XgBoost Algorithm

Coefficient Crusaders: Shane Sarabdial, Noelle Kiesz, Kendra Johnson, Miko Le

TABLE OF CONTENTS

1. INTRO TO XGBOOST
   What it is and how it works?

2. DATASET 1: CARS93
   Predict Car Prices

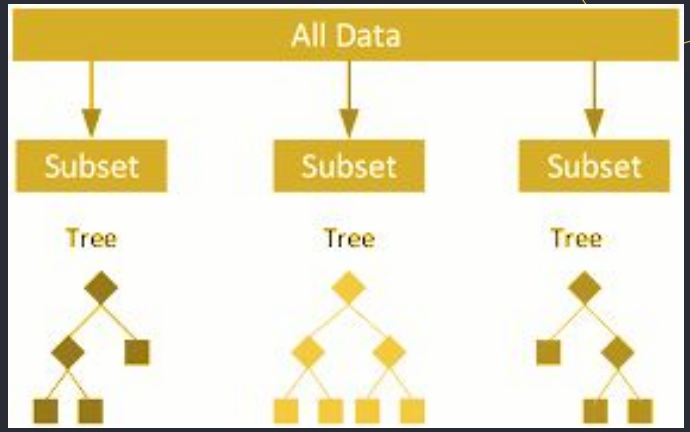
3. DATASET 2: POKEMON
   Determine Legendary Pokemon

4. DATASET 3: HEART DISEASE
   Heart Disease Classification

1

What is XGBoost?

- ★ XGBoost: Gradient boosting library with decisions tree as a base model
  - Commonly used for classification and regression
- ★ How it Works: sequences of decision trees and correct errors made by previous trees
- ★ Target dataset: Categorical and Continuous variables
- ★ Hyperparameters:
  - Learning rate
  - The number of trees
  - Maximum depth of each tree



ADVANTAGES

- Speed and Accuracy
- Ability to handle missing values
- Prevent Overfitting
  - Regularization
  - Sampling
  - Pruning
  - Early Stopping

DISADVANTAGES

- Not ideal with outliers
- Manually create dummy variable
- Requires careful parameter tuning

XGBOOST vs LINEAR REGRESSION

- XGBoost is more complex, better with big datasets
- Linear Regression is easier to implement and faster to train, more fitted with small datasets.

Scoring Metris

1. Mean Squared Error: Takes the squared distance between the actual and predicted. The closer the MSE is to 0 the better our model is.

2. ROC Curve: The ratio between true positive rate(TPR) and false positive rate (FPR). TPR/FPR is optimal when TRP is 1 and FRP is 0 resulting in a vertical line.

2

Car Price Prediction with XGBoost

Dataset Characteristics

Source

The data was sourced from kaggle.

Size

The data 92 rows and 27 columns

Data Types

The data has a combination of categorical and numeric values

Features

1. Manufacturer
2. Model
3. Type
4. Minimum Price
5. Price
6. Max Price
7. MPG City
8. MPG HighWay
9. AirBags
10. Drive Train
11. Cylinders
12. Engine Size
13. Horsepower
14. RPM
15. Rev per mile
16. Manual Transition
17. Fuel Tank Capacity
18. Passengers
19. Length
20. Wheelbase
21. Width
22. Turn Circle
23. Rear seat room
24. Luggage room
25. Weight
26. Origin
27. Make

Data preprocessing

STAGE 1

Converted columns to proper data types.
Checked for nulls and outliers.

STAGE 2

Dropped Luggage and Rear room columns.

