Heart Disease Dataset
Using
Decision Tree, Random forest, Adaboosting

The heart as we know is an important organ of the human body that pumps blood through the body. If circulation of blood in the body is inefficient the organs like the brain suffer and if the heart stops working altogether, death occurs within minutes. Life is completely dependent on efficient functioning of the heart. The term Heart disease refers to disease of the heart. A number of factors have been shown in the data set that increases the risk of Heart disease:

- age
- sex
- chest pain type (4 values)
- resting blood pressure
- serum cholestoral in mg/dl
- fasting blood sugar > 120 mg/dl
- resting electrocardiographic results (values 0,1,2)
- maximum heart rate achieved
- exercise induced angina
- oldpeak = ST depression induced by exercise relative to rest
- the slope of the peak exercise ST segment
- number of major vessels (0-3) colored by flourosopy
- thal: 3 = normal; 6 = fixed defect; 7 = reversable defect

Import libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# To generate confusion matrix
from sklearn.metrics import confusion_matrix
#To split dataset to training and test set‘
from sklearn.model_selection import train_test_split

#classifiers
from tree import DecisionTreeClassifier

#to obtain accuracy score of model
from sklearn.metrics import accuracy_score

#to generate classification report
from sklearn.metrics import classification_report
```

Load Dataset

```
df = pd.read_csv('heart.csv')
df.head()
```

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

Data exploration: Data analysis and visualization

The data set we have consists of factors that might lead to heart disease. Information of dataset, It shows we have 13 features that will help to predict target i.e heart disease true and false

```
df.info()
```

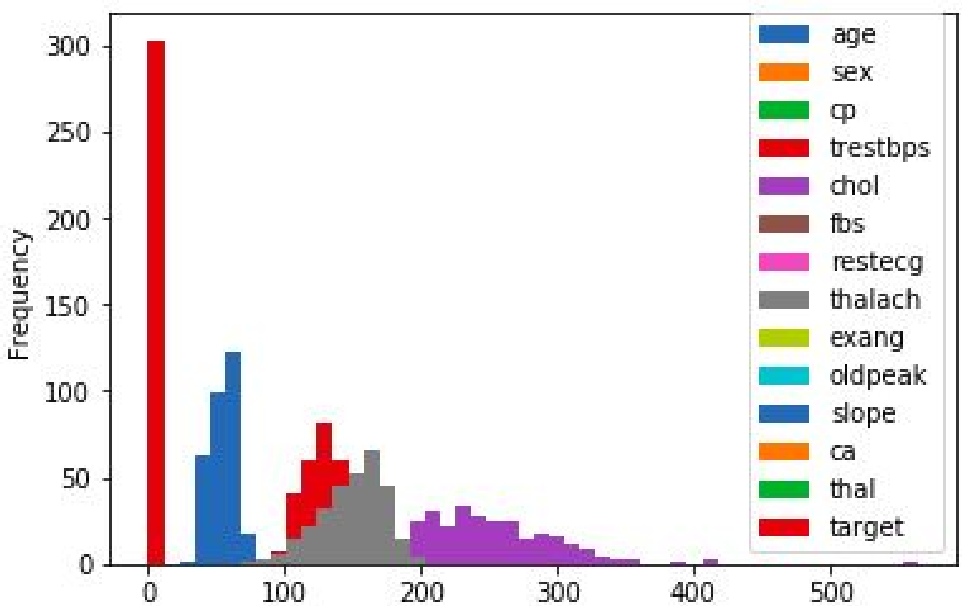
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

Obtaining further description on the data set

