Applied Machine Learning

Homework 1

Anand Kamble
Department of Scientific Computing
Florida State University

Enviroment Setup

For this homework, I will be using Python as my programming language with the `sklearn` package for the decision tree and random forest training and implementation.
I am using Conda for creating the environment and to manage the packages as well.

To set up the environment I used:

```
conda create -n "homework1" python3.11
pip install scikit-learn numpy matplotlib ipython jupyter --no-cache-dir
```

Note: `ipykernel` and `jupyter` are development dependencies.

The code used for this homework is available on my GitHub:
`https://github.com/anand-kamble/FSU-assignments/blob/main/Machine%20Learning/HW01/main.py`

1. A

First, we will load the MADELON dataset using `numpy` for both the training and test datasets.

Listing 1: Loading MADELON dataset

```python
import numpy as np

X_train: np.ndarray[np.float64] = np.loadtxt('./MADELON/madelon_train.data')
X_test: np.ndarray[np.float64] = np.loadtxt('./MADELON/madelon_valid.data')
Y_train: np.ndarray[np.float64] = np.loadtxt('./MADELON/madelon_train.labels')
Y_test: np.ndarray[np.float64] = np.loadtxt('./MADELON/madelon_valid.labels')
```

Next, we define two arrays to store the misclassification errors for both training and testing datasets.

Listing 2: Initializing error arrays

```python
train_error: list = []
test_error: list = []
```

Now we will train the decision trees with maximum depth ranging from 1 to 12. For this, we are using a for loop where the tree depth increases every iteration and for each depth we are calculating the misclassification errors on the training and test datasets. These errors are then appended to the error arrays.

Listing 3: Training Decision Trees with varying depths

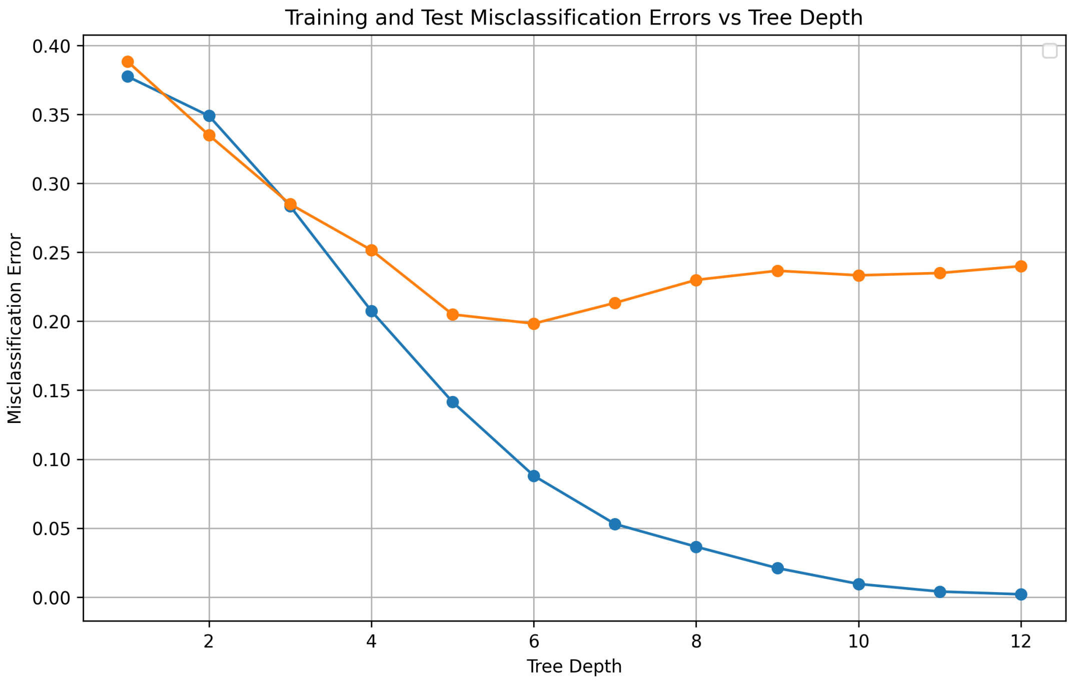
```python
for i in range(12):
    tree = DecisionTreeClassifier(max_depth=i+1)
    tree.fit(X_train, Y_train)
    Y_pred_test: np.ndarray = tree.predict(X_test)
    test_error.append(1 - accuracy_score(Y_test, Y_pred_test))
    Y_pred_train: np.ndarray = tree.predict(X_train)
    train_error.append(1 - accuracy_score(Y_train, Y_pred_train))
```

After successful execution of this code, we can plot the misclassification errors vs tree depth graph using `matplotlib`.

