Assessment - 3

Course code: PMCA507P

Course Name: Machine Learning Lab

Programme: MCA

Faculty handling the course: Dr. Parimala M and Dr. Anitha A

School of Computer Science Engineering & Information Systems

Due Date: 2/4/2025(B1 slot) and 4/4/2025 (B2 slot)

Ex #1. Implementing Random Forest Regression in Python

Import Libraries

Here we are importing all the necessary libraries required.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import warnings

from sklearn.preprocessing import LabelEncoder
from sklearn.impute import KNNImputer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
warnings.filterwarnings('ignore')
```

Import Dataset

Now let's load the dataset in the panda's data frame. Create a dataset "salaries.csv" using the dataset given below

Position	Level	Salary
0	Business Analyst	45000
1	Junior Consultant	50000
2	Senior Consultant	60000
3	Manager	80000
4	Country Manager	110000
5	Region Manager	150000
6	Partner	200000
7	Senior Partner	300000
8	C-level	500000

9	CEO	1000000

```
df=  pd.read_csv('salaries.csv')
print(df)
```

```
df.info()
```

Data Preparation

Extracting Features: It extracts the features from the DataFrame and stores them in a variable named X.
Extracting Target Variable: It extracts the target variable from the DataFrame and stores it in a variable named y.

```
X  =  df.iloc[:,1:2].values
y  =  df.iloc[:,2].values
```

Random Forest Regressor Model

The code processes categorical data by encoding it numerically, combines the processed data with numerical data, and trains a Random Forest Regression model using the prepared data.

```
import  pandas  as  pd
from  sklearn.ensemble  import  RandomForestRegressor
from  sklearn.preprocessing  import  LabelEncoder


  Check  for  and  handle  categorical  variables
label_encoder  =  LabelEncoder()
x_categorical  =  df.select_dtypes(include=['object']).apply(label_encoder.fit_transform)
x_numerical  =  df.select_dtypes(exclude=['object']).values
x  =  pd.concat([pd.DataFrame(x_numerical),  x_categorical],  axis=1).values


regressor  =  RandomForestRegressor(n_estimators=10,  random_state=0,  oob_score=True)


regressor.fit(x,  y)
```

- `RandomForestRegressor`: It builds multiple decision trees and combines their predictions.
- `n_estimators=10`: Defines the number of decision trees in the Random Forest (10 trees in this case).
- `random_state=0`: Ensures the randomness in model training is controlled for reproducibility.
- `oob_score=True`: Enables out-of-bag scoring which evaluates the model's performance using data not seen by individual trees during training.
- `LabelEncoder()`: Converts categorical variables (object type) into numerical values, making them suitable for machine learning models.
- `select_dtypes()`: Selects columns based on data type—`include=['object']` selects categorical columns, and `exclude=['object']` selects numerical columns.
- `apply(label_encoder.fit_transform)`: Applies the LabelEncoder transformation to each categorical column, converting string labels into numbers.
- `concat()`: Combines the numerical and encoded categorical features horizontally into one dataset, which is then used as input for the model.
- `fit()`: Trains the Random Forest model using the combined dataset (`x`) and target variable (`y`).

5. Make predictions and Evaluation

The code evaluates the trained Random Forest Regression model:

- out-of-bag (OOB) score, which estimates the model's generalization performance.
- Makes predictions using the trained model and stores them in the 'predictions' array.
- Evaluates the model's performance using the Mean Squared Error (MSE) and R-squared (R2) metrics.

