(21/4/25)

LAB-5:

→ Random Forest:

```
import pandas as pd
from google.colab import files
from sklearn.model_selection import train_test_split
uploaded = files.upload()
for filename in uploaded.keys():
        df = pd.read_csv(filename)
        display(df.head())
x = df.iloc[:, :-1]
y = df.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
                                          = 0.2, random_state = 42)

rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of Random Forest Model: {accuracy*100.2f}%")
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Output:

```
Accuracy of Random Forest Model: 72.08%
Classification Report:
        precision   recall   f1-score   support
0         0.79       0.78      0.78        99
1         0.61       0.62      0.61        55
```

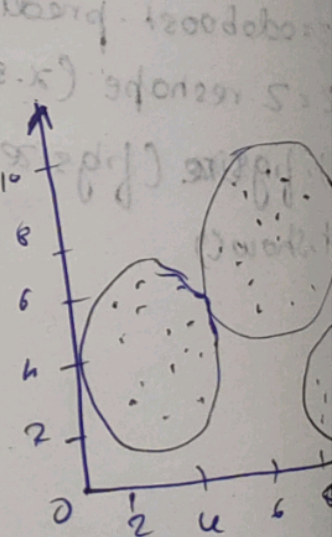
→ Boosting:

```
import numpy as np
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier.

sns.set(style = "whitegrid")
```

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

    def fit(self, X, y):
        n_samples, n_features = X.shape
        w = np.ones(n_samples)/n_samples
      for estimator in range(self.n_estimators):
            model = DecisionTreeClassifier(max_depth=1)
            model.fit(X, y, sample_weight=w)
            y_pred = model.predict(X)
            alpha = 0.5*np.log((1-err)/err) if err<1 else 0
            w = w * np.exp(-alpha*y*y_pred)
            w = w/np.sum(w)

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

X, y = make_classification(n_samples=500, n_features=2, n_info=2,
                       n_red=0, n_classes=2, random_state=42)

y = 2*y-1
adaboost = AdaBoost(n_estimator=50)
adaboost.fit(X,y)
x_min, x_max = X[:,0].min()-1, X[:,0].max()+1
y_min, y_max = X[:,1].min()-1, X[:,1].max()+1
Z = adaboost.predict(np.c_[x.ravel(), y.ravel(),])
Z = Z.reshape(x.shape)
plt.figsize(figsize=(10,6))
plt.show()
```

→ K-Means Clustering :

```
class K_Means:
    def __init__(self, k=2, tolerance=0.001, max_iter=500):
        self.k = k
        self.max_it = max_iter
        self.tolerance = tolerance

    def predict(self, data):
        self.centroids = {}
        classification = distances.index(min(distances))
        return classification.

    def fit(self, data):
        for i in range(self.k):
            self.centroids[i] = data[i]

        for i in range(self.max_iter):
            self.classes = {}
            for j in range(self.k):
                self.classes[j] = []

            for cluster_index in self.classes:
                self.centroids[cluster_index] = np.average(self.classes[
                    cluster_index], axis=0)


def main():
    K = 3
    center_1 = np.array([1, 1])
    center_2 = np.array([5, 5])
    center_3 = np.array([8, 1])

    cluster_1 = np.random.rand.(100, 2) + center_1
    cluster_2 = np.random.rand(100, 2) + center_2
    cluster_3 = np.random.rand(100, 2) + center_3

    k_means = K_Means(K)
    k_means.fit(data)

    for centroid in k_means.centroids:
        plt.scatter(s=130, marker="X")
```



```
for cluster_index in k_means.classes:
    color = colors [cluster_index]
    for features in k_means. classes [cluster_index]:
        plt.scatter (features[0], features[1], color = color, s = 30)

if __name__ == "__main__":
    main()
```

→ Principle Component Analysis (PCA):

```
import pandas as pd
from sklearn.decomposition import PCA
uploaded = files.upload()
for filename in uploaded.keys():
    df = pd.read_csv(filename)
    display (df.head())
numeric_df = df.select_dtypes (include = [np.number])
selected_features = numeric_df.columns
x = numeric_df [selected_features]. dropna ()
x_scaled = StandardScaler (). fit_transform(x)
pca = PCA (n_components = 2)
principal_components = pca . fit_transform (x_scaled)
pca_df = pd.DataFrame (data = principal_components, columns = ['PC1',
plt. figure (figsize = (8,6))
plt.grid (True)
plt. show ()
print ("Explained variance Ratio:", pca.explained_var_ratio)
```

Output:

```
Explained variance Ratio : [0.5216, 0.2863]
```