LAB-05

PAGE NO :
DATE :

1) Random Forest Ensemble Algorithm

```
import numpy as np
from collections . import Counter
from random import sample.

def train



import pandas as pd
from sklearn. model_select import train_test_split
from Sklearn. ensemble import RandomForest classifier
from google.colab import files.

uploaded = files. upload ()

for filename in uploaded. key ():
    df = pd. read_csv(filname)
    print (f" Data loded : { filename }")
    display (df. head ())

x = df. iloc [: , :-1]
y = df. iloc [: , -1]

x_train, x_test, y_train, y_test = train test
    = train_test_split (x, y, test_size=0.2,
        random_state = 42)

rf_model = RandomForestclassifier (n_estimators=100,
        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 (f" Accuracy of Random forest : {accuracy *100:
                                       .2f} %")
print (" classification Report : ")
print (classification_report (y_test, Y_pred))
```

output :-

```
Student.csv.
Data loaded from: Student.csv


Accuracy of Random Forest Model : 100.00 %.
classification Report :
              precision    recall   f1-score   support
     0          1.00        1.00      1.00        1
     1          1.00        1.00      1.00        1

accuracy                              1.00        2
macro avg       1.00        1.00      1.00        2
weighted avg    1.00        1.00      1.00        2
```

② Principle component

```
import pandas as pd
import numpy as np
from ~~numpy as np~~ sklearn.decomposition import
                                   PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from google.colab import files
```

```
uploaded = files.upload()

for filename in uploaded.key():
    df = pd.read_csv(filename)
    print(f"uploaded:{filename}")
    display(df.head())

numeric_df = df.select_dtypes(include=
                        [np.number])
print("Numerical feature found:", list(numericdf.
                        columns))

selected_features = numeric_df.colums

x = numeric_df[selected_features].dropna()
x_scaled = standardscaler().fit_transform(X)

pca  = PCA(n_components = 2)
principal_components = pca.fit_transform(x_scal

pca_df = pd.DataFrame(data= principal_
        components, colums =['PC1', 'PC2'])

plt.figure(figure=(8,6))
 plt.scatter(pca_df['pc1'], pca_df['pc2'],
                        alpha= 0.7)
plt.xlabel('principal component 1')
plt.ylabel('principal component 2')
plt.title('2D PCA visualization')
plt.grid('~~2D PCA~~ Vi due')
 plt.show.
```

Output!-

```
Student.csv
uploaded : student.csv

Numerical features found : ['Hours_Studied', '
                 'Passed']
```

③ **K-Mean**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
import seaborn as sns
from sklearn.cluster import KMeans

iris = datasets.load_iris()
print("Dataset loaded sucessfuly")

Data = pd.DataFrame(iris.data, colums = iris.
            feature_names)

X = Data.iloc[:,0:3].values
css = []
kmean = KMeans(n_clusters=3, init='k-means
    ++', max_iter=100, n_init=10, random_state=0)
```

PAGE NO :

DATE :

```
y_Kmeans = Kmeans.fit_predict (x)
kmeans. cluster_centers .

plt. Scatter (x [Y_Kmeans = = 0 , 0], x[Y_Kmeans ==0,1]
  , s = 100, c = 'red', label = "Iris-setosa')
plt. scatter (x [Y_Kmeans = = 1, 0], x[ Y_Kmeans = = 1,1]
  s = 100, c = 'blue', label = 'Iris-versicolor')
plt. Scatter (x [Y_Kmeans = = 2, 0], x [Y_means ==
  2, 1], s = 100, c = 'green', label = 'Iris-vir'

plt. scatter (Kmeans. cluster_centers_[: , 0],
  Kmeans. cluster_centers_ [: , 1], s = 100, c = 'black'
  , label = 'Centroids')

plt. legend ()
```

(4) Booster cluster

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn. tree import DecisionTreeClassifier
from sklearn. datasets import make_classification
from sklearn. metrics import accuracy_score
from sklearn. decomposition import PCA

sns. set (style = "whitegrid")

class AdaBoost :
  def __init__ (self, n_estimators = 50):
        self.n_estimators = n_estimators.
        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)

            err = np.sum(w * (y_pred != y)) / np.sum(w)
            self.errors.append(err)

            alpha = 0.5 * np.log((1-err) / err) if err<1
                                         else 0.
            self.alphas.append(alpha)
            self.models.append(model)

        ~~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)
```

PAGE NO :
DATE :

```
def score (self, x, y):
    return accuracy_score (y_self.predict
                                   (x))

x, y = make_classification(n_samples= 500,
    n_features = 2, n_informative = 2, n_redundant=0,
    n_classes= 2, random_state=42)

y = 2 * y - 1

addboost = AddBoost (n_estimators=50)
adaboost.fit (x, y)

accuracy = adaboost.score(X, y)
print (f" model accuracy: {acc :.4f}")
```

21.04