**Machine learning project**

**Participants:**

**Omar abdelfattah 212325**

[**Omar.abdelfattah2@msa.edu.eg**](mailto:Omar.abdelfattah2@msa.edu.eg)

**Youssef sharabas 212091**

[**youssef.hazem2@msa.edu.eg**](mailto:youssef.hazem2@msa.edu.eg)

**mohammed magdy 213059**

[**Mohamed.Magdi4@msa.edu.eg**](mailto:Mohamed.Magdi4@msa.edu.eg)

**In this project we used multiple machine learning classifiers on a 250,000+ record database we displayed the accuracy, precision, recall and f1score for all six classifiers in a table to make it easy to compare between them. The six classifiers are Naïve Bays, SVM, MLP, Gradient Boosting, KNN and Logistic Regression**

**Dataset:**

**Our dataset was on the effect of multiple factors on the chance of having a heart problem or attack it is a binary classification problem each value in the dataset refers to a decided range for example (1 in age column means from 20 to 24 years old)**

**Naïve Bays:**

Code:  
#train

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, recall\_score, precision\_score, f1\_score

from sklearn import preprocessing

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

# Load the training data

df = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/Machine Project 2023/heart\_disease\_health\_indicators\_BRFSS2015.csv')

from google.colab import drive

drive.mount('/content/drive')

X1 = df.drop(['HeartDiseaseorAttack'] , axis = 1)

y1 = np.array(df['HeartDiseaseorAttack'])

X\_train1, X\_test1, y\_train1, y\_test1 = train\_test\_split (X1, y1, test\_size=0.25)

clf = GaussianNB()

clf.fit(X\_train1, y\_train1)

y\_pred1 = clf.predict(X\_test1)

from sklearn import preprocessing

label\_encoder = preprocessing.LabelEncoder()

y1 = label\_encoder.fit\_transform(y1)

y\_test1 = label\_encoder.fit\_transform(y\_test1)

y\_pred1 = label\_encoder.fit\_transform(y\_pred1)

accuracy1 = accuracy\_score(y\_test1, y\_pred1)

print('Accuracy:', accuracy1)

recall1 = recall\_score(y\_test1, y\_pred1)

print('Recall:', recall1)

precision1 = precision\_score(y\_test1, y\_pred1)

print('Precision:', precision1)

f11 = f1\_score(y\_test1,y\_pred1)

print('F1\_score:', f11)

Results:

Accuracy: 0.8183538315988647

Recall: 0.5501509560550151

Precision: 0.2706717280079221

F1\_score: 0.36283185840707965

Brief:

Naïve bayes is used in text classification, spam filtering, sentiment analysis, recommended systems, medical diagnosis.

Naïve bayes is supervised machine learning Algorithm.

Naïve bayes has discrete and continues features. The continues we need to estimate the mean and variance for each class then use it in the following equation:

while in the discrete we only use the following equation to calculate the probability of each class then do the same for the testing phase: P(H|X) =P(X|H)P(H)/P(X)

**SVM:**

Code:

#train

import pandas as pd

import numpy as np

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, recall\_score, precision\_score, f1\_score

from sklearn import preprocessing

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

df = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/Machine Project 2023/heart\_disease\_health\_indicators\_BRFSS2015.csv')

X2 = df.drop(['HeartDiseaseorAttack'] , axis = 1)

y2 = np.array(df['HeartDiseaseorAttack'])

X\_train2, X\_test2, y\_train2, y\_test2 = train\_test\_split(X2, y2, test\_size=0.25, random\_state=42)

print(len(X\_train2))

print(len(X\_test2))

svm = SVC(C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache\_size=200, class\_weight=None, verbose=False, max\_iter=-1, decision\_function\_shape='ovr', break\_ties=False, random\_state=None)

svm.fit(X\_train2, y\_train2)

y\_pred2 = svm.predict(X\_test2)

accuracy2 = accuracy\_score(y\_test2, y\_pred2)

print('Accuracy:', accuracy2)

recall2 = recall\_score(y\_test2, y\_pred2,pos\_label=1)

print('Recall:', recall2)

precision2 = precision\_score(y\_test2, y\_pred2,pos\_label=1)

print('Precision:', precision2)

f12 = f1\_score(y\_test2,y\_pred2,pos\_label=1)

print('F1\_score:', f12)

Results:

Accuracy: 0.906401766004415

Recall: 0.0

Precision: 0.0

F1\_score: 0.0

Brief:

SVM is a binary classification. Also, it is a Supervised machine learning Algorithm.

