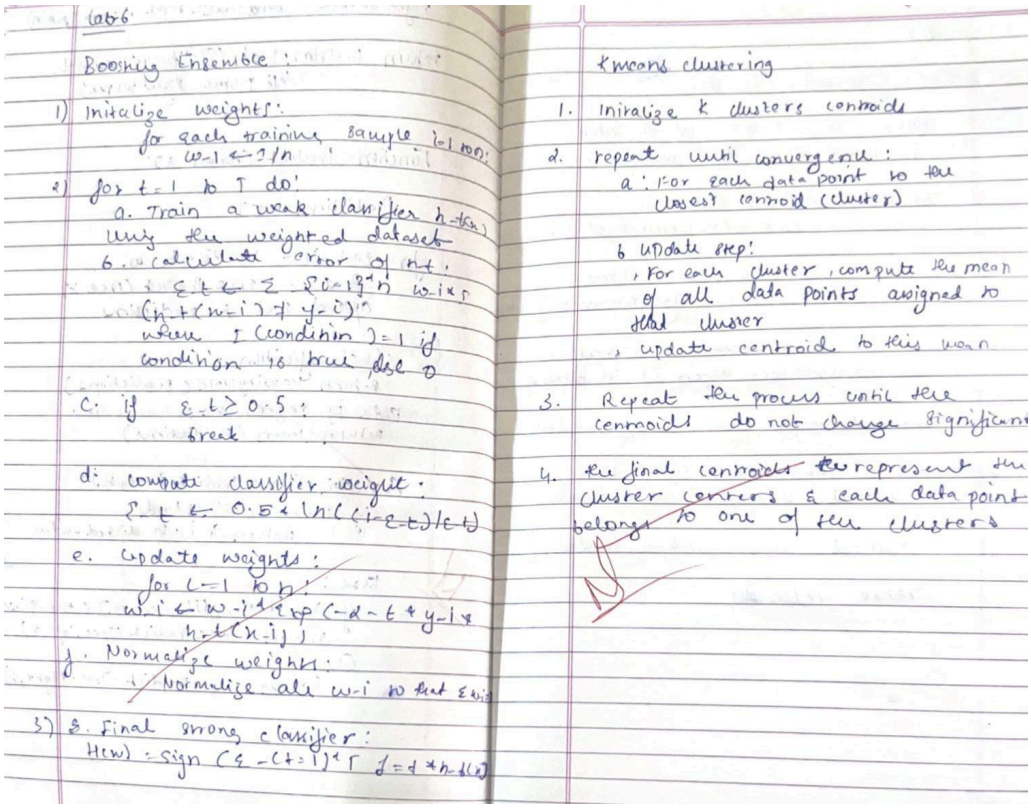
Implement Boosting ensemble and k means clustering method on a given dataset



Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Load your dataset from the local path
df = pd.read_csv("/content/data.csv")

# Drop any unnamed columns (like 'Unnamed: 32')
df.drop(columns=[col for col in df.columns if "Unnamed" in col],
inplace=True)

# Encode labels: M = 1 (Malignant), B = -1 (Benign)
df['diagnosis'] = df['diagnosis'].map({'M' : 1, 'B': -1})

# Drop ID column if present
```

```
if 'id' in df.columns:
    df.drop(columns=['id'], inplace=True)

# Features and labels
X = df.drop(['diagnosis'], axis=1).values
y = df['diagnosis'].values

# Normalize features
X = (X - X.mean(axis=0)) / X.std(axis=0)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Decision Stump class
class DecisionStump:
    def __init__(self):
        self.feature_index = None
        self.threshold = None
        self.polarity = 1
        self.alpha = None

    def predict(self, X):
        n = X.shape[0]
        predictions = np.ones(n)
        feature_values = X[:, self.feature_index]
        if self.polarity == 1:
            predictions[feature_values < self.threshold] = -1
        else:
            predictions[feature_values > self.threshold] = -1
        return predictions

