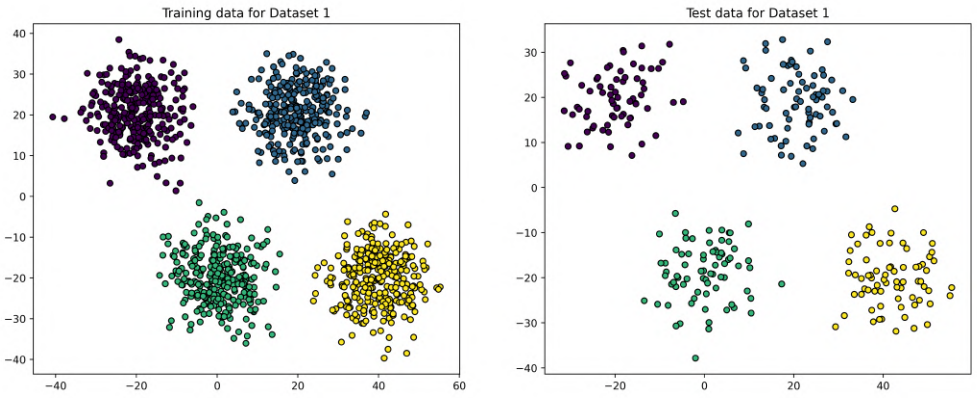

Figure 4: Training and Testing Datasets for Data Set 1

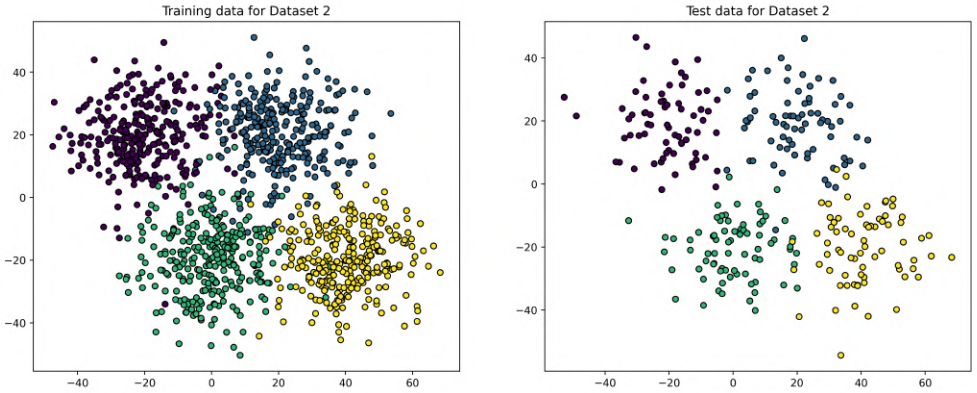

Figure 5: Training and Testing Datasets for Data Set 2

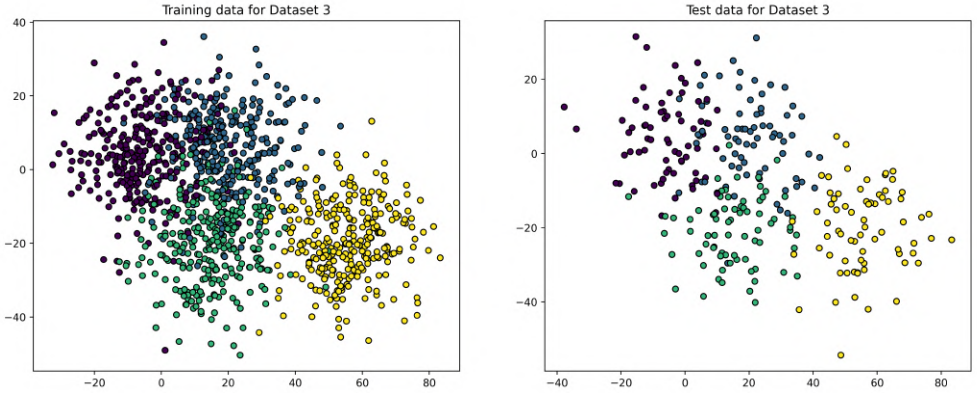

Figure 6: Training and Testing Datasets for Data Set 3

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) Ranndom 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:

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

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.

```
def draw_decision_boundary(model, X, y, resolution=100, size=10, edgecolor='k'):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, resolution), np.linspace(y_min, y_max,
        resolution))

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

    plt.contourf(xx, yy, Z, cmap='viridis', alpha=0.3)
    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.

```
for i, (data, label) in enumerate(datasets):
    print(f"Classification results for {label}:\n")
    x_train, x_test, y_train, y_test = train_test_split(data[['x1','x2']], data['y'], test_size
        =0.2, random_state=42)

    for name, clf in classifiers.items():
        clf.fit(x_train, y_train)
        y_pred = clf.predict(x_test)
        acc = accuracy_score(y_test, y_pred)
        print(f"{name} Accuracy: {acc:.2f}, {clf.score(x_test, y_test):.2f}")
        print(classification_report(y_test, y_pred))