Listing 4: Plotting misclassification errors vs. tree depth

```
plt.figure(figsize=(10, 6),dpi=300)
plt.xlabel('Tree Depth')
plt.ylabel('Misclassification Error')
plt.title('Training and Test Misclassification Errors vs Tree Depth')
plt.legend()
plt.grid(True)
plt.plot(range(1,13),train_error, label='Training Error', marker='o')
plt.plot(range(1,13),test_error, label='Test Error', marker='o')
plt.show()
```

The plot looks like this:



Depth	Train Error	Test Error
1	0.3775	0.3883
2	0.349	0.3350
3	0.2835	0.2850
4	0.2075	0.2517
5	0.1415	0.2067
6	0.0890	0.1933
7	0.0530	0.2117
8	0.0355	0.2217
9	0.0215	0.2183
10	0.0090	0.2550
11	0.0055	0.2433
12	0.0020	0.2383

Table 1: Train and Test Errors at Various Depths

The tree with the depth 6 has the lowest Test error. The 6th row, which corresponds to a tree depth of 6, has been highlighted in yellow as it has the lowest test error.

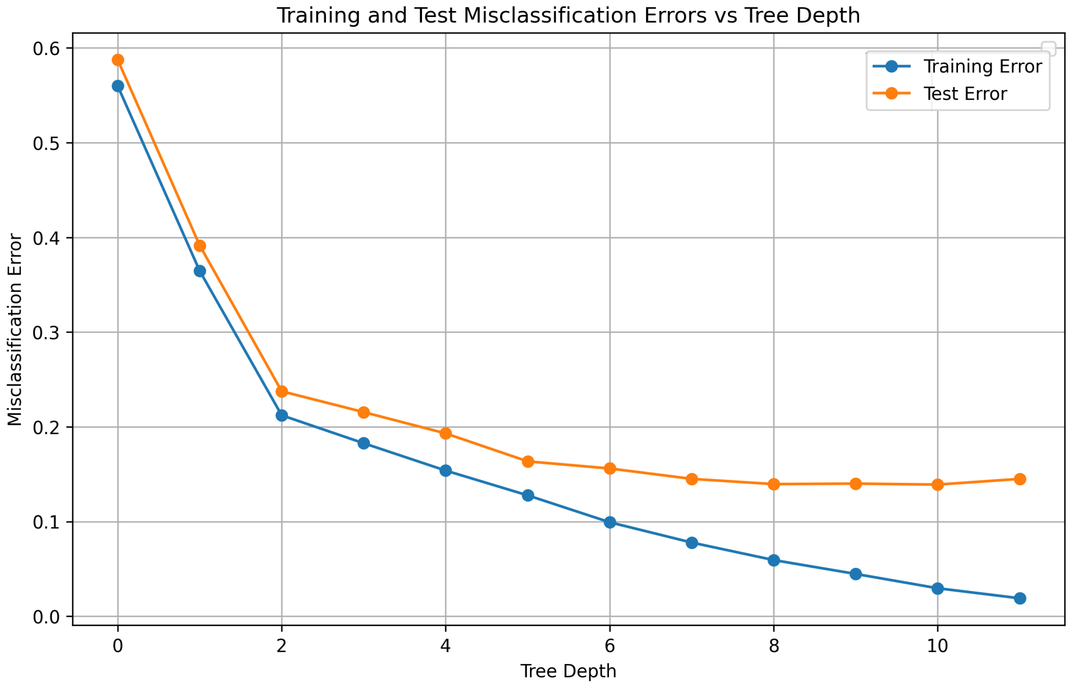
1. B

Now we will load the `satimage` dataset using `numpy`.

