MACHINE LEARNING
CSE4020

J COMPONENT

HEART DISEASE PREDICTION AND COMPARISON OF ALGORITHMS
KNN,SVM AND RANDOM FOREST

Submitted to : Dr. Naga Raja G
Slot: B1

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Allen Saldanha - 18BCB0088
Dhiren - 18BCB0089
Kshiti Sinha - 17BCE2340

PROBLEM DEFINITION

The aim of the project is to classify if the person has a heart disease or not. Along with this we also want to compare the results of three algorithms which are KNN,SVM and RANDOM FOREST CLASSIFIER on the basis of precision and accuracy.

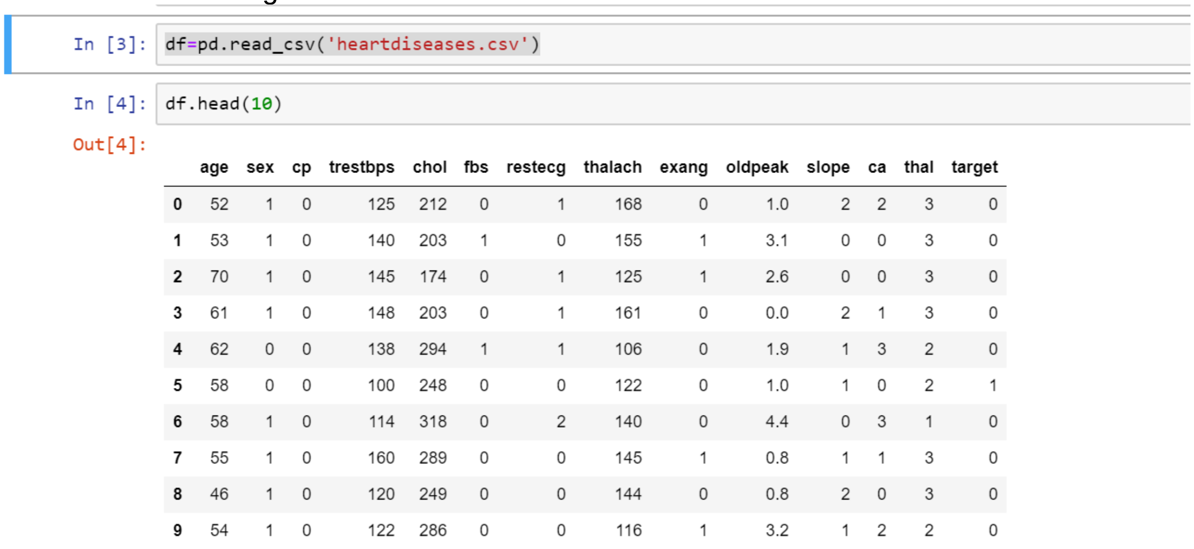
This data set dates from 1988 and consists of four databases: Cleveland, Hungary, Switzerland, and Long Beach V. It contains 76 attributes, including the predicted attribute, but all published experiments refer to using a subset of 14 of them. The "target" field refers to the presence of heart disease in the patient. It is integer valued 0 = no disease and 1 = disease.

Attributes that are present in the dataset- age,sex,chest pain type(4types),resting bp,serum cholesterol,fasting blood sugar,resting electrocardiographic results,maximum heart rate received,exercise induced angina,oldpeak = ST depression induced by exercise relative to rest,
The slope of the peak exercise ST segment,number of major vessels,thal: 0 = normal; 1 = fixed defect; 2 = reversible defect