```
df.describe()
```

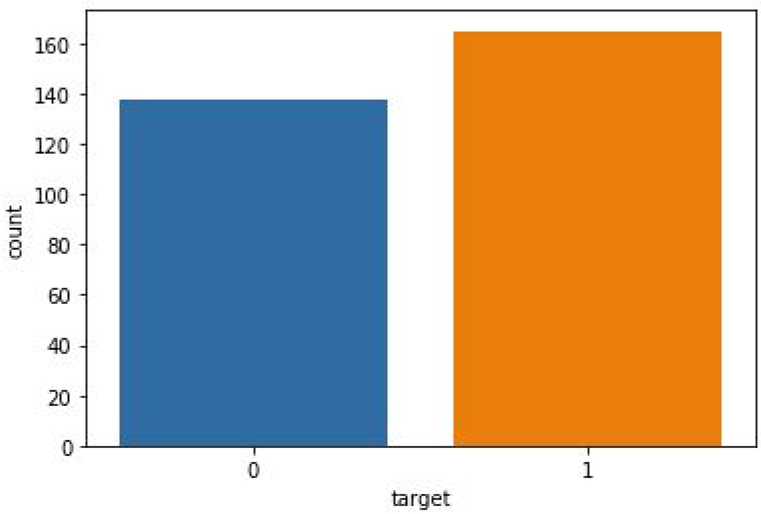
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000

```
df.plot.hist(bins=50)
```



The above histogram shows the frequency plot diagram of the features

```
sns.countplot(x="target", data=df)
```



Countplot diagram shows about 160 of the record is Positive cases and around 140 cases Negetive cases

SPlit data to train and test set

```
X = df.drop(columns='target')
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1,
stratify=y
```

Splitting the data to train and test set considering 80% of data to train and 20% to test to predict target i.e

0: Has Disease
1: No Disease

Features: Extraction and normalization

OneHotEncoding

One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. The categorical value represents the numerical value of the entry in the dataset

```
cat_columns = ['cp', 'exang', 'slope', 'thal']
num_columns = [c for c in X_train.columns if c not in cat_columns]
```

Categorical columns : ['cp', 'exang', 'slope', 'thal']
Num_columns: ['age', 'sex', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'oldpeak', 'ca']

Importing OneHotEncoder library using:

```python
from sklearn.preprocessing import OneHotEncoder
# OneHotEncoder()
num_columns + list(cat_columns)
```

```python
# create oneHotEncoder instance
encoder = OneHotEncoder(handle_unknown='ignore')


#Fit on categorical columns
encoder.fit(X_train[cat_columns])


#Transform on training data
encoder.categories_
X_train_cat_encoded = encoder.transform(X_train[cat_columns])

column_names = encoder.get_feature_names(input_features = cat_columns)
print(column_names)
X_train_cat_encoded_df = pd.DataFrame(X_train_cat_encoded.todense(),
                                      columns=column_names,
                                      index=X_train.index)
X_train_cat_encoded_df
```

```
['cp_0' 'cp_1' 'cp_2' 'cp_3' 'exang_0' 'exang_1' 'slope_0' 'slope_1'
 'slope_2' 'thal_0' 'thal_1' 'thal_2' 'thal_3']
```

	cp_0	cp_1	cp_2	cp_3	exang_0	exang_1	slope_0	slope_1	slope_2	thal_0	thal_1	thal_2	thal_3
0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0
2	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
3	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0
4	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
237	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
238	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
239	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0

```python
X_train_encoded = pd.concat([X_train[num_columns],X_train_cat_encoded_df], axis=1)
X_train_encoded
X_train_encoded.columns
```

```
Index(['age', 'sex', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'oldpeak', 'ca', 'cp_0', 'cp_1', 'cp_2', 'cp_3', 'exang_0', 'exang_1',
       'slope_0', 'slope_1', 'slope_2', 'thal_0', 'thal_1', 'thal_2',
       'thal_3'],
      dtype='object')
```

Similarly encoding the test data sets using one hotEncoder.

Model Build

SVM Model

```
from sklearn.svm import SVC
```

```
def svm_model(X_train_v, y_train):
    model = SVC(random_state = 1)
    model.fit(X_train_v, y_train)
    return model

```

```
model= svm_model(X_train_enc, y_train)
model
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=1, shrinking=True, tol=0.001,
    verbose=False)
```

Decision Tree classifier

1. **Simple Decision Tree classifier**

