Assessment 3

April 4, 2025

1 Name: Akash Kumar Banik

2 Reg. No: 24MCA0242

2.1 Excerise 1 Implementing Random Forest Regression in Python

[1]:
```
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')
```

2.1.1 Importing the Dataset

[2]:
```
df= pd.read_csv('salaries.csv')
print(df)
df.info()
```

```
            Position  Level  Salary
0   Business Analyst      0   45000
1  Junior Consultant      1   50000
2  Senior Consultant      2   60000
3            Manager      3   80000
4    Country Manager      4  110000
5     Region Manager      5  150000
6            Partner      6  200000
7     Senior Partner      7  300000
8            C-level      8  500000
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 9 entries, 0 to 8
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Position  9 non-null      object
 1   Level     9 non-null      int64 
 2   Salary    9 non-null      int64 
dtypes: int64(2), object(1)
memory usage: 348.0+ bytes
```

2.1.2 Data Preparation

[3]:
```
X = df.iloc[:,1:2].values
y = df.iloc[:,2].values
```

2.1.3 Regressor Model

[4]:
```
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)
```

[4]:
```
RandomForestRegressor(n_estimators=10, oob_score=True, random_state=0)
```

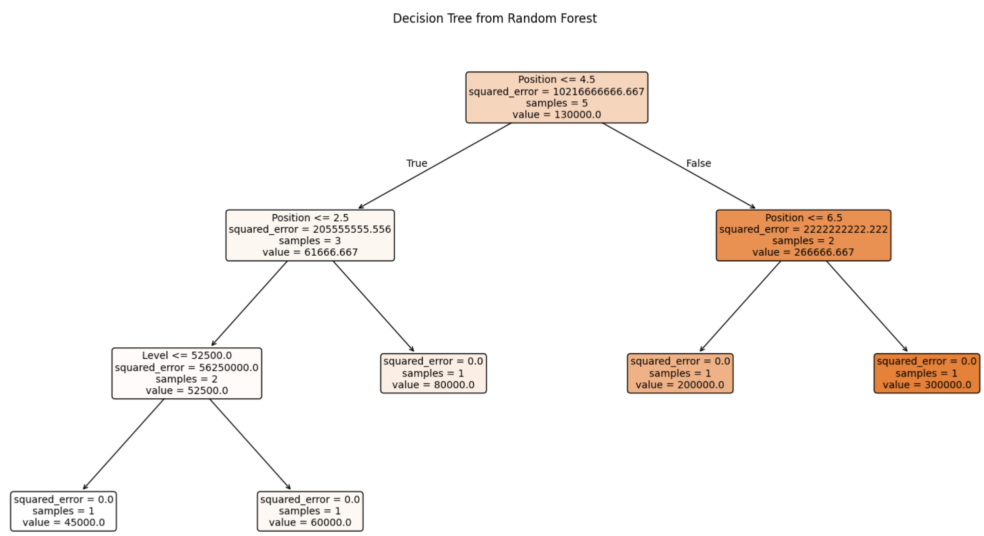
2.1.4 Making predictions and Evaluation

[5]:
```
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}')
```

```
Out-of-Bag Score: 0.48618707817464146
Mean Squared Error: 2826472222.2222223
R-squared: 0.8592930674205642
```

2.1.5 Visualizing a Single Decision Tree

[6]:
```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()
```



2.2 Exercise 2 Python implementation of AdaBoost

[8]:
```python
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
```

2.2.1 Defining the AdaBoost Class

[23]:
```python
class AdaBoost:
    def __init__(self, n_estimators=50):
        self.n_estimators = n_estimators
        self.alphas = []
```

```
        self.models = []

    # Training the AdaBoost Model
    def fit(self, X, y):
        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)

    # 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)
```

[24]:
```
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)
    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}%")
```

```
Accuracy: 84.0%
```