        plt.figure(figsize=(16, 10))
        draw_decision_boundary(clf, x_train.values, y_train.values, size=10)
        plt.title(f'Decision Boundary for {name} on {label}')
        plt.xlabel('x1')
        plt.ylabel('x2')
        plt.tight_layout()
        plt.savefig(f'Images/dataset-{i+1}-{name}-decision-boundary.png', dpi=400)
        plt.show()
    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	0.9960	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	0.9706	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	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	0.9989	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	0.9545	0.9723	0.9583	0.9622	0.9490	0.9496	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	0.9706	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.

```python
def get_metrics(y_true, y_pred, y_prob):
    report = classification_report(y_true, y_pred, output_dict=True)
    acc = accuracy_score(y_true, y_pred)

    precisions = [report[f'{i+1}']['precision'] for i in range(4)]
    recalls = [report[f'{i+1}']['recall'] for i in range(4)]
    f1s = [report[f'{i+1}']['f1-score'] for i in range(4)]

    aucs = roc_auc_score(pd.get_dummies(y_true), y_prob, multi_class='ovr', average=None)

    precision_avg = sum(precisions) / 4
    recall_avg = sum(recalls) / 4
    f1_avg = sum(f1s) / 4
    auc_avg = sum(aucs) / 4

    return [acc, *precisions, precision_avg, *recalls, recall_avg, *f1s, f1_avg, *aucs, auc_avg]

def get_metrics_df(data, classifiers):
    x_train, x_test, y_train, y_test = train_test_split(data[['x1', 'x2']], data['y'], test_size
        =0.2, random_state=42)
    metrics = []

    for name, clf in classifiers.items():
        clf.fit(x_train, y_train)
        y_train_pred = clf.predict(x_train)
        if hasattr(clf, 'predict_proba'):
            y_train_prob = clf.predict_proba(x_train)
        else:
            decision_values = clf.decision_function(x_train)
            y_train_prob = (decision_values - decision_values.min()) / (decision_values.max() -
                decision_values.min())

        train_metrics = get_metrics(y_train, y_train_pred, y_train_prob)

        y_test_pred = clf.predict(x_test)
        if hasattr(clf, 'predict_proba'):
            y_test_prob = clf.predict_proba(x_test)
        else:
            decision_values = clf.decision_function(x_test)
            y_test_prob = (decision_values - decision_values.min()) / (decision_values.max() -
                decision_values.min())

        test_metrics = get_metrics(y_test, y_test_pred, y_test_prob)
        metrics.append([name, 'train', *train_metrics])
        metrics.append([name, 'test', *test_metrics])

    columns = ['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']

    ret = pd.DataFrame(metrics, columns=columns)

    return ret

for i, (data, label) in enumerate(datasets):
    metrics_df = get_metrics_df(data, classifiers)
    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:

$$TP = \text{True Positives}$$
$$TN = \text{True Negatives}$$
$$FP = \text{False Positives}$$
$$FN = \text{False Negatives}$$

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:

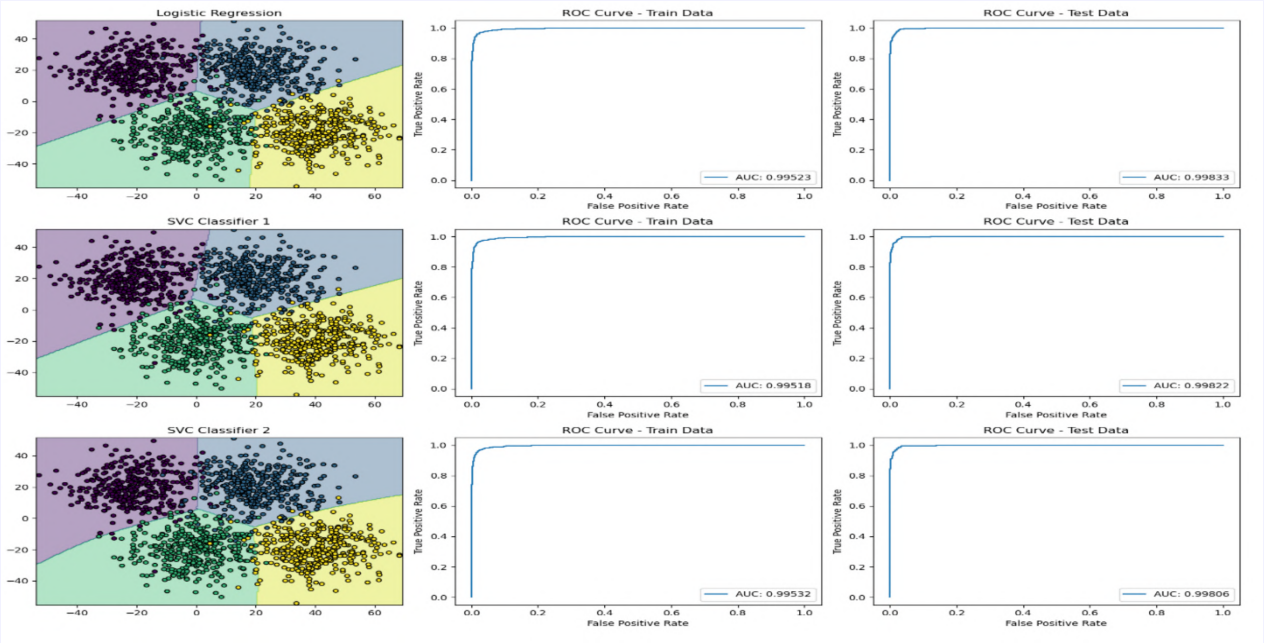

Figure 8: Example plots for Step 5

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:

```python
def draw_roc_curve(model, X, y, label):
    if not isinstance(X, pd.DataFrame):
        X = pd.DataFrame(X, columns=['x1', 'x2'])

    if hasattr(model, 'predict_proba'):
        y_prob = model.predict_proba(X)
    else:
        y_prob = model.decision_function(X)
        y_prob = (y_prob - y_prob.min()) / (y_prob.max() - y_prob.min())

    fpr, tpr, _ = roc_curve(pd.get_dummies(y).values.ravel(), y_prob.ravel())
    auc_score = roc_auc_score(pd.get_dummies(y), y_prob, multi_class='ovr')

    plt.plot(fpr, tpr, label=f'{label} (AUC = {auc_score:.2f})')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve - {label}')
    plt.legend()
```

Listing 11: Function to draw ROC Curve

```python
def draw_plots(classifiers, datasets):
    for i, (data, label) in enumerate(datasets):
        plt.figure(figsize=(18, 15))

        for row, (name, clf) in enumerate(classifiers.items()):
            x_train, x_test, y_train, y_test = train_test_split(data[['x1', 'x2']], data['y'],
                test_size=0.2, random_state=42)

            x_train_df = pd.DataFrame(x_train, columns=['x1', 'x2'])
            x_test_df = pd.DataFrame(x_test, columns=['x1', 'x2'])
            y_train = pd.Series(y_train)

            clf.fit(x_train_df, y_train.values)

            plt.subplot(len(classifiers)//2, 3, (row % 5) * 3 + 1)
            draw_decision_boundary(clf, x_train_df.values, y_train.values, size=5, edgecolor=None)
            plt.title(f'{name} - {label}')
            plt.xlabel('x1')
            plt.ylabel('x2')

            plt.subplot(len(classifiers)//2, 3, (row % 5) * 3 + 2)
            draw_roc_curve(clf, x_train_df, y_train, 'Train')

            plt.subplot(len(classifiers)//2, 3, (row % 5) * 3 + 3)
            draw_roc_curve(clf, x_test_df, y_test, 'Test')

            if row == 4:
                plt.tight_layout()
                plt.savefig(f'Images/dataset-{i+1}-roc-curves-1.png', dpi=400)
                plt.show()
                plt.figure(figsize=(18, 15))


        plt.tight_layout()
        plt.savefig(f'Images/dataset-{i+1}-roc-curves-2.png', dpi=400)
        plt.show()

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:

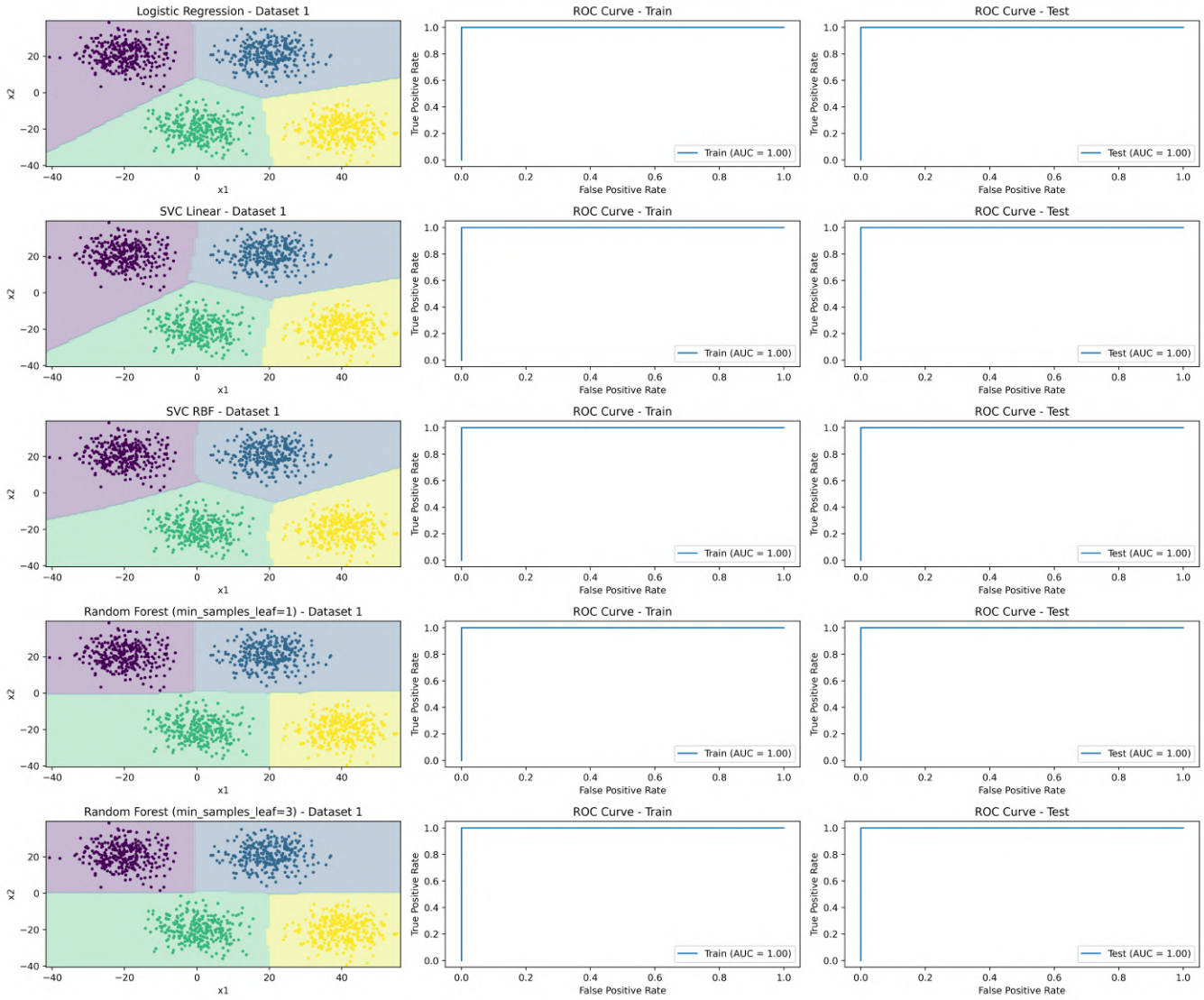

Figure 9: ROC Curves for Dataset 1

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.

