Heart Failure Prediction Using Supervised Machine Learning

Shivam Kumar

School of Computer Science and Engineering

Lovely Professional University,

Phagwara, Punjab, India, 144411

rahul.shiva64@gmail.com

Abstract- Heart Failure is one of the biggest problems in the world for common people. For rich person it is very easy to go into a big hospital and get report after so many tests but for the common, middle class and lower-class family it is not possible to get correct result at right time. So, if we can predict the head failure by using some basic data then it will be highly beneficial. In this paper I am going to discuss some of the Machine Learning model to predict heart failure prediction and its accuracy. I have taken data from Kaggle for training and testing purpose. To predict the outcome, one need to provide few data such as Age, Gender, Chest Pain, Resting BP, Cholesterol, Fasting BS, Resting EC, Max HR, Exercise A, Old Peak and ST Slope.

1. Introduction

Heart failure sometimes known as congestive heart failure occurs when the hear muscle doesn't pump blood as well as it should. When this happens, blood often backs up and fluid can build up in the lungs, causing shortness of breath.

Heart failure signs and symptoms may include:

- Shortness of breadth with activity or when lying down.
- Fatigue and weakness
- Swelling in the legs, ankles and feet
- Rapid or irregular heartbeat
- Reduced or irregular heartbeat
- Reduced ability to exercise
- Persistent cough or wheezing with white or pink blood-tinged mucus
- Swelling of the belly area
- Very rapid weight gain from fluid build-up
- Chest pain

To predict heart failure, I am going to use that information of a patient which can play important role:

1) Age of the patient

2) Gender of the patient

3) Chest pain type

4) Resting blood pressure

5) Serum cholesterol

6) Fasting blood sugar

7) Resting electrocardiogram results

8) Maximum heart rate achieved

9) Exercise-induced angina

10) Old peak

11) The slope of the peak exercise ST segment

2. Literature Review

I took data set for building model from Kaggle which is an open-source platform. This data set is the combination of 4 different research. At first, I performed data pre-processing on the data which includes data cleaning, normalization, standardization and feature selection, then Exploratory data analysis, feature engineering, training of model and finding accuracy of model respectively.

```
#df=pd.read_csv('heart.csv')
#df.head()
URL = 'https://drive.google.com/file/d/1Yt7Kr9tbitUscefZdk-5Gt2YN_QQKVmW/view?usp=sharing'
path = 'https://drive.google.com/uc?export=download&id='+URL.split('/')[-2]
df = pd.read_csv(path)
df.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0

The dataset contains 918 rows and 12 features, it doesn't contain any null value, outlier and dirty value therefore we not require to perform any extra work on data cleaning part.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

The dataset doesn't contain any duplicate value so we don't require to remove any row.

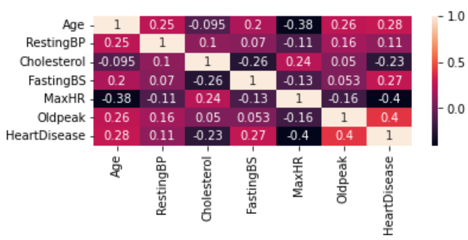
```
df.duplicated().sum()
```

```
0
```

The data is less correlated so we don't require to perform feature reduction.