SVM separates data from each other so it categorize it into two classes then we get maximum margin classifier then in each class we need to get support vectors.

**KNN:**

Code:

import pandas as pd

import numpy as np

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, recall\_score, precision\_score, f1\_score

from sklearn import preprocessing

# Load the training data

df = pd.read\_csv('/content/heart\_disease\_health\_indicators\_BRFSS2015.csv')

# Create feature and target arrays

X = df.drop(['HeartDiseaseorAttack'], axis=1)

y = np.array(df['HeartDiseaseorAttack'])

# Preprocess the data to handle missing values

X = X.fillna(X.mean())  # Replace missing values with the mean

# Split into training and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize and train the KNN classifier

knn = KNeighborsClassifier(n\_neighbors=21)

knn.fit(X\_train, y\_train)

# Predict on the test set

pred\_y = knn.predict(X\_test)

# Encode the target values

label\_encoder = preprocessing.LabelEncoder()

y\_test\_encoded = label\_encoder.fit\_transform(y\_test)

pred\_y\_encoded = label\_encoder.transform(pred\_y)

# Calculate evaluation metrics

accuracy = accuracy\_score(y\_test\_encoded, pred\_y\_encoded)

recall = recall\_score(y\_test\_encoded, pred\_y\_encoded)

precision = precision\_score(y\_test\_encoded, pred\_y\_encoded)

f1 = f1\_score(y\_test\_encoded, pred\_y\_encoded)

# Print the evaluation metrics

print('Accuracy:', accuracy)

print('Recall:', recall)

print('Precision:', precision)

print('F1\_score:', f1)

Results:

Accuracy: 0.9060036266162094

Recall: 0.02915268456375839

Precision: 0.4982078853046595

F1\_score: 0.055082227065583515

Brief:

* K-Nearest Neighbors (KNN) is a supervised machine learning algorithm
* Used for both classification and regression tasks.
* It calculates the distance between a new data point and existing points, selects the k closest neighbors, and predicts the class or value based on the neighbors. KNN is popular in recommendation systems, pattern recognition, and data mining.

**Logistic regression:**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, recall\_score, precision\_score, f1\_score

from sklearn.impute import SimpleImputer

from sklearn import preprocessing

# Load the data

data = pd.read\_csv('/content/heart\_disease\_health\_indicators\_BRFSS2015.csv')

# Create feature and target arrays

X = data.drop(['HeartDiseaseorAttack'], axis=1)

y = np.array(data['HeartDiseaseorAttack'])

# Handle missing values using mean imputation

imputer = SimpleImputer(strategy='mean')

X\_imputed = imputer.fit\_transform(X)

# Split into training and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_imputed, y, test\_size=0.2, random\_state=42)

# Initialize and train the logistic regression classifier

classifier = LogisticRegression(solver='lbfgs', random\_state=0)

classifier.fit(X\_train, y\_train)

# Predict on the test set

predicted\_y = classifier.predict(X\_test)

# Encode the target values

label\_encoder = preprocessing.LabelEncoder()

y\_test\_encoded = label\_encoder.fit\_transform(y\_test)

pred\_y\_encoded = label\_encoder.transform(predicted\_y)

# Calculate evaluation metrics

accuracy = accuracy\_score(y\_test\_encoded, pred\_y\_encoded)

recall = recall\_score(y\_test\_encoded, pred\_y\_encoded)

precision = precision\_score(y\_test\_encoded, pred\_y\_encoded)

f1 = f1\_score(y\_test\_encoded, pred\_y\_encoded)

# Print the evaluation metrics

print('Accuracy:', accuracy)

print('Recall:', recall)

print('Precision:', precision)

print('F1\_score:', f1)

Result:

Accuracy: 0.9053476443768997

Recall: 0.12538540596094552

Precision: 0.45607476635514016

F1\_score: 0.1966948810963321

Brief:

Logistic regression is a statistical model used for binary classification tasks, where the goal is to predict the probability of an outcome belonging to a specific class.