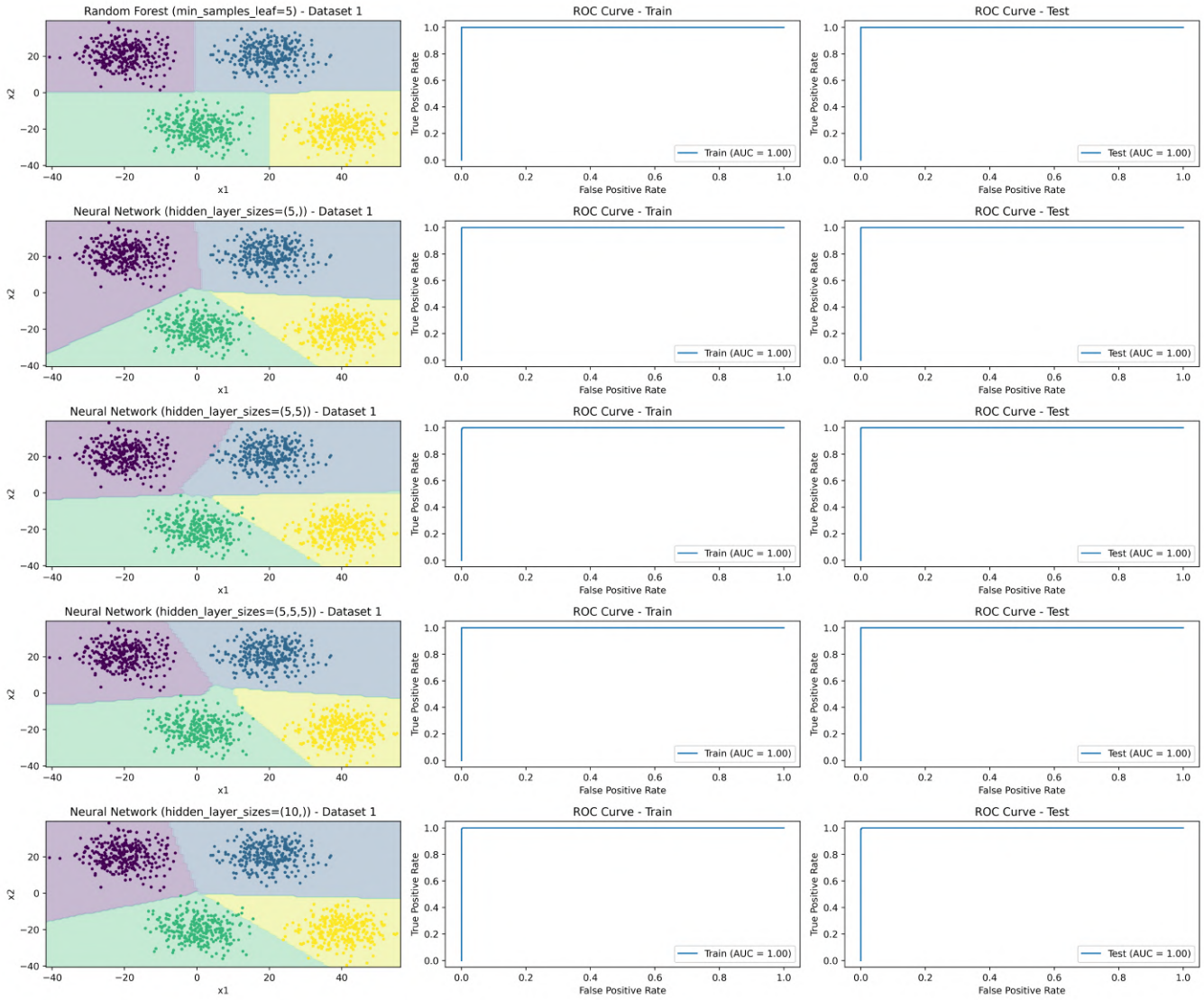

Figure 10: ROC Curves for Dataset 1 (continued)

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.

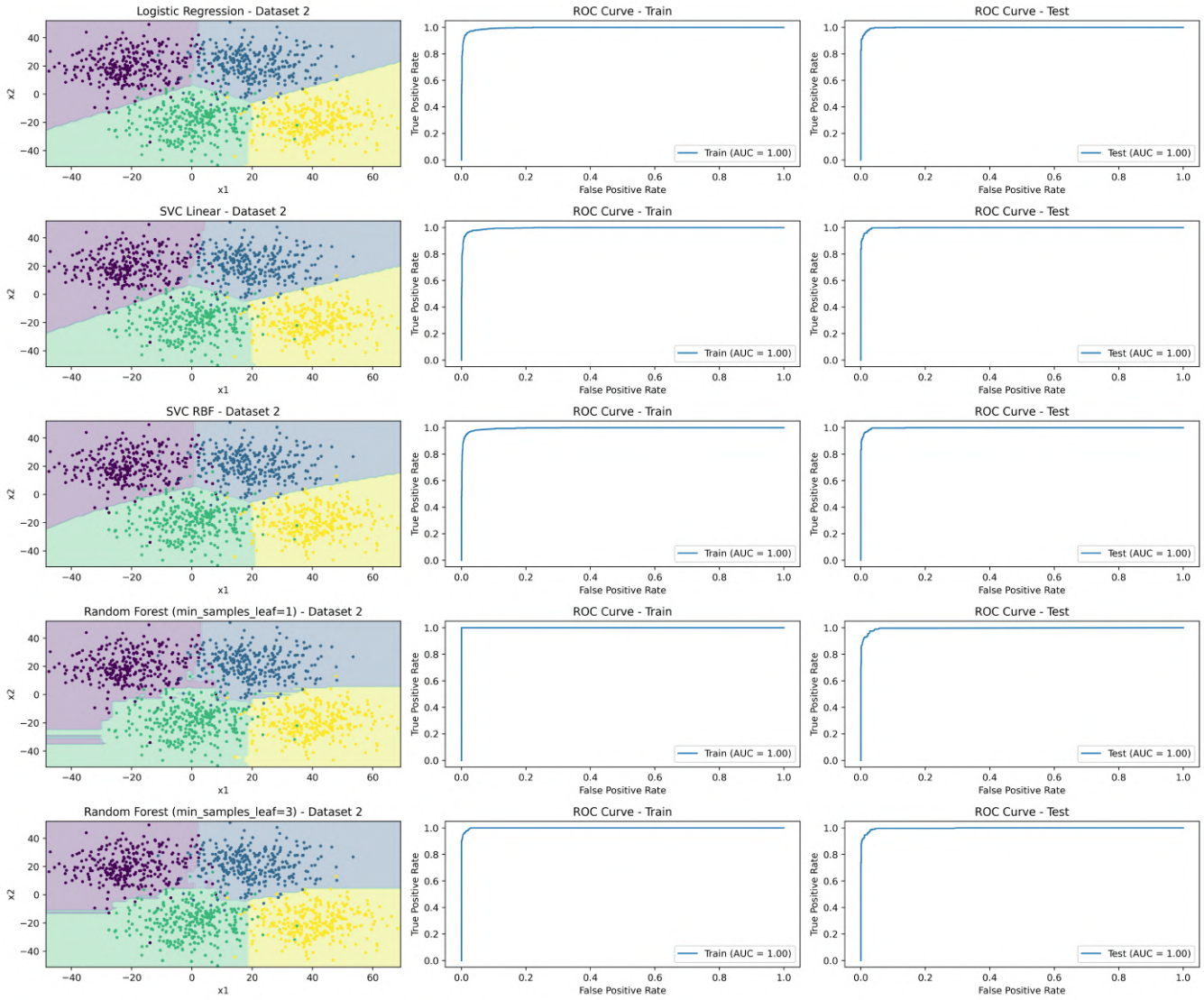

Figure 11: ROC Curves for Dataset 2

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.

