##BITS F464 - Semester 1 - MACHINE LEARNING

PROJECT - MACHINE LEARNING FOR SUSTAINABLE DEVELOPMENT GOALS (SDGs)

Team number: 27

(*In Title case, separated with commas*) ***Full names of all students in the team: Mohammed Imran, Nallabolu Sreethi Reddy, Samhitha Vaddempudi, Jonnalagadda Sripada Reddy, Prasanth VV.***

(*Separated by commas*) ***Id number of all students in the team: 2021A7PS0001H, 2021A7PS0020H, 2021A7PS0175H, 2021A7PS2813H, 2021A3PS0780H.***

Please refer to the email providing the assignment of project and follow the instructions provided in the project brief.

1. Preprocessing of Dataset

The respective dataset has been shared in the project brief. Please refer to it.

```
!pip install catboost



import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

file_path = 'Heart_Disease.csv'

# Read the CSV file without column labels
df = pd.read_csv(file_path, header=None)

column_names = ['hospital', 'age', 'sex', 'cp', 'trestbps', 'chol',
'fbs', 'restecg', 'thalach',
                'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num']

# Assign the column names to the DataFrame
df.columns = column_names

# Replace "?" with a standard notation (e.g., NaN for missing values)
```

```
df.replace('?', pd.NA, inplace=True)

# Define the mapping for the 'hospital' column
hospital_mapping = {
    'Cleveland': 0,
    'Hungarian': 1,
    'Switzerland': 2,
    'VA': 3
}

df['hospital'] = df['hospital'].replace(hospital_mapping)

df['num'] = np.clip(df['num'], 0, 1)


# Display the DataFrame with the new column names
print(df)

# Save the DataFrame to a new CSV file
output_file_path = 'Heart_data_labeled.csv'
df.to_csv(output_file_path,index=False)
```

```
     hospital  age  sex  cp trestbps chol fbs restecg thalach exang
oldpeak  \
0           0   63    1   1      145  233   1       2     150     0
2.3
1           0   67    1   4      160  286   0       2     108     1
1.5
2           0   67    1   4      120  229   0       2     129     1
2.6
3           0   37    1   3      130  250   0       0     187     0
3.5
4           0   41    0   2      130  204   0       2     172     0
1.4
..        ...  ...  ...  ..      ...  ...  ..     ...     ...   ...
...
915         3   54    0   4      127  333   1       1     154     0
0
916         3   62    1   1     <NA>  139   0       1    <NA>  <NA>
<NA>
917         3   55    1   4      122  223   1       1     100     0
0
918         3   58    1   4     <NA>  385   1       2    <NA>  <NA>
<NA>
919         3   62    1   2      120  254   0       2      93     1
0

    slope   ca  thal  num
0       3    0     6    0
1       2    3     3    1
```

```
2       2    2     7    1
3       3    0     3    0
4       1    0     3    0
..    ...  ...   ...  ...
915   NaN  NaN  <NA>    1
916   NaN  NaN  <NA>    0
917   NaN  NaN     6    1
918   NaN  NaN  <NA>    0
919   NaN  NaN  <NA>    1

[920 rows x 15 columns]
```

```
def resumetable(df):
    print(f"Dataset Shape: {df.shape}")
    summary = pd.DataFrame(df.dtypes,columns=['dtypes'])
    summary = summary.reset_index()
    summary['Name'] = summary['index']
    summary = summary[['Name','dtypes']]
    summary['Missing'] = df.isnull().sum().values
    summary['Uniques'] = df.nunique().values
    summary['First Value'] = df.loc[0].values
    summary['Second Value'] = df.loc[1].values
    return summary

file_loc =  'Heart_data_labeled.csv'

df = pd.read_csv(file_loc)

# Shuffle the data
df = df.sample(frac=1).reset_index(drop=True)

df.head()
```

```
   hospital  age  sex  cp  trestbps   chol  fbs  restecg  thalach
exang  \
0         3   63    1   4     160.0  267.0  1.0      1.0     88.0
1.0
1         0   44    1   4     112.0  290.0  0.0      2.0    153.0
0.0
2         3   61    1   1     142.0  200.0  1.0      1.0    100.0
0.0
3         0   58    0   3     120.0  340.0  0.0      0.0    172.0
0.0
4         0   50    1   4     150.0  243.0  0.0      2.0    128.0
0.0

   oldpeak  slope   ca  thal  num
0      2.0    NaN  NaN   NaN    1
1      0.0    1.0  1.0   3.0    1
2      1.5    3.0  NaN   NaN    1
```

```
3      0.0    1.0  0.0   3.0    0
4      2.6    2.0  0.0   7.0    1
```

```
print ("Total number of rows in dataset = {}".format(df.shape[0]))
print ("Total number of columns in dataset = {}".format(df.shape[1]))
```

```
Total number of rows in dataset = 920
Total number of columns in dataset = 15
```

```
result = resumetable(df)
result
```

```
Dataset Shape: (920, 15)
```

