SCHOOL OF COMPUTER SCIENCES

UNIVERSITI SAINS MALAYSIA

CPC152: Foundations and Programming for Data Analytics

Project (20%)

(Group Project)

25 June 2023

Prepared by: Group 9

Name	Matric Number	Student Email
Nurul Afiqah binti Azhar	160335	afiqahazhar@student.usm.my
Danish Raimi Bin Fazrul Edlin	164108	danishraimi77@student.usm.my
Tan Yin Xuan	164467	tanyinxuan4@student.usm.my
Muhammad Lutfi bin Yahya	160382	ymuhammadlutfi@student.usm.my
Li Zi Liang	160757	LIZILIANG@student.usm.my
Ainul Mardhiah Binti Abdul Mutalip	161836	ainulmrdh@student.usm.my

Lecturer:

Ts. Dr. Chew XinYing

Table Of Content:

1. Abstract

Machine learning algorithms are computational models created to automatically learn the patterns or attributes and make forecasting or decisions according to the datasets. Via the machine learning algorithms, the machines, which are computers, can learn from examples, identify patterns, and make data-driven predictions or classifications. Various types of machine learning algorithms facilitate the data predictions and help solve real-world convoluted problems.

Firstly, our report will introduce the brief background of the 3 selected machine learning algorithms, which are k-nearest neighbours algorithm (KNN), decision tree and support vector machine (SVM). Next, we state our objectives of the experiment clearly. Then, we justify our choice that had been made in the selection of 3 machine learning algorithms from 7 common machine learning algorithms. We also show the whole steps to build both 3 machine learning algorithms, followed by the codes. Moreover, we make comparisons between the 3 machine learning algorithms according to the results of the experiments and 1 machine learning algorithm is recommended as our final champion model. Furthermore, we discuss and look into the results of the experiments and the reasons we choose that machine learning algorithm as our champion model. Lastly, the lessons learned from this project are stated and conclusions of the selection of champion model and overall project are made.

2. Introduction

a. K-Nearest Neighbours (KNN)

The abbreviations of k-nearest neighbours algorithm are KNN or k-NN. It is a simple machine learning algorithm that has the characteristics of a non-parametric and supervised learning classifier. Thus, it does not comprise any model training or parameter estimation. KNN employs vicinity to classify and predict the clustering of a new data point. It can be implemented for classification and regression issues as the training data serves as the reference points for classification or regression. However, it is normally used to resolve the classification issues, by the hypothesis that analogous points can be obtained near each other.

KNN functions to store the entire accessible cases in memory and classify the new data point or case according to a resemblance measure. It is commonly implemented to categorise a data point according to the way its neighbours are categorised. In KNN, K is a hyperparameter that signifies the number of nearest neighbours to be included in the preponderance of the voting process. A smaller K value makes the algorithm more sensitive to local variations, while a larger K value smooths out the decision boundaries. Parameter tuning is the process of selecting the right value of K and it is significant in accomplishing higher accuracy. KNN is also popular as it is known as a lazy learner since it does not learn a discriminative function from the training datasets. It "memorises" the training dataset instead of learning the data.

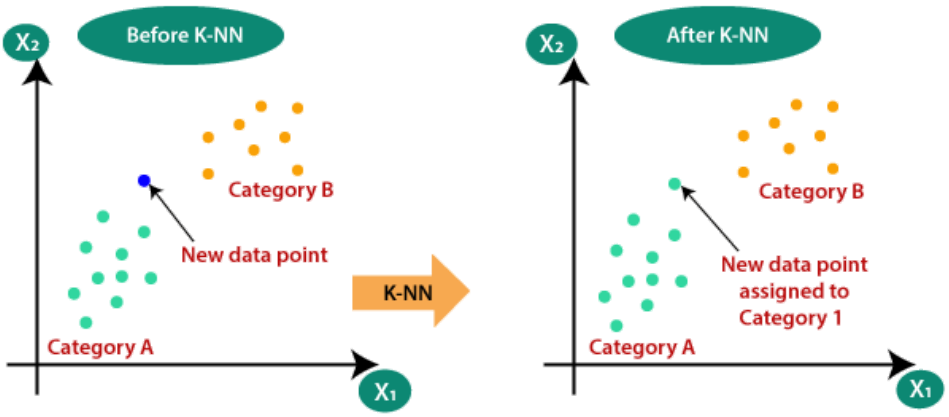

Figure 2.1

For the classification, when a new unlabeled data point is presented, the KNN algorithm calculates the metric distance (i.e., Euclidean distance, Manhattan distance, Minkowski distance) between the new data point and the entire labelled data points in the training datasets. Then, it chooses the 'k' nearest neighbours (i.e., the data points with the smallest distances) and assigns the class label that is most predominant among those. For the regression, the KNN algorithm computes the distances between the new data point and the labelled data points. It then decides the 'k' nearest neighbours

and accounts the average or weighted average of their corresponding target values as the forecasted value for the new data point.

b. Decision Tree

Another supervised learning algorithm that is well-known and likely to be used is the Decision Tree algorithm. It is useful for solving regression and classification problems too. The purpose of applying Decision Tree is to produce a training model that can be utilised to forecast the value or class of the target variable by learning simple decision rules inferred from training data. The forecasting of a class label for a record begins from the tree root. The values of the root attribute are evaluated with the record's attribute. Via the evaluation and comparison, we follow the branch of that corresponding value and hop to the successive node.

One of the famous Decision Tree types is classification and regression trees (CART). It was introduced by Leo Breiman. This algorithm utilises Gini impurity for the decision in the splitting of ideal attributes. Gini impurity is used to measure how frequently a randomly chosen attribute is misclassified. A lower value of Gini impurity is more ideal for the splitting. At each internal node of the tree, the algorithm selects a splitting criterion to divide the data according to a specific feature and its corresponding threshold value. The often used splitting criteria include Gini impurity and information gain (using measures like entropy or the Gini index). These criteria compute the impurity of the target variable within each partition.

The Decision Tree algorithm uses a top-down, greedy approach to construct the tree. It selects the best splitting criterion and partitions the data starting from the root node. This process is recursively repeated for each child node until a halting condition is met. Stopping conditions can include achieving a maximum depth, a minimum impurity threshold, or a minimum number of samples per leaf. The resultant tree structure can be visualised and easily understood. This has contributed to a great advantage in human-friendly explanations of decision-making. The Decision Tree algorithm is broadly used in various domains due to its efficient interpretability and ability to capture complex decision boundaries.

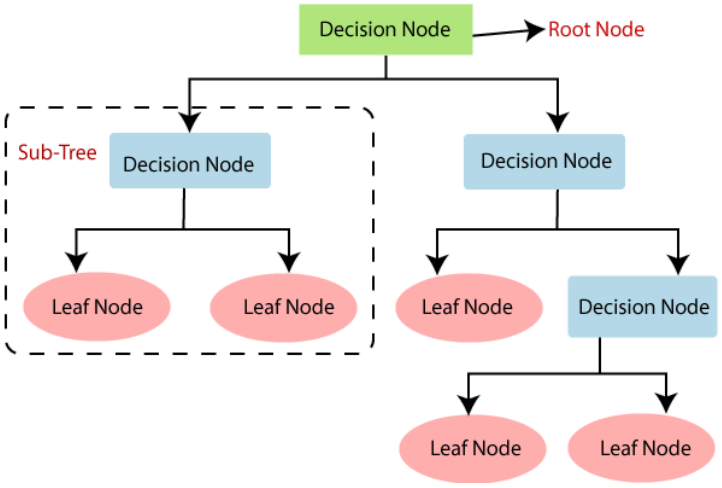

Figure 2.2

c. Support Vector Machine (SVM)

Support Vector Machine or SVM is also included in the most familiar and well-known supervised learning algorithms. It is used for classification as well as regression problems. Yet commonly, it is implemented for resolving classification issues in machine learning. In the 1990s, this algorithm was developed by Vladimir Vapnik with his coworkers at AT&T Bell Laboratories. The ultimate goal of the SVM algorithm is to find the most suitable decision boundary or line that can isolate one or more dimensional space into several different classes. Thus, we can simply place the new data point in the right category in data prediction.