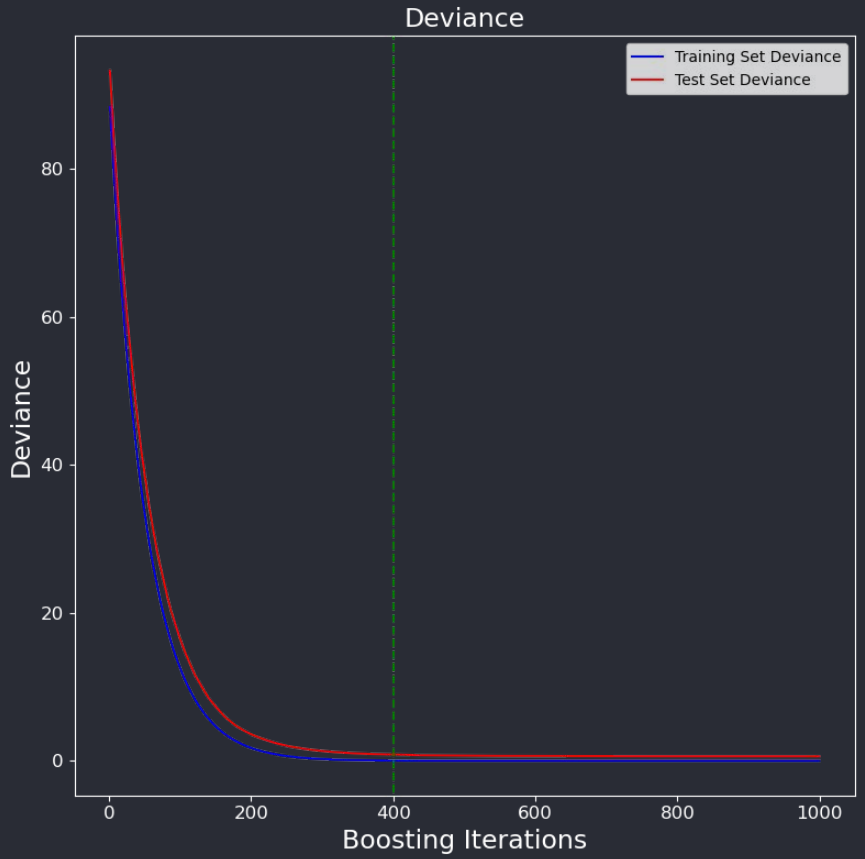
STAGE 3

Create dummy variables for categorical columns (8 columns).

STAGE 4

Split data into train and test.

Testing

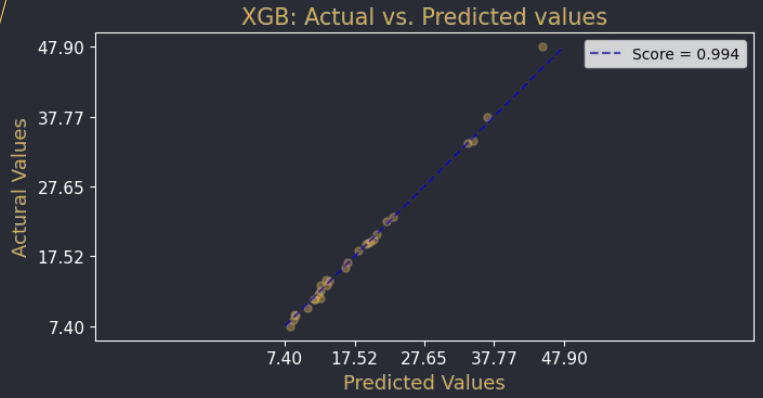


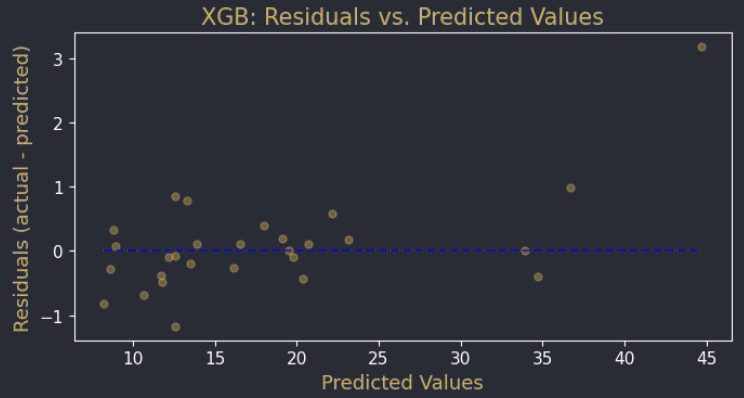

```
params = {
    "n_estimators": 1000,
    "max_depth": 4,
    "min_samples_split": 10,
    "learning_rate": 0.01,
    "loss": "squared_error",
}
```