```
model = DecisionTreeClassifier(random_state = 1)
model.fit(X_train_encoded, y_train)
preds = model.predict(X_test_encoded)

print(classification_report(y_test, preds))
```

```
              precision    recall  f1-score   support

           0       0.65      0.71      0.68        28
           1       0.73      0.67      0.70        33

    accuracy                           0.69        61
   macro avg       0.69      0.69      0.69        61
weighted avg       0.69      0.69      0.69        61
```

2. **Using entropy**

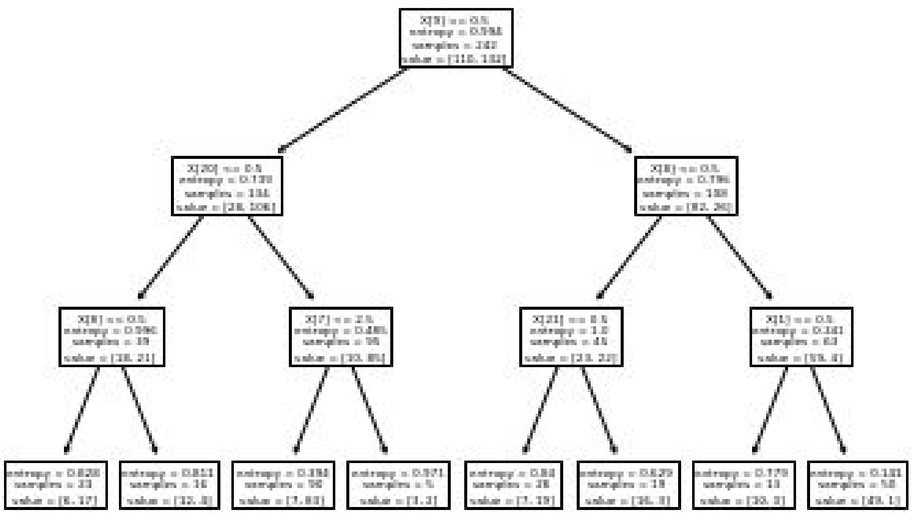
Entropy is the measure of uncertainty of a random variable, it characterizes the impurity of data. Entropy is the parameter used in Decision tree classifier, a function used to measure the quality of split. Higher the entropy more information content.

```
def train_using_entropy(X_train, X_test, y_train):
    clf_entropy = DecisionTreeClassifier(
            criterion = "entropy", random_state = 100,
            max_depth = 3, min_samples_leaf = 5)

    return clf_entropy
```

Plot Decision Tree:

```
from sklearn.tree import export_graphviz
from sklearn import tree
export_graphviz(model, out_file='tree.dot',  feature_names = X_train_encoded.columns,
            class_names = ['0', '1'],   rounded = True, proportion = False,
            precision = 2, filled = True)
tree.plot_tree(clf_entropy.fit(X_train_encoded, y_train))
```



Model evaluation using the test set.

Now, will run the model on the train and test set using SVC(Support Vector Classifier) ,
and then use the test set to see what kind of prediction results we get
using the test data set for the Support Vector Machines as well as the Random Forest Mode ab
other models

```
from sklearn.metrics import accuracy_score

preds=model.predict(X_test_enc)
accuracy_score(y_test, preds)
```

```
0.639344262295082
```

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, preds))
```

```
              precision    recall  f1-score   support

           0       0.69      0.39      0.50        28
           1       0.62      0.85      0.72        33

    accuracy                           0.64        61
   macro avg       0.65      0.62      0.61        61
weighted avg       0.65      0.64      0.62        61
```

Above classification report is from SVM model evaluation with accuracy score of **63.93%**

Prediction Model:

```
def prediction(X_test, clf_object):

    # Predicton on test with giniIndex
    y_pred = clf_object.predict(X_test)
    print("Predicted values:")
    print(y_pred)
    return y_pred
```

Accuracy check model:

```
# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

    print("Confusion Matrix: ",
    confusion_matrix(y_test, y_pred))


    print ("Accuracy : ",
    accuracy_score(y_test,y_pred)*100)



    print("Report : ",
    classification_report(y_test, y_pred))
```

clf_entropy = train_using_entropy(X_train_encoded, X_test_encoded, y_train)
y_pred_entropy = prediction(X_test_encoded, clf_entropy)

cal_accuracy(y_test, y_pred_entropy)

```
Predicted values:
[1 0 1 1 1 1 1 1 0 0 0 1 0 1 0 1 1 1 1 0 1 0 0 1 0 1 0 1 0 0 1 1 1 0 1 0 0
 0 1 1 1 0 1 1 1 1 1 0 0 1 1 1 0 1 0 1 0 0 0 1 1]
Confusion Matrix:  [[21  7]
 [ 4 29]]
Accuracy :  81.9672131147541
Report :               precision    recall  f1-score   support

           0       0.84      0.75      0.79        28
           1       0.81      0.88      0.84        33

    accuracy                           0.82        61
   macro avg       0.82      0.81      0.82        61
weighted avg       0.82      0.82      0.82        61
```

