Heart Disease

Finding the best classification model to predict heart disease

Presented by Charla Gaddy

Dataset

https://archive.ics.uci.edu/ml/datasets/heart+Disease

This database contains 76 attributes, 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.

	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 attributes

* age
* sex
* chest pain
* resting blood pressure
* cholesterol
* fasting blood sugar
* resting electrocardiographic
* maximum heart rate
* exercised induced angina
* depression induced by exercise
* slope of the peak exercise
* major blood vessels
* thallium stress test
* target

Project Design

The main question to answer is whether a model can accurately predict if a person has heart disease or not, based on the attributes of the data.

The objective is to create the best model by increasing accuracy, in the hold test sample and increasing the AUC -- area under the ROC curve.

The recall and precision will also be calculated to make sure the model isn't predicting a lot of false positives.

The data will first be run on 6 baseline models and then 4 models will be tuned via gridsearch.

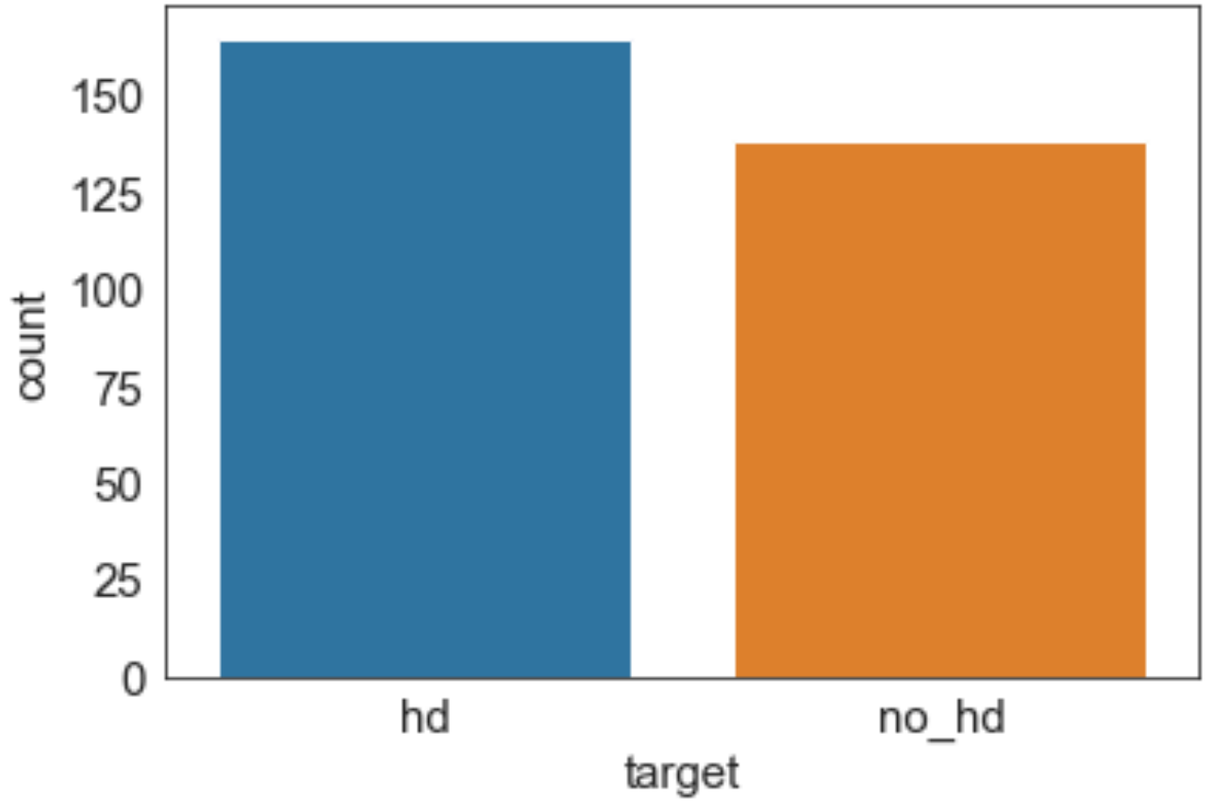
54.46% of the people in the dataset have heart disease.

```
target = (sum(df['target'])/len(df['target'].index))*100
print('My baseline : {:.2f}%'.format(target))
```

```
My baseline : 54.46%
```