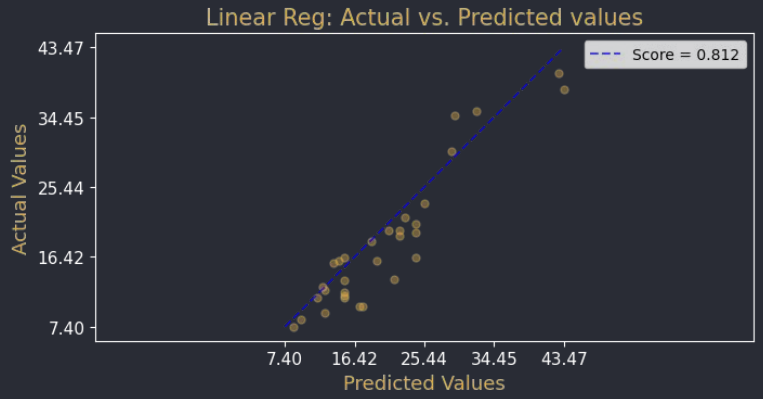
Our model is not overfitted and performs slightly worse than the training MSE. However, after 400 iterations we get marginal improvements in score and it's not worth the extra processing power.

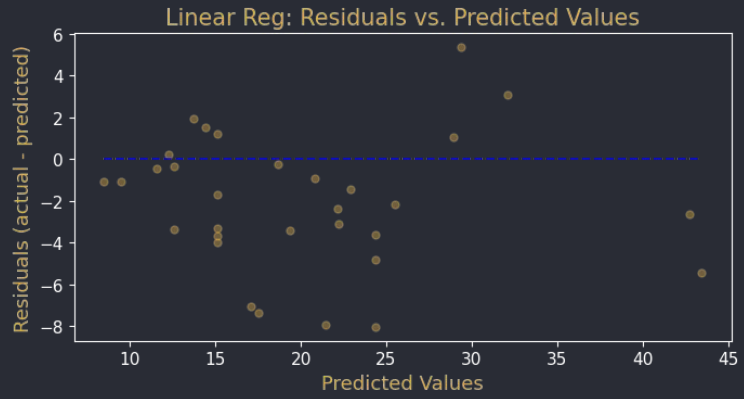
XGB compared to Linear Regression

Plotting cross-validated predictions









3

Pokemon Classification with XGBoost

Dataset Characteristics

Source

The data was sourced from Kaggle

Size

The data was 800 rows and 13 columns

Data Types

The data has a combination of categorical and numeric values and a Boolean

Features

1. Name
2. Type 1
3. Type 2
4. Total
5. HP
6. Attack
7. Defense
8. Sp. Attack
9. Sp. Defense
10. Speed
11. Generation
12. Legendary

Data Processing

STAGE 1

Check for data types and nulls.

STAGE 2

Replacing Nulls with ‘None’