# AdaBoost class
class AdaBoost:
    def __init__(self, n_clf=10):
        self.n_clf = n_clf
        self.clfs = []

    def fit(self, X, y):
        n_samples, n_features = X.shape
```

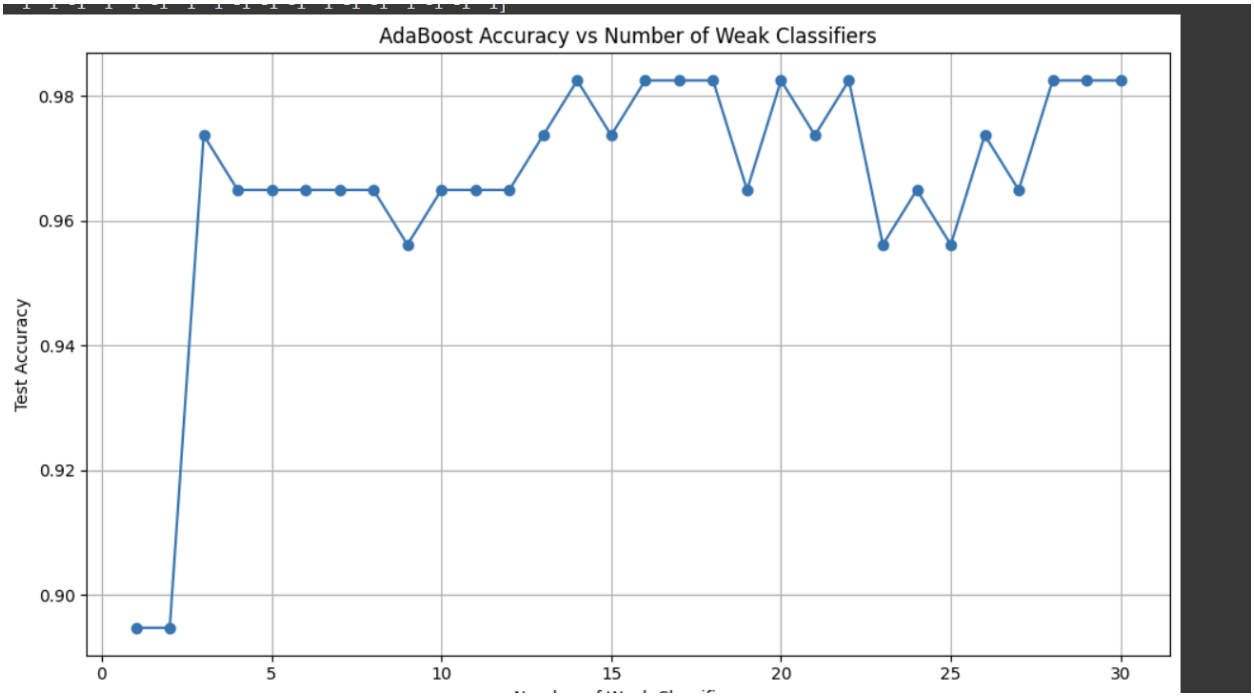
```
        weights = np.ones(n_samples) / n_samples

        for _ in range(self.n_clf):
            clf = DecisionStump()
            min_error = float('inf')

            for feature_i in range(n_features):
                feature_values = X[:, feature_i]
                thresholds = np.unique(feature_values)
                for threshold in thresholds:
                    for polarity in [1, -1]:
                        predictions = np.ones(n_samples)
                        if polarity == 1:
                            predictions[feature_values < threshold] = -1
                        else:
                            predictions[feature_values > threshold] = -1

                        error = np.sum(weights[y != predictions])