```python
from sklearn.metrics import mean_squared_error, r2_score

oob_score = regressor.oob_score_
print(f'Out-of-Bag Score: {oob_score}')

predictions = regressor.predict(x)

mse = mean_squared_error(y, predictions)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y, predictions)
print(f'R-squared: {r2}')
```

- **`mean_squared_error`**: Calculates the difference between true and predicted values (MSE).
- **`r2_score`**: Measures how well the model fits the data (R-squared value).
- **`oob_score_`**: Retrieves the out-of-bag score for model performance evaluation.
- **`predict()`**: Makes predictions using the trained Random Forest model.
- **`print()`**: Displays the model evaluation metrics: out-of-bag score, MSE, and R-squared.

Visualizing a Single Decision Tree from the Random Forest Model

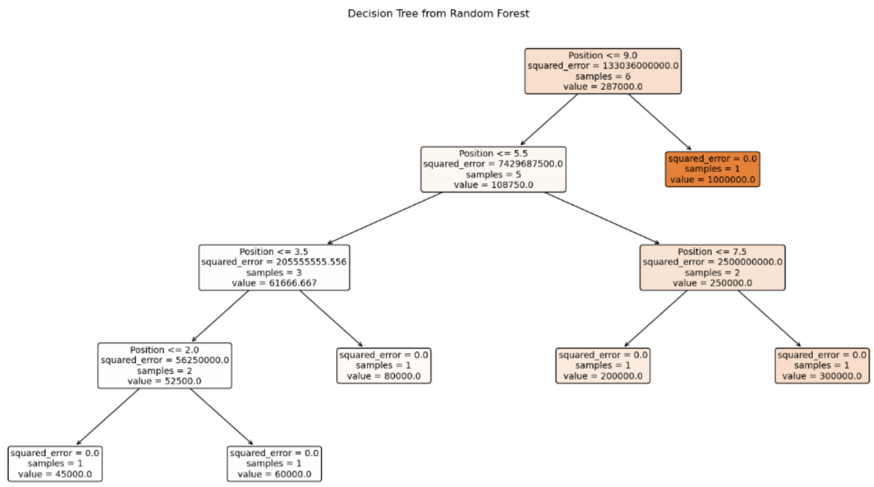
The code visualizes one of the decision trees from the trained Random Forest model. Plots the selected decision tree, displaying the decision-making process of a single tree within the ensemble.

```python
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Assuming regressor is your trained Random Forest model
# Pick one tree from the forest, e.g., the first tree (index 0)
tree_to_plot = regressor.estimators_[0]

# Plot the decision tree
plt.figure(figsize=(20, 10))
plot_tree(tree_to_plot, feature_names=df.columns.tolist(), filled=True, rounded=True, fontsize=10)
plt.title("Decision Tree from Random Forest")
plt.show()
```

Output:



Code

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import warnings
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import KNNImputer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
warnings.filterwarnings('ignore')
df= pd.read_csv('salaries.csv')
print(df)

df.info()

# data partition
X = df.iloc[:,1:2].values
y = df.iloc[:,2].values


# RF Regression model
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder

 #Check for and handle categorical variables
label_encoder = LabelEncoder()
x_categorical = df.select_dtypes(include=['object']).apply(label_encoder.fit_transform)
x_numerical = df.select_dtypes(exclude=['object']).values
x = pd.concat([pd.DataFrame(x_numerical), x_categorical], axis=1).values

regressor = RandomForestRegressor(n_estimators=10, random_state=0, oob_score=True)

regressor.fit(x, y)

# make prediction
from sklearn.metrics import mean_squared_error, r2_score

oob_score = regressor.oob_score_
print(f'Out-of-Bag Score: {oob_score}')

predictions = regressor.predict(x)

mse = mean_squared_error(y, predictions)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y, predictions)
print(f'R-squared: {r2}')

from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Assuming regressor is your trained Random Forest model
# Pick one tree from the forest, e.g., the first tree (index 0)
tree_to_plot = regressor.estimators_[0]

# Plot the decision tree
plt.figure(figsize=(20, 10))
plot_tree(tree_to_plot, feature_names=df.columns.tolist(), filled=True, rounded=True,
fontsize=10)
plt.title("Decision Tree from Random Forest")
plt.show()
```

output produced:

```
Position                  Level   Salary   Unnamed: 3
```

```
0          0   Business Analyst       1       45000
1          1  Junior Consultant       2       50000
2          2  Senior Consultant       3       60000
3          3            Manager       4       80000
4          4    Country Manager       5      110000
5          5     Region Manager       6      150000
6          6            Partner       7      200000
7          7     Senior Partner       8      300000
8          8            C-level       9      500000
9          9                CEO      10     1000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Position    10 non-null     int64
 1   Level       10 non-null     object
 2   Salary      10 non-null     int64
 3   Unnamed: 3  10 non-null     int64
dtypes: int64(3), object(1)
memory usage: 452.0+ bytes
Out-of-Bag Score: 0.9095977461691747
Mean Squared Error: 0.07699999999999999
R-squared: 0.9906666666666667
```