DATASET

Heart Diseases
LINK - https://www.kaggle.com/johnsmith88/heart-disease-dataset

First 10 rows including all the columns



```
In [3]: df=pd.read_csv('heartdiseases.csv')

In [4]: df.head(10)

Out[4]:
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
5	58	0	0	100	248	0	0	122	0	1.0	1	0	2	1
6	58	1	0	114	318	0	2	140	0	4.4	0	3	1	0
7	55	1	0	160	289	0	0	145	1	0.8	1	1	3	0
8	46	1	0	120	249	0	0	144	0	0.8	2	0	3	0
9	54	1	0	122	286	0	0	116	1	3.2	1	2	2	0

This dataset was obtained from the UCI archive and is also available on kaggle too. It consists of data of 1025 individuals. This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "goal" field refers to the presence of heart disease in the patient. It is integer-valued from 0 (no presence) to 4. Experiments with the Cleveland database have concentrated on simply attempting to distinguish presence (values 1,2,3,4) from absence (value 0).

ALGORITHM USED

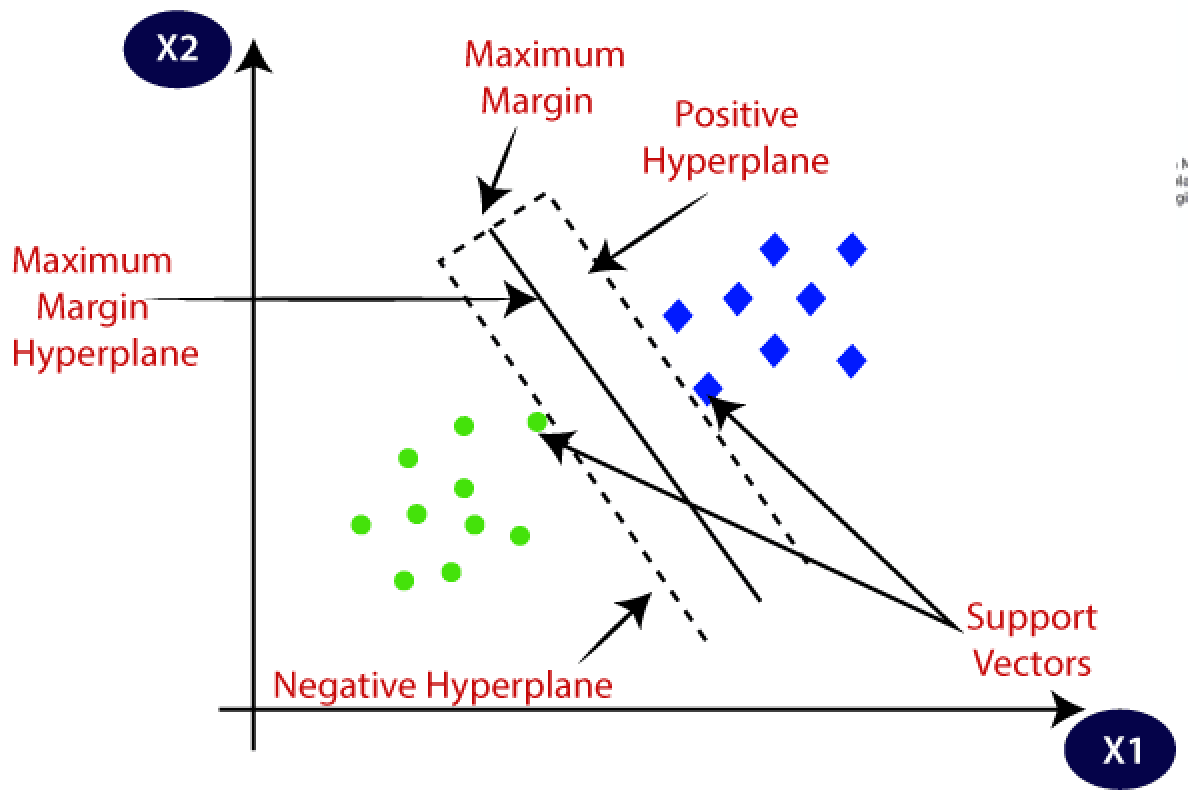
Our objective is to classify if the person has heart disease or not and we therefore use 3 algorithms for the same to also create a comparison

1.Support Vector Machine

Support vector machines (SVMs) is a set of supervised learning methods which can be used for classification, regression and outliers detection.
SVM or Support Vector Machine is a linear model for classification and regression problems. It can solve linear and non-linear problems and work well for many practical problems. The idea of SVM is simple: The algorithm creates a line or a hyperplane which separates the data into classes. Assuming that there are data points which are already labelled (as this is supervised learning). Our purpose is to divide them into two seperate classes with homogeneous data in each class and also teach the model to classify the new data into one of these using learning and modelling. A Support Vector Machine (**SVM**) performs classification by finding the **hyperplane** that maximizes the margin between the two classes. The vectors (cases) that define the **hyperplane** are the support vectors. The **vectors** (cases) that define the hyperplane are the **support vectors**.Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications. It calculates the distance between the two opposite classes to create a margin.The **SVM** in particular defines the criterion to be looking for a decision surface that is maximally far away from any data point. This distance from the decision surface to the closest data point determines the **margin** of the classifier.