Accuracy: 81.967%

Model Selection

Grid search is the process of performing hyper parameter tuning in order to determine the optimal values for a given model. In this case parameters we use are max_tree_depth, depth, CV to fit the model and train the data and select the best model to predict the output

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline
```

Parameter tuning for SVM model

```
grid_param = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                     'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}
```

```
from sklearn.metrics import make_scorer, f1_score
scorer = make_scorer(f1_score, average='micro')
```

```
scorer = make_scorer(f1_score, average='micro')
clf = GridSearchCV(SVC(), grid_param,scoring=scorer)
clf.fit(X_train_enc, y_train)
```

```
print(clf.best_score_, clf.best_params_)
```

```
0.8142857142857143 {'C': 1, 'kernel': 'linear'}
```

Model selection shows that it is 81.4% accurate and best parameters is C:1, kernel: linear

Hyper parameter tunning Decision Tree Classifier

```
#Build a pipeline
tree_steps = [('tree', DecisionTreeClassifier(random_state =42))]
tree_pipe = Pipeline(tree_steps)

tree_params = {"tree__max_depth":list(range(2,20))}

tree = GridSearchCV(tree_pipe, param_grid = tree_params, cv=10, scoring = "accuracy")
tree.fit(X_train_encoded, y_train)
```

Best Parameter:

```
#best hyperparameters
tree.best_params_

{'tree__max_depth': 3}
```

Classification report:

```
Confusion Matrix:  [[20  8]
 [11 22]]
Accuracy :  68.85245901639344
Report :               precision    recall  f1-score   support

           0       0.65      0.71      0.68        28
           1       0.73      0.67      0.70        33

    accuracy                           0.69        61
   macro avg       0.69      0.69      0.69        61
weighted avg       0.69      0.69      0.69        61
```

Accuracy : 68.85%

```
print("Training accuracy is: {}".format(accuracy(X_train_encoded, y_train,tree)))
print("Testing accuracy is: {}".format(accuracy(X_test_encoded, y_test,tree)))

Training accuracy is: 0.86
Testing accuracy is: 0.82
```

Training accuracy: 86%
Testing accuracyL 82%

Random Forest

Model1:

```
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

#Build a random forest pipeline
rf_steps = [('rf', RandomForestClassifier(random_state=42))]
rf_pipe = Pipeline(rf_steps)

#Hyperparameter tuning
#n_estimators == no. of trees
rf_params = {"rf__n_estimators":list(range(2,11))}
rf = GridSearchCV(rf_pipe,param_grid = rf_params, cv=10)
rf.fit(X_train_encoded, y_train)
```

Best parameter:

rf.best_params

{'rf__n_estimators': 9}

```
#Performance
print("Training accuracy is: {}".format(accuracy(X_train_encoded, y_train,rf)))
print("Testing accuracy is: {}".format(accuracy(X_test_encoded, y_test,rf)))
```

```
Training accuracy is: 0.98
Testing accuracy is: 0.74
```

Training accuracy: 98%
Testing accuracyL 74%

Above accuracy result showed that , random forest model overfit on the training data. As we are not controlling the maximum depth of each tree in the ensemble like in simple decision tree model.

Model 2:

```
rf2_steps = [('rf2', RandomForestClassifier(random_state=42))]
rf2_pipe = Pipeline(rf2_steps)

rf2_params = {"rf2__n_estimators":list(range(2,11)),
              "rf2__min_samples_leaf":list(range(5,50,5))}
rf2 = GridSearchCV(rf2_pipe,param_grid = rf2_params, cv=10)
rf2.fit(X_train_encoded, y_train)
```

Best parameter:

rf2.best_params

{'rf2__min_samples_leaf': 5, 'rf2__n_estimators': 9}

