Moghees Ahmad

BSCS – 7B

Applied Machine Learning

Assignment - 02

Question No. 1: Decision Tree & Random Forest

(a)

The primary motivation behind ensemble methods in machine learning is to improve predictive accuracy, reduce overfitting, enhance model robustness, and handle complex data by combining multiple models to create a more accurate and reliable overall prediction. Ensembles are also more stable, less sensitive to data variations, and can offer insights into feature importance and model interpretability, making them versatile tools for various problem domains.

(b)

Training Random Forest:

- Select a random subset of the training data (with replacement).
- The size of this sample is typically the same as the original dataset but with random instances (this is called bagging).
- For each subset, build a decision tree with the following rules:
- a. Select a random subset of features (a subset of the original features) at each node of the tree.
- b. Split nodes to maximize information gain (or another suitable criterion).
- Repeat steps 1 and 2 to create a predefined number of decision trees, typically denoted as n_estimators.
- The ensemble of decision trees forms the Random Forest.

Pseudo code for training random forests:

```
forest = []

for i in range(n_estimators):

    sample_X, sample_y = get_subset_of_dataset(X, y)

    tree = build_decision_tree(sample_X, sample_y)

    forest.append(tree)

    return forest
```

Inference Random Forest:

- For each input data point, pass it through each decision tree in the forest.
- Collect the predictions from each tree.
- For classification tasks, use majority voting to determine the final class prediction.

Pseudo Code for Inference with a Random Forest:

predictions = []

for tree in forest:

tree_prediction = tree.predict(X)

predictions.append(tree_prediction)

final_prediction = majority_vote(predictions)

(c)

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

Reading Data

```
# Reading dataset
dataset = pd.read_csv("/content/Iris.csv", header = None)
# dropping id column and separating labels
X = dataset.iloc[1:,1:-1]
y = dataset.iloc[1:,-1]

# splitting dataset into test and train set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42, test_size=0.3)

# shapes of data
print("X_train Shape : " + str(X_train.shape))
print("Y_train Shape : " + str(y_train.shape))

print("X_test Shape : " + str(X_test.shape))
print("Y_test Shape : " + str(y_test.shape))
```

```
X_train Shape : (105, 4)
Y_train Shape : (105,)
X_test Shape : (45, 4)
Y_test Shape : (45,)
```

Decision Trees

```
DecisionTree = DecisionTreeClassifier()
DecisionTree.fit(X_train, y_train)

print("Decision Tree's Accuracy on Training Set : ", DecisionTree.score(X_train, y_train) * 100)
```

```
Decision Tree's Accuracy on Training Set :  100.0
```

```
# Predicting classes
y_pred = DecisionTree.predict(X_test)
# Accuracy
accuracy = accuracy_score(y_test,y_pred)
print("Decision Tree's Accuracy on Test Set : ", accuracy * 100)
```

```
Decision Tree's Accuracy on Test Set :  100.0
```

Random Forests

```
RandomForest = RandomForestClassifier()
RandomForest.fit(X_train, y_train)

print("Random Forest's Accuracy on Training Set : ", RandomForest.score(X_train, y_train) * 100)
```

```
Random Forest's Accuracy on Training Set :  100.0
```

```
# Predicting classes
y_pred = RandomForest.predict(X_test)
# Accuracy
accuracy = accuracy_score(y_test,y_pred)
print("Random Forest's Accuracy on Test Set : ", accuracy * 100)
```

```
Random Forest's Accuracy on Test Set :  100.0
```

How much better does it perform compared to the Decision Tree classifier?

Random forests are better in this case because they offer several advantages over a single decision tree. While the accuracy may appear similar, random forests provide greater robustness and generalization.

- **Reduced Overfitting:** Decision trees are prone to overfitting the training data, which means they can become too specific to the training data and perform poorly on new, unseen data. Random forests mitigate this issue by combining multiple decision trees, reducing overfitting.
- **Improved Generalization:** Random forests reduce the variance of the model. By averaging the results of multiple decision trees, they improve generalization, making them more reliable when applied to new data.
- **Handling Outliers:** Decision trees can be sensitive to outliers, leading to unusual splits in the data. Random forests can handle outliers more effectively because they aggregate the results from multiple trees.
- **Feature Importance:** Random forests can provide insights into feature importance. By tracking which features are used to split the data and how much they contribute to the model's performance, you can gain a better understanding of the data.
- **Ensemble Learning:** Random forests utilize ensemble learning, which combines multiple models to make predictions. This typically results in more stable and accurate predictions than a single decision tree.

Question No. 2: Support Vector Machine (SVM)

(a)

In Support Vector Machines (SVM), a support vector is a data point that is closest to the decision boundary (hyperplane) and has a non-zero value for the slack variable. Support vectors play a crucial role in defining the decision boundary of the SVM, and they are the data points that are most challenging to classify correctly. The decision boundary is defined by a weighted combination of support vectors, and these vectors effectively "support" the hyperplane.