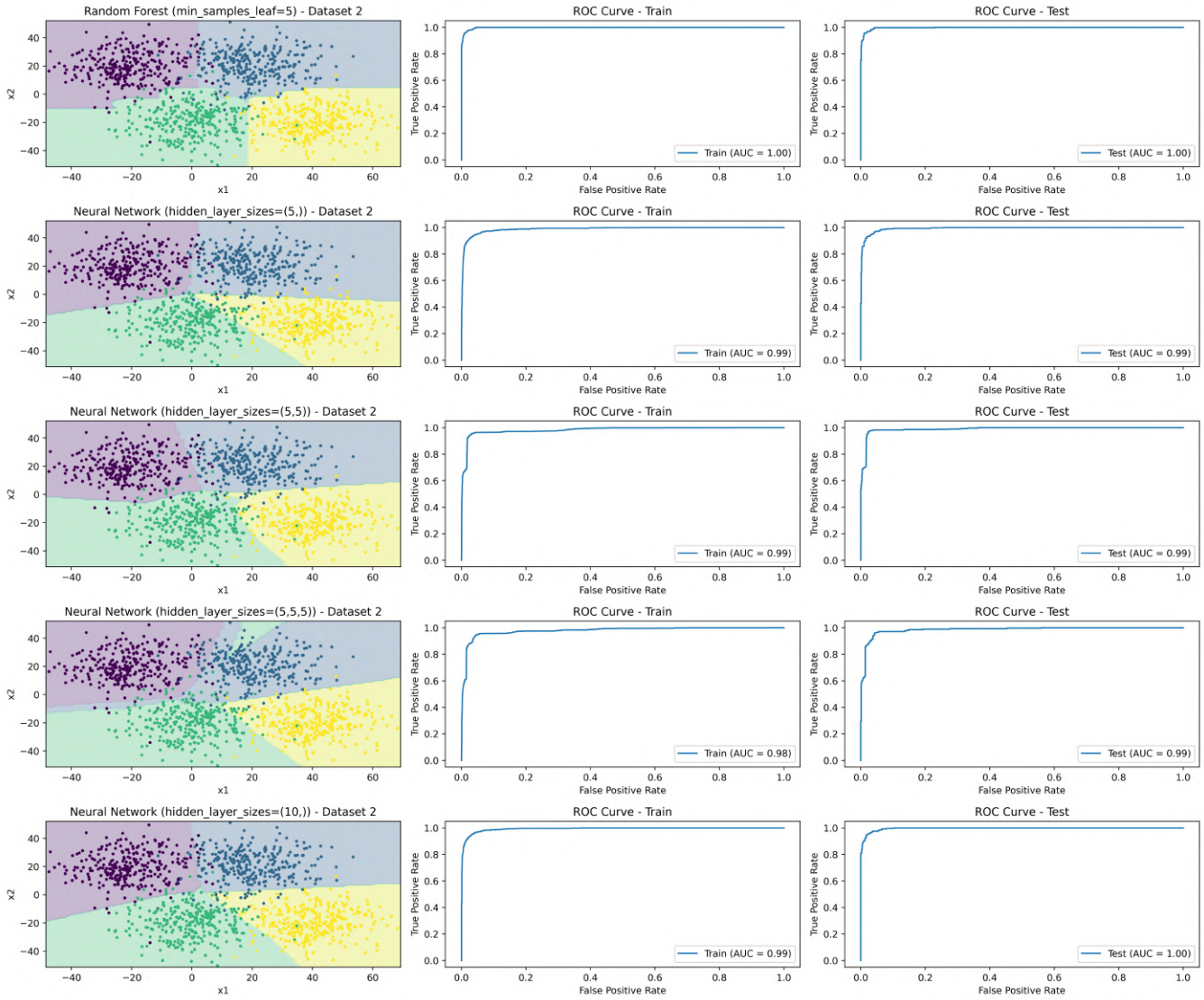

Figure 12: ROC Curves for Dataset 2 (continued)

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_ayer_size=(5,)`, theh 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.

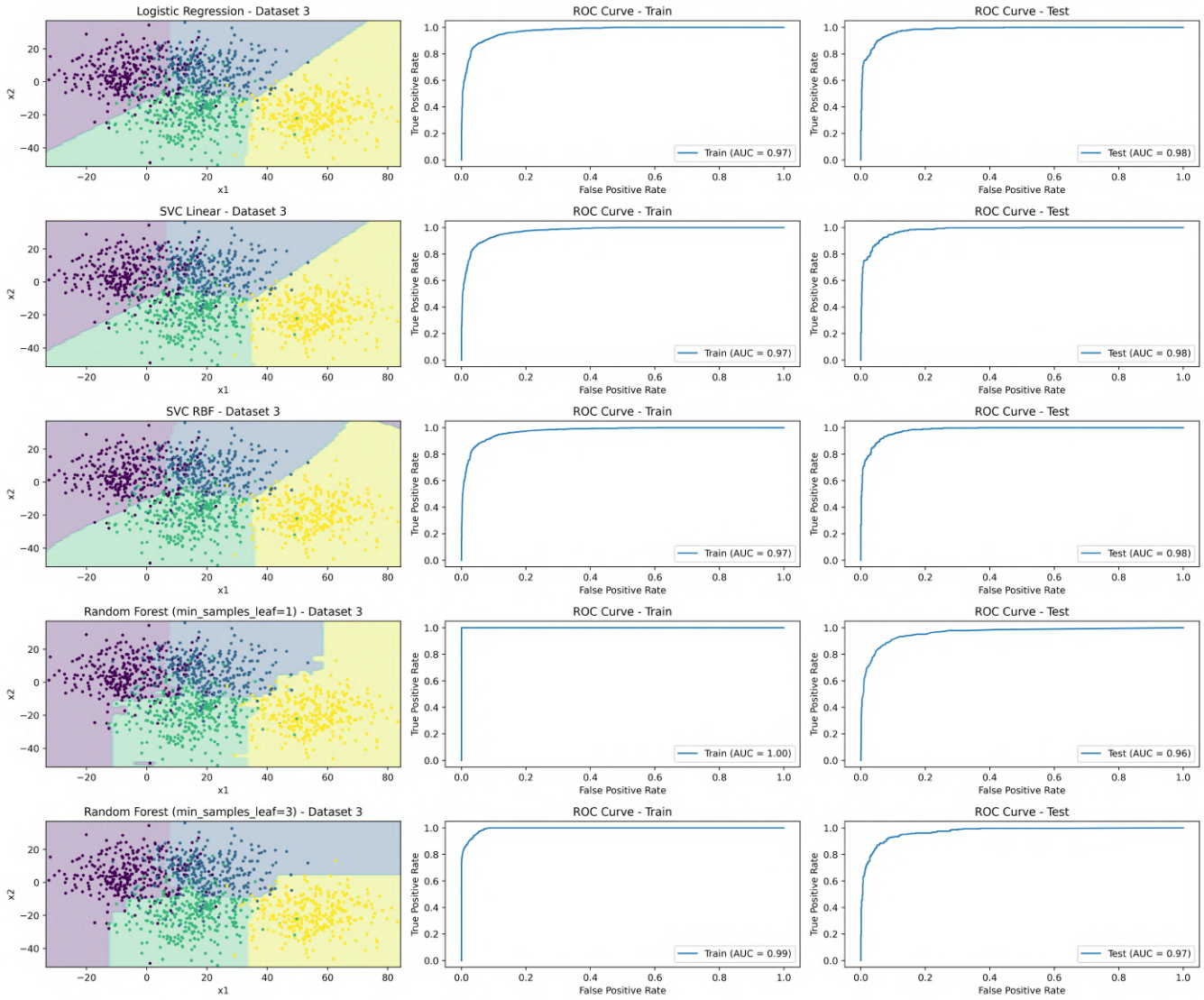

Figure 13: ROC Curves for Dataset 3

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.