It has several applications including medical diagnoses, credit scoring, and market analysis. Fraud detection and natural language processing

**MLP (**multi-layer perceptron**)**

#Train

import pandas as pd

import numpy as np

from sklearn.metrics import accuracy\_score, recall\_score, precision\_score, f1\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.datasets import load\_iris

# Load sample data

df = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/Machine Project 2023/heart\_disease\_health\_indicators\_BRFSS2015.csv')

df

x4=df.drop(['HeartDiseaseorAttack'],axis=1)

x4

y4=np.array(df['HeartDiseaseorAttack'])

x\_train4,x\_test4,y\_train4,y\_test4=train\_test\_split(x4,y4,test\_size=0.25)

# Create an MLP classifier

mlp = MLPClassifier(hidden\_layer\_sizes=(100, ),        # Number of hidden layers and units per layer

                    activation='relu',                 # Activation function ('identity', 'logistic', 'tanh', 'relu')

                    solver='adam',                     # Solver for weight optimization ('lbfgs', 'sgd', 'adam')

                    alpha=0.0001,                      # L2 penalty (regularization term) parameter

                    batch\_size='auto',                 # Size of minibatches for stochastic optimizers

                    learning\_rate='adaptive',          # Learning rate schedule ('constant', 'invscaling', 'adaptive')

                    learning\_rate\_init=0.005,          # The initial learning rate

                    power\_t=0.5,                       # The exponent for inverse scaling learning rate

                    max\_iter=200,                      # Maximum number of iterations

                    shuffle=True,                      # Whether to shuffle samples in each iteration

                    random\_state=None,                 # Seed for the random number generator

                    tol=0.0001,                        # Tolerance for the optimization

                    verbose=False,                     # Whether to print progress messages

                    warm\_start=False,                  # Reuse the previous solution

                    momentum=0.9,                      # Momentum for gradient descent update

                    nesterovs\_momentum=True,           # Whether to use Nesterov's momentum

                    early\_stopping=False,              # Terminate training when validation score is not improving

                    validation\_fraction=0.1,           # Proportion of training data to set aside as validation set

                    beta\_1=0.9,                        # Exponential decay rate for estimates of first moment vector in adam

                    beta\_2=0.999,                      # Exponential decay rate for estimates of second moment vector in adam

                    epsilon=1e-8,                      # Value for numerical stability in adam

                    n\_iter\_no\_change=10,               # Maximum number of epochs without any improvement in the loss

                    max\_fun=15000)                     # Maximum number of function calls for the solver

# Train the MLP classifier

mlp.fit(x\_train4, y\_train4)

# Make predictions

y\_pred4 = mlp.predict(x\_test4)

from sklearn import preprocessing

label\_encoder = preprocessing.LabelEncoder()

y4 = label\_encoder.fit\_transform(y4)

test\_y4 = label\_encoder.fit\_transform(y\_test4)

pred\_y4 = label\_encoder.fit\_transform(y\_pred4)

accuracy4 = accuracy\_score(test\_y4, pred\_y4)

print('Accuracy:', accuracy4)

recall4 = recall\_score(test\_y4, pred\_y4)

print('Recall:', recall4)

precision4= precision\_score(test\_y4, pred\_y4)

print('Precision:', precision4)

f14 = f1\_score(test\_y4,pred\_y4)

print('F1\_score:', f14)

**results:**

Accuracy: 0.9070167139703563

Recall: 0.09656545545047288

Precision: 0.562862669245648

F1\_score: 0.1648491715054525

brief:

MLP is a deep learning algorithm

MLP is a neural network connecting multiple layers in a directed graph, which means that the signal path through the nodes only goes one way. Each node, apart from the input nodes, has a nonlinear activation function. An MLP uses backpropagation as a supervised learning technique. Since there are multiple layers of neurons, MLP is a deep learning technique.