In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where the number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.



2. K Nearest Neighbors

K-nearest neighbors (KNN) algorithm is a part of a supervised learning algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. These are the properties of KNN

1.Lazy learning algorithm − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.

2.Non-parametric learning algorithm − KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

Predictions are made for a new instance (x) by searching through the entire training set for the K most similar instances (the neighbors) and summarizing the output variable for those K instances. For regression this might be the mean output variable, in classification this might be the mode (or most common) class value.
To determine which of the K instances in the training dataset are most similar to a new input a distance measure is used. For real-valued input variables, the most popular distance measure is Euclidean distance.
Euclidean distance is calculated as the square root of the sum of the squared differences between a new point (x) and an existing point (xi) across all input attributes j.

EuclideanDistance(x, xi) = sqrt(sum((xj – xij)^2))

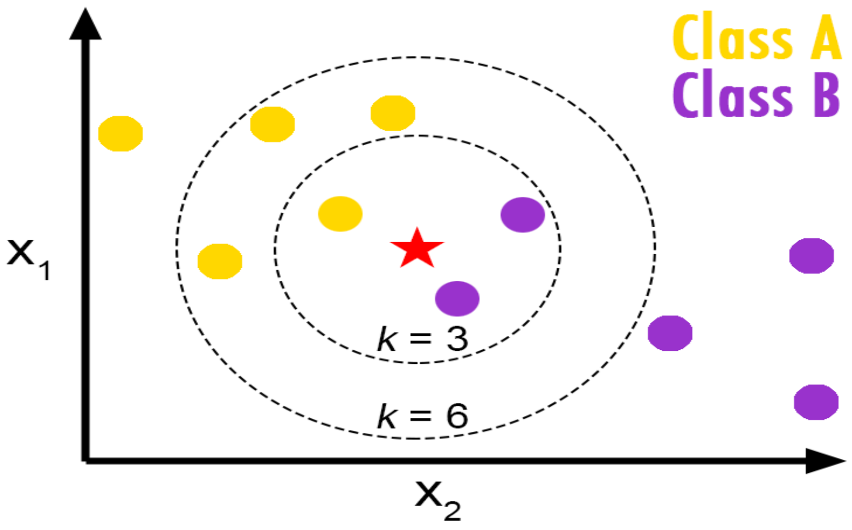
Other popular distance measures include:

- **Hamming Distance**: Calculate the distance between binary vectors.
- **Manhattan Distance**: Calculate the distance between real vectors using the sum of their absolute difference. Also called City Block Distance
- **Minkowski Distance**: Generalization of Euclidean and Manhattan distance

There are many other distance measures that can be used, such as Tanimoto, Jaccard, Mahalanobis and cosine distance. You can choose the best distance metric based on the properties of your data. If you are unsure, you can experiment with different distance metrics and different values of K together and see which mix results in the most accurate models.

The red star represents our test data point whose value is (2 , 1 , 3) . Our test point is surrounded by yellow and blue dots which represent our 2 classes. Now , we find out the distance from our test point to each of the dots present on the graph. Since there are 10 dots , we get 10 distances. We determine the lowest distance and predict that it belongs to the same class of its nearest neighbor. If a yellow dot is the closest ,then we predict that our test data point is also a yellow dot.