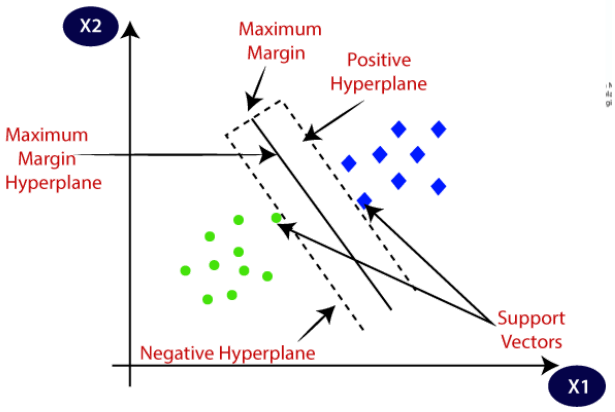

Figure 2.3

From *Figure 2.3,* SVM is based on the thought of finding a hyperplane that best divides a dataset into 2 classes. The best decision boundary is called a hyperplane. Margins are the perpendicular distances between the line and those dots closest to the line. A low margin will tend to increase the probability of misclassification, so it is encouraged to maximise the margin. SVM chooses the extreme vectors or points that aid in locating the hyperplane. Since these extreme cases are acknowledged as support vectors, the algorithm

is known as Support Vector Machine. Plus, it is not affected by the presence of the outliers.

Furthermore, SVM can efficiently handle non-linear decision boundaries by applying the concepts of the Soft Margin and Kernel Tricks. Firstly, the finding of decision boundaries in soft margin allows the toleration of one or few misclassified dots. The degree of the tolerance is represented as the parameter (C), which can be regulated to control the balance between maximising the margin and minimising the classification error. A lower C value allows for a wider margin but may lead to misclassified points, while a higher C value will lead to an accurate classification at the expense of a narrower margin. The kernel function converts the input data into a higher-dimensional feature space in order to find a non-linear decision boundary. This permits SVM to explore and capture complex relationships between the data points. Several kernels or transformations like linear, polynomial, radial basis function (RBF), Gaussian, sigmoid, and precomputed, are provided to create the non-linear decision boundary.

3. Objective of the experiment

For this project, we want to develop machine learning using three different algorithms which are Support Vector Machine, Decision Trees, and K-Nearest Neighbour to accurately predict the value of target which means to predict whether someone has the risk of having heart disease or not. The predictions are being made based on several features.

The features are:

age	the age of the patient
sex	(the sex of the patient (1 = male, 0 = female)
cp	chest pain type (typical angina=0, atypical angina=1, non-anginal pain=2, or asymptomatic=3)
trestbps	resting blood pressure in mmHg
exang	exercise-induced angina (1 = yes, 0 = no)
oldpeak	ST depression induced by exercise relative to rest

chol	serum cholesterol in mg/dl
fbs	fasting blood sugar > 120 mg/dl (1 = true, 0 = false)
restecg	resting electrocardiographic results (normal, ST-T wave abnormality, or left ventricular hypertrophy)
thalach	maximum heart rate achieved
slope	the slope of the peak exercise ST segment (upsloping (0), flat (1), or downsloping (2))
ca	number of major vessels coloured by fluoroscopy (0-3
thal	thalassemia (3 = normal, 6 = fixed defect, 7 = reversible defect)
target	the presence of heart disease (1 = heart disease present, 0 = no heart disease)

Table 3.1

The goal is to achieve the following in particular:

- Clean up, preprocess, and handle missing numbers,and consistency issues.
- Choose the dataset's most pertinent features that significantly aid in forecasting whether a person will develop heart disease.
- Implement and train the K-Nearest Neighbours (KNN), Support Vector Machines (SVM), and Decision Trees machine learning models.
- Evaluate each model's performance based on the accuracy, precision, recall, F1-score. To choose the most reliable and accurate model(champion model), compare the outcomes produced by the three.
- Based on the evaluation metrics, decide which model will be the champion model or to forecast heart disease.

By achieving these goals, we want to create precise machine learning models that, based on the provided dataset, can accurately predict the presence or absence of cardiac disease. This could potentially save lives and improve healthcare outcomes by assisting in the early detection and prevention of heart disease.

4. Justification of choice

There are some reasons why we choose to use Decision Trees, Support Vector Machine, Decision and K-Nearest Neighbour as the Machine learning model for our experiments on the heartdisease.csv dataset.

Decision Trees:

- **Interpretability**

  Decision Tree is simple to interpret and understand in order to create predictions. Decision trees employ a set of if-then-else conditions. Every node in the decision tree indicates a choice based on a certain trait, and every branch denotes the potential results of that choice. The decision criteria that come from this hierarchical structure are simple to understand. In a heart disease prediction model, as an example, a decision rule might read: "If age > 50 and cholesterol > 200, then predict the presence of heart disease."

- **Handling Mixed Data Types**

  The data that we use in our experiment which is heartdisease.csv contain a combination of medical features represented as numerical values(example:age, cholesterol level) and categorical values(example:sex,chest pain type). Decision trees can make prediction using this mixed data type without requiring explicit encoding or preprocessing.Hence, it is suitable to predict the presence of the heart disease for patients.

- **Feature Importance**

  The insights into the importance of different features in predicting heart disease can be viewed by us by using Decision Trees.We can determine which features play an important role in classification process by examining the splits and the structure of the tree.This information is very useful for us to prioritise specific medical tests or attributes when diagnosing or assessing the risk of heart disease.

- **Handling Nonlinear Relationships**

  Decision trees are capable of capturing nonlinear relationships between features and the target.In the case of our experiment to predict the presence of heart disease among patients,the relationships between certain medical attributes (e.g., cholesterol level, blood pressure, age) and the presence of heart disease are not linear. Decision trees can model such complex relationships by performing splits based on different thresholds. Hence, Decision Trees can capture nonlinear patterns in the data effectively.

Support Vector Machine:

- **Effective in High-Dimensional Spaces**

  For the high-dimensional datasets,where the number of features is large,SVM can perform very well.For our experiment to predict the presence of heart disease,there are multiple medical features such as age,blood pressure,cholesterol level and others.SVM is well suited for capturing complex correlations among these features and effectively categorising examples since it can handle high-dimensional spaces.Plus,the model is also ideal for working with data when the number of features exceeds the number of samples because its complexity is O(n-features * n2 samples).

- **Robust to Outliers**

  SVM is also robust to outliers in the dataset. In terms of heartdisease.csv that we use, there may be some outliers that can affect the result of prediction. SVM aims to find the decision boundary that maximises the margin between classes. Hence, it is very effective to ignore outliers that fall far from the decision boundary.

- **Offers several Kernel Selection**

  SVM offers numerous kernel functions that can be customised to the unique properties of the dataset. The radial basis function or polynomial kernel are suitable choices for kernels that allow SVM to handle a variety of data types and accurately represent certain relationships.