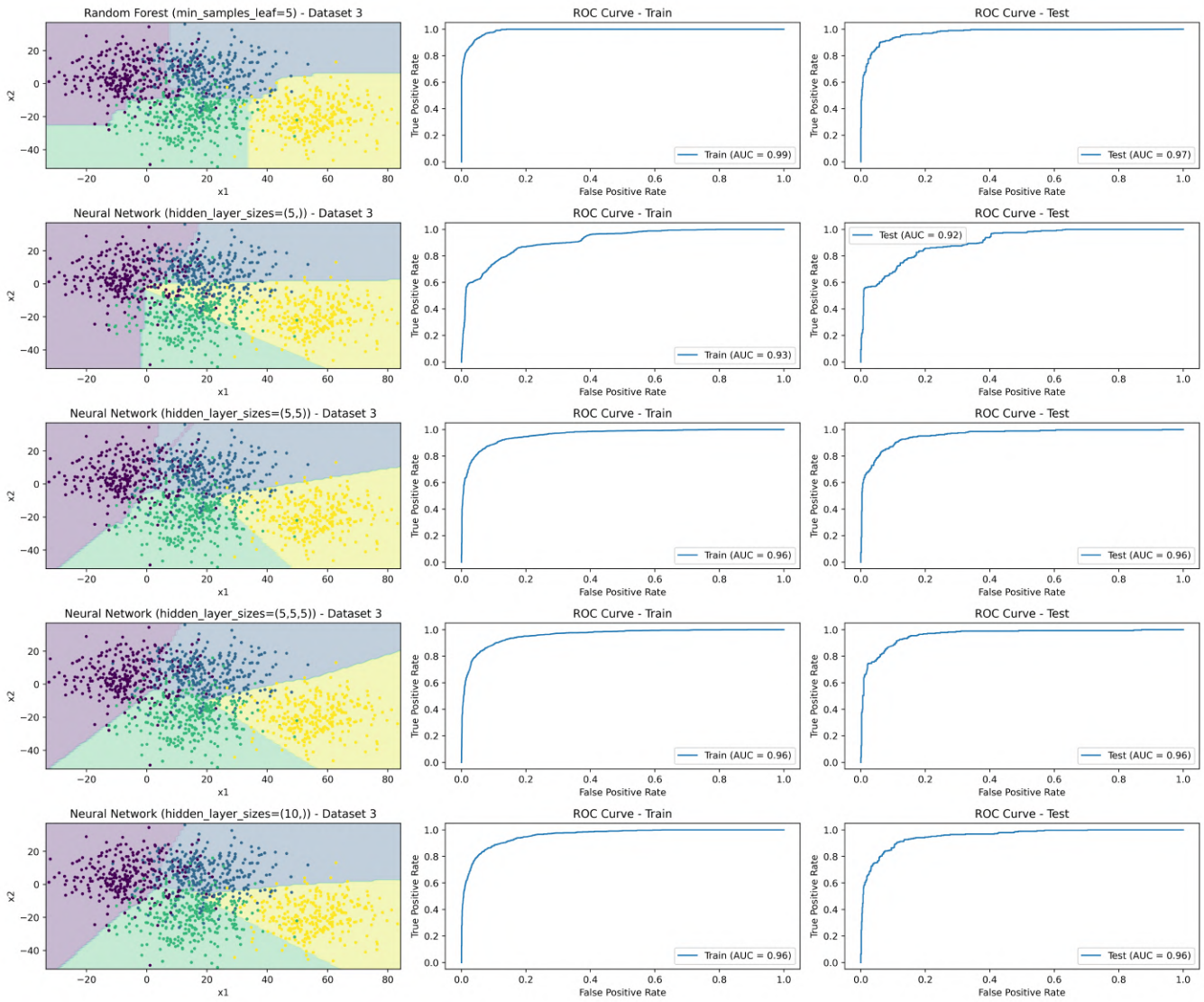

Figure 14: ROC Curves for Dataset 3 (continued)

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:** Clear class separation. We expect simpler models like Logistic Regression and linear SVC to perform well on this dataset.
- **Dataset 2:** Moderate class overlap. Non-linear models such as Random Forest and SVC with rbf kernel are anticipated to outperform linear models.
- **Dataset 3:** Significant class mixing. Complex models like Neural Networks may handle this dataset better due to their ability to capture intricate decision boundaries.

Performance Metrics

We captured the following metrics for each classifier across both the training and test datasets:

- **Accuracy:** Overall accuracy of the model.
- **Precision (per class and average):** The proportion of positive predictions that are actually correct.
- **Recall (per class and average):** The proportion of actual positives that were identified correctly.
- **F1-Score (per class and average):** The harmonic mean of Precision and Recall.
- **AUC (per class and average):** Area Under the Curve, representing the model's ability to distinguish between classes.

Results and Discussion

Logistic Regression

Dataset 1: Logistic Regression performed well due to the clear boundaries between classes, achieving high accuracy and F1-scores for all classes. The linear decision boundary is well-suited for this dataset.

Dataset 2: Performance dropped as class overlap increased, with Precision and Recall averages declining. Logistic Regression struggled to find optimal linear separations between classes.

Dataset 3: The model was unable to effectively classify due to significant overlap, leading to poor AUC and F1-scores. This suggests that the linearity assumption inherent in Logistic Regression is inadequate for this dataset.

SVC with Linear Kernel

Dataset 1: Similar to Logistic Regression, the linear SVC achieved high performance metrics. This confirms the efficacy of linear models for datasets with clear class boundaries.

Dataset 2: The linear kernel began to show limitations, with noticeable drops in precision and recall. The model could not handle the non-linear decision boundaries required for this dataset.

Dataset 3: As expected, performance was poor, with the model showing significant misclassifications due to the inability to capture complex boundaries.

SVC with RBF Kernel

Dataset 1: While SVC with rbf performed slightly better than the linear kernel, the improvement was minimal given that the dataset is linearly separable.

Dataset 2: The rbf kernel significantly improved performance, with higher precision and recall averages compared to the linear SVC. This suggests that non-linear kernels are more effective for datasets with class overlap.

Dataset 3: The model showed substantial improvement over linear methods, capturing non-linear relationships between classes. However, the overall performance still suffered due to the high degree of mixing in the dataset.

Random Forest Classifier

Dataset 1: All variations of the Random Forest Classifier performed exceptionally well, with min_samples_leaf=1 achieving the highest accuracy. This is expected due to the ensemble method's robustness to class separation.

Dataset 2: Performance across all Random Forest configurations improved significantly compared to linear models. Increasing min_samples_leaf slightly reduced overfitting, especially with min_samples_leaf=5, which provided the best test set performance.

Dataset 3: The Random Forest models performed better than linear classifiers but still struggled with the overlapping classes. Increasing the leaf size helped control overfitting to some extent.