```
plt.figure(figsize=(6,2))
sns.heatmap(df.corr(),annot=True)
```

```
<AxesSubplot:>
```



There were some numerical columns and some categorical column.

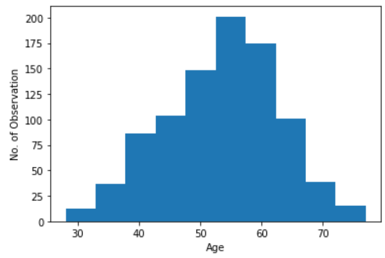
```
df.dtypes
```

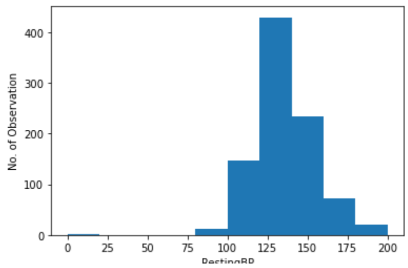
```
Age                 int64
Sex                object
ChestPainType      object
RestingBP           int64
Cholesterol         int64
FastingBS           int64
RestingECG         object
MaxHR               int64
ExerciseAngina     object
Oldpeak           float64
ST_Slope           object
HeartDisease        int64
dtype: object
```

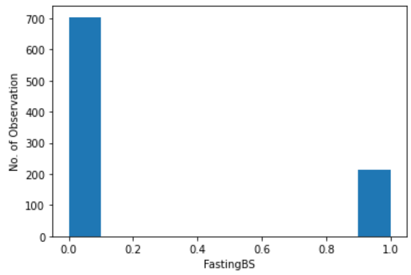
Numerical columns are: Age, RestingBP, Cholesterol, FastingBS, MaxHR, Oldpeak.

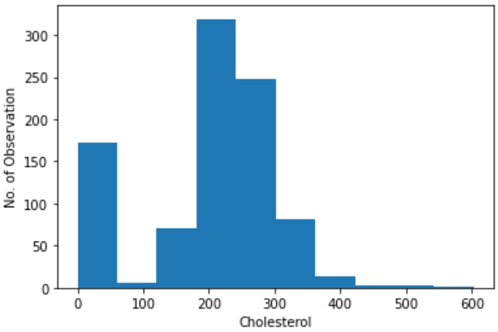
Categorical columns are: Sex, ChestPainType,RestingECG, ExerciseAngina, ST_Slope.

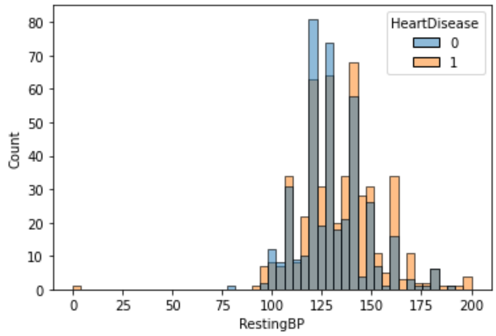
Frequency are not evenly distributed as per the feature:

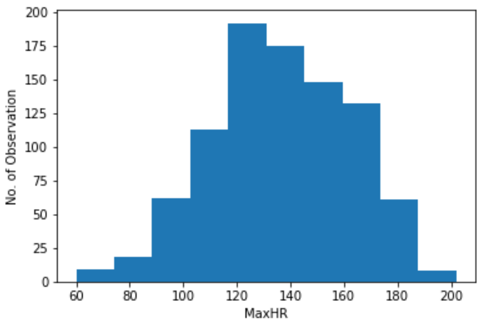


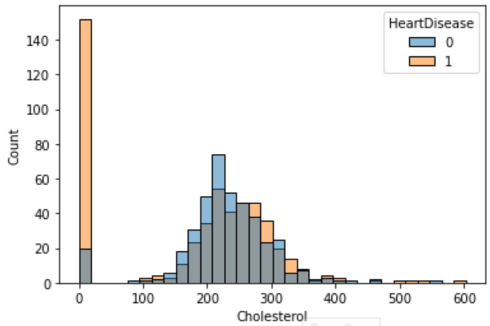