- **Ability to handle Imbalanced Data**

  Since SVM does not place a significant emphasis on the class distribution, it handles unbalanced data well .The algorithm seeks to find the best decision boundary regardless of class proportions, mitigating issues related to imbalanced datasets.

K-Nearest Neighbors:

- **Easy to understand and implement**

  The k-nearest neighbour is a machine learning model that is straightforward and easy to understand and implement.This is because it only operates using the principle of similarity which means it classifies an instance based on the class labels of its nearest neighbours.

- **No Training Phase**

  There is no explicit training phase for model construction or parameter estimation in k-NN. The method runs all the calculations during the prediction phase while the training data is kept in memory.

- **Model Flexibility**

  Since k-NN does not presuppose any particular data distribution or functional form, it can be applied to various dataset types. It can naturally handle both category and numerical characteristics, making it possible to easily incorporate the many medical attributes found in the "heartdisease.csv" dataset.

- **Single Hyperparameters**

  By using the k-NN machine learning model,it only involves a single hyperparameter, the value of K. This makes hyper parameter tuning easy for our experiment.

5. Steps to build the machine learning models

Data Extraction and Cleansing

```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# read heart disease data from csv file
dataset = pd.read_csv('heartDisease.csv')
print(" ")
print(dataset.head())
print(" ")
print(dataset.isnull().sum())
```

Figure 5.1.1

- Firstly, import the needed libraries that are necessary, which are 'Pandas' for data frame, 'NumPy', and 'matplotlib. pylot'.
- Read the dataset using the pandaframe function which is pd.read_csv. The file read is 'heartDisease.csv'.
- Print a few rows of the dataset using 'dataset.head()' to look a quick overview of the data

```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1

age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
...
ca          0
thal        0
target      0
dtype: int64
```

Figure 5.1.2

- Here Is the result of the code, as we can see there is no null data in the data frame.

Split the Data Frame .

Next we will split data frame into Two variable which are X for Features and y for Target or Output

```
#Split the data into features and target
#split dataset
X = dataset.iloc[:, 0:13]
y = dataset.iloc[:, 13]
```

Figure 5.1.3

- The line X = dataset.iloc[:, 0:13] selects all rows of the dataset and the columns from index 0 to 12 (columns 1 to 13). It assigns these columns to the variable X, which represents the features of the dataset.
- The line y = dataset.iloc[:, 13] selects all rows of the dataset and the column with index 13 (the 14th column). It assigns the values from this column to the variable y, which represents the target variable or the output variable of the dataset.

Features Selection

Next, we will done the Features Selection Operation to Choose The Top 6 Best out of them

Full Code:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.model_selection import train_test_split
import pandas as pd

bestfeatures = SelectKBest(score_func=chi2, k=13)
fit = bestfeatures.fit(X, y)

dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)

featureScores = pd.concat([dfcolumns, dfscores], axis=1)
featureScores.columns = ['Feature', 'Score']  # Rename the columns

top_features = featureScores.nlargest(13, 'Score')
print(top_features)

#use the top 6 features
X = dataset[['cp', 'thalach', 'exang', 'oldpeak', 'ca', 'thal']]
y = dataset.iloc[:, 13]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.2)
```

Figure 5.1.4

- The code uses SelectKBest with the chi2 scoring function to perform feature selection and select the top 13 features.
- The feature scores are stored in the fscores DataFrame and the feature names are stored in the df columns DataFrame.
- The featureScores DataFrame is created by concatenating dfcolumns and dfscores horizontally.
- The columns of featureScores are renamed as 'Feature' and 'Score'.
- The top 13 features with the highest scores are selected and stored in the top_features DataFrame.
- The top_features DataFrame is printed to display the selected features and their scores.
- The dataset is subsetted to include only the top 6 features: 'cp', 'thalach', 'exang', 'oldpeak', 'ca', and 'thal'.
- The target variable (y) is assigned to the 14th column of the dataset.
- The dataset is split into training and testing sets using the train_test_split() function, with 80% of the data used for training and 20% for testing. The random_state parameter is set to 42 for reproducibility.

Perform Standard Scaler

Full code:

```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Figure 5.1.5

- The code imports the StandardScaler class from the sklearn.preprocessing module, which is used for feature scaling.
- An instance of StandardScaler is created and assigned to the variable sc.
- The fit_transform() method of StandardScaler is applied to X_train, which performs two steps: fitting the scalar to the training data and transforming the training data using the calculated mean and standard deviation. This process standardised the features, making them have a mean of 0 and a standard deviation of 1.
- The transformed training data is assigned back to X_train, overwriting the original values.
- The transform() method of StandardScaler is applied to X_test, which performs only the transformation step. It uses the mean and standard deviation calculated from the training data to scale the testing data accordingly.
- The transformed testing data is assigned back to X_test, overwriting the original values.

Finally, we have already done our Data Preparation. Now we can start to use our machine learning model.

Decision Tree

1) **First we need to import the needed library for the Decision Tree and Library to determine the accuracy of the Machine Learning Model.**

```
#import libraries needed for Decision Tree
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier

# This library is for  determined the accuracy of the model
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import precision_score, recall_score
from sklearn import tree
```

Figure 5.2.1

- The code imports the necessary libraries for implementing a decision tree model for classification tasks: train_test_split for data splitting, StandardScaler for feature scaling, and DecisionTreeClassifier for building the decision tree model.
- It also imports evaluation metrics such as accuracy_score, confusion_matrix, f1_score, cross_val_score, precision_score, and recall_score for assessing the performance of the model

2) **Hyperparameter Tuning**

(A) Using GridSearchCV function

```
#find the best parameters for the model
from sklearn.model_selection import GridSearchCV

parameters = [{'criterion': ['entropy'], 'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11], 'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]}]

grid_search = GridSearchCV(estimator=DecisionTreeClassifier(random_state=0), param_grid=parameters, scoring='accuracy', cv=10)
grid_search.fit(X_train, y_train)

best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_

print("Best Accuracy: {:.2f}%".format(best_accuracy * 100))
print("Best Parameters:", best_parameters)
```

Figure 5.2.2

- The code utilises grid search, implemented through GridSearchCV from sklearn.model_selection, to find the best hyperparameters for a decision tree model.
- It specifies a dictionary of hyperparameter values to search over, including criterion (set to 'entropy'), max_depth (ranging from 1 to 11), and min_samples_leaf (ranging from 1 to 11).
- The grid search is performed by fitting the GridSearchCV object to the training data and subsequently obtaining the best accuracy score and corresponding best parameters through the best_score_ and best_params_ attributes, respectively.

Output:

```
Best Accuracy: 80.60%
Best Parameters: {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 7}
```

Figure 5.2.3

As we can see this is the output for the above code.

(B) Plot Accuracy Vs Max_depth

```
#test all the parameter values to find the higher accurancy using for loops and plot in graph
accuracy = []
for i in range(1, 11):
    dtree = DecisionTreeClassifier(criterion = 'entropy', max_depth = i, random_state = 0)
    dtree.fit(X_train, y_train)
    y_pred = dtree.predict(X_test)
    accuracy.append(accuracy_score(y_test, y_pred))
    print("Accuracy for max depth {}: {}".format(i, accuracy_score(y_test, y_pred)))
plt.figure(figsize = (10, 6))
plt.plot(range(1, 11), accuracy, color = 'red', linestyle = 'dashed', marker = 'o', markerfacecolor = 'blue', markersize = 10)
plt.title('Accuracy vs. Max Depth')
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
```

Figure 5.2.4

- The code evaluates the accuracy of a decision tree classifier with different values of the max_depth parameter.
- It uses a for loop to iterate over the values 1 to 10 for max_depth and trains a decision tree classifier for each value.
- The accuracy of each classifier is calculated using the accuracy_score function and stored in a list called accuracy.
- Finally, a line plot is created to visualise the relationship between max_depth and accuracy using matplotlib.