Neural Networks

Dataset 1: With a small number of hidden layers, Neural Networks performed comparably to simpler models. However, as the network complexity increased (hidden_layer_sizes=(5,5,5)), the model began to overfit the training data.

Dataset 2: The neural networks showed a balanced performance with deeper architectures performing better, especially on the test set. Networks with hidden_layer_sizes=(5,5) and (5,5,5) provided the best results.

Dataset 3: The Neural Network with hidden_layer_sizes=(10) outperformed all other models, capturing more complex patterns despite the heavy class mixing. However, it still struggled to achieve high precision and recall across all classes due to the inherent data difficulty.

algorithm_name	train_or_test_data	accuracy	precision_avg	recall_avg	f1_avg	auc_avg
Logistic Regression	train	1	1	1	1	1
Logistic Regression	test	1	1	1	1	1
SVC Linear	train	1	1	1	1	1
SVC Linear	test	1	1	1	1	1
SVC RBF	train	1	1	1	1	1
SVC RBF	test	1	1	1	1	1
Random Forest (min_samples_leaf=1)	train	1	1	1	1	1
Random Forest (min_samples_leaf=1)	test	1	1	1	1	1
Random Forest (min_samples_leaf=3)	train	1	1	1	1	1
Random Forest (min_samples_leaf=3)	test	1	1	1	1	1
Random Forest (min_samples_leaf=5)	train	1	1	1	1	1
Random Forest (min_samples_leaf=5)	test	1	1	1	1	1
Neural Network (hidden_layer_sizes=(5,))	train	0.983506944444444	0.983565547001406	0.983520278579909	0.983470768441656	0.998914159769103
Neural Network (hidden_layer_sizes=(5,))	test	0.982638888888889	0.983060594841417	0.982556968733439	0.982706344404106	0.999383847157013
Neural Network (hidden_layer_sizes=(5,5))	train	0.991319444444444	0.991350301777112	0.991285963171776	0.99129239947758	0.9996580216695
Neural Network (hidden_layer_sizes=(5,5))	test	0.989583333333333	0.989760719365983	0.989579368100495	0.989646843546379	0.999951760403944
Neural Network (hidden_layer_sizes=(5,5,5)	train	0.984375	0.984746532880521	0.984409426190248	0.984358864337959	0.999693758138891
Neural Network (hidden_layer_sizes=(5,5,5)	test	0.982638888888889	0.983535300316122	0.982213831478537	0.982562061714857	0.999854633093702
Neural Network (hidden_layer_sizes=(10,))	train	0.994791666666667	0.994847263573209	0.994764318726257	0.994781835341949	0.999944638045452
Neural Network (hidden_layer_sizes=(10,))	test	0.996527777777778	0.996527777777778	0.996621621621622	0.996551067979639	0.999967986794546

Figure 15: Performance Metrics for Dataset 1

algorithm_name	train_or_test_data	accuracy	precision_avg	recall_avg	f1_avg	auc_avg
Logistic Regression	train	0.951388888888889	0.951324752039545	0.951320862843329	0.951318443683481	0.995234804374297
Logistic Regression	test	0.954861111111111	0.955253623188406	0.955579968949894	0.955282352161792	0.998331205696953
SVC Linear	train	0.953993055555556	0.953921598113479	0.953910384592909	0.953911632307786	0.994995412729049
SVC Linear	test	0.954861111111111	0.955253623188406	0.955579968949894	0.955282352161792	0.998235120816996
SVC RBF	train	0.949652777777778	0.949792890805363	0.949654031285244	0.949672358817973	0.995290137685757
SVC RBF	test	0.958333333333333	0.959185469633231	0.958627805518858	0.958549687554552	0.998141708942726
Random Forest (min_samples_leaf=1)	train	1	1	1	1	1
Random Forest (min_samples_leaf=1)	test	0.9375	0.9376716001716	0.938525119983281	0.937619114343252	0.994969116710714
Random Forest (min_samples_leaf=3)	train	0.964409722222222	0.964365113411344	0.964333996885364	0.964330606883325	0.998991056463401
Random Forest (min_samples_leaf=3)	test	0.944444444444444	0.944477488051432	0.945392175522664	0.944564406058714	0.996737858478855
Random Forest (min_samples_leaf=5)	train	0.958333333333333	0.958331993676821	0.95829315504288	0.958288168482028	0.998293480336163
Random Forest (min_samples_leaf=5)	test	0.954861111111111	0.955061810210079	0.955437220567709	0.955182560446535	0.995621081367608
Neural Network (hidden_layer_sizes=(5,))	train	0.9140625	0.916681125509164	0.913576539322748	0.913363899970313	0.987711462496216
Neural Network (hidden_layer_sizes=(5,))	test	0.888888888888889	0.895379988060798	0.891943915746733	0.888299014580293	0.990347641565361
Neural Network (hidden_layer_sizes=(5,5))	train	0.926215277777778	0.92650655781998	0.926106762704477	0.925999997016153	0.990068667941812
Neural Network (hidden_layer_sizes=(5,5))	test	0.944444444444444	0.944321054054799	0.945269635833016	0.944648321011322	0.994307787610529
Neural Network (hidden_layer_sizes=(5,5,5)	train	0.940972222222222	0.941023862070459	0.940828344342969	0.940837176037885	0.994680228617463
Neural Network (hidden_layer_sizes=(5,5,5)	test	0.958333333333333	0.958622789367786	0.95915873620099	0.958450843987268	0.997804160992312
Neural Network (hidden_layer_sizes=(10,))	train	0.930555555555556	0.931633068543045	0.930290769792988	0.930192817714554	0.993583943313632
Neural Network (hidden_layer_sizes=(10,))	test	0.934027777777778	0.934882349653476	0.935069246960299	0.934534445106214	0.995226258014734