But , in some cases , you can also get two distances which are exactly equal. Here , we take into consideration a third data point and calculate its distance from our test data. In the above diagram , our test data lies in between the yellow and the blue dot . We considered the distance from thee 3rd data point and predicted that our test data is of class BLUE .
Advantages of kNN

1. Simple Implementation.
2. Makes no prior assumption of the data.



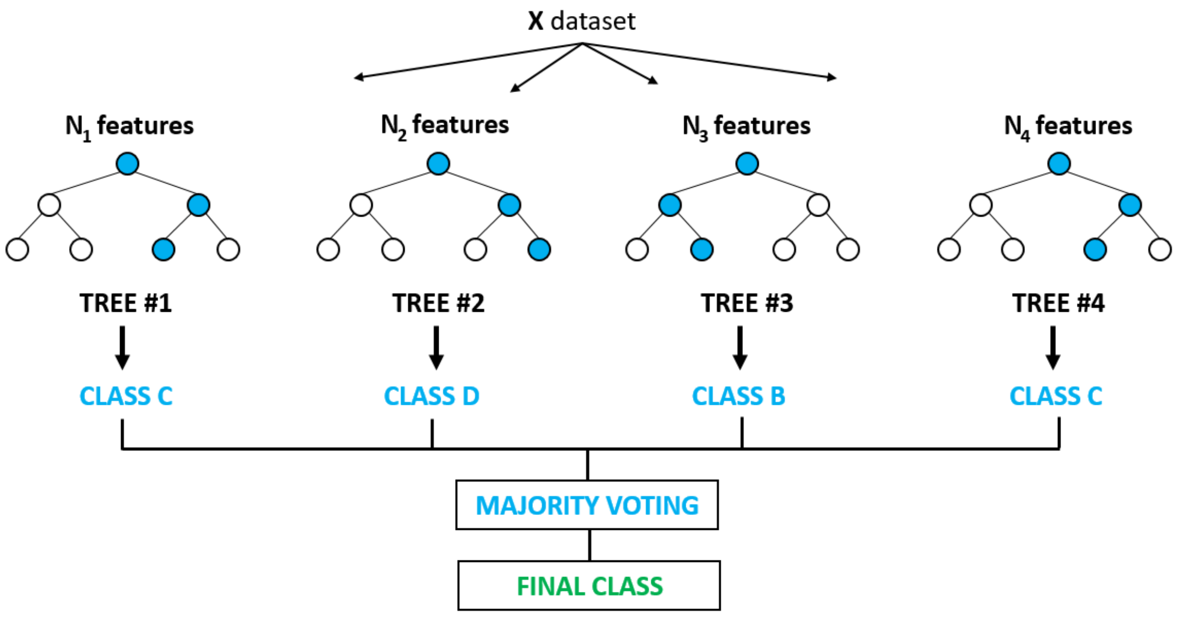
3. Random Forest Classifier
Most votes become our model’s prediction.Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. ***Ensembled algorithms*** are those which combine more than one algorithms of the same or different kind for classifying objects. For example, running predictions over Naive Bayes, SVM and Decision Tree and then taking a vote for final consideration of class for the test object. Each individual tree in the random forest spits out a class prediction and the class with the .A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So the prerequisites for random forest to perform well are:

1. There needs to be some actual signal in our features so that models built using those features do better than random guessing.
2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

Bagging stands for *bootstrap aggregation*. The aim is the same as for pruning. We want to reduce the variance of a decision tree. By the way we can use bagging for any kinds of learning algorithms. The principle is that we can reduce the variance by averaging a set of observations.

If we have a **X** set of **n** independent varibles $\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_n}$ each with variance **V** then the variance of the mean **X** (the mean of the $\mathbf{x_1}, \mathbf{x_2} \dots \mathbf{x_n}$ variables) is $\frac{\mathbf{V}}{\mathbf{n}}$



Proposed Architecture

1.Obtain the dataset of the patients
2. Perform data analysis and cleaning of data
3.Perform the different algorithms (SVM,KNN,Random Forest Classifier)
4. Prediction of the patient's health - that is if the patient has disease or not

CODE

MACHINE LEARNING PROJECT - PERFORMANCE COMPARISON FOR SVM, RANDOM FOREST AND KNN