Output :

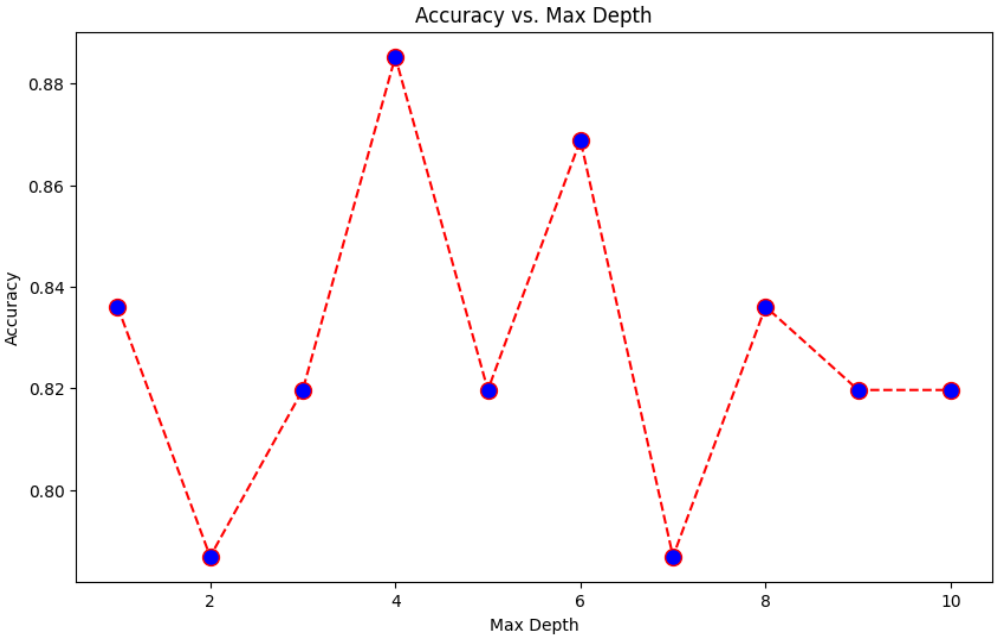

Figure 5.2.5

```
Accuracy for max depth 1: 0.8360655737704918
Accuracy for max depth 2: 0.7868852459016393
Accuracy for max depth 3: 0.819672131147541
Accuracy for max depth 4: 0.8852459016393442
Accuracy for max depth 5: 0.819672131147541
Accuracy for max depth 6: 0.8688524590163934
Accuracy for max depth 7: 0.7868852459016393
Accuracy for max depth 8: 0.8360655737704918
Accuracy for max depth 9: 0.819672131147541
Accuracy for max depth 10: 0.819672131147541
```

Figure 5.2.6

As we can see the highest accuracy that we get is 0.8852 with max_depth of 4.(

(C) Plot Accuracy VS Min sample leaf

Code:

```
#find the best parameters min_samples_leaf usning for loops and plot in graph
accuracy = []
for i in range(1, 11):
    dtree = DecisionTreeClassifier(criterion = 'entropy', min_samples_leaf = i, random_state = 0)
    dtree.fit(X_train, y_train)
    y_pred = dtree.predict(X_test)
    accuracy.append(accuracy_score(y_test, y_pred))
plt.figure(figsize = (10, 6))
plt.plot(range(1, 11), accuracy, color = 'red', linestyle = 'dashed', marker = 'o', markerfacecolor = 'blue', markersize = 10)
plt.title('Accuracy vs. Min Samples Leaf')
plt.xlabel('Min Samples Leaf')
plt.ylabel('Accuracy')
```

Figure 5.2.7

- The code calculates the accuracy of a decision tree classifier with different values of the min_samples_leaf parameter.
- It uses a for loop method to iterate over the values 1 to 10 for min_samples_leaf and trains a decision tree classifier for each value.
- The accuracy of each classifier is calculated using the accuracy_score function and stored in a list called accuracy.
- Finally, a line plot is created to visualise the relationship between min_samples_leaf and accuracy, helping to identify the optimal value for min_samples_leaf.

Output:

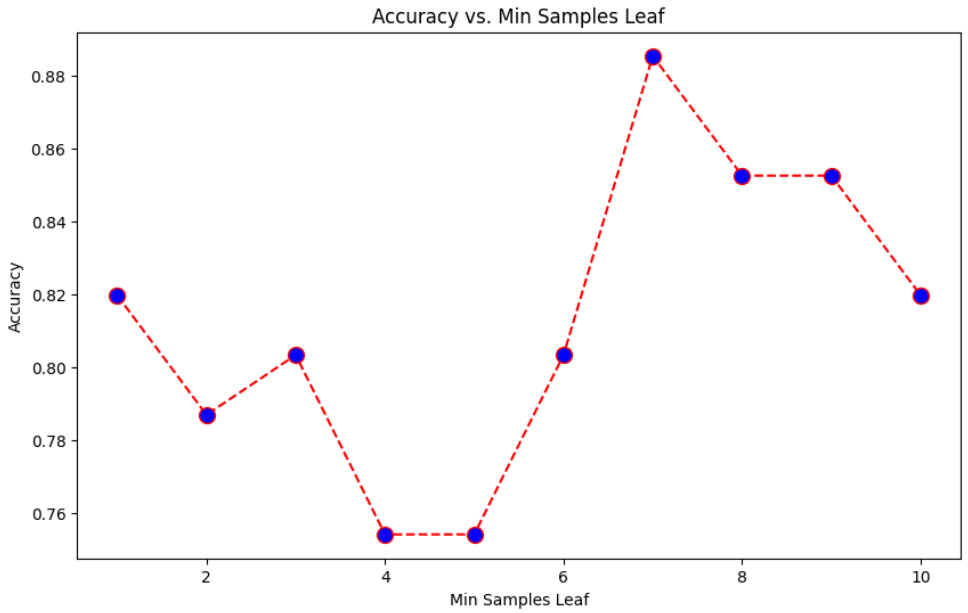

Figure 5.2.8

As we can see from the plot, the highest Accuracy is 0.88 with min Samples Lead of 7.

3) Fit the model with the Best parameter

```
#function to perform training with entropy
clf_entropy = DecisionTreeClassifier(criterion = 'entropy',min_samples_leaf = 7, random_state = 0)
clf_entropy.fit(X_train, y_train)
```

✓ 0.0s

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', min_samples_leaf=7, random_state=0)
```

Figure 5.2.9

Predict Test Set

```
#Predit Test Set
y_pred = clf_entropy.predict(X_test)
print(y_pred)
```

✓ 0.0s

```
[0 0 1 0 1 1 1 0 0 1 1 0 1 0 1 1 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 0 0 0 0
 1 1 1 1 1 1 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0]
```

Figure 5.3

Evaluate The Model:

```
#Evaluate Model
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred))
print("Accuracy: ", accuracy_score(y_test, y_pred)*100)
print("F1 Score: ", f1_score(y_test, y_pred, average='macro')*100)
print ('Recall:', recall_score(y_test, y_pred, average="weighted")*100)
print ('Precision:', precision_score(y_test, y_pred, average="weighted")*100)

print("")

#accuracy for training set
print("Training set score: {:.3f}".format(clf_entropy.score(X_train, y_train)*100))