STAGE 3

Creating Dummy Variables for Type 1 and Type 2

STAGE 4

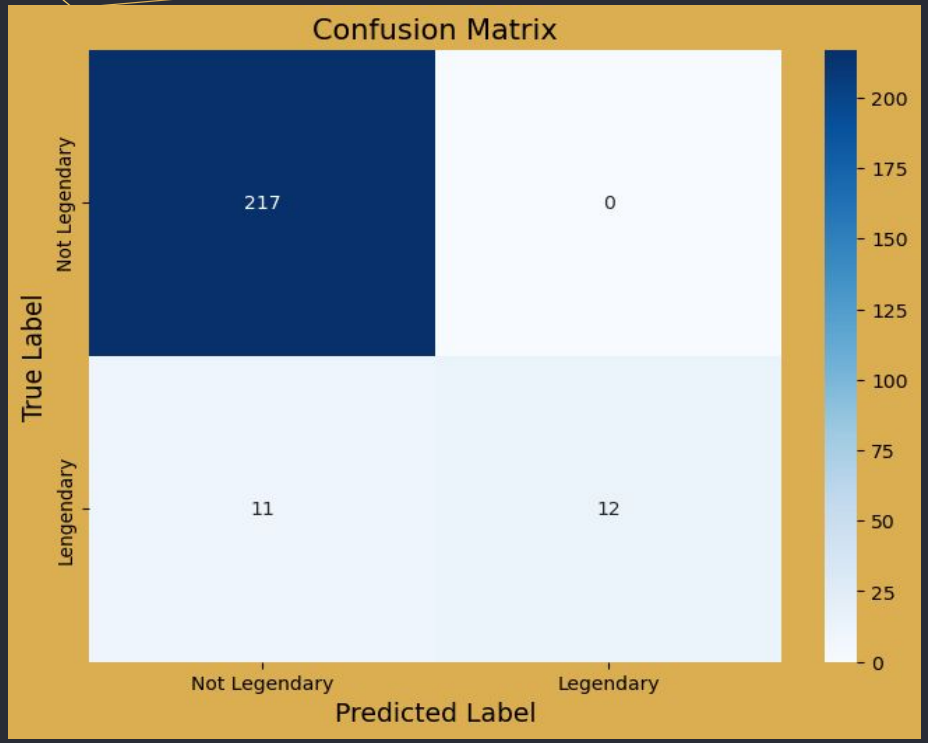
Split into test and train data

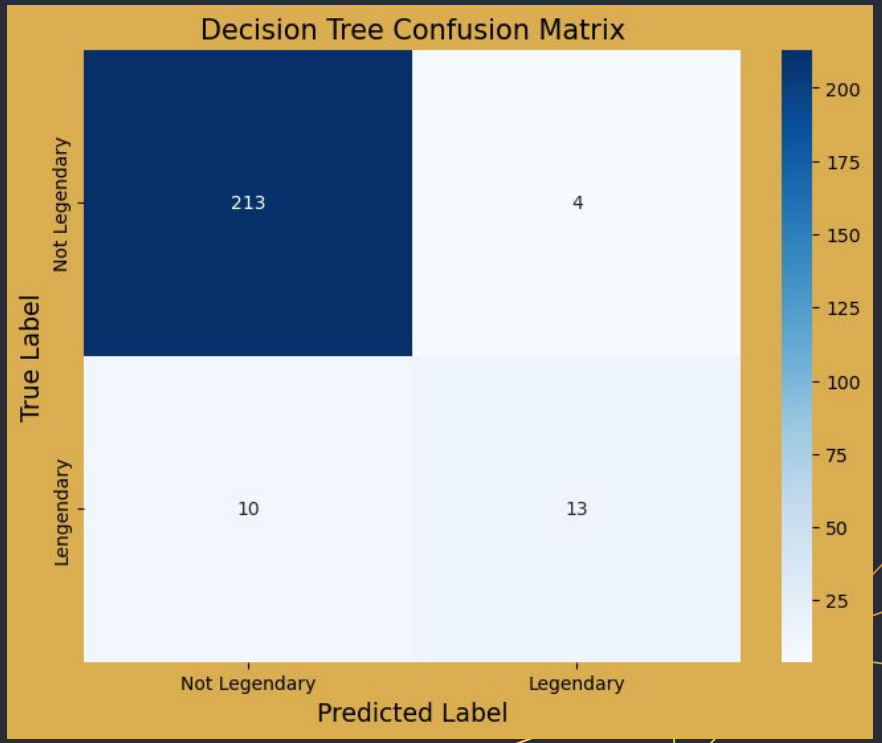
Data and Accuracy

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False

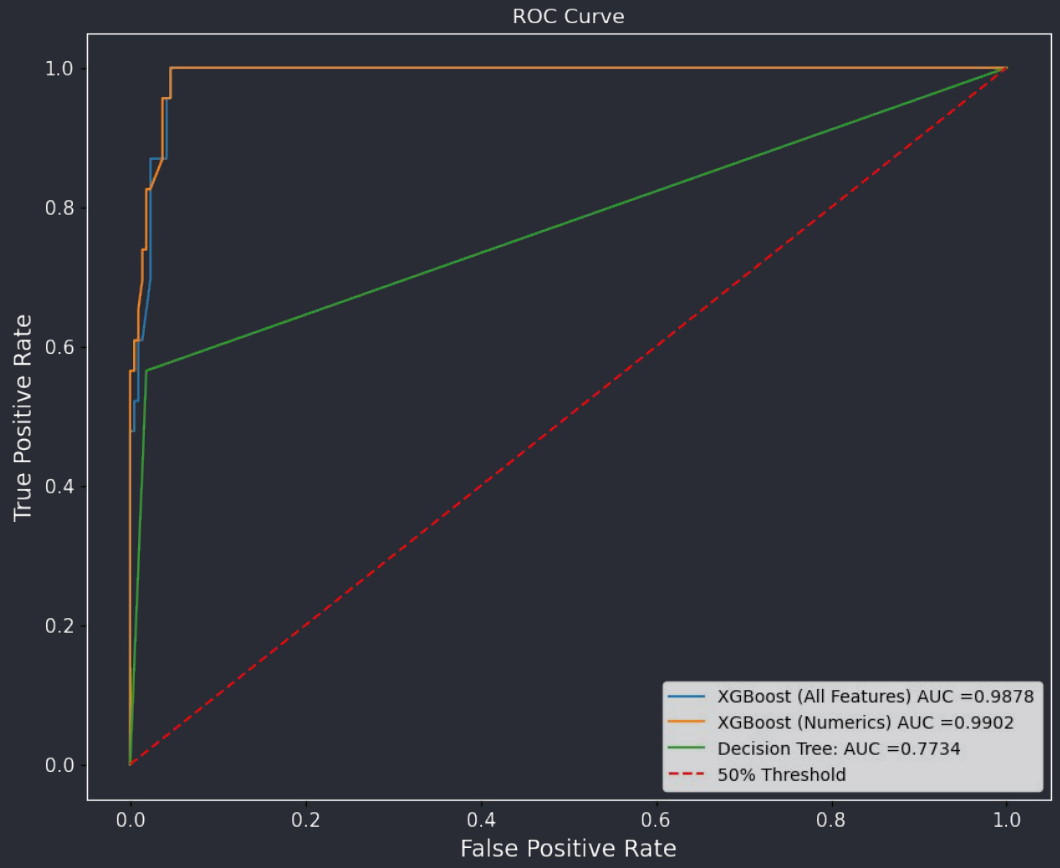
```
Accuracy for Gradient Boosting for all Features: 0.9417
Accuracy for Gradient Boosting with only Numeric Features: 0.9542
```

Confusion Matrix Compared to Decision Tree





Testing



Both models trained by XGBoost perform similarly.

4

Heart Disease Classification with XGBoost

Dataset Characteristics

Source

The data was sourced from kaggle.

Balance

The data is imbalanced with 292,422 negatives vs 27,373 positive cases

Size

The data has 311,936 rows and 18 columns

Data Types

The data has a combination of categorical and numeric values

Features

1. Heart Disease
2. BMI
3. Smoking
4. Alcohol Drinking
5. Stroke
6. Physical Health
7. Mental Health
8. Driff Walking
9. Sex
10. Age Group
11. Race
12. Diabetic
13. Physical Activity
14. General Health
15. Sleep Time
16. Asthma
17. Kidney Disease
18. Skin Cancer

Data preprocessing

STAGE 1

Check for data types and nulls.