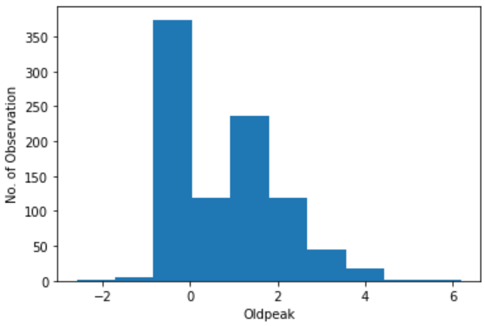


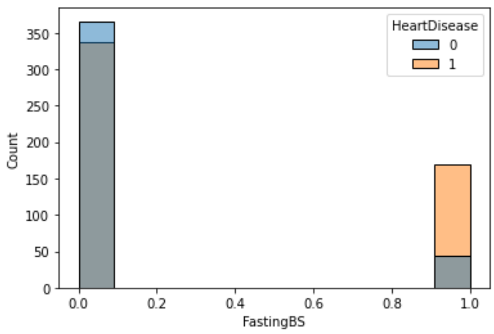

No. of Observation
0 50 100 150 200 250 300
0 100 200 300 400 500 600
Cholesterol

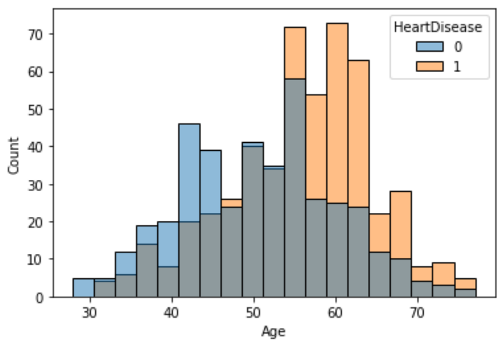

HeartDisease
0
1
Count
0 10 20 30 40 50 60 70 80
0 25 50 75 100 125 150 175 200
RestingBP

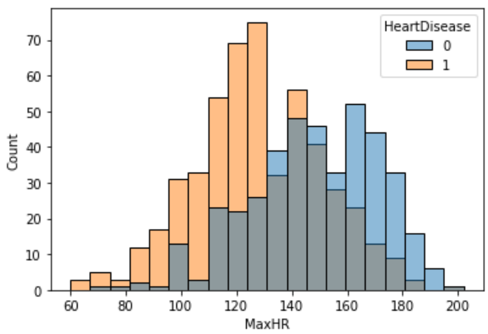

No. of Observation
0 25 50 75 100 125 150 175 200
60 80 100 120 140 160 180 200
MaxHR

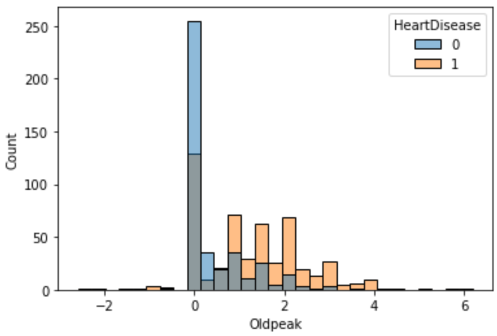

HeartDisease
0
1
Count
0 20 40 60 80 100 120 140
0 100 200 300 400 500 600
Cholesterol


No. of Observation
0 50 100 150 200 250 300 350
−2 0 2 4 6
Oldpeak


HeartDisease
0
1
Count
0 50 100 150 200 250 300 350
0.0 0.2 0.4 0.6 0.8 1.0
FastingBS


HeartDisease
0
1
Count
0 10 20 30 40 50 60 70
30 40 50 60 70
Age


HeartDisease
0
1
Count
0 10 20 30 40 50 60 70
60 80 100 120 140 160 180 200
MaxHR


HeartDisease
0
1
Count
0 50 100 150 200 250
−2 0 2 4 6
Oldpeak

Out of 918 rows, 504 row have positive outcome and 410 have negative outcome.