#accuracy for test set
print("Test set score: {:.3f}".format(clf_entropy.score(X_test, y_test)*100))

#there is no overfitting or underfitting because the accuracy for training set and test set are almost the same
```

Figure 5.3.1

- This code is for evaluates the Machine Learning model of KNN by calculating various metrics and scores
- It print the confusion matrix , which provides information about the models prediction for different classes
- The accuracy , F1 score , recall and precision scores are calculated and printed to assess the model's overall performances
- The accuracy of the model on the training and test sets is calculated and printed to evaluate its fit and generalisation capabilities

Output:

```
Confusion Matrix:
[[28  1]
 [ 6 26]]
Accuracy:  88.52459016393442
F1 Score:  88.51224105461394
Recall: 88.52459016393442
Precision: 89.66748812457588

Training set score: 87.603
Test set score: 88.525
```

Figure 5.3.2

K-Nearest Neighbour Model.

(1) First we import needed Library for K-NN Machine Learning Model

```
#import library for KNN
from sklearn.neighbors import KNeighborsClassifier
```

Figure 5.3.3

In this case , KneighborClassifier has been imported for this Machine Learning

(2) HyperParameter Tuning

(A) Find Error Rate K Value

As we know K value is very important in order to classify this data . For this Machine Learning we will use For loop to find the best Value Of K and Visualise it using Matplotlib.

```
#finding the k value
error = []
k_range = range(1,25)
# Trying to find the error for K values between 1 and 30
for i in range(1, 30):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))
plt.figure(figsize=(12, 6))
plt.plot(range(1, 30), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
print("Minimum error:-",min(error),"at K =",error.index(min(error))+1)
```

Figure 5.3.4

- The code flows through a range of values for K from 1 to 29.
- For each K value, it trains a KNeighborsClassifier on the training data and makes predictions on the test data to find the error .
- It calculates the error rate by comparing the predicted labels with the true labels and stores it in a list called error.
- Afterwards, it plots the error rates for different K values and prints the minimum error rate along with the corresponding value of K where the error is minimised.

Output:

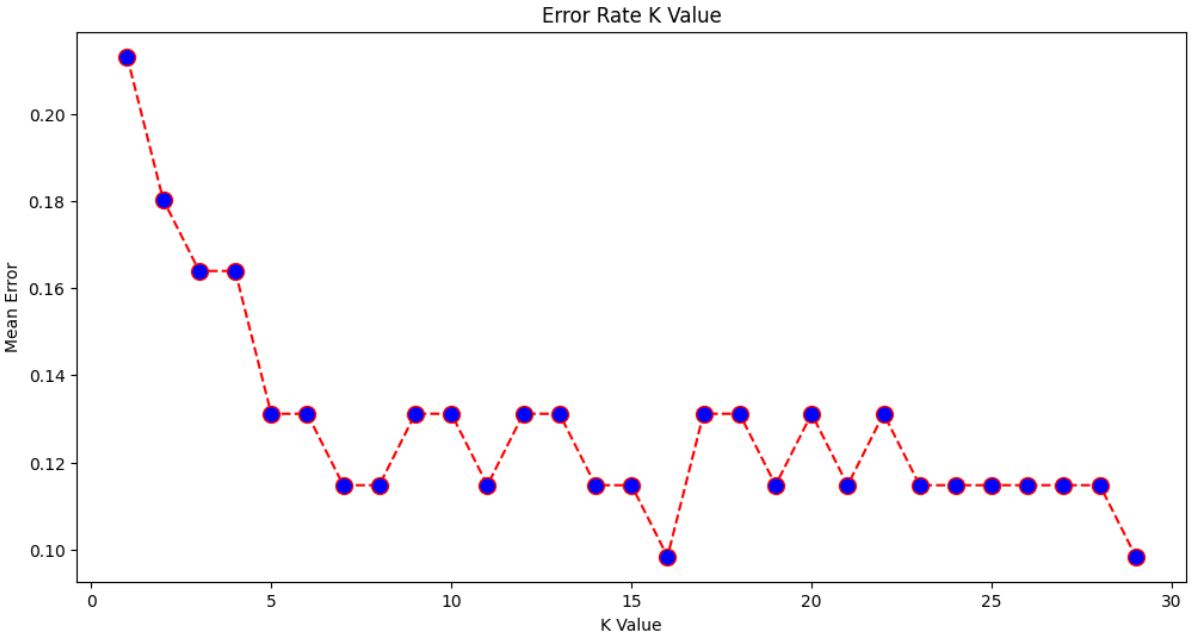

Figure 5.3.5

(B) Plot Accuracy vs K Value

Code:

```
from sklearn import metrics
#Try running from k=1 through 30 and record testing accuracy
k_range = range(1,30)
scores = {}
scores_list = []
for k in k_range:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train,y_train)
        y_pred=knn.predict(X_test)
        scores[k] = metrics.accuracy_score(y_test,y_pred)
        scores_list.append(metrics.accuracy_score(y_test,y_pred))
#plot the relationship between K and the testing accuracy
plt.plot(k_range,scores_list)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
```

Figure 5.3.6

- The code uses the k-nearest neighbours algorithm, a classification method, to predict labels for a given dataset and calculate it .
- It tests different values of the parameter k value from 1 to 30 to find the best value that yields the highest accuracy to be chosen as a k value .

- For each value of "k", the code trains a k-nearest neighbours classifier on the training data and predicts labels for the test data.
- The accuracy of the predictions is calculated and stored for each "k" value, and a plot is created to show how the accuracy changes as "k" varies.

Output:

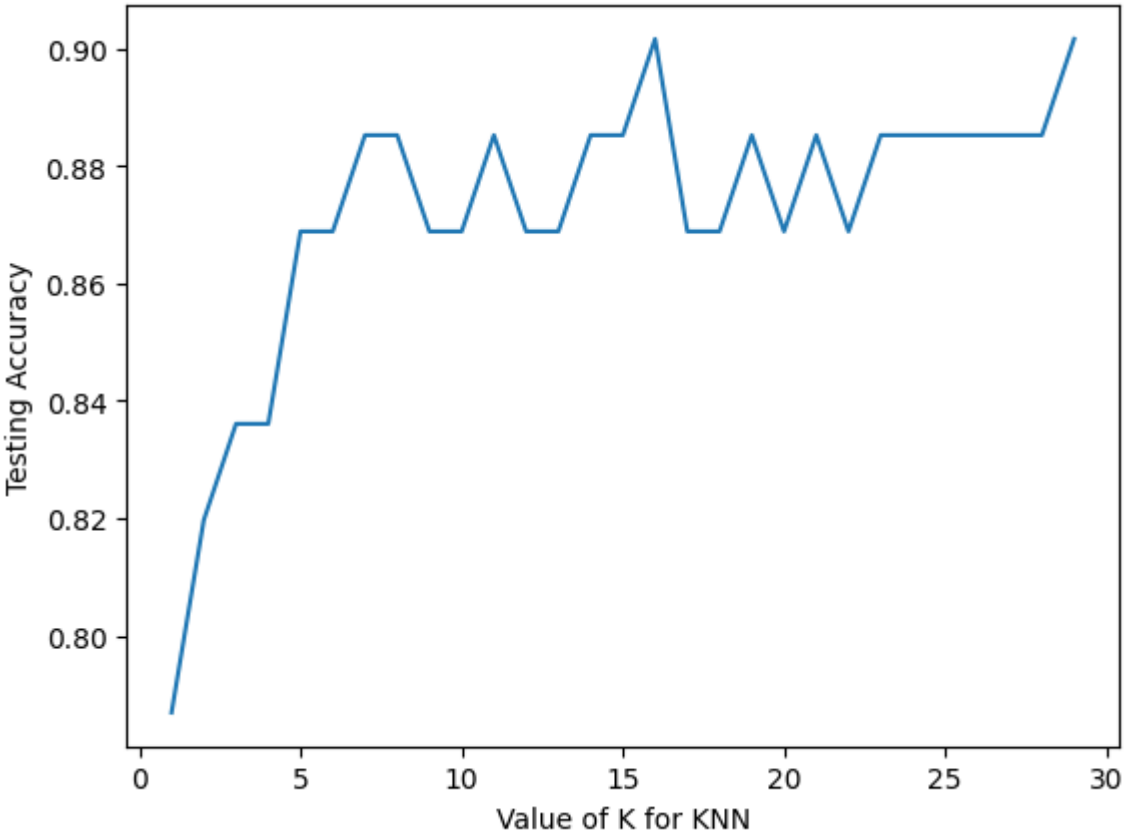

Figure 5.3.7

(3) Fit The Data into Machine Learning model