STAGE 2

Removed outliers that were 3 standard deviations away in BMI and Sleep Time columns.

STAGE 3

Create dummy variables for categorical columns (14 columns).

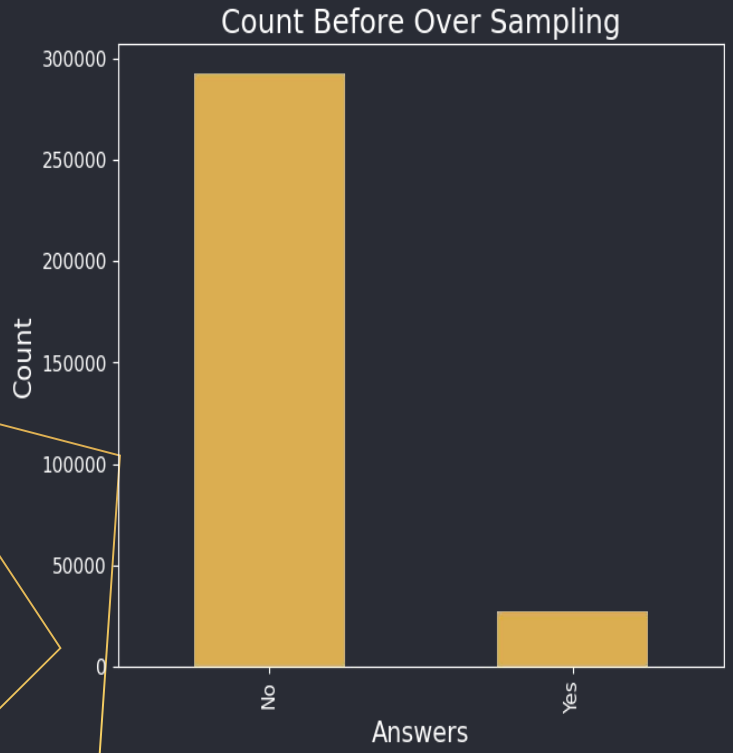
STAGE 4

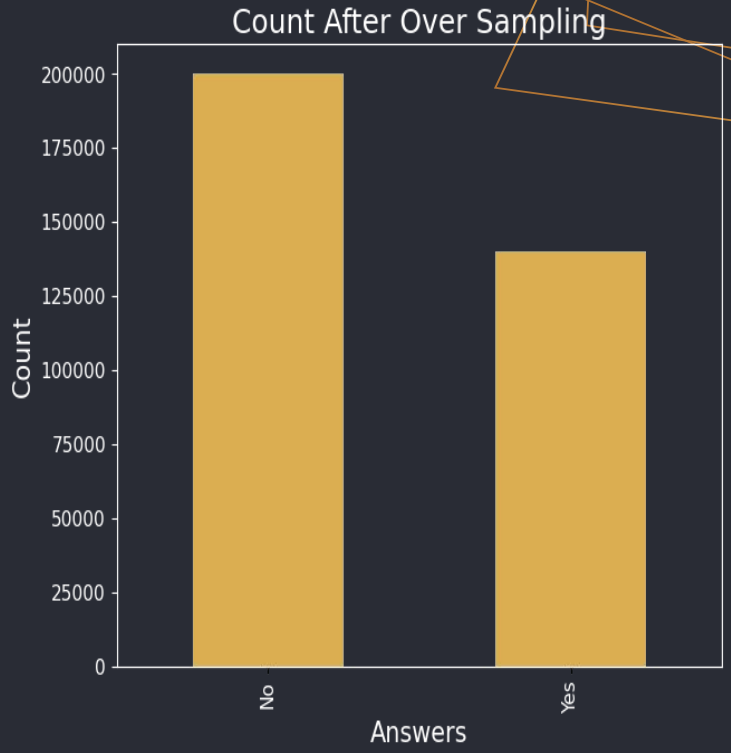
Split data into train and test.