```
HaveHeartDisease=df[df['HeartDisease']==1].shape[0]
```

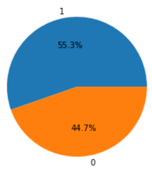
```
NotHaveHeartDisease=df[df['HeartDisease']==0].shape[0]
```

```
print("No. of Patient Having Heart Disease: ",HaveHeartDisease)
print("No. of Patient don't Have Heart Disease: ",NotHaveHeartDisease)
```

```
No. of Patient Having Heart Disease:  508
No. of Patient don't Have Heart Disease:  410
```

```
plt.pie([HaveHeartDisease,NotHaveHeartDisease],autopct='%1.1f%%',labels=['1','0'])
```

```
: ([<matplotlib.patches.Wedge at 0x175a7a521f0>,
   <matplotlib.patches.Wedge at 0x175a7a52910>],
  [Text(-0.1835941114214546, 1.0845705151124876, '1'),
   Text(0.1835941114214547, -1.0845705151124876, '0')],
  [Text(-0.10014224259352068, 0.5915839173340841, '55.3%'),
   Text(0.10014224259352074, -0.5915839173340841, '44.7%')])
```



As we are using supervised machine learning we need to store data as feature and target.

```
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

We require to split data into 2 parts so that we can perform training and testing efficiently without any bias.

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3)
```

```
xtrain.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope
801	56	M	ASY	132	184	0	LVH	105	Y	2.1	Flat
158	44	M	ASY	130	290	0	Normal	100	Y	2.0	Flat
596	57	M	ASY	122	264	0	LVH	100	N	0.0	Flat
816	58	M	ASY	125	300	0	LVH	171	N	0.0	Up
132	56	M	ASY	170	388	0	ST	122	Y	2.0	Flat

```
xtest.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope
503	51	M	ASY	132	227	1	ST	138	N	0.2	Up
771	55	M	ASY	140	217	0	Normal	111	Y	5.6	Down
815	68	M	NAP	118	277	0	Normal	151	N	1.0	Up
888	52	M	ASY	128	204	1	Normal	156	Y	1.0	Flat
227	38	M	ASY	92	117	0	Normal	134	Y	2.5	Flat

Data contains categorical column we require to convert it into numerical form by using label encoder or OneHotEncoder. I am using label encoder.

```
obj={}

#categorical_column=['Sex','ChestPainType','RestingECG','ExerciseAngina','ST_Slope']
for i in categorical_column:
    le=LabelEncoder()
    obj[i]=le
    xtrain[i]=obj[i].fit_transform(xtrain[i])
    xtest[i]=obj[i].transform(xtest[i])
```

After converting categorical data into numeric form. Data is:

```
xtrain.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope
801	56	1	0	132	184	0	0	105	1	2.1	1
158	44	1	0	130	290	0	1	100	1	2.0	1
596	57	1	0	122	264	0	0	100	0	0.0	1
816	58	1	0	125	300	0	0	171	0	0.0	2
132	56	1	0	170	388	0	2	122	1	2.0	1

```
xtest.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope
503	51	1	0	132	227	1	2	138	0	0.2	2
771	55	1	0	140	217	0	1	111	1	5.6	0
815	68	1	2	118	277	0	1	151	0	1.0	2
888	52	1	0	128	204	1	1	156	1	1.0	1
227	38	1	0	92	117	0	1	134	1	2.5	1

Data is not in standardized. So, we will standardize it:

```
sc=StandardScaler()
x_train_scaled=sc.fit_transform(xtrain)
x_test_scaled=sc.fit_transform(xtest)
```

```
x_train_scaled
```

```
array([[ 0.29600495,  0.51117185, -0.8315057 , ...,  1.22793017,
         1.22148071, -0.59360286],
       [-1.00116351,  0.51117185, -0.8315057 , ...,  1.22793017,
         1.12496494, -0.59360286],
       [ 0.40410232,  0.51117185, -0.8315057 , ..., -0.81437856,
        -0.80535042, -0.59360286],
       ...,
       [ 0.18790758,  0.51117185, -0.8315057 , ...,  1.22793017,
         0.83541764, -2.24335624],
       [ 0.72839444,  0.51117185, -0.8315057 , ...,  1.22793017,
        -0.32277158, -0.59360286],
       [ 1.05268655,  0.51117185, -0.8315057 , ...,  1.22793017,
         1.60754378, -2.24335624]])
```