```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import time
```

```
In [3]: df=pd.read_csv('heartdiseases.csv')
```

```
In [4]: df.head(10)
```

Out[4]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
5	58	0	0	100	248	0	0	122	0	1.0	1	0	2	1
6	58	1	0	114	318	0	2	140	0	4.4	0	3	1	0
7	55	1	0	160	289	0	0	145	1	0.8	1	1	3	0
8	46	1	0	120	249	0	0	144	0	0.8	2	0	3	0
9	54	1	0	122	286	0	0	116	1	3.2	1	2	2	0

SVM

```
In [5]: y=df['target'].values
        x1=df.drop('target',axis=1)
```

```
In [ ]: Normalising the values
```

```
In [7]: x = (x1 - np.min(x1))/(np.max(x1)-np.min(x1)).values
```

```
In [8]: #Split the data For Train and Test
        from sklearn.model_selection import train_test_split
        xtrain, xtest, ytrain, ytest = train_test_split(x,y,test_size=0.3, random_state=42)
```

```
In [9]: #Now we are going to check the SVM accuracy
        from sklearn.svm import SVC

        SVM = SVC(random_state=42)
        start_time=time.time()
        SVM.fit(xtrain,ytrain)
        end_time=time.time()
        run_time=(end_time-start_time)*1000

        print ("SVM Accuracy:", SVM.score(xtest,ytest))

        SVMscore = SVM.score(xtest,ytest)

        SVM Accuracy: 0.8051948051948052
```

```
In [10]: from sklearn.metrics import confusion_matrix
```

```
In [11]: l=confusion_matrix(SVM.predict(xtest),ytest)
```

```
In [12]: precision=(l[1][1]/(l[1][1] + l[0][1]))
         recall=(l[1][1]/(l[1][0] + l[1][1]))
         accuracy=((l[0][0]+l[1][1])/(l[0][0]+l[0][1]+l[1][0]+l[1][1]))
         f1_score=2*((precision*recall)/(precision+recall))
```

```
In [13]: print("Accuracy :",accuracy)
         print('\n')
         print("Precision :",precision)
         print('\n')
         print("Recall :",recall)
         print('\n')
         print("F1score :",f1_score)

         print('\n')

         print('Running time :',run_time)

         Accuracy : 0.8051948051948052


         Precision : 0.9194630872483222


         Recall : 0.7405405405405405


         F1score : 0.8203592814371257
```

RANDOM FOREST

```
In [14]: df=df.rename(columns={0: 'age', 1:'sex', 2:'cp', 3:'trestbps',4: 'chol',5: 'fbs',
         6: 'restecg',7: 'thalach',8: 'exang',9: 'oldpeak',10: 'slope',11: 'ca',12: 'thal',13:'target'})
```

```
In [15]: from sklearn.ensemble import RandomForestClassifier
```

```
In [ ]:
```

```
In [17]: x=df.drop('target',axis=1)
         y=df['target']
```

```
In [18]: from sklearn.ensemble import RandomForestClassifier
```

```
In [20]: randomtree=RandomForestClassifier(n_estimators=300,random_state=101)
```

```
In [21]: randomtree.fit(x,y)
```

```
Out[21]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=300, n_jobs=None,
                     oob_score=False, random_state=101, verbose=0, warm_start=False)
```

```
In [22]: #For cp number 3 = no pain, while 0-2 refers to pain,
         #hence 3 will be grouped as disease -ve and 0-2 as disease +ve
```

```
In [22]: #For cp number 3 = no pain, while 0-2 refers to pain,
         #hence 3 will be grouped as disease -ve and 0-2 as disease +ve
```

```
In [23]: number=[0,1,2]
         for col in df.itertuples():
             if (col.cp in number):
                 df['cp'].replace(to_replace=col.cp,value=1,inplace=True)
```

```
In [24]: df.head()
```

Out[24]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	1	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	1	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	1	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	1	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	1	138	294	1	1	106	0	1.9	1	3	2	0

```
In [25]: df_top9 = df.loc[:,['cp','oldpeak','thal','ca','thalach','age','chol','trestbps','exang']]
```

```
In [27]: #With the inbuilt functions
         from sklearn.metrics import confusion_matrix,precision_recall_fscore_support,accuracy_score
         from sklearn.model_selection import train_test_split
         from sklearn import  metrics
         from sklearn.metrics import precision_score
```

```
In [28]: x_train,x_test,y_train,y_test = train_test_split(df_top9,y,test_size=0.4,random_state=101)
         ranforclass=RandomForestClassifier()
         ranforclass.fit(x_train,y_train)
         predictions=ranforclass.predict(x_test)
         accuracy=accuracy_score(predictions,y_test)
         cm=confusion_matrix(predictions,y_test)
         prfs=precision_recall_fscore_support(predictions,y_test)

         print('Accuracy :',accuracy)
         print('\n')
         print('Confusion_Matrix: ',cm)
         print('\n')
         print('Precision: ',prfs[0])
         print('Recall: ',prfs[1])
         print('Fscore: ',prfs[2])
         print('Support: ',prfs[3])
```

```
Accuracy : 0.9414634146341463


Confusion_Matrix:  [[189  17]
 [  7 197]]


Precision:  [0.96428571 0.92056075]
Recall:  [0.91747573 0.96568627]
Fscore:  [0.94029851 0.94258373]
Support:  [206 204]
```

KNN