```
#Performance
print("Training accuracy is: {}".format(accuracy(X_train_encoded, y_train,rf2)))
print("Testing accuracy is: {}".format(accuracy(X_test_encoded, y_test,rf2)))
```

```
Training accuracy is: 0.88
Testing accuracy is: 0.8
```

Training accuracy: 88%
Testing accuracyL 80%

By controlling the minimum number of leaves in each tree limits the depth of the tree. With a classification accuracy of 80% on the test set,

Model3:

```
from sklearn.ensemble import RandomForestClassifier

forest_clf = RandomForestClassifier(n_estimators = 100,
                                    random_state = 42)
forest_scores = cross_val_score(forest_clf, X_train_enc,
                                y_train, cv=10)
forest_scores.mean()
```

```
0.826
```

AdaBoosting

Boosting is an ensemble method that is fundamentally different from bagging and random forests. RF combines decision trees in parallel, boosting combines models additively. The ensemble model in boosting is a linear combination of simpler trees. AdaBoost, which stands for Adaptive Boosting. It helps to improve the performance of weak base learners by training new trees on points that were incorrectly classified. Thus, the complex model at the end of this iterative process delivers higher classification accuracy.

```
from sklearn.preprocessing import StandardScaler

#Build an adaboost pipe
boost_steps = [('standardizer', StandardScaler()),
               ('boost', AdaBoostClassifier(random_state=42))]
boost_pipe = Pipeline(boost_steps)
```

```
#hyperparamter tuning
boost_params = {"boost__n_estimators":list(range(2,20)),
                "boost__learning_rate":[0.001,0.01,0.1,1,1.4,1.8,2,3,4]}
boost = GridSearchCV(boost_pipe,param_grid = boost_params, cv=10)
boost.fit(X_train_encoded, y_train)
```

Best parameter:

boost.best_params

{'boost__learning_rate': 1, 'boost__n_estimators': 3}

```
#Performance
print("Training accuracy is: {}".format(accuracy(X_train_encoded, y_train,boost)))
print("Testing accuracy is: {}".format(accuracy(X_test_encoded, y_test,boost)))
```

```
Training accuracy is: 0.86
Testing accuracy is: 0.82
```

Training accuracy: 86%
Testing accuracyL 82%

Accuracy obtained from AdaBoost classification is similar to simple decision tree.

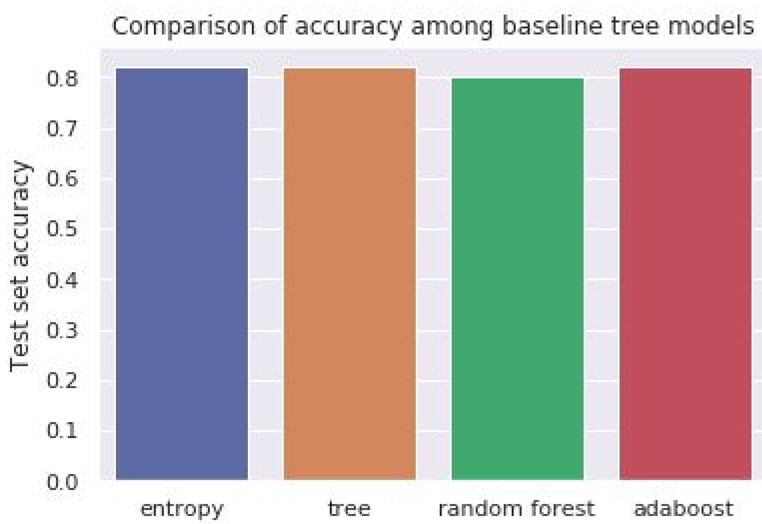
Comparing the models

```
acc_dict = {}

all_models = [clf_entropy, tree, rf2, boost]
all_model_names = ["entropy","tree","random forest","adaboost"]
for i in range(len(all_models)):
    acc_dict[all_model_names[i]] = accuracy(X_test_encoded,y_test,all_models[i])
```