```
xtrain=pd.DataFrame(x_train_scaled,columns=xtrain.columns[:])
xtest=pd.DataFrame(x_test_scaled,columns=xtest.columns[:])
xtrain.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope
0	0.296005	0.511172	-0.831506	-0.009811	-0.130499	-0.554559	-1.506339	-1.263179	1.227930	1.221481	-0.593603
1	-1.001164	0.511172	-0.831506	-0.120313	0.847123	-0.554559	0.045940	-1.462697	1.227930	1.124965	-0.593603
2	0.404102	0.511172	-0.831506	-0.562323	0.607329	-0.554559	-1.506339	-1.462697	-0.814379	-0.805350	-0.593603
3	0.512200	0.511172	-0.831506	-0.396570	0.939352	-0.554559	-1.506339	1.370459	-0.814379	-0.805350	1.056151
4	0.296005	0.511172	-0.831506	2.089736	1.750963	-0.554559	1.598218	-0.584818	1.227930	1.124965	-0.593603

After completion of data pre-processing, exploratory data analysis, standardization and normalization, the remaining is training and calculating accuracy.

```
m1=LogisticRegression()
m2=RidgeClassifier(alpha=0.25)
m3=DecisionTreeClassifier(criterion='gini',max_depth=5)
m4=RandomForestClassifier(criterion='gini',n_estimators=50)
m5=AdaBoostClassifier(algorithm='SAMME',learning_rate=0.1,n_estimators=150)
m6=GradientBoostingClassifier(criterion='friedman_mse',learning_rate=0.05,n_estimators=100)
m7=XGBClassifier(learning_rate=0.1,n_estimators=50)
m8=KNeighborsClassifier(n_neighbors=7)
```

```
dictionary={'Logistic Regression':m1,'Ridge':m2,'Decision Tree':m3,'Random Forest':m4,'Ada Booster':m5,'Gradient Boost':m6,'X
```

```
best=0
bestName='Logistic Regression'
ScoreName={}
for i in dictionary.keys():
    model=dictionary[i]
    model.fit(xtrain,ytrain)
    pred=model.predict(xtest)
    score=model.score(xtest,ytest)
    print("Model Name: ",i)
    print("Accuracy: ",accuracy_score(ytest,pred))
    print()
    print(classification_report(ytest,pred))
    print('Confusion Matrix: \n',confusion_matrix(ytest,pred))
    print('####################################################################')
    if(score>best):
        best=score
        bestName=i
    ScoreName[i]=score
```

3. Comparison of algorithm's accuracy

Training Accuracy will be as follows:

```
Model Name:  Logistic Regression
Accuracy:  0.8043478260869565

              precision    recall  f1-score   support

           0       0.75      0.81      0.78       116
           1       0.85      0.80      0.83       160

    accuracy                           0.80       276
   macro avg       0.80      0.81      0.80       276
weighted avg       0.81      0.80      0.81       276

Confusion Matrix:
 [[ 94  22]
 [ 32 128]]
```

```
Model Name:  Ridge
Accuracy:  0.7971014492753623

              precision    recall  f1-score   support

           0       0.73      0.82      0.77       116
           1       0.86      0.78      0.82       160

    accuracy                           0.80       276
   macro avg       0.79      0.80      0.79       276
weighted avg       0.80      0.80      0.80       276

Confusion Matrix:
 [[ 95  21]
 [ 35 125]]
```

```
Model Name:  Decision Tree
Accuracy:  0.8369565217391305

              precision    recall  f1-score   support

           0       0.80      0.81      0.81       116
           1       0.86      0.86      0.86       160

    accuracy                           0.84       276
   macro avg       0.83      0.83      0.83       276
weighted avg       0.84      0.84      0.84       276

Confusion Matrix:
 [[ 94  22]
 [ 23 137]]
```

```
Model Name:  Random Forest
Accuracy:  0.8369565217391305

              precision    recall  f1-score   support

           0       0.80      0.82      0.81       116
           1       0.87      0.85      0.86       160

    accuracy                           0.84       276
   macro avg       0.83      0.83      0.83       276
weighted avg       0.84      0.84      0.84       276

Confusion Matrix:
 [[ 95  21]
 [ 24 136]]
```

```
Model Name:  Ada Booster
Accuracy:  0.822463768115942

              precision    recall  f1-score   support

           0       0.77      0.82      0.79       116
           1       0.86      0.82      0.84       160

    accuracy                           0.82       276
   macro avg       0.82      0.82      0.82       276
weighted avg       0.82      0.82      0.82       276

Confusion Matrix:
 [[ 95  21]
 [ 28 132]]
```

```
Model Name:  Gradient Boost
Accuracy:  0.8514492753623188

              precision    recall  f1-score   support

           0       0.83      0.82      0.82       116
           1       0.87      0.88      0.87       160

    accuracy                           0.85       276
   macro avg       0.85      0.85      0.85       276
weighted avg       0.85      0.85      0.85       276

Confusion Matrix:
 [[ 95  21]
 [ 20 140]]
```