```
In [29]: df=pd.read_csv('heartdiseases.csv')
```

```
In [30]: df.head(10)
```

Out[30]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
5	58	0	0	100	248	0	0	122	0	1.0	1	0	2	1
6	58	1	0	114	318	0	2	140	0	4.4	0	3	1	0
7	55	1	0	160	289	0	0	145	1	0.8	1	1	3	0
8	46	1	0	120	249	0	0	144	0	0.8	2	0	3	0
9	54	1	0	122	286	0	0	116	1	3.2	1	2	2	0

```
In [31]: df.target.value_counts()

Out[31]: 1    526
         0    499
         Name: target, dtype: int64
```

```
In [32]: from sklearn.preprocessing import StandardScaler
```

```
In [33]: scaler=StandardScaler()
```

```
In [34]: scaler.fit(df.drop('target',axis=1))

         C:\Users\AllenAlben\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:645: DataConversionWarning: Data with input dtype
         int64, float64 were all converted to float64 by StandardScaler.
           return self.partial_fit(X, y)

Out[34]: StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
In [35]: scaled_features=scaler.transform(df.drop('target',axis=1))

         C:\Users\AllenAlben\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: DataConversionWarning: Data with input dtype int64, fl
         oat64 were all converted to float64 by StandardScaler.
           """Entry point for launching an IPython kernel.
```

```
In [36]: scaled_features

Out[36]: array([[-0.26843658,  0.66150409, -0.91575542, ...,  0.99543334,
                  1.20922066,  1.08985168],
                [-0.15815703,  0.66150409, -0.91575542, ..., -2.24367514,
                 -0.73197147,  1.08985168],
                [ 1.71659547,  0.66150409, -0.91575542, ..., -2.24367514,
                 -0.73197147,  1.08985168],
                ...,
                [-0.81983438,  0.66150409, -0.91575542, ..., -0.6241209 ,
                  0.23862459, -0.52212231],
                [-0.4889957 , -1.51170646, -0.91575542, ...,  0.99543334,
                 -0.73197147, -0.52212231],
                [-0.04787747,  0.66150409, -0.91575542, ..., -0.6241209 ,
                  0.23862459,  1.08985168]])
```

```
In [37]: df_feat=pd.DataFrame(scaled_features,columns=df.columns[:-1])
```

```
In [38]: df_feat.head()
```

Out[38]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	-0.268437	0.661504	-0.915755	-0.377636	-0.659332	-0.418878	0.891255	0.821321	-0.712287	-0.060888	0.995433	1.209221	1.089852
1	-0.158157	0.661504	-0.915755	0.479107	-0.833861	2.387330	-1.004049	0.255968	1.403928	1.727137	-2.243675	-0.731971	1.089852
2	1.716595	0.661504	-0.915755	0.764688	-1.396233	-0.418878	0.891255	-1.048692	1.403928	1.301417	-2.243675	-0.731971	1.089852
3	0.724079	0.661504	-0.915755	0.936037	-0.833861	-0.418878	0.891255	0.516900	-0.712287	-0.912329	0.995433	0.238625	1.089852
4	0.834359	-1.511706	-0.915755	0.364875	0.930822	2.387330	0.891255	-1.874977	-0.712287	0.705408	-0.624121	2.179817	-0.522122

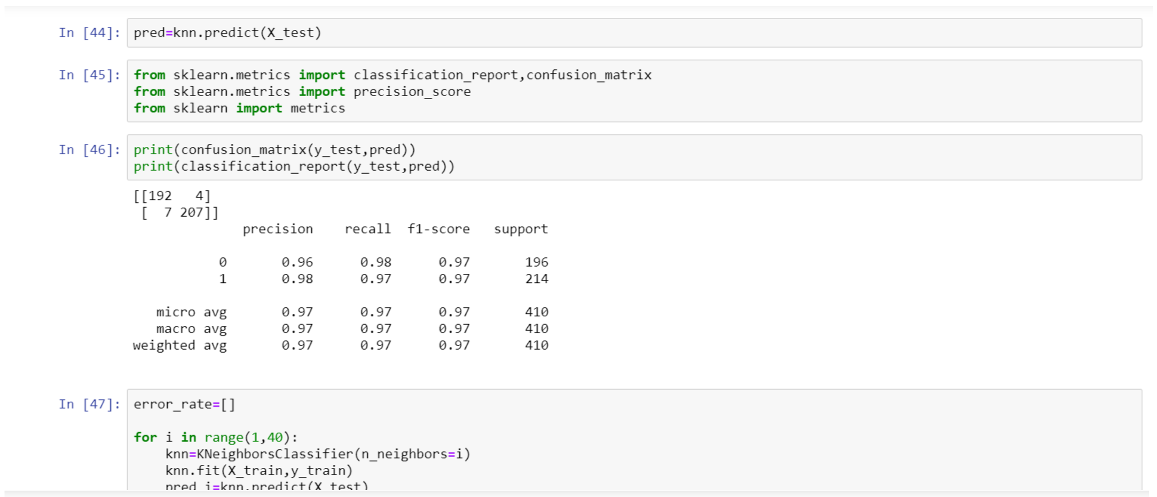
```
In [39]: from sklearn.model_selection import train_test_split
```