Ex # 2. Python implementation of AdaBoost

The Adaboost was explained using the iris dataset.

1. Import Libraries

Let's begin with importing important libraries that we will require to do our classification task:

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

2. Defining the AdaBoost Class

```
class AdaBoost:
    def __init__(self, n_estimators=50):
        self.n_estimators = n_estimators
        self.alphas = []
        self.models = []
```

- **AdaBoost** class is initialized with the number of weak learners (`n_estimators`).
- **self.alphas**: Stores the weight of each model based on its performance.
- **self.models**: Stores the weak classifiers (decision stumps) used in AdaBoost.

3. Training the AdaBoost Model (Fit Method)

```
    def fit(self, X, y):
        n_samples, n_features = X.shape
        w = np.ones(n_samples) / n_samples
```

- **n_samples, n_features**: Retrieves the number of samples and features from the dataset.
- **w**: Initializes sample weights uniformly.

```
        for _ in range(self.n_estimators):
```

```
                model = DecisionTreeClassifier(max_depth=1)
                model.fit(X, y, sample_weight=w)
                predictions = model.predict(X)
                err = np.sum(w * (predictions != y)) / np.sum(w)
                alpha = 0.5 * np.log((1 - err) / (err + 1e-10))
                self.alphas.append(alpha)
                self.models.append(model)
                w = w * np.exp(-alpha * y * predictions)
                w = w / np.sum(w)
```

- **err**: Computes the weighted error, penalizing misclassified samples more.
- **alpha**: Calculates the model weight based on its error. Models with lower error receive higher weight (`alpha`).
- **self.alphas.append(alpha)**: Appends the model's weight to the list.
- **self.models.append(model)**: Appends the trained weak classifier to the list.
- **w**: Updates the sample weights based on whether they were correctly or incorrectly classified

4. Making Predictions

```
    def predict(self, X):
        strong_preds = np.zeros(X.shape[0])
        for model, alpha in zip(self.models, self.alphas):
            strong_preds += alpha * model.predict(X)
        return np.sign(strong_preds).astype(int)
```

- **strong_preds**: Stores the aggregated predictions from all weak classifiers.