Listing 5: Loading satimage dataset

```python
import numpy as np

X_train: np.ndarray[np.float64] = np.loadtxt('./satimage/X.dat')
X_test: np.ndarray[np.float64] = np.loadtxt('./satimage/Xtest.dat')
Y_train: np.ndarray[np.float64] = np.loadtxt('./satimage/Y.dat')
Y_test: np.ndarray[np.float64] = np.loadtxt('./satimage/Ytest.dat')
```

We will use the same code as Part A to train the trees and predict the labels that are listed in listing 6 and we will also plot the misclassification errors vs tree depth graph using listing 4.



Depth	Train Error	Test Error
1	0.5599	0.5875
2	0.3646	0.3915
3	0.2122	0.2375
4	0.1826	0.2155
5	0.1538	0.1925
6	0.1276	0.1645
7	0.0992	0.1555
8	0.0778	0.1470
9	0.0593	0.1430
10	0.0449	0.1380
11	0.0300	0.1395
12	0.0189	0.1370

Table 2: Train and Test Errors at Various Depths

The tree with the depth 9 has the lowest Test error. The 9th row, which corresponds to a tree depth of 9, has been highlighted in yellow as it has the lowest test error.

1. C

For random forest we are using the `RandomForestClassifier` class from the `sklearn` package.

Listing 6: Training a Random Forest Classifier with varying number of trees

```
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(n_estimators=k_value)
```

The `RandomForestClassifier` class also allows us to define the split attribute by passing an argument `max_features` while creating the class. [1] To use $\sim \sqrt{500}$ features we will have to specify `max_features` as `sqrt`.

Listing 7: Training a Random Forest Classifier with a limited number of features per split

```
RandomForestClassifier(n_estimators=k_value, max_features='sqrt')
```

We are also specifying the number of trees by passing the argument `n_estimators`. We have stored all the values for k in a list. We are also using this k list to iterate over it and generate random forests with different number of trees.

Listing 8: Training a Random Forest Classifier with varying numbers of trees and calculating training and test errors using a limited number of features per split (square root of total features)

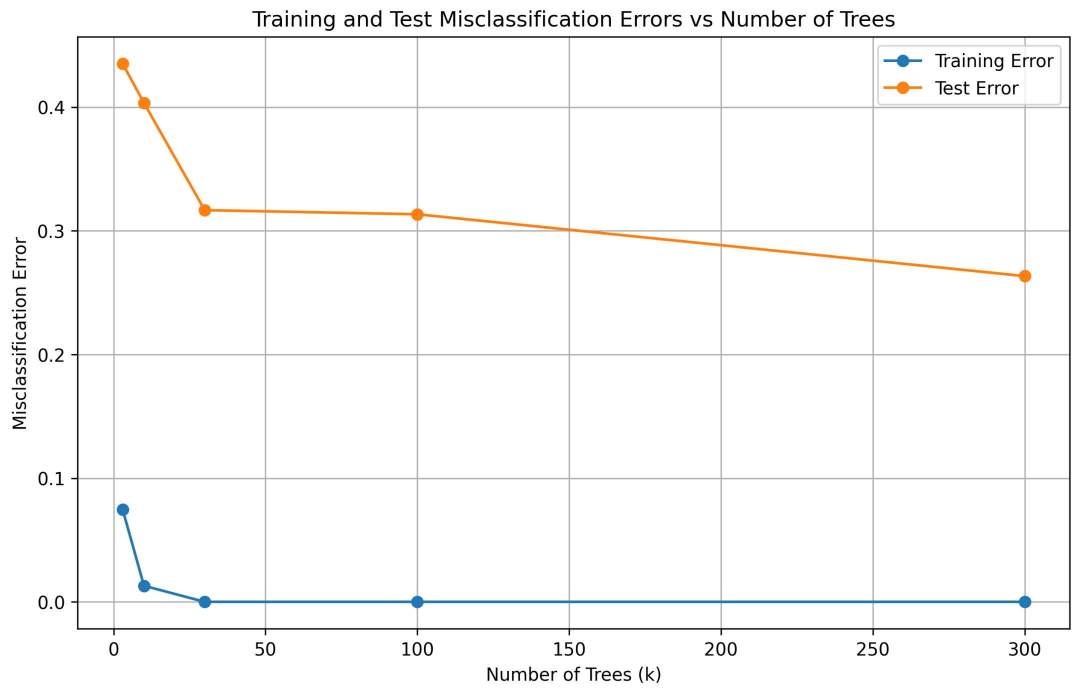
```
k: list[int] = [3, 10, 30, 100, 300]

train_error: list = []
test_error: list = []

for k_value in k:
    forest = RandomForestClassifier(n_estimators=k_value, max_features='sqrt')
    forest.fit(X_train, Y_train)

    Y_pred_test: np.ndarray = forest.predict(X_test)
    test_error.append(1 - accuracy_score(Y_test, Y_pred_test))

    Y_pred_train: np.ndarray = forest.predict(X_train)
    train_error.append(1 - accuracy_score(Y_train, Y_pred_train))
```