```
In [40]: X=df_feat
         y=df['target']
         X_train,X_test,y_train,y_test=train_test_split(X, y, test_size=0.4, random_state=101)
```

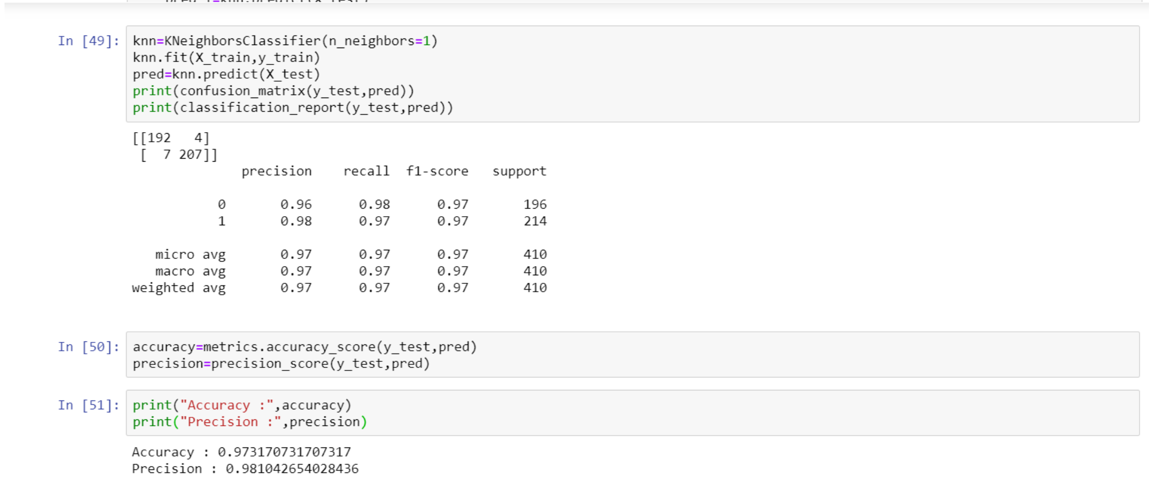
```
In [41]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [42]: knn=KNeighborsClassifier(n_neighbors=1)
```

```
In [43]: knn.fit(X_train,y_train)

Out[43]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                    weights='uniform')
```



```
In [44]: pred=knn.predict(X_test)

In [45]: from sklearn.metrics import classification_report,confusion_matrix
         from sklearn.metrics import precision_score
         from sklearn import metrics

In [46]: print(confusion_matrix(y_test,pred))
         print(classification_report(y_test,pred))

         [[192   4]
          [  7 207]]
                       precision    recall  f1-score   support

                    0       0.96      0.98      0.97       196
                    1       0.98      0.97      0.97       214

            micro avg       0.97      0.97      0.97       410
            macro avg       0.97      0.97      0.97       410
         weighted avg       0.97      0.97      0.97       410


In [47]: error_rate=[]

         for i in range(1,40):
             knn=KNeighborsClassifier(n_neighbors=i)
             knn.fit(X_train,y_train)
             pred_i=knn.predict(X_test)
```



```
In [49]: knn=KNeighborsClassifier(n_neighbors=1)
         knn.fit(X_train,y_train)
         pred=knn.predict(X_test)
         print(confusion_matrix(y_test,pred))
         print(classification_report(y_test,pred))

         [[192   4]
          [  7 207]]
                       precision    recall  f1-score   support