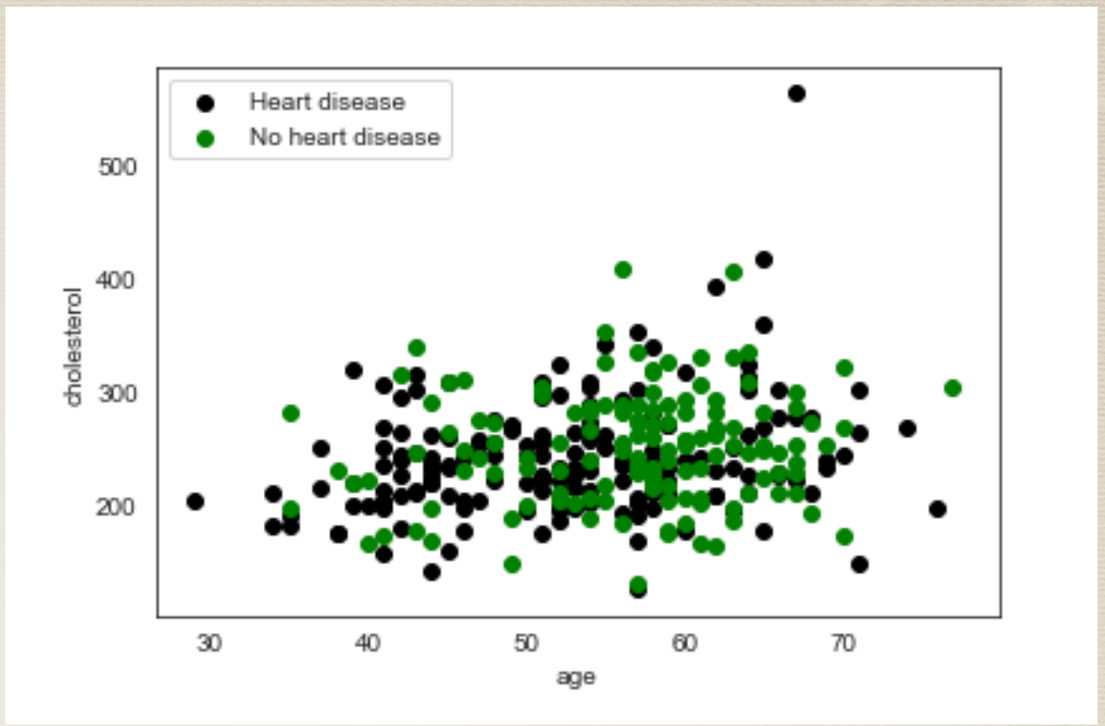
```
#baseline 165/(165+138) - if predict all ones
heart = df["target"]
sns.countplot(heart)
heart_temp = df.target.value_counts()
print(heart_temp)
```

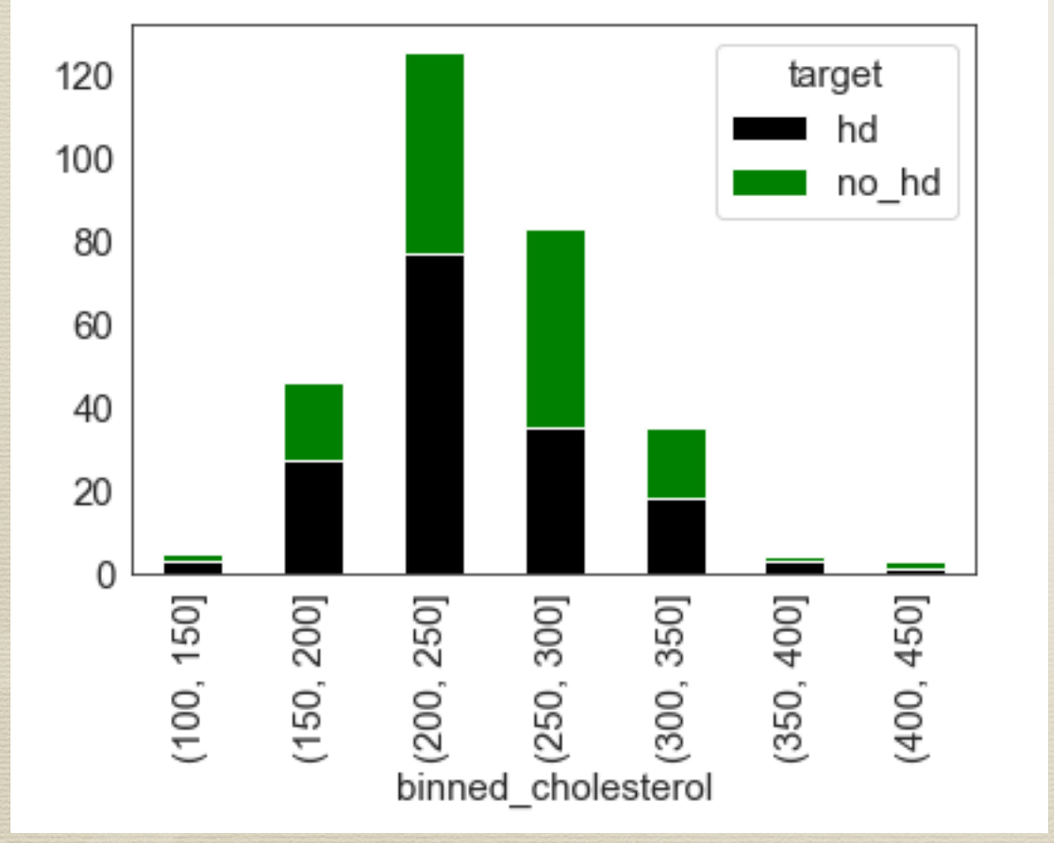
```
hd       165
no_hd    138
Name: target, dtype: int64
```