```
#Plot performance comparison bar plot
sns.barplot(list(acc_dict.keys()), list(acc_dict.values()))
plt.title("Comparison of accuracy among baseline tree models")
plt.ylabel("Test set accuracy")
```

Below is the bar plot of model accuracy



Final Model

Comparing the models from the above bar plot, Simple decision tree, decision tree with entropy and adaboost had almost the same accuracy.
So using adaboost for final model prediction

And considering SVM model using best parameters we get:

```
final_model = SVC(random_state=1, kernel="linear", C=1.0)
final_model.fit(X_train_enc, y_train)
y_pred = final_model.predict(X_test_enc)
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.79      0.81        28
           1       0.83      0.88      0.85        33

    accuracy                           0.84        61
   macro avg       0.84      0.83      0.83        61
weighted avg       0.84      0.84      0.84        61
```

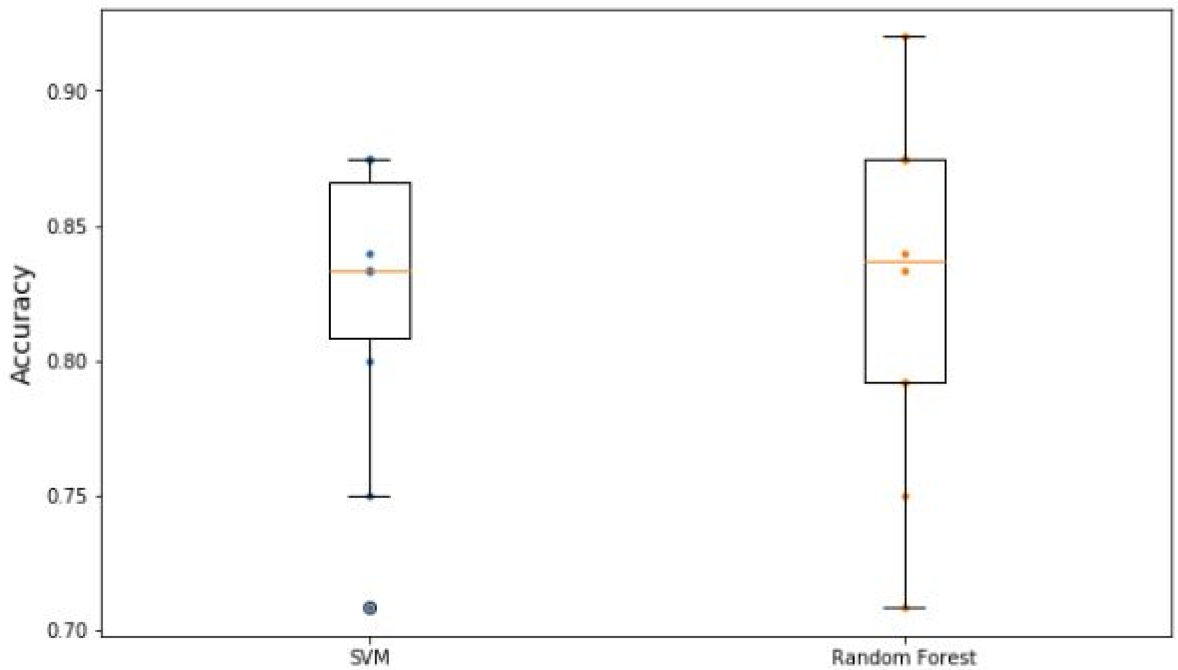
```
svm_scores = cross_val_score(model, X_train_enc, y_train, cv=10)
svm_scores.mean()
```

```
0.8223333333333335
```

Accuracy of model : 82.23%

As we can see that the from Random Forests model 3 is slightly better than that of the SVM Model and below graphically as well.

```
plt.figure(figsize=(10,6))
plt.plot([1]*10, svm_scores, ".")
plt.plot([2]*10, forest_scores, ".")
plt.boxplot([svm_scores, forest_scores], labels=("SVM",
                                                 "Random Forest"))
plt.ylabel("Accuracy", fontsize=14)
plt.show()
```



I

Code link

https://github.com/Kristiee/Hear_disease_classifier/blob/master/DT_RF_Adaboost.ipynb

https://github.com/Kristiee/Hear_disease_classifier/blob/master/SVM_final_Heart_Disease.ipynb