                    0       0.96      0.98      0.97       196
                    1       0.98      0.97      0.97       214

            micro avg       0.97      0.97      0.97       410
            macro avg       0.97      0.97      0.97       410
         weighted avg       0.97      0.97      0.97       410


In [50]: accuracy=metrics.accuracy_score(y_test,pred)
         precision=precision_score(y_test,pred)

In [51]: print("Accuracy :",accuracy)
         print("Precision :",precision)

         Accuracy : 0.9731707317073171
         Precision : 0.981042654028436
```

RESULTS

From the above implementation of three different algorithms,we compare the different values of accuracy values

To conclude the accuracy for

SVM: 80.51948%

RANDOM FOREST: 94.14634%

KNN: 97.31707%

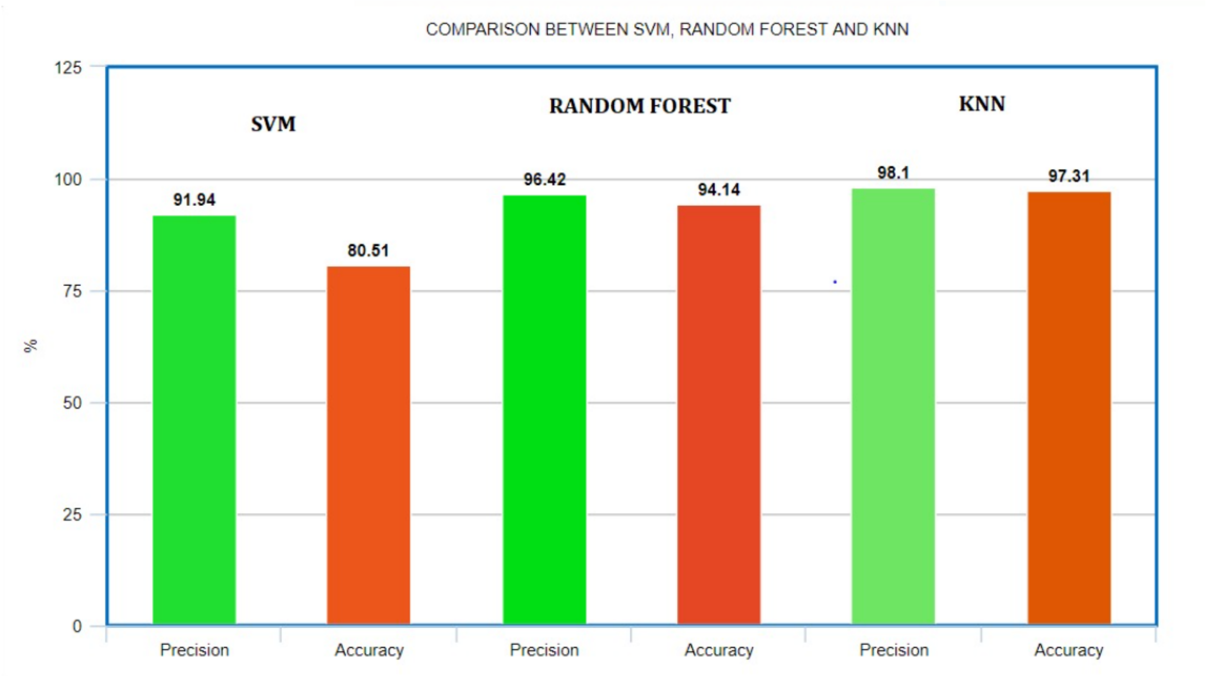
Hence we can conclude that KNN is the most accurate

KNN provides the highest value of accuracy

DISCUSSIONS

Each and every dataset has its own limitations which a particular machine learning algorithm can make use of to achieve maximum accuracy. In this case, where we had data of 1025 individuals and we were able to notice that the algorithm which provides the highest accuracy and thereby the best performance for this particular dataset was the K-Nearest Neighbours algorithm.
In SVM. Two classes were created where one class is people with heart disease and other without heart disease.
In KNN, with groups are made based on characteristics with a particular value of k.
In Random Forest Classifier, a large number of decision trees are made to arrive at a conclusion if the person has heart disease or not.

Accuracy and precision comparison between KNN, SVM, Random Forest:


COMPARISON BETWEEN SVM, RANDOM FOREST AND KNN
SVM
RANDOM FOREST
KNN
Precision
Accuracy
Precision
Accuracy
Precision
Accuracy