(b)

Soft Margin and Hard Margin classifiers are concepts in Support Vector Machines (SVM). They differ in how they handle data points that are not perfectly separable by a hyperplane. Here's a differentiation between the two:

Hard Margin Classifier:

- **Perfect Separation**: A Hard Margin SVM seeks to find a hyperplane that perfectly separates the two classes of data. It assumes that the data is linearly separable, meaning that there exists a hyperplane that can completely separate all data points without any misclassification.
- **No Tolerance for Misclassification**: In a Hard Margin SVM, it is not allowed to have any data points within the margin (the region between the two parallel hyperplanes). All data points must be correctly classified and lie on the correct side of the hyperplane.
- **Risk of Overfitting:** Hard Margin SVMs can be sensitive to outliers or noisy data. If the data is not perfectly linearly separable or contains outliers, it may not be possible to find a hyperplane that satisfies the hard margin criteria. This can lead to overfitting.

Soft Margin Classifier:

- **Handling Non-Separable Data:** A Soft Margin SVM, on the other hand, allows for a certain degree of misclassification or overlapping of data points between classes. It is used when the data is not perfectly linearly separable, or when there's a desire to make the model more robust to noise or outliers.
- **Tolerance for Misclassification:** In a Soft Margin SVM, a "slack variable" is introduced to account for misclassified data points or those that fall within the margin. The optimization objective is to minimize both the margin width and the misclassification errors, with a trade-off parameter (C) that controls the balance between the two.
- **Robustness to Noise and Outliers:** Soft Margin SVMs are more robust to noisy data and outliers because they allow for a certain level of misclassification. The parameter C controls how much tolerance is allowed for misclassification.

(c)

```
!pip install idx2numpy

import numpy as np
import pandas as pd
import idx2numpy
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

Reading Dataset

```
X_train = idx2numpy.convert_from_file('/content/train-images.idx3-ubyte')
y_train = idx2numpy.convert_from_file('/content/train-labels.idx1-ubyte')

X_test = idx2numpy.convert_from_file('/content/t10k-images.idx3-ubyte')
y_test = idx2numpy.convert_from_file('/content/t10k-labels.idx1-ubyte')
```

```
print("X_train Shape : " + str(X_train.shape))
print("Y_train Shape : " + str(y_train.shape))

print("X_test Shape : " + str(X_test.shape))
print("Y_test Shape : " + str(y_test.shape))
```

```
X_train Shape : (60000, 28, 28)
Y_train Shape : (60000,)
X_test Shape : (10000, 28, 28)
Y_test Shape : (10000,)
```

Reshaping Data

```
# reshaping data
X_train = X_train.reshape(60000,-1)
X_test = X_test.reshape(10000,-1)

print("X_train Shape : " + str(X_train.shape))
print("Y_train Shape : " + str(y_train.shape))

print("X_test Shape : " + str(X_test.shape))
print("Y_test Shape : " + str(y_test.shape))
```

```
X_train Shape : (60000, 784)
Y_train Shape : (60000,)
X_test Shape : (10000, 784)
Y_test Shape : (10000,)
```

```
[30] linear_svm = SVC(kernel='linear', decision_function_shape='ovr')
     linear_svm.fit(X_train, y_train)

     print("Linear SVM's Accuracy on Training Set : ", linear_svm.score(X_train, y_train) * 100)

     Linear SVM's Accuracy on Training Set :  100.0

     # Predicting
     y_pred = linear_svm.predict(X_test)

     # Accuracy
     accuracy = accuracy_score(y_test,y_pred)
     print("Linear SVM's Accuracy on Test Set : ", accuracy * 100)

     Linear SVM's Accuracy on Test Set :  91.45
```

Standardizing Data

```
     # Standardization
     scaler = StandardScaler()

     scaler.fit(X_train)
     standardized_X_train = scaler.transform(X_train)

     scaler.fit(X_test)
     standardized_X_test = scaler.transform(X_test)
```

+ Code + Text

Linear SVM with Standardized Data

```
[33] linear_svm_with_standard_data = SVC(kernel='linear', decision_function_shape='ovr')
     linear_svm_with_standard_data.fit(standardized_X_train, y_train)

     print("Linear SVM with Standardized Data Accuracy on Training Set : ", linear_svm_with_standard_data.score(standardized_X_train, y_train) * 100)

     Linear SVM with Standardized Data Accuracy on Training Set :  99.60666666666667

     # Predicting
     y_pred = linear_svm_with_standard_data.predict(standardized_X_test)

     # Accuracy
     accuracy = accuracy_score(y_test,y_pred)
     print("Linear SVM with Standardized Data Accuracy on Test Set : ", accuracy * 100)

     Linear SVM with Standardized Data Accuracy on Test Set :  92.0
```

+ Code + Text

Non-Linear SVM with Standardized Data

```
[35] non_linear_svm = SVC(kernel='rbf', decision_function_shape='ovr')
     non_linear_svm.fit(standardized_X_train, y_train)

     print("Non-Linear SVM with Standardized Data Accuracy on Training Set : ", non_linear_svm.score(standardized_X_train, y_train) * 100)

     Non-Linear SVM with Standardized Data Accuracy on Training Set :  98.55000000000001

     # Predicting
     y_pred = non_linear_svm.predict(standardized_X_test)

     # Accuracy
     accuracy = accuracy_score(y_test,y_pred)
     print("Non-Linear SVM with Standardized Data Accuracy on Test Set : ", accuracy * 100)

     Non-Linear SVM with Standardized Data Accuracy on Test Set :  95.78999999999999
```

+ Code + Text