Figure 16: Performance Metrics for Dataset 2

algorithm_name	train_or_test_data	accuracy	precision_avg	recall_avg	f1_avg	auc_avg
Logistic Regression	train	0.863715277777778	0.862920518873092	0.863491925978704	0.863135770393	0.968972035685042
Logistic Regression	test	0.861111111111111	0.864778007771562	0.863684102495204	0.860162799719306	0.977844529277323
SVC Linear	train	0.864583333333333	0.863984831365745	0.864381260040502	0.864067192689714	0.968735356710566
SVC Linear	test	0.864583333333333	0.867928213210924	0.866764388663726	0.863313693305432	0.977571962707087
SVC RBF	train	0.864583333333333	0.864529482409575	0.864408481953537	0.86419290627969	0.968772588512345
SVC RBF	test	0.871527777777778	0.876584191164635	0.872834870910679	0.871008074797013	0.978809423235598
Random Forest (min_samples_leaf=1)	train	1	1	1	1	1
Random Forest (min_samples_leaf=1)	test	0.829861111111111	0.830739709851552	0.833140576367585	0.825630895295939	0.958513176761961
Random Forest (min_samples_leaf=3)	train	0.916666666666667	0.916651975378596	0.916519224923451	0.916547318742388	0.993294218666698
Random Forest (min_samples_leaf=3)	test	0.854166666666667	0.856765651554821	0.856674298573636	0.852885920050099	0.964203099483248
Random Forest (min_samples_leaf=5)	train	0.893229166666667	0.893458046976959	0.893105760233850	0.893186998504226	0.988534920923559
Random Forest (min_samples_leaf=5)	test	0.861111111111111	0.863071970602931	0.863042873030446	0.860124726129107	0.968905569774674
Neural Network (hidden_layer_sizes=(5,))	train	0.796006944444444	0.800078971038575	0.795115568342399	0.790942278565374	0.944150722312837
Neural Network (hidden_layer_sizes=(5,))	test	0.774305555555556	0.791415852905846	0.778883502392202	0.768910123251627	0.94494957995459
Neural Network (hidden_layer_sizes=(5,5))	train	0.801215277777778	0.801495523096703	0.800894366724692	0.798347872779753	0.950914824068182
Neural Network (hidden_layer_sizes=(5,5))	test	0.784722222222222	0.793914571425593	0.786015510124872	0.782603326559129	0.948478540802374
Neural Network (hidden_layer_sizes=(5,5,5)	train	0.8203125	0.819467767644964	0.819741614826474	0.816754272919839	0.953700866242091
Neural Network (hidden_layer_sizes=(5,5,5)	test	0.788194444444444	0.795386904761905	0.791457694231099	0.782508409652142	0.959137343781956
Neural Network (hidden_layer_sizes=(10,))	train	0.836805555555556	0.835650719493302	0.836306587668935	0.834558104029744	0.96091036786021
Neural Network (hidden_layer_sizes=(10,))	test	0.8125	0.821272540849686	0.816171189831090	0.809635416666667	0.962689100033015

Figure 17: Performance Metrics for Dataset 3

Conclusion

Across all datasets, model performance varied significantly based on the complexity of the data and the classification algorithm used. Logistic Regression and linear SVC excelled on Dataset 1 with clear class separation but failed on more complex datasets. Non-linear models like SVC with rbf kernel and Random Forest performed well on moderately mixed datasets, and Neural Networks showed the best performance on the most complex dataset, albeit with risks of overfitting. Ultimately, model choice should be driven by the nature of the data, with simpler models being more appropriate for clearly separable classes and more complex models like Neural Networks or Random Forests for datasets with significant overlap.

Main Learnings

This report summarizes the key learnings from the classification exercise, where multiple machine learning algorithms were applied to three datasets exhibiting varying degrees of class separability. Through this assignment, I explored the strengths and limitations of several classification models, including Logistic Regression, Support Vector Classifiers (SVCs), Random Forests, and Neural Networks, across different data complexity levels. The assignment not only reinforced my understanding of the theoretical aspects of these algorithms but also provided insights into their practical performance under diverse conditions.

Key Learnings

Importance of Data Structure and Class Boundaries

A significant takeaway from this assignment was the impact of data structure, particularly class separation, on model performance. In Dataset 1, where the class boundaries were distinct, simpler models such as Logistic Regression and SVC with a linear kernel performed exceptionally well. Their linear decision boundaries were well-suited to this dataset, and increasing model complexity did not yield substantial performance gains.

In contrast, Datasets 2 and 3, which exhibited increasing levels of class overlap, posed greater challenges for linear models. The overlap led to decreased performance metrics such as accuracy, precision, and recall. This highlighted the fact that data with non-linear class boundaries requires more sophisticated models, such as SVC with an rbf kernel or Neural Networks, which are capable of capturing complex relationships between features.

Effect of Model Complexity

This assignment also taught me the importance of balancing model complexity with the data at hand. For Dataset 1, simple models like Logistic Regression and linear SVC achieved high accuracy without overfitting. However, in more complex datasets, non-linear classifiers like Random Forest and Neural Networks outperformed linear models. The

ability of these complex models to handle non-linear patterns proved crucial, especially for Dataset 3, which had a high degree of class mixing.

However, increasing model complexity comes with risks, particularly overfitting. Neural Networks with deeper architectures, such as `hidden_layer_sizes=(5,5,5)`, demonstrated higher accuracy but also exhibited signs of overfitting, especially in Dataset 1. This reinforced the lesson that while deep models may have greater expressive power, they should be used judiciously, with careful tuning of hyperparameters such as the number of hidden layers or minimum leaf sizes in Random Forests.

Model Selection Based on Data Complexity

The assignment underscored the necessity of selecting models based on the complexity of the data. For simpler, linearly separable data, Logistic Regression or SVC with a linear kernel proved effective. These models are computationally efficient and provided near-optimal performance on Dataset 1.

However, for more complex datasets like Dataset 2 and Dataset 3, where class boundaries were non-linear or highly mixed, non-linear classifiers like SVC with an rbf kernel, Random Forests, and Neural Networks were essential. The Random Forest classifier with `min_samples_leaf=5`, for example, struck a good balance between bias and variance, reducing overfitting while maintaining strong performance on the test sets.

This learning emphasized that no single model is universally optimal across all datasets. The right choice depends on the characteristics of the data, including class separability and the presence of non-linear patterns.

Evaluating Models with Comprehensive Metrics

One of the most valuable aspects of this assignment was the experience of evaluating models using a broad range of metrics, including accuracy, precision, recall, F1-score, and AUC. I learned that accuracy alone can be misleading, especially in datasets with class imbalances or overlapping classes. Metrics like precision and recall, which provide class-specific insights, were crucial in understanding model performance, particularly for models that struggled with certain classes.

Additionally, the Area Under the Curve (AUC) metric proved useful for evaluating the classifier's ability to differentiate between classes, especially for the more complex datasets. This comprehensive evaluation enabled a more nuanced understanding of each model's strengths and weaknesses across different datasets.

Conclusion

Overall, this assignment provided invaluable insights into the behavior of various classification algorithms across datasets with different levels of complexity. I learned that data characteristics, particularly class boundaries, heavily influence model performance and that increasing model complexity should be done with caution to avoid overfitting. Finally, I gained an appreciation for the importance of using a wide range of evaluation metrics to fully assess a model's effectiveness. These lessons will be instrumental in future machine learning projects, where model selection, tuning, and evaluation will be critical for achieving optimal results.