```
#Fit into MAchince Learning Model
knn=KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train,y_train)
```

✓ 0.0s

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

Figure 5.3.8

After finding the best value of K , we will assign 7 as K value into Machine Learning.

(4) Evaluate The Machine Learning Model :

Code:

```
#Evaluate Model
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred))
print("Accuracy: ", accuracy_score(y_test, y_pred)*100)
print("F1 Score: ", f1_score(y_test, y_pred, average='macro')*100)
print ('Recall:', recall_score(y_test, y_pred, average="weighted")*100)
print ('Precision:', precision_score(y_test, y_pred, average="weighted")*100)
```

Figure 5.3.9

- Evaluate all the Machine Learning Model and print all the various metrics and scores
- Print Confusion Matrix, Accuracy , F1 Score , Recall, and Precision

Output:

```
Confusion Matrix:
[[25  4]
 [ 2 30]]
Accuracy:  90.1639344262295
F1 Score:  90.09740259740259
Recall: 90.1639344262295
Precision: 90.30679667130968
```

Figure 5.4

<u>Support Vector Machine</u>

(1) Import Important Library for Support Vector Machine (SVM)

```
# Import library for Support Vector Machine (SVM) algorithm
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
```

✓ 0.0s

Figure 5.4.1

(2) Crate a Linear SVM classifier

```
#·Create·a·linear·SVM·classifier
model·=·SVC(kernel='linear',·C=1)
```

✓ 0.0s

Figure 5.4.2

(3) Fit the Model to the training data

```
# Fit the model to the training data
model = model.fit(X_train, y_train)
✓ 0.0s
```

Figure 5.4.3

(4) Make prediction ot the testing data

```
# Make predictions on the testing data
y_predict = model.predict(X_test)
✓ 0.0s
```

Figure 5.4.4

(5) Evaluate the model

Code:

```
# import libraries for evaluation of model
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_score, f1_score

print ('Accuracy:', accuracy_score(y_test, y_predict)*100)
print ('Recall:', recall_score(y_test, y_predict, average="weighted")*100)
print ('Precision:', precision_score(y_test, y_predict, average="weighted")*100)
confusion = confusion_matrix(y_test, y_predict)
print('Confusion matrix:')
print(confusion)
print("F1 Score: ", f1_score(y_test, y_predict, average='macro')*100)
```

Figure 5.4.5

- Evaluate all the Machine Learning Model and print all the various metrics and scores
- Print Confusion Matrix, Accuracy , F1 Score , Recall, and Precision

Output :