	Name	dtypes	Missing	Uniques	First Value	Second Value
0	hospital	int64	0	4	3.0	0.0
1	age	int64	0	50	63.0	44.0
2	sex	int64	0	2	1.0	1.0
3	cp	int64	0	4	4.0	4.0
4	trestbps	float64	59	61	160.0	112.0
5	chol	float64	30	217	267.0	290.0
6	fbs	float64	90	2	1.0	0.0
7	restecg	float64	2	3	1.0	2.0
8	thalach	float64	55	119	88.0	153.0
9	exang	float64	55	2	1.0	0.0
10	oldpeak	float64	62	53	2.0	0.0
11	slope	float64	309	3	NaN	1.0
12	ca	float64	611	4	NaN	1.0
13	thal	float64	486	3	NaN	3.0
14	num	int64	0	2	1.0	1.0

```
target_col = "num"
X = df.loc[:, df.columns != target_col]
y = df.loc[:, target_col]
```

```
import numpy as np
import pandas as pd
```

```
# Define the split ratio (e.g., 67% training, 33% testing)
split_ratio = 0.67
split_index = int(len(df) * split_ratio)

# Split the data into training and testing sets using Pandas
train_data = df.iloc[:split_index, :]
test_data = df.iloc[split_index:, :]

# Use Pandas to get X_train, y_train, X_test, and y_test
X_train = train_data.drop('num', axis=1)
```

```
y_train = train_data['num']

X_test = test_data.drop('num', axis=1)
y_test = test_data['num']

X_train.head()
```

```
   hospital  age  sex  cp  trestbps   chol  fbs  restecg  thalach
exang  \
0         3   63    1   4     160.0  267.0  1.0      1.0     88.0
1.0
1         0   44    1   4     112.0  290.0  0.0      2.0    153.0
0.0
2         3   61    1   1     142.0  200.0  1.0      1.0    100.0
0.0
3         0   58    0   3     120.0  340.0  0.0      0.0    172.0
0.0
4         0   50    1   4     150.0  243.0  0.0      2.0    128.0
0.0

   oldpeak  slope   ca  thal
0      2.0    NaN  NaN   NaN
1      0.0    1.0  1.0   3.0
2      1.5    3.0  NaN   NaN
3      0.0    1.0  0.0   3.0
4      2.6    2.0  0.0   7.0
```

```
features = list(X_train.columns)

cat_features = ["hospital", "sex", "cp", "fbs", "restecg", "exang",
"slope", "ca", "thal"]
```

2. ML Model 1

```
import numpy as np


def unit_step_func(x):
    return np.where(x > 0 , 1, 0)

class Perceptron:

    def __init__(self, learning_rate=0.01, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.activation_func = unit_step_func
        self.weights = None
        self.bias = None
```

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

        # init parameters
        self.weights = np.zeros(n_features)
        self.bias = 0

        y_ = np.where(y > 0 , 1, 0)

        # learn weights
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self.activation_func(linear_output)

                # Perceptron update rule
                update = self.lr * (y_[idx] - y_predicted)
                self.weights += update * x_i
                self.bias += update


    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        y_predicted = self.activation_func(linear_output)
        return y_predicted


# Testing
if __name__ == "__main__":

    p = Perceptron(learning_rate=0.01, n_iters=1000)
    p.fit(X_train, y_train)
    predictions = p.predict(X_test)

    print("Perceptron classification accuracy", accuracy(y_test,
predictions))

    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    plt.scatter(X_train[:, 0], X_train[:, 1], marker="o", c=y_train)

    x0_1 = np.amin(X_train[:, 0])
    x0_2 = np.amax(X_train[:, 0])

    x1_1 = (-p.weights[0] * x0_1 - p.bias) / p.weights[1]
    x1_2 = (-p.weights[0] * x0_2 - p.bias) / p.weights[1]

    ax.plot([x0_1, x0_2], [x1_1, x1_2], "k")

    ymin = np.amin(X_train[:, 1])
```

```
    ymax = np.amax(X_train[:, 1])
    ax.set_ylim([ymin - 3, ymax + 3])

    plt.show()
```

3. ML Model 2

```
import numpy as np
from collections import Counter

def euclidean_distance(x1, x2):
    distance = np.sqrt(np.sum((x1-x2)**2))
    return distance

class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        predictions = [self._predict(x) for x in X]
        return predictions

    def _predict(self, x):
        # compute the distance
        distances = [euclidean_distance(x, x_train) for x_train in
self.X_train]