After successful training and evaluation, we can plot the training and test errors vs number of trees k as two separate curves using `matplotlib` as follows.

Listing 9: Plotting training and test misclassification errors against varying numbers of trees

```
plt.figure(figsize=(10, 6),dpi=300)
plt.plot(k, train_error, label='Training Error', marker='o')
plt.plot(k, test_error, label='Test Error', marker='o')
plt.xlabel('Number of Trees (k)')
plt.ylabel('Misclassification Error')
plt.title('Training and Test Misclassification Errors vs Number of Trees')
plt.legend()
plt.grid(True)
plt.show()
```

The plot is shown below:



Number of Trees (k)	**Training Error**	**Test Error**
3	0.0710	0.4250
10	0.0105	0.3700
30	0.0000	0.3683
100	0.0000	0.3017
300	0.0000	0.2633

Table 3: Training and Test Errors for Different Numbers of Trees in a Random Forest Classifier

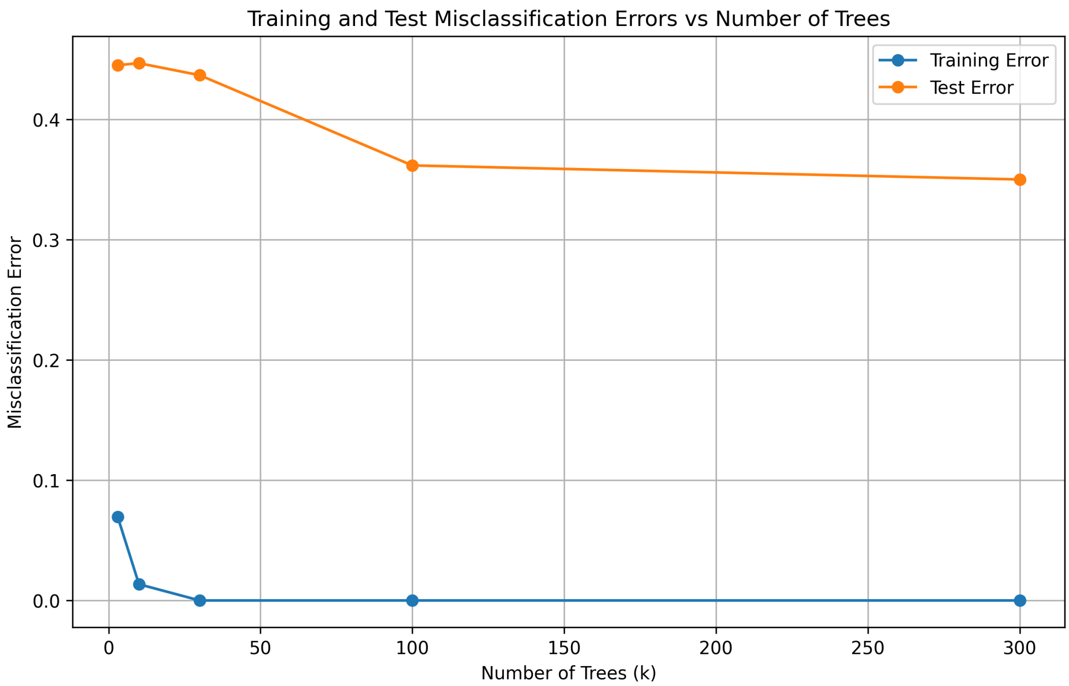
1. D

To specify the split attribute such that at each node it is chosen from a random subset of $\sim log_2(500)$ features we need to specify the `max_features` as `log2` as shown in listing 10.