5. Example Usage

```
if __name__ == "__main__":

    X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    adaboost = AdaBoost(n_estimators=50)
    adaboost.fit(X_train, y_train)

    predictions = adaboost.predict(X_test)

    accuracy = accuracy_score(y_test, predictions)
    print(f"Accuracy: {accuracy * 100}%")
```

```
# Adaboost for iris data set from scratch
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Define Adaboost class
class AdaBoost:
    def __init__(self, n_estimators=50):
        self.n_estimators = n_estimators
        self.alphas = []
        self.models = []

    #Training the AdaBoost Model (Fit Method)
    def fit(self, X, y): # Define fit as a method within AdaBoost class
        n_samples, n_features = X.shape
        w = np.ones(n_samples) / n_samples
        for _ in range(self.n_estimators):
            model = DecisionTreeClassifier(max_depth=1)
            model.fit(X, y, sample_weight=w)
            predictions = model.predict(X)
            err = np.sum(w * (predictions != y)) / np.sum(w)
            alpha = 0.5 * np.log((1 - err) / (err + 1e-10))
            self.alphas.append(alpha)
            self.models.append(model)
            w = w * np.exp(-alpha * y * predictions)
            w = w / np.sum(w)
```

```
    def predict(self, X): # Define predict as a method within AdaBoost class
        strong_preds = np.zeros(X.shape[0])
        for model, alpha in zip(self.models, self.alphas):
            strong_preds += alpha * model.predict(X)
        return np.sign(strong_preds).astype(int)

from sklearn.datasets import make_classification
if __name__ == "__main__":
    X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
    ad = AdaBoost(n_estimators=50)
    print(ad)
    ad.fit(X_train, y_train)
    predictions = ad.predict(X_test) # Call predict on the 'ad' instance
    accuracy = accuracy_score(y_test, predictions)
    print(f"Accuracy: {accuracy * 100}%")
```

```
# Adaboost using sklearn inbuilt for iris dataset (Another program)

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris

# 1. Load the Iris dataset
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.DataFrame(iris.target, columns=['target'])

# 2. Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 3. Create an AdaBoost classifier
#   - n_estimators: Number of weak learners (decision trees) to use
#   - learning_rate: Contribution of each weak learner to the final prediction
#   - random_state: For reproducibility
adaboost = AdaBoostClassifier(n_estimators=50, learning_rate=0.5, random_state=42)

# 4. Train the model
adaboost.fit(X_train, y_train.values.ravel())

# 5. Make predictions on the test set
y_pred = adaboost.predict(X_test)

# 6. Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

Exercise 3.

K-Means algorithm K-means clustering algorithm is one of the well-known algorithms for clustering the data. The algorithm acts in an iterative way and is parametrised by K : the numbers of clusters we want to get.

1. Import the following python modules : matplotlib.pyplot, seaborn sns.set(), numpy and KMeans.
2. Execute the following code and observe what you obtain :

```
from sklearn.datasets.samples_generator import make_blobs
X, y = make_blobs(n_samples = 500, centers = 4,
        cluster_std = 0.40, random_state = 0)
plt.scatter(X[:, 0], X[:, 1], s = 50);
plt.show()
```

3. Initialise km to be the K-means algorithm, with the required parameter of how many clusters (n_clusters).
4. Train the K-means model with the input data.
5. Execute the following code to visualise the result of your training :

```
y_kmeans = km.predict(X)
plt.scatter(X[:, 0], X[:, 1], c = y_kmeans, s = 50, cmap = 'viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c = 'black', s = 200, alpha = 0.5);
plt.show()
```

****

Exercise - 4
One interesting application of clustering is in colour compression within images. For example, imagine you have an image with millions of colours. In most images, a large number of the colours will be unused, and many of the pixels in the image will have similar or even identical colours. Execute the following code :

```python
from sklearn.datasets import load_sample_image
china = load_sample_image("china.jpg")
ax = plt.axes(xticks=[], yticks=[])
ax.imshow(china);
```

It shows an image from datasets of `sklearn`.

1. Explain the result of the following instruction :

   ```python
   china.shape
   ```

2. On can see the image as a set of pixels and hence as as a cloud of points in a three-dimensional colour space. Give the instructions to reshape the data to `[n_samples x n_features]`, and rescale the colours so that they lie between 0 and 1.

You can use the function `plot_pixels` in the given file to visualise a subset of pixels.

Now we will use k-means algorithm to reduce these 16 million colours to just 16 colours. We will use a slightly different implementation of this algorithm. We will use the mini batch k-means which operates on subsets of the data to compute the result much more quickly than the standard k-means algorithm.

1. Give `python` instructions to obtain the new colours (the min batch k-means algorithm is implemented using `MiniBatchKMeans` function in `sklearn.cluster` module.

2. Execute the following instructions to observe the result :

   ```python
   china_recolored = new_colors.reshape(china.shape)
   fig, ax = plt.subplots(1, 2, figsize=(16, 6),
                          subplot_kw=dict(xticks=[], yticks=[]))
   fig.subplots_adjust(wspace=0.05)
   ax[0].imshow(china)
   ax[0].set_title('Original Image', size=16)
   ax[1].imshow(china_recolored)
   ax[1].set_title('16-color Image', size=16);
   ```

****