        # get the closest k
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]

        # majority voye
        most_common = Counter(k_nearest_labels).most_common()
        return most_common[0][0]

clf = KNN(k=5)
clf.fit(X_train, y_train)
predictions = clf.predict(X_test)

print(predictions)

acc = np.sum(predictions == y_test) / len(y_test)
print(acc)
```

4. ML Model 3

```
import numpy as np


# Decision stump used as weak classifier
class DecisionStump:
    def __init__(self):
        self.polarity = 1
        self.feature_idx = None
        self.threshold = None
        self.alpha = None

    def predict(self, X):
        n_samples = X.shape[0]
        X_column = X[:, self.feature_idx]
        predictions = np.ones(n_samples)
        if self.polarity == 1:
            predictions[X_column < self.threshold] = -1
        else:
            predictions[X_column > self.threshold] = -1

        return predictions


class Adaboost:
    def __init__(self, n_clf=5):
        self.n_clf = n_clf
        self.clfs = []

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

        # Initialize weights to 1/N
        w = np.full(n_samples, (1 / n_samples))

        self.clfs = []

        # Iterate through classifiers
        for _ in range(self.n_clf):
            clf = DecisionStump()
            min_error = float("inf")

            # greedy search to find best threshold and feature
            for feature_i in range(n_features):
                X_column = X[:, feature_i]
                thresholds = np.unique(X_column)

                for threshold in thresholds:
                    # predict with polarity 1
```

```
                    p = 1
                    predictions = np.ones(n_samples)
                    predictions[X_column < threshold] = -1

                    # Error = sum of weights of misclassified samples
                    misclassified = w[y != predictions]
                    error = sum(misclassified)

                    if error > 0.5:
                        error = 1 - error
                        p = -1

                    # store the best configuration
                    if error < min_error:
                        clf.polarity = p
                        clf.threshold = threshold
                        clf.feature_idx = feature_i
                        min_error = error

            # calculate alpha
            EPS = 1e-10
            clf.alpha = 0.5 * np.log((1.0 - min_error + EPS) /
(min_error + EPS))

            # calculate predictions and update weights
            predictions = clf.predict(X)

            w *= np.exp(-clf.alpha * y * predictions)
            # Normalize to one
            w /= np.sum(w)

            # Save classifier
            self.clfs.append(clf)

    def predict(self, X):
        clf_preds = [clf.alpha * clf.predict(X) for clf in self.clfs]
        y_pred = np.sum(clf_preds, axis=0)
        y_pred = np.sign(y_pred)

        return y_pred


# Testing
if __name__ == "__main__":

    # Adaboost classification with 5 weak classifiers
    clf = Adaboost(n_clf=5)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
```

```
    acc = accuracy(y_test, y_pred)
    print("Accuracy:", acc)
```

5. ML Model 4 (Based on research literature)

```
import pandas as pd

# Assuming X_train is your training data
columns_to_convert = ['hospital', 'age', 'sex', 'cp', 'trestbps',
'chol', 'fbs', 'restecg', 'thalach',
                'exang', 'oldpeak', 'slope', 'ca', 'thal']

X_train[columns_to_convert] = X_train[columns_to_convert].astype(str)

cat_features = ["hospital", "sex", "cp", "fbs", "restecg", "exang",
"slope", "ca", "thal"]


from catboost import CatBoostClassifier


model_cb = CatBoostClassifier(task_type='GPU', iterations=200,
                              random_state = 2021,
                              eval_metric="F1")


import pandas as pd
from catboost import CatBoostClassifier

# Assuming X_train is your training data
# Convert specified columns to string type
columns_to_convert = ['hospital', 'age', 'sex', 'cp', 'trestbps',
'chol', 'fbs', 'restecg', 'thalach',
                      'exang', 'oldpeak', 'slope', 'ca', 'thal']
X_train[columns_to_convert] = X_train[columns_to_convert].astype(str)
X_test[columns_to_convert] = X_test[columns_to_convert].astype(str)


# Specify categorical features
cat_features = [0,2,3,6,7,9,11,12,13]  # Update with the correct
indices of categorical features

# Initialize and fit the CatBoost model
model_cb = CatBoostClassifier()
model_cb.fit(X_train, y_train, cat_features=cat_features, plot=True,
eval_set=(X_test, y_test))
```

7. References

1. pandas library
2. catboost