Listing 10: Training a Random Forest Classifier with a limited number of features per split (log base 2 of the total features)

```
RandomForestClassifier(n_estimators=k_value, max_features='log2')
```

After successful training and evaluation, we get the plot of training and test errors vs number of trees k as two separate curves using `matplotlib` which is shown below:



Number of Trees (k)	Training Error	Test Error
3	0.0825	0.4867
10	0.0145	0.4700
30	0.0000	0.3733
100	0.0000	0.3450
300	0.0000	0.3417

Table 4: Training and Test Errors for Different Numbers of Trees in a Random Forest Classifier

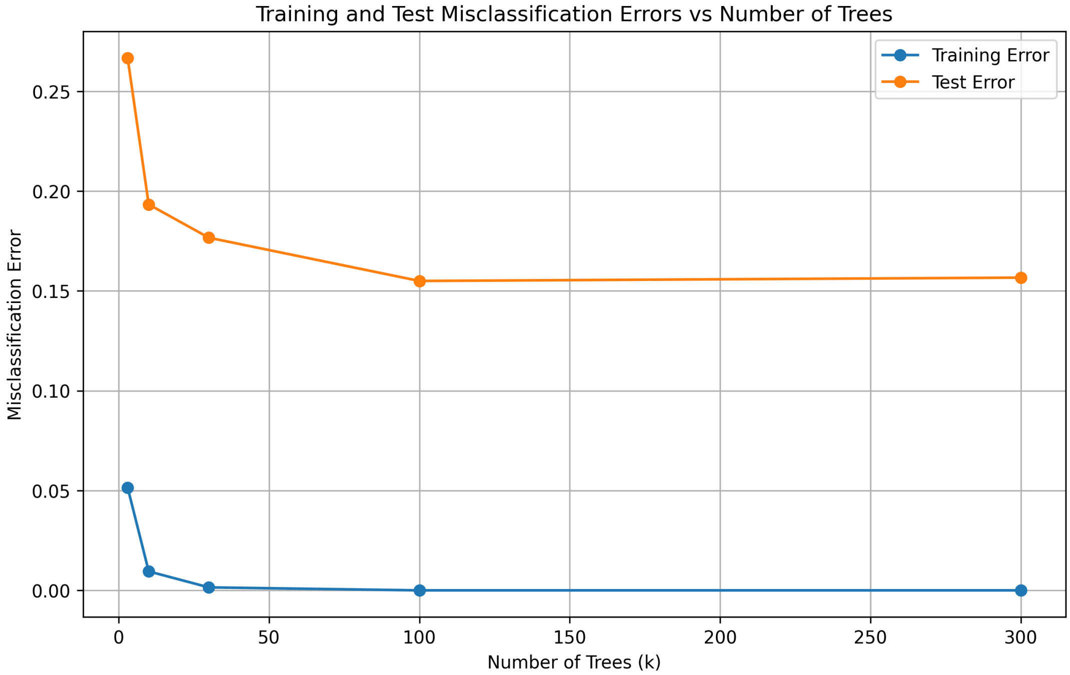
1. E

To specify that the split attribute at each node is chosen from all 500 features, we have to pass `max_features` as `None` as shown in listing 11.

Listing 11: Training a Random Forest Classifier with no limit on the number of features per split (using all features at each split)

```
RandomForestClassifier(n_estimators=k_value, max_features=None)
```

Below is the plot and the results when using all 500 features:



Number of Trees (k)	**Training Error**	**Test Error**
3	0.0445	0.2483
10	0.0075	0.2033
30	0.0010	0.1617
100	0.0000	0.1500
300	0.0000	0.1417

Table 5: Training and Test Errors for Different Numbers of Trees in a Random Forest Classifier

References

[1] Scikit-learn. *sklearn.ensemble.RandomForestClassifier — scikit-learn 1.0.2 documentation*. Accessed: 2024-09-04. 2024. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`.

[2] Scikit-learn. *sklearn.metrics.accuracy_score — scikit-learn documentation*. Accessed: 2024-09-04. 2024. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html`.

[3] Scikit-learn. *sklearn.tree.DecisionTreeClassifier — scikit-learn documentation*. Accessed: 2024-09-04. 2024. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`.