```
Accuracy: 86.88524590163934
Recall: 86.88524590163934
Precision: 86.88524590163934
Confusion matrix:
[[25  4]
 [ 4 28]]
F1 Score:  86.85344827586206
```

Figure 5.4.6

6. Comparison and recommendation

COMPARISON

When comparing the performance of the Decision Tree, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN) models, it becomes evident that each model displays strengths and weaknesses in different areas.

I. Decision Tree

The Decision Tree model showcases strong performance across all evaluated metrics, making it a reliable choice for classification tasks. With an accuracy of 88.52%, the model correctly classifies around 88.52% of instances, indicating a high level of correctness in its predictions. This shows the model's ability to make accurate classifications based on the given features.

The F1 score of 88.51% suggests a good balance between precision and recall, indicating that the model effectively identifies positive instances while maintaining a low number of false positives and false negatives. This balance is further supported by the recall value of 88.52%, which showcases the model's success in capturing a significant proportion of the positive instances present in the dataset.

Moreover, the precision value of 89.67% highlights the model's accuracy in predicting positive instances. It signifies that when the Decision Tree model labels an instance as positive, it is correct approximately 89.67% of the time. This indicates a relatively low rate of false positives, making a point of the model's ability to make precise positive predictions.

Decision Tree provide a moderate level of tunability. Decision Tree have various hyperparameters that can be modified to control the complexity of the tree and prevent overfitting. Some tuning hyperparameters in this machine

learning include the maximum depth of the tree, the minimum number of samples required to split a node, and the split criterion. By adjusting the hyperparameter, the Decision Tree can control the depth and width of the tree, which directly affects complexity. Tuning hyperparameters in the Decision Tree allows to adapt to different datasets and optimise performance.

Decision Tree has a high level of flexibility, Decision Tree can handle categorical and numerical data that makes well-suited for feature spaces with mixed data types. Decision Tree can show linear and non-linear relationships between features, which make Decision Tree can be adaptable to a wide range of classification and regression problems. The flexibility of Decision Tree lies in the ability to represent complex decision boundaries and accurately model complex relationships.

Decision Trees also have low space complexity. The memory requirement of Decision Trees is determined by the structure of the tree and the associated rule conditions. The memory usage increases with the depth and complexity of the tree. However, Decision Trees typically have lower space complexity since they store the tree structure and rule conditions rather than the entire training dataset. The space required to store a Decision Tree model depends on factors such as the number of nodes, features, and branching factors.

Decision Trees display different levels of model complexity. The complexity of a Decision Tree depends on factors such as the depth of the tree and the number of splits made at each node. Shallow Decision Tree with splits tend to represent simpler models, while deeper trees with numerous splits can capture more complex relationships and interactions between features. Decision Tree can represent both linear and non-linear decision boundaries and can handle both categorical and numerical data. Their ability to split the feature space allows Decision Tree to capture complex patterns and interactions in the data. Decision Trees also offer interpretability, as the resulting tree structure can be visualised and understood in terms of decision rules.

In summary, the Decision Tree model demonstrates strong performance in terms of accuracy, F1 score, recall, and precision. It consistently achieves high accuracy by correctly classifying instances, and the balanced F1 score reflects its ability to accurately identify positive instances while minimising false positives and false negatives. The model's high recall value suggests its effectiveness in capturing positive instances, while the precision value underscores its ability to make accurate positive predictions. Overall, the Decision Tree model is a dependable option for classification tasks, offering a good balance between accuracy and precision.

II. Support Vector Machine (SVM)

The SVM (Support Vector Machine) model demonstrates solid performance across all evaluated metrics, making it a reliable choice for classification tasks. With an accuracy of 86.89%, the model correctly classifies approximately 86.89% of instances, indicating a strong overall performance and a high level of correctness in its predictions.

The F1 score of 86.85% reflects a balanced trade-off between precision and recall, indicating that the SVM model effectively identifies positive instances while maintaining a reasonable number of false positives and false negatives. This balance is further supported by the recall value of 86.89%, which highlights the model's ability to capture a significant proportion of the positive instances present in the dataset.

Moreover, the precision value of 86.89% emphasises the model's accuracy in predicting positive instances. It suggests that when the SVM model labels an instance as positive, it is correct approximately 86.89% of the time. This indicates a relatively low rate of false positives, indicating the model's ability to make accurate positive predictions.

When seeing the tunability of SVM, SVM shows high tunability due to its various hyperparameters. Hyperparameter in SVM is the choice of the kernel function, which allows for different ways to modify the input data and capture complex relationships. Additionally, the regularisation parameter (C) can be tuned to control the trade-off between placing the training data precisely and preventing overfitting. By modifying the hyperparameter, SVM can be fine-tuned to best suit the specific characteristics of the dataset and the desired trade-off between bias and variance. Tuning the SVM hyperparameter can significantly impact the model’s performance and flexibility to different datasets.

Flexibility on SVM provides high flexibility as SVM can handle both linear and non-linear classification problems. By using various kernel functions such as linear, polynomial, or radial basis functions, SVM can capture complex decision boundaries and deal with different types of data distributions. Flexibility in SVM, enables SVM to adjust to various problem domains. SVM can be extended for regression tasks and further enhance their versatility.

SVM generally has a higher space complexity compared to KNN and Decision Trees. In SVM, the memory usage depends on the number of support vectors. Support vectors are the subset of training samples that lie closest to the decision boundary. As the number of support vectors increases, the memory requirements of the SVM model grow accordingly. Additionally, non-linear SVM also requires storing the kernel matrix, which can be memory-intensive,

especially for large datasets. Therefore, SVM may require a significant amount of memory to store the model and associated data structures.

SVM generally exhibits medium to high model complexity. SVM targets to find the optimal decision boundary that maximally separates different classes in the feature space. By using different kernel functions, SVM can model linear and non-linear relationships between features. This allows SVM to represent complex decision boundaries and capture intricate patterns in the data. As the complexity of the kernel function increases, SVM can handle more intricate and non-linear relationships, but this also leads to higher model complexity. However, the complexity of SVM can sometimes result in overfitting, especially if the model is not properly regularised or if the kernel function is overly complex.

In summary, the SVM model exhibits solid performance in terms of accuracy, F1 score, recall, and precision. It achieves high accuracy by correctly classifying instances, and the balanced F1 score underscores its ability to accurately identify positive instances while minimize false positives and false negatives. The model's recall value demonstrates its effectiveness in capturing positive instances, while the precision value highlights its ability to make accurate positive predictions. Overall, the SVM model proves to be a dependable option for classification tasks, providing a good balance between accuracy and precision.

III. K-Nearest Neighbors (KNN)

The KNN (K-Nearest Neighbors) model stands out among the compared models with its impressive performance metrics. With an accuracy of 90.79%, the KNN model demonstrates a high level of correctness in classifying instances, correctly identifying approximately 90.79% of instances. This highlights its reliability in making accurate predictions.

The F1 score of 90.71% indicates a strong balance between precision and recall, reflecting the model's ability to accurately classify positive instances while minimising false positives and false negatives. The recall value of 90.79% showcases the model's exceptional capability to identify a large portion of the positive instances, ensuring that it captures around 90.79% of the positive instances present in the dataset.

Moreover, the precision value of 90.80% highlights the model's accuracy in predicting positive instances. It suggests that when the KNN model labels an instance as positive, it is correct approximately 90.80% of the time.

This demonstrates a low rate of false positives and underlines the model's effectiveness in correctly identifying positive cases.

On the other hand, KNN has limited tunability compared to SVM and Decision Tree. KNN main hyperparameter is K, representing the number of nearest neighbours considered for classification. Adjusting value K allows balancing between underfitting which is a large value K and overfitting which is a small value K, but tunability options are relatively constrained compared to SVM and Decision Tree. By adjusting K, one can control the level of smoothing or roughness in the decision boundary. However, the tunability in KNN is more straightforward compared to SVM and Decision Tree. The selection of the K value depends on the nature of the datasets and the complexity of the underlying relationships between data points.

KNN in terms of flexibility, particularly in dealing with non-linear classification problems. KNN is a non-parametric algorithm that makes predictions based on the nearest neighbours in the feature space. This enables KNN to capture complex decision boundaries and modify them to various types of data distributions. KNN flexibility lies in the ability to model both linear and non-linear relationships between features, making it suitable for a wide range of classification tasks. However, KNN may run into challenges with high-dimensionality, as the effectiveness of the nearest neighbour search can be compromised.

KNN has relatively low space complexity during training. KNN models do not require storing any explicit model parameters. Instead, the memory requirements for KNN lie in storing the training samples themselves, along with their corresponding labels. During inference, the memory usage increases as the model needs to search and store the K nearest neighbours in the feature space. The space complexity of KNN depends on the size of the training dataset and the dimensionality of the feature space.