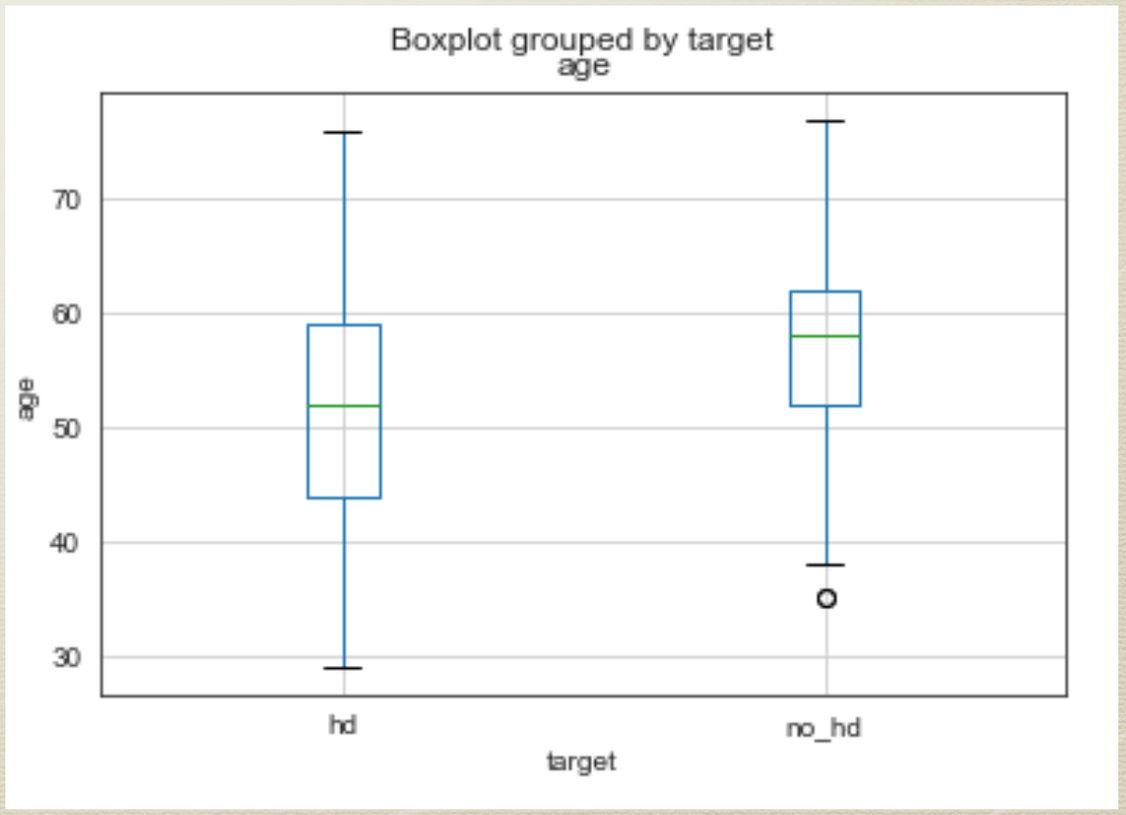


Age and Cholesterol

- The average age for heart disease is lower than those who don't have the disease.
- Outliers are minimal.
- Cholesterol between 200 and 250 have the highest amount of heart disease patients.







Description of the data

From the distribution shown:

- Insignificant outliners
- no null values

```
# Checking outliers at 25%,50%,75%,90%,95% and 99%
numerical_df.describe(percentiles=[.25,.5,.75,.90,.95,.99])
```

	age	resting_bp	cholesterol	max_heart_rate	oldpeak
count	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	131.623762	246.264026	149.646865	1.039604
std	9.082101	17.538143	51.830751	22.905161	1.161075
min	29.000000	94.000000	126.000000	71.000000	0.000000
25%	47.500000	120.000000	211.000000	133.500000	0.000000
50%	55.000000	130.000000	240.000000	153.000000	0.800000
75%	61.000000	140.000000	274.500000	166.000000	1.600000
90%	66.000000	152.000000	308.800000	176.600000	2.800000
95%	68.000000	160.000000	326.900000	181.900000	3.400000
99%	71.000000	180.000000	406.740000	191.960000	4.200000
max	77.000000	200.000000	564.000000	202.000000	6.200000

VIF to check for multicolinerity

VIF is low which is an indication of no correlation.

```
##Lets look at Variance Inflation Factor and check for multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant
X = add_constant(df)
pd.Series([variance_inflation_factor(X.values, i)
                for i in range(X.shape[1])],
               index=X.columns)
```

```
const                   212.998773
age                       1.443937
sex                       1.231356
chest_pain                1.397152
resting_bp                1.180747
cholesterol               1.152971
fasting_blood_sugar       1.087698
resting_ecg               1.066721
max_heart_rate            1.653567
exercise_angina           1.440147
oldpeak                   1.744666
slope                     1.662325
num_major_vessels         1.290729
thalium_stress_test       1.191528
target                    2.072754
dtype: float64
```