2.3 Exercise 3 K-Means clustering

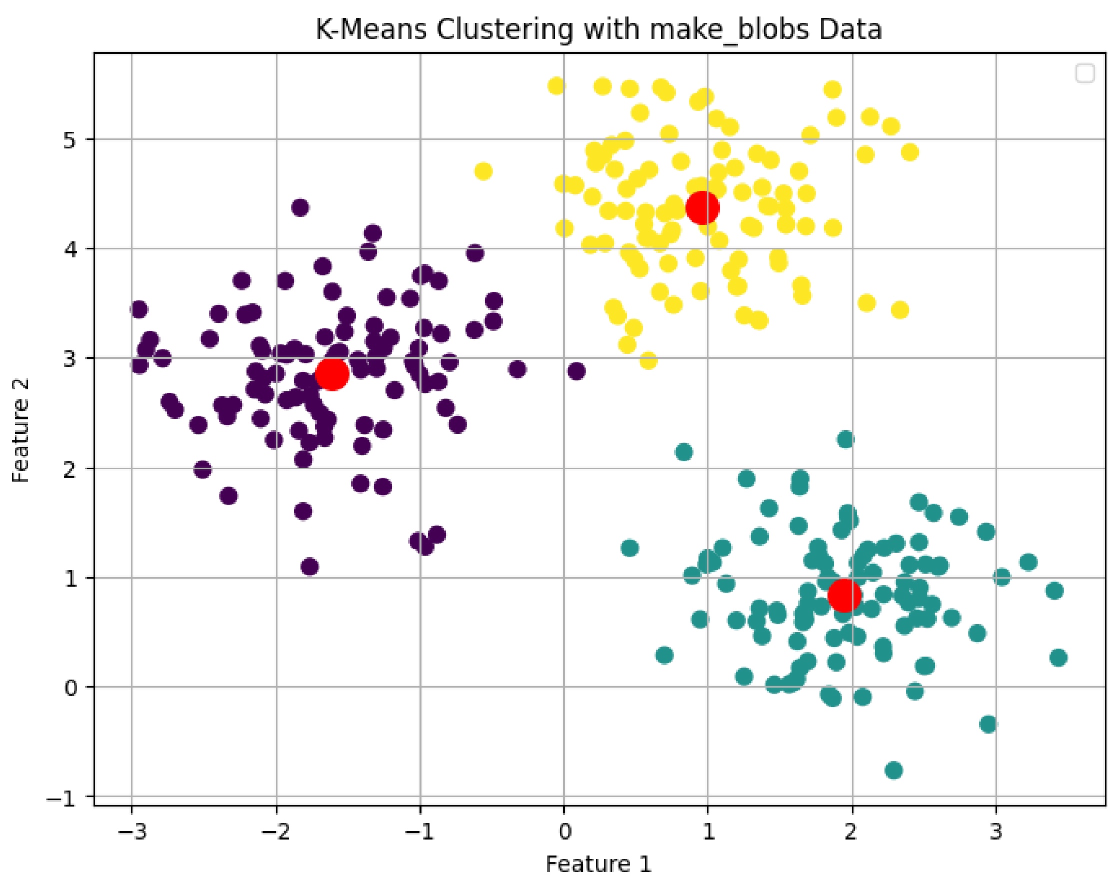
[31]:
```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Generate synthetic dataset
```

```
X, y_true = make_blobs(n_samples=300, centers=3, cluster_std=0.60,
 ↪random_state=0)

# Apply KMeans clustering
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(X)
labels = kmeans.labels_

# Plotting the clusters
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=200, c='red')
plt.title('K-Means Clustering with make_blobs Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid(True)
plt.show()
```



2.4 Exercise 4

2.4.1 One interesting application of clustering is in colour compression within images. For

2.4.2 example, imagine you have an image with millions of colours. In most images, a large

2.4.3 number of the colours will be unused, and many of the pixels in the image will have

2.4.4 similar or even identical colours. Execute the following code :

[30]:
```python
# Import required libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_sample_image
from sklearn.cluster import MiniBatchKMeans

# Load a sample image (china.jpg) from sklearn
china_image = load_sample_image("china.jpg")

# Prepare the image data for clustering
# Convert pixel values from 0-255 to 0-1 for easier processing
image_normalized = china_image / 255.0

# Reshape the 3D image (height x width x RGB) into a 2D array (num_pixels x 3)
# Each row is a pixel's RGB value
pixels = image_normalized.reshape(-1, 3)

# Apply MiniBatchKMeans to reduce the number of colors
# We choose 16 clusters (colors)
kmeans = MiniBatchKMeans(n_clusters=16, random_state=42)
kmeans.fit(pixels)                              # Fit the model to the pixel data

# Replace each pixel with the nearest color from the 16 found by k-means
new_colors = kmeans.cluster_centers_[kmeans.predict(pixels)]

# Reshape the new pixel data back to the original image shape
image_recolored = new_colors.reshape(china_image.shape)

# Show the original and the 16-color version side by side
fig, axes = plt.subplots(1, 2, figsize=(16, 6), subplot_kw=dict(xticks=[],
 ↪yticks=[]))
fig.subplots_adjust(wspace=0.05)

axes[0].imshow(china_image)
axes[0].set_title('Original Image', size=16)
```

```
axes[1].imshow(image_recolored)
axes[1].set_title('16-Color Image', size=16)

plt.show()
```