**Gradient boosting**

#Train

# Import models and utility functions

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, recall\_score, precision\_score, f1\_score

from sklearn.datasets import load\_digits

from sklearn.metrics import classification\_report, confusion\_matrix

import pandas as pd

import numpy as np

# Setting SEED for reproducibility

SEED = 23

df = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/Machine Project 2023/heart\_disease\_health\_indicators\_BRFSS2015.csv')

df

x3=df.drop(['HeartDiseaseorAttack'],axis=1)

x3

y3=np.array(df['HeartDiseaseorAttack'])

# Importing the dataset

# Splitting dataset

train\_x3, test\_x3, train\_y3, test\_y3 = train\_test\_split(x3, y3,

                                                    test\_size = 0.25,

                                                    random\_state = SEED)

# Instantiate Gradient Boosting Regressor

gbc = GradientBoostingClassifier(n\_estimators=300,

                                 learning\_rate=0.05,

                                 random\_state=100,

                                 max\_features=5 )

# Fit to training set

gbc.fit(train\_x3, train\_y3)

# Predict on test set

pred\_y3= gbc.predict(test\_x3)

from sklearn import preprocessing

label\_encoder = preprocessing.LabelEncoder()

y3 = label\_encoder.fit\_transform(y3)

test\_y3 = label\_encoder.fit\_transform(test\_y3)

pred\_y3 = label\_encoder.fit\_transform(pred\_y3)

accuracy3 = accuracy\_score(test\_y3, pred\_y3)

print('Accuracy:', accuracy3)

recall3 = recall\_score(test\_y3, pred\_y3)

print('Recall:', recall3)

precision3 = precision\_score(test\_y3, pred\_y3)

print('Precision:', precision3)

f13 = f1\_score(test\_y3,pred\_y3)

print('F1\_score:', f13)

Accuracy: 0.9077262693156732

Recall: 0.11639591358231452

Precision: 0.5468135326514555

F1\_score: 0.191935929301298

Brief:

gradient Boosting is the grouping of **Gradient descent and Boosting**. In gradient boosting, each new model minimizes the loss function from its predecessor using the Gradient Descent Method. This procedure continues until a more optimal estimate of the target variable has been achieved

Unlike other ensemble techniques, the idea in gradient boosting is that they build a series of trees where every other tree tries to correct the mistakes of its predecessor tree.

Results:

╒═════════════════════╤════════════╤═══════════╤═════════════╤════════════╕

│ Classsifier name │ Accuracy │ Recall │ Precision │ f1-score │

╞═════════════════════╪════════════╪═══════════╪═════════════╪════════════╡

│ Naive bayes │ 0.818354 │ 0.550151 │ 0.270672 │ 0.362832 │

├─────────────────────┼────────────┼───────────┼─────────────┼────────────┤

│ SVM │ 0.906402 │ 0 │ 0 │ 0 │

├─────────────────────┼────────────┼───────────┼─────────────┼────────────┤

│ Gradient Boosting │ 0.907726 │ 0.116396 │ 0.546814 │ 0.191936 │

├─────────────────────┼────────────┼───────────┼─────────────┼────────────┤

│ MLP │ 0.907017 │ 0.0965655 │ 0.562863 │ 0.164849 │

├─────────────────────┼────────────┼───────────┼─────────────┼────────────┤

│ Logistic regression │ 0.907222 │ 0.119946 │ 0.51895 │ 0.194855 │

├─────────────────────┼────────────┼───────────┼─────────────┼────────────┤

│ KNN │ 0.906276 │ 0.0267857 │ 0.48773 │ 0.0507825 │

╘═════════════════════╧════════════╧═══════════╧═════════════╧════════════╛

Accuracy was our main focus when trying to get best results from each classifier

the gradient boosting classifier got the highest accuracy