Categorical variables were dummied

Chest Pain Type
- 0: Typical angina
- 1: Atypical angina
- 2: Non-anginal pain
- 3: Asymptomatic

Resting electrocardiographic results
- 0: Nothing to note
- 1: ST-T Wave abnormality Can range from mild symptoms to severe problems
  Signals non-normal heart beat
- 2: Possible or definite Left ventricular hypertrophy Enlarged heart's main
  pumping chamber

slope: the slope of the peak exercise ST segment
- 0: upsloping
- 1: flatsloping
- 2: downsloping

Thalium stress test result - Sees how blood moves through your heart while exercising
- 1: Normal
- 6: fixed defect
- 7; reversible defect

Summary of the data

* Variables are not correlated.
* There were no null values.
* Data was scaled due to the different parameters.
* Dummy variables were created for categorical data.
* The target variable was balanced.
* A 3 fold cross validation was used for all.

Plan

- The objective is to create the best model by increasing the accuracy and the area under the curve scores.
- A classification model will be used to determine which class a patient belongs.
- First we will calculate the baseline score.
- Second we will tune the model to get a better baseline score.
- Third we will compare the scores of each algorithm to see which has the best score.
- Finally we will determine the best algorithm for determining heart disease.

Base Models

Naive Bayes

KNN Classifier

Logistic Regression

Support Vector Machine

Random Forest

XGBoost

Model Tuning

Four models were tuned to see if the score would change for the better. Grid-search was used to find the best combination of parameters with a 3 fold cross validation to yield the best accuracy score.

* KNN Classifier
* Random Forest
* Support Vector Machines
* Extreme Gradient boosting

KNN tuning: n_neighbors

A range of n_neighbors were looped through the classifier and 42 produced the best score ~ 0.8350.

```
Num Neighbors 42 --> Average Accuracy:    0.8350
Num Neighbors 43 --> Average Accuracy:    0.8317
Num Neighbors 44 --> Average Accuracy:    0.8284
Num Neighbors 45 --> Average Accuracy:    0.8317
Num Neighbors 46 --> Average Accuracy:    0.8383
Num Neighbors 47 --> Average Accuracy:    0.8350
Num Neighbors 48 --> Average Accuracy:    0.8317
Num Neighbors 49 --> Average Accuracy:    0.8317
```

```
#Adding the best n_neighbors value of 42
knn = KNeighborsClassifier(n_neighbors=42)

knn_pipeline = make_pipeline(preprocessing.StandardScaler(), knn)
res = cross_val_score(knn_pipeline, X, y, cv=3)
print("n_neighbors of 42 has an Average Accuracy score: {0:.4f}".format(np.mean(res)))
```

```
n_neighbors of 42 has an Average Accuracy score: 0.8350
```