4.Outcome

After comparison of all above algorithm the higher accuracy is of Gradient boosting algorithm.

```
print("Maximum Score: ",best)
print("Name: ",bestName)
```

```
Maximum Score:  0.8514492753623188
Name:  Gradient Boost
```

5.Heart failure overview

Now a days most of the people uses mobile phones and computers even in rural areas. If we can host the model on the internet then the most of the people will be benefitted in spite of location, financial condition or time. Those who belongs to rural area or poor family, who don't go for regular check-ups they will be really benefitted.

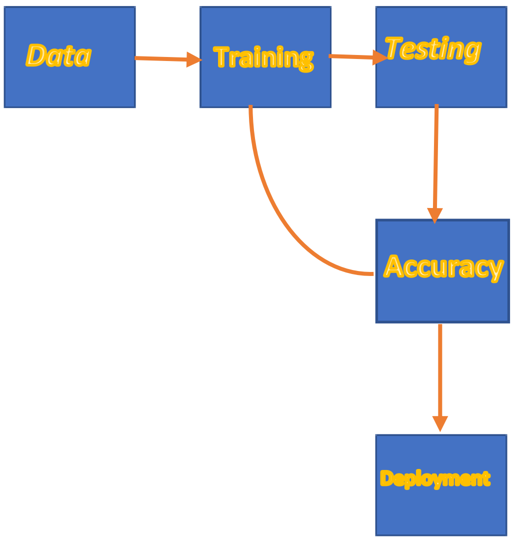
6.Supervised Machine learning

Machine learning is the branch of computer science where we create/ build a machine which can learn from data and take decision for unseen data without being explicitly programmed.

Supervised machine learning is a type of machine learning where we train model with feature data as well as target data.

Supervised machine learning is of two types. The first one is regression where output will be numerical data and second one is classification where output will be discrete value.

7. Working of model



8.Logistic Regression

Logistic regression is a type of supervised classification machine learning. This algorithm is used when we have binary discrete target outcome such as true and false, 0 and 1, yes or no etc. Sigmoidal function is used as target function to build this model.

9. Ridge Regression

When logistic regression gets overfit/ in case of multicollinearity. The variance gets high then we require to increase bias. The bias is added as penalty term in Ridge Regression.

10. Decision Tree

In classification problem, we use decision tree to predict output. It makes a hierarchy of question and takes decision based on the output of the question. Every node in the tree represents the question and edge represents the decision. Finally leaf nodes are the decision.

11. Random Forest

Random Forest is the combination of Decision Tree. This is bagging technique in which data are used as with replacement for every Decision tree. Random forest uses more than one algorithm.

12.Ada Boost

It is the type of ensemble learning, boosting technique in which more the one algorithm is used in sequential form. Data used by every algorithm is without replacement. One by one model will pass the result to subsequent model.

13.Gradient Boosting

Gradient Boosting is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor's error. In contrast to Adaboost, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of predecessor as labels.

14. References

1)https://classroom.google.com/c/NDQ4MTk1MDU0Mjgw/p/NDY3Nzc4MDc3NzA4/details

2)https://doi.org/10.1371/journal.pone.0181001

3)https://doi.org/10.1186/s12911-020-1023-5

4)https://doi.org/10.1186/s12911-020-1023-5

15. Implementation of Model

https://colab.research.google.com/drive/1MckOfR4Me8GH5jOvAEeajzwyheh96OEs?usp=sharing