KNN has relatively low model complexity compared to SVM. KNN does not explicitly learn a model but instead relies on the stored training instances for making predictions. The complexity of KNN lies in the number of neighbours which is K and the distance metric used. KNN captures relationships in the data based on the closeness of instances in the feature space. However, KNN can also be a limitation as it does not learn explicit decision boundaries or capture complex interactions between features. KNN is more suited for simpler patterns or when the relationships in the data can be suitably captured by local proximity.

In summary, the KNN model showcases outstanding performance across all evaluated metrics. Its high accuracy, F1 score, recall, and precision values illustrate its competence in accurately classifying instances. With a

strong balance between precision and recall, the model achieves high accuracy while minimizing misclassifications. The KNN model's ability to capture a large proportion of positive instances with high precision makes it a reliable choice for various classification tasks.

RECOMMENDATION

Based on the comparison, it can be seen that KNN has high accuracy among SVM and Decision Tree. The best recommendation would be to use the KNN model. The KNN model is consistently better and higher than the Decision Tree and SVM models in terms of accuracy, F1 score, recall and precision. KNN achieves the highest scores in all these metrics, indicating. With its robustness in classification tasks, high accuracy, and precision, and a balanced F1 score and recall, the KNN model showcases the ability to make accurate predictions while effectively capturing positive instances. Therefore, the KNN model is recommended for this specific task due to its superior performance across all evaluations.

	K-Nearest Neighbors (KNN)	**Support Vector Machine (SVM)**	**Decision Tree**
Accuracy	90.79%	86.89%	88.52%
F1 Score	90.71%	86.85%	88.51%
Recall	90.79%	86.89%	88.52%
Precision	90.80%	86.89%	89.67%
Confusion Matrix	31 4 3 38	25 4 4 28	28 1 6 26

Table 6.1

7. Results and discussion

In this report,the heart disease data set was used. The goal of this report is to select the most suitable machine learning to accurately predict whether a person will likely get heart disease or not. Several evaluation metrics are used to evaluate the performance for these three machine learning which are K-Nearest Neighbors (KNN), Decision Tree and Support Vector Machine (SVM).

Based on table 6.1, the evaluation metrics such as accuracy, F1 score, recall and precision were compared for those three machine learning algorithms. The champion model is KNN as it has the highest accuracy which is 90.79% compared to SVM and Decision Tree. This shows that the proportion of correctly classified instances is the highest among SVM and Decision Tree. KNN demonstrates the minimum misclassification. Other than that, KNN also has the highest precision which is 90.80%. This demonstrates that of all the positive instances predicted and KNN has the highest ability to predict them correctly. Out of all instances that were predicted to be positive, KNN exhibits the highest accuracy in correctly identifying the positive instances because it has the highest recall values. F1 score represents a harmonic mean of precision and recall that provides a balanced measure.In comparison to SVM and Decision Tree, KNN has the highest F1 score.

The performance of those three machine learning that used in this report was compared. K-Nearest Neighbors (KNN) model achieved the highest accuracy, F1 score, recall and precision values among all models. Thus, it demonstrated the ability to accurately predict whether heart disease will exist or not. Hence, the champion model for this report is KNN.

8. Concluding remarks

We have learned a lot about machine learning models through learning, among which there are seven types of machine learning models that we are familiar with. The machine learning algorithms provided respectively are KNN (K-Nearest Neighbours), Decision Tree, SVM (Support Vector Machine), Naive Bayes Classifier, K-Means Clustering, Hierarchical Clustering, and Regression. We chose three of these seven machine learning models KNN, Decision Tree, and SVM (Support Vector Machine). Through a series of data comparison and analysis, we finally selected our champion model, which is KNN with the highest accuracy.

Compared with the other two machine learning models, KNN has some unique points that lead us to select it as our champion model. Among all models, the KNN model has the highest value of accuracy, F1 score, recall rate and accuracy.

9. Lesson learned from the project

Through such a group project, we learned a treasure trove of useful knowledge and experienced some difficulties. In the process of completing this project, each member of our team worked together, completed their own part, helped each other, expressed their own opinions, cultivated the thinking of solving problems, realised the importance of teamwork and communication, took the team as the centre, and mastered the skills of data analysis. Every group member could master effective tools used in machine learning, the ways to coordinate the work of team members, understand the ability of team members to allocate work reasonably, utilise one's advantages, and finally obtain the best solution via the discussion.In this process, we also encountered some problems.

In the process of this project, we also encountered a series of problems, and we had different opinions on data selection and data analysis. Regarding different opinions and ideas, we first discussed in the group and solved the problem by integrating the opinions of other members. If the problem could not be solved, we would open face-to-face meeting mode to discuss offline, and everyone would express their own ideas and solve the problem together.

About how to determine which algorithm is the best fit, we have also learned relevant knowledge through the following points, such as accuracy, precision, recall rate, f1 score and other indicators. Accuracy reflects a situation where the data is close to the reality, while F1 score is an indicator used to judge the performance of classification models, which combines recall rate and accuracy. It is a very important parameter in the algorithm.

As for how to select the model: we select KNN (K-Nearest Neighbours), Decision Tree and SVM (Support Vector Machine). Three algorithms are used to analyse data, and KNN algorithm is selected based on the comprehensive performance of the three algorithms. Through this project, we have also learned the advantages and disadvantages of the three algorithms. KNN algorithm is our target algorithm, which is simple to implement and easy to understand, has a large scope of application, and is not sensitive to outliers. The disadvantage of it is that the complexity of the calculation is relatively high, and if the number of samples is large, its complexity will be greater. Decision tree algorithms can deal with continuous features, can also be used for classification and classification, does not need normalisation, but is easy to overfit, while SVM is suitable for high-dimensional space, has a high generalisation ability, but it needs to optimise parameters, the calculation is too complex.

In the process of machine learning, if the model compounds training data, there will be an overfitting problem, resulting in a decline in data

generalisation performance. In order to solve this problem, we have learned several solutions in the process of the project, including cross-validation and hyperparameter regularisation, and the method of early stop. Regularisation refers to adding brave regularisation terms through loss functions. It is used to limit the size of data and improve generalisation performance. Cross-validation divides a training data into multiple data, alternately uses one of the separated data as our verification data, and selects the optimal model. Early stop means to stop training in time when the verification data no longer changes in performance during the verification process. The above methods can avoid the problem of data generalisation performance degradation.

10. Conclusion

This project mainly aims to analyse the data of human heart disease through three different algorithms of Machine learning, namely KNN (K-Nearest Neighbours), Decision Tree and SVM (Support Vector Machine). Based on the latest data, KNN (K-Nearest Neighbours), Decision Tree, SVM (Support Vector Machine) are used to further determine, predict and detect the risk of heart disease and the corresponding probability. By analysing the data of the same heart disease and comparing the results, we can analyse three different algorithms to determine which one combines our selection requirements, judge the accuracy, precision, recall rate, f1 score and area under ROC curve of the three algorithms, and then select the machine learning algorithm with the highest accuracy to predict the risk and probability of suffering from heart disease.

In summary, the KNN model showcases outstanding performance across all evaluated metrics. Its high accuracy, F1 score, recall, and precision values illustrate its competence in accurately classifying instances. With a strong balance between precision and recall, the model achieves high accuracy while minimising misclassifications. From the above analysis, we come to the conclusion that the accuracy of KNN is 90.79%, the score of F1 is 90.71%, the accuracy of SVM is 86.88%, the score of F1 is 88.51%, and the accuracy of Decision Tree is 83.60%, the score of F1 is 88.51%. Accuracy determines the reliability of an algorithm, and the score of F1 is an evaluation index that comprehensively considers accuracy and recall rate. By summing up the above data, we can know that KNN has the highest accuracy and F1 also has the highest score, thus we can know that the error rate of KNN algorithm is very small and it has a high accuracy. The performance of these three algorithms is also extremely excellent, so we choose the most reliable and accurate model -KNN, which has a very small error. KNN algorithms can be used to predict and judge whether the tester has heart disease more accurately and reliably.

11. Reference List

a. *What is the k-nearest neighbors algorithm? | IBM*. (n.d.). https://www.ibm.com/topics/knn#:~:text=The%20k%2Dnearest%20neighbors%20algorithm%2C%20also%20known%20as%20KNN%20or,of%20an%20individual%20data%20point.

b. *KNN Regression*. (n.d.). https://www.saedsayad.com/k_nearest_neighbors_reg.htm

c. Bajpai, I. (2020). Top 13 Data Mining Algorithms. *Geeky Humans*. https://geekyhumans.com/de/top-13-data-mining-algorithms/

d. Ghosh, S. (2023). How to Compare Machine Learning Models and Algorithms. *neptune.ai*. https://neptune.ai/blog/how-to-compare-machine-learning-models-and-algorithms

e. K, Dhiraj. "Top 5 Advantages and Disadvantages of Decision Tree Algorithm." *Medium*, 7 Feb. 2023, https://dhirajkumarblog.medium.com/top-5-advantages-and-disadvantages-of-decision-tree-algorithm-428ebd199d9a

f. Jim Woodruff,"What Are the Advantages of Decision Trees?" *Small Business - Chron.com*, 24 Jan. 2019, https://smallbusiness.chron.com/advantages-decision-trees-75226.html

g. Laura Auria1,Rouslan A. Moro. "Support Vector Machines (SVM) as a Technique for Solvency Analysis", Berlin, August. 2008, https://www.diw.de/documents/publikationen/73/diw_01.c.88369.de/dp811.pdf

h. "How Does KNN Algorithm Work ? What Are the Advantages and Disadvantages of KNN ?" *Machine Learning Interviews*, 14 Feb. 2019, https://machinelearninginterview.com/topics/machine-learning/how-does-knn-algorithm-work-what-are-the-advantages-and-disadvantages-of-knn/

i. Naik, S. (2023). Decision Tree Advantages and Disadvantages. *EDUCBA*. https://www.educba.com/decision-tree-advantages-and-disadvantages/

j. K, D. (2023, February 7). Top 4 advantages and disadvantages of Support Vector Machine or SVM. *Medium*. https://dhirajkumarblog.medium.com/top-4-advantages-and-disadvantages-of-support-vector-machine-or-svm-a3c06a2b107