Support Vector Machines tuning

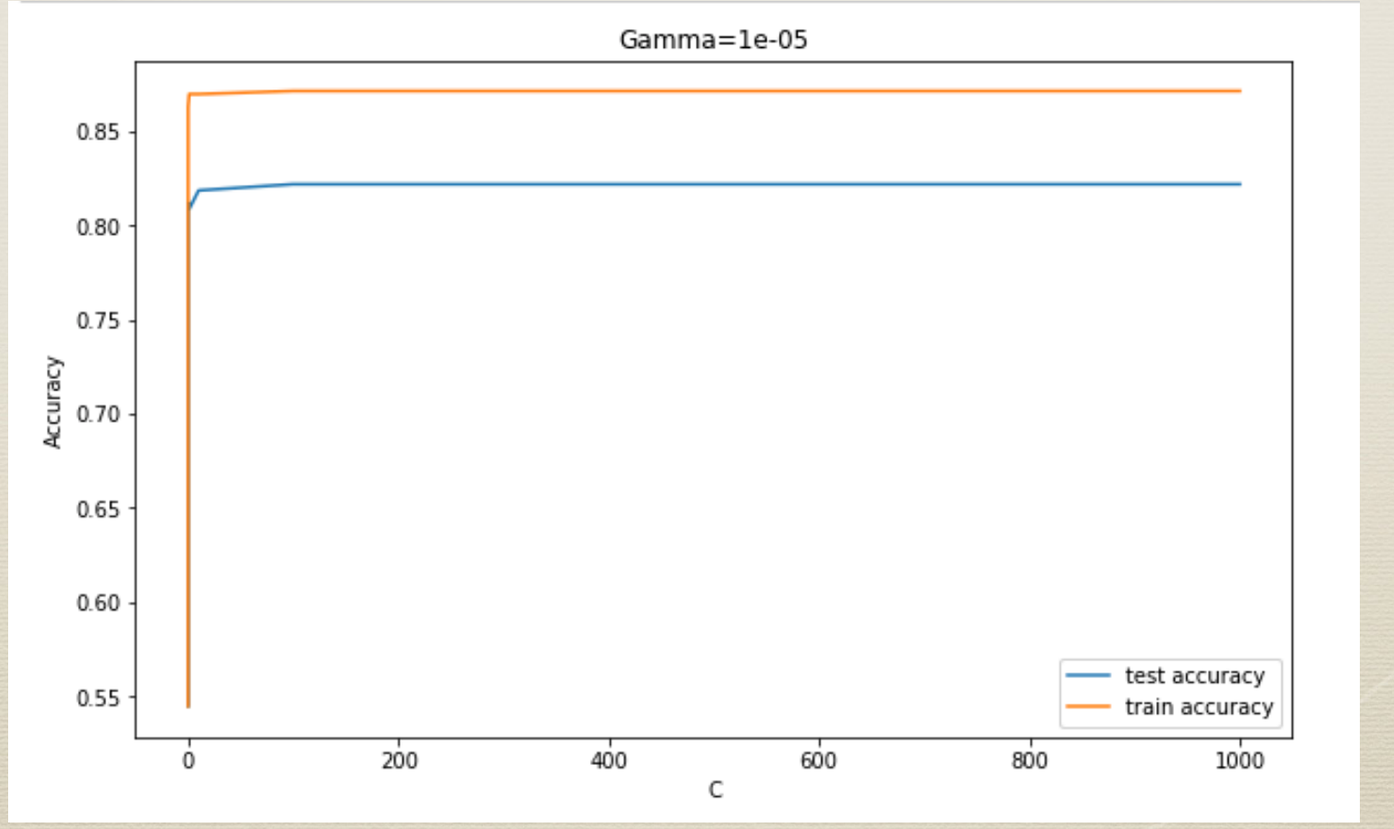
1. gamma
2. C

```
params = {"SVM__C": [0.0001, 0.001, 0.1, 1, 10, 100, 1000],
          "SVM__gamma" :[0.00001,0.0001, 0.001, 0.1, 1, 10, 100]}

folds = KFold(n_splits = 3, shuffle = True, random_state = 99)
```

Results

```
The highest accuracy score is 0.8218  {'SVM__C': 100, 'SVM__gamma': 1e-05}
```



Random Forest tuning

1. max depth
2. minimum samples leaf
3. minimun samples split
4. n_estimators
5. max features

```
rf_final = RandomForestClassifier(bootstrap=True,
                                  max_depth=2,
                                  min_samples_leaf=30,
                                  min_samples_split=5,
                                  max_features=4,
                                  n_estimators=5)
```

Results

```
The best RFC accuracy score is 0.8449 and has the parameters
 {'max_depth': 2, 'max_features': 4, 'min_samples_leaf': 30, 'min_samples_split': 5, 'n_estimators': 5}
```

AdaBoostClassifier was also used to boost Random Forest but the final results didn't increase the score. There wasn't an improvement.

Extreme Gradient Boost tuning

1. max depth
2. learning rate
3. column sample by tree
4. column sample by level
5. n_estimators
6. subsample

```
xgb_param_grid = {
 'max_depth':[2],
 'n_estimators':[30],
 'learning_rate':[0.3],
 'subsample':[0.3],
 'colsample_bytree':[0.15],
 'colsample_bylevel':[0.0001],}
xgb_final = XGBClassifier()
```

Results

```
The best XGB accuracy score is 0.8482 with parameters
 {'colsample_bylevel': 0.0001, 'colsample_bytree': 0.15, 'learning_rate': 0.3, 'max_depth': 2, 'n_estimators': 30, 's
ubsample': 0.3}
```

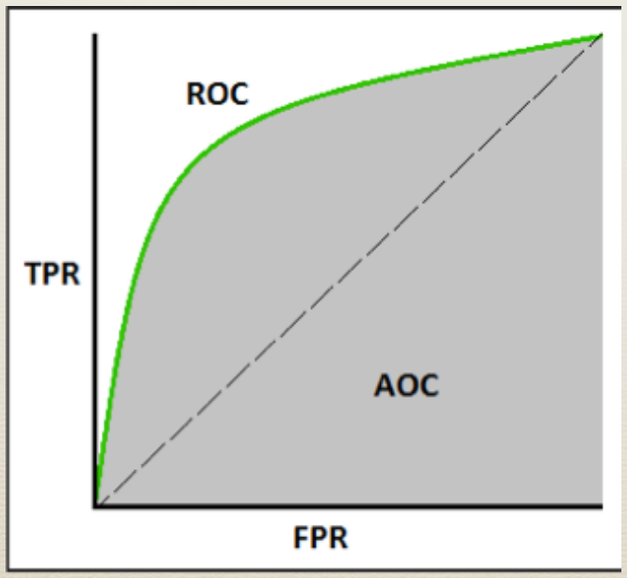
Model Comparison:

The metrics used to evaluate the models are the accuracy score and ROC/AUC.