                        if error < min_error:
                            clf.polarity = polarity
                            clf.threshold = threshold
                            clf.feature_index = feature_i
                            min_error = error
```

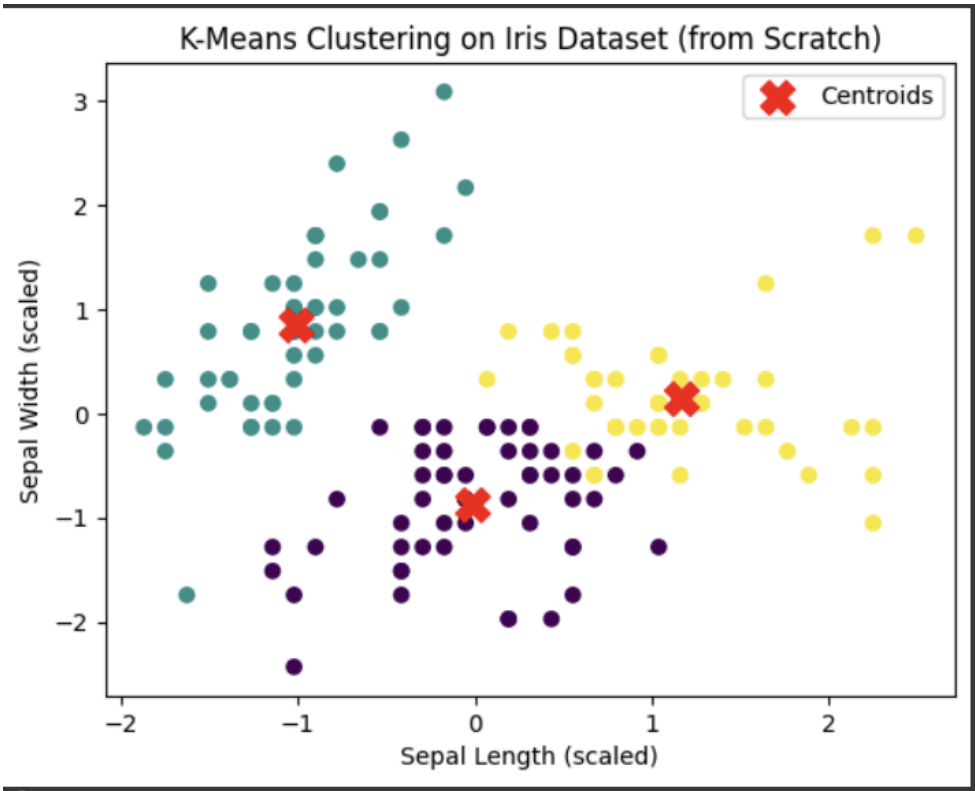
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (sepal length, sepal width, etc.)

# Step 2: Normalize the data (optional but helps in K-Means)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Define the K-Means function (from scratch)
```

```
def k_means(X, k, max_iters=100, tol=1e-4):
      # Step 3a: Randomly initialize the centroids by picking random
points from the data
    np.random.seed(42)
    centroids = X[np.random.choice(X.shape[0], k, replace=False)]

    # Step 3b: Iteratively update the centroids
    for _ in range(max_iters):
        # Step 3c: Assign each point to the closest centroid
        distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)
        labels = np.argmin(distances, axis=1)

        # Step 3d: Update the centroids (mean of points in each cluster)
         new_centroids = np.array([X[labels == i].mean(axis=0) for i in
range(k)])

          # Step 3e: Check for convergence (if centroids don't change,
stop)
        if np.all(np.abs(new_centroids - centroids) < tol):
            break

        centroids = new_centroids

    return centroids, labels

# Step 4: Apply the K-Means algorithm
k = 3  # As there are 3 species in Iris dataset
centroids, labels = k_means(X_scaled, k)

# Step 5: Visualizing the clusters (using the first two features for
simplicity)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='viridis',
marker='o')
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=200,
marker='X', label='Centroids')
plt.title('K-Means Clustering on Iris Dataset (from Scratch)')
plt.xlabel('Sepal Length (scaled)')
plt.ylabel('Sepal Width (scaled)')
plt.legend()
plt.show()
```

```
# Output Cluster Centers
print("Cluster Centers:\n", centroids)
```

Output:






```
Cluster Centers:
 [[-0.01139555 -0.87600831  0.37707573  0.31115341]
 [-1.01457897  0.85326268 -1.30498732 -1.25489349]
 [ 1.16743407  0.14530299  1.00302557  1.0300019 ]]
```