AUC is the area under the ROC Curve . ROC Curve (receiver operating characteristic) shows the performance of the classification model at all classification thresholds.

- True Positive Rate = TP/(TP+FN)
- False Positive Rate = FP/(FP+TN)

Our goal is to not tell a person they have heart disease when they don't and that they don't have heart disease when they actually have it. So we need rates to be low.

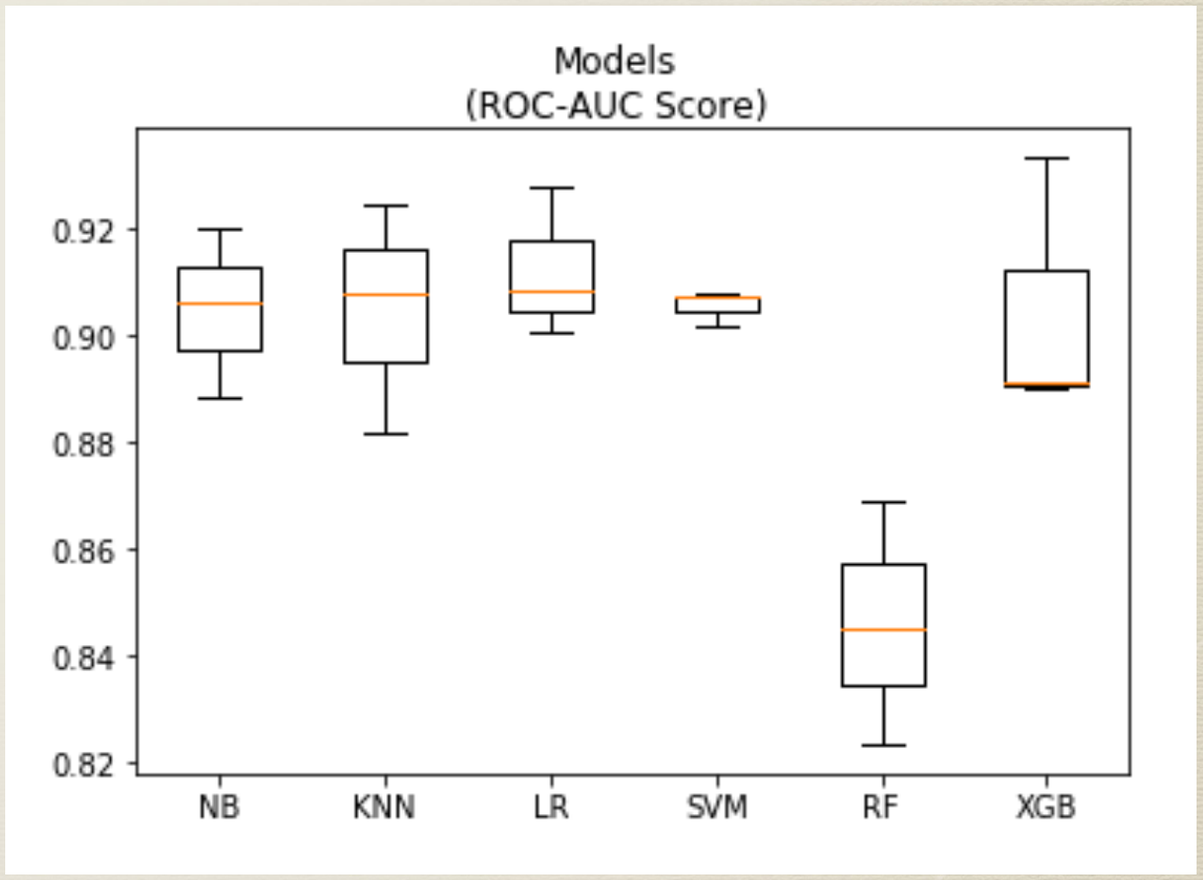


ROC/AUC Model Comparison

All the tuned models were put into a new jupyter notebook and compared side by side.

- LR has the best score
- SVM is the most stable
- 3 fold evaluation

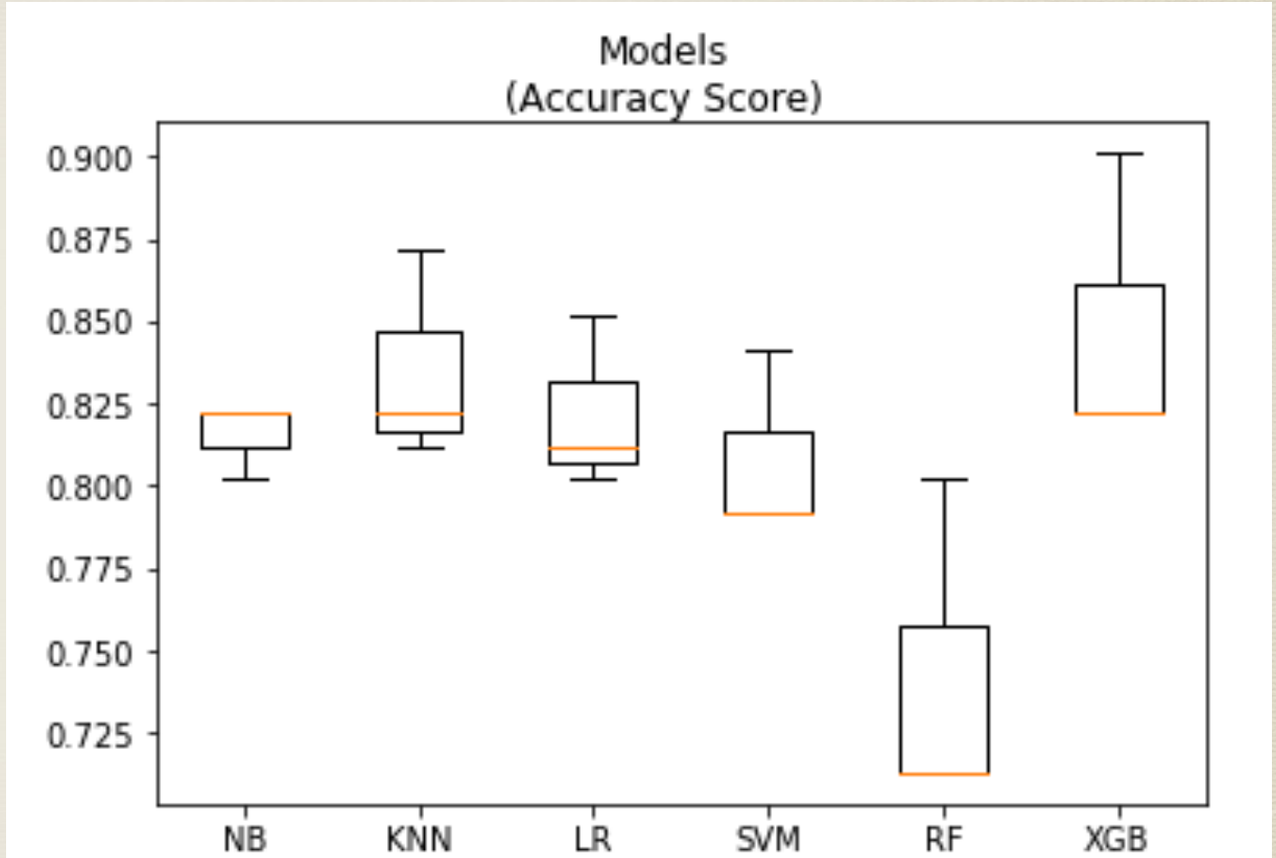
	ROC/AUC	
	score	std
NB:	0.9047	(0.0129)
KNN:	0.9047	(0.0176)
LR:	0.9121	(0.0115)
SVM:	0.9057	(0.0026)
RF:	0.8460	(0.0186)
XGB:	0.9047	(0.0201)



Accuracy Score Model Comparison

- XGB has the best score
- NB is the most stable
- 3 fold evaluation

	Accuracy	
	score	std
NB:	0.8152	(0.0093)
KNN:	0.8350	(0.0260)
LR:	0.8218	(0.0214)
SVM:	0.8086	(0.0233)
RF:	0.7426	(0.0420)
XGB:	0.8482	(0.0373)



The Best Model

XGB is the winning model
tuned average score was 85%
tuned roc/auc score 90%

```
best_model = XGBClassifier(bootstrap=True,
                           subsample= 0.3,
                           n_estimators=30,
                           max_depth=2,
                           learning_rate=0.3,
                           colsample_bytree=0.15,
                           colsample_bylevel=0.0001)
```

XGB evaluation

* 85.52% of predictions were correct - TP/(TP + FP)
* 87.88% of the positive cases were caught - TP/(TP + FN)

```
          precision
          score    stdev
XGB:    0.8552 (0.0735)

           recall
           score    stdev
XGB:    0.8788 (0.0171)
```

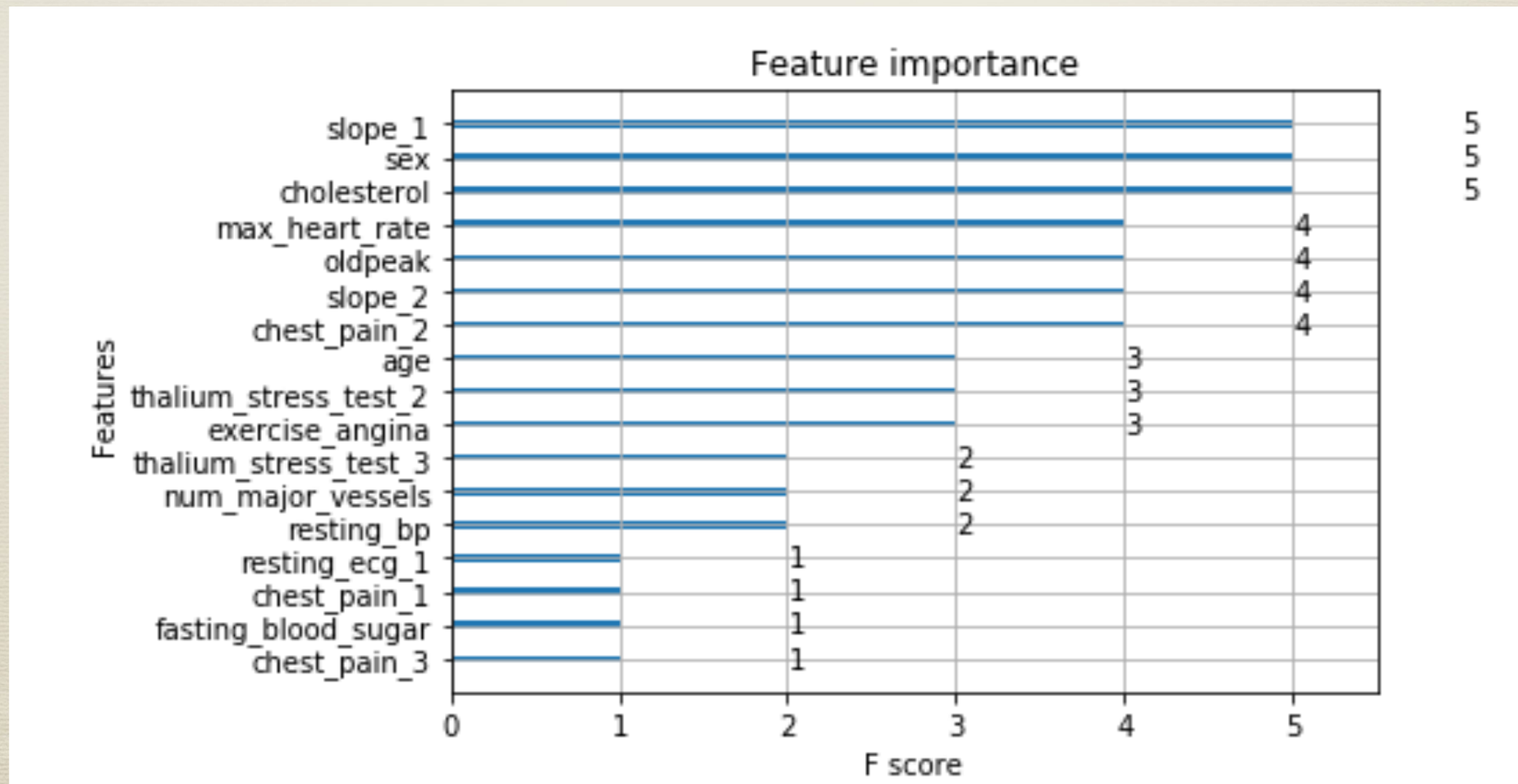
- Precision(positive predictive value) TP/(TP + FP) - What percent of positive predictions were correct? 85.52% TP/(TP + FP)
- Recall(sensitivity) TP/(TP + FN) - What percent of the positive cases did you catch? 87.88%

```
#F1score= 2 * (precision*recall) / (precision+recall)
Fmeasure = 2*(0.8552 * 0.8788) / (0.8552 + 0.8788)
print('F1score =  {0:.2%}'.format(Fmeasure))
```

```
F1score =  86.68%
```

Feature Importance

The most significant features predicted for heart disease using the extreme gradient boost model are seen below.



Summary

	ROC/AUC	
	score	std
NB:	0.9047	(0.0129)
KNN:	0.9047	(0.0176)
LR:	0.9123	(0.0114)
SVM:	0.9057	(0.0026)
RF:	0.8451	(0.0209)
XGB:	0.9047	(0.0201)

	Accuracy	
	score	std
NB:	0.8152	(0.0093)
KNN:	0.8350	(0.0260)
LR:	0.8218	(0.0214)
SVM:	0.8086	(0.0233)
RF:	0.7261	(0.0337)
XGB:	0.8482	(0.0373)

* Naive Bayes scored good in ROC/AUC and wasn't that far behind in accuracy from XGB.
* K-Nearest Neighbor scored good in ROC/AUC and wasn't that far behind in accuracy from XGB. Based on distances between data point and not on errors like XGB.
* Logistic Regression decides on a particular cutoff score for the probabilities and the cutoff boundary will always be linear and therefore misclassifying some data points.
* Support Vector Machine did ok. It score ok mostlikely due to the fact that it looks at the distances between the data points.
* XGBoost has more parameters to tune, like sub sample (helps with overfitting) and learning rate (reduces the size of the previous loss function gradient). It models on the errors in steps and trees are formed sequentially then combined to form a result.

Some short comings

- Maybe include new features for a better prediction such as family history, inflammation and blood sugar for diabetics, as the Reynolds Risk Score* for heart disease suggests.
- Check whether any of the original 76 attributes would be beneficial to add back.
- A larger dataset or a combined dataset from several states and not just Cleveland.

*http://www.reynoldsriskscore.org/