STAGE 5

Rebalanced train data with a Random Over Sampler from imbalance learn library.

Over Sampling





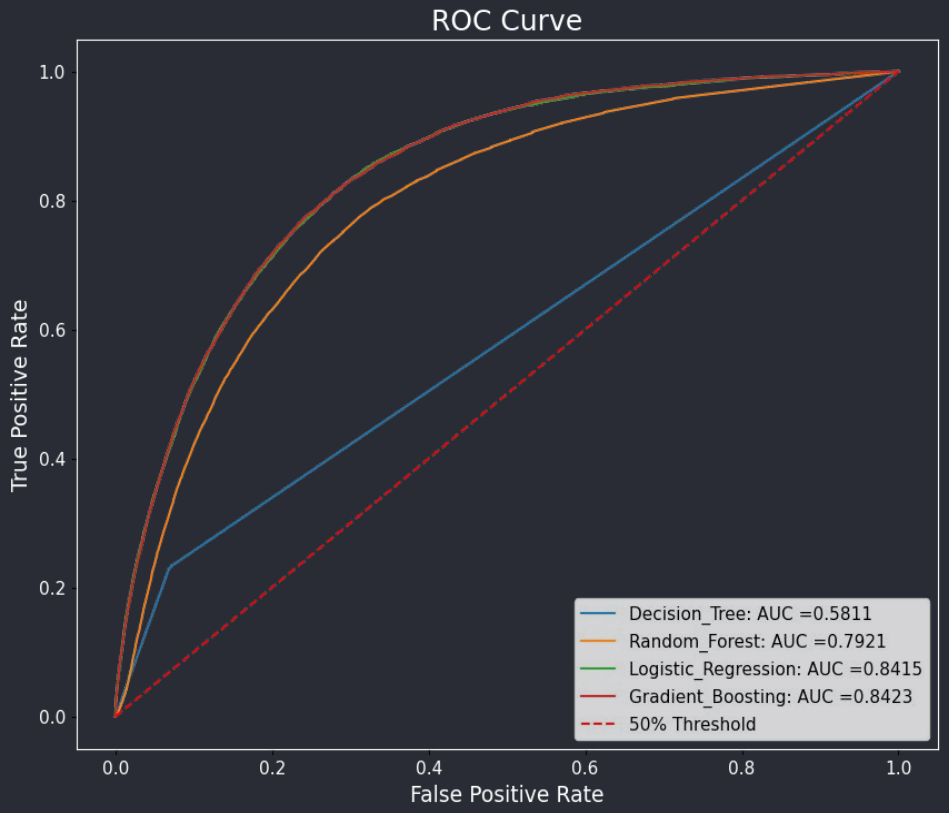
GridSearchCV

```
algos = {
    'Decision_Tree': {
        'model': DecisionTreeClassifier(),
        'params': {
            'splitter': ['best'],
            'max_features': ['sqrt','log2'],
            'random_state': [42]
        }
    },
    'Random_Forest' : {
        'model' : RandomForestClassifier(),
        'params' : {
            'n_estimators': [150,250],
            'random_state': [42],
            'max_depth': [30,50],
            'max_features': ['sqrt'],
            'min_samples_leaf': [1,2],
            'max_leaf_nodes': [None,8,16]
        }
    },
    'Logistic_Regression' :{
        'model' : LogisticRegression(),
        'params' : {
            'max_iter': [100,200],
            'random_state': [42],

        }
    },
    'Gradient_Boosting' :{
        'model' : GradientBoostingClassifier(),
        'params' : {
            'learning_rate': [.1,.25,],
            'n_estimators': [600],
            'random_state': [42],
            'n_iter_no_change': [10],
            'max_features': ['sqrt',],
        }
    }

}
```

```
scores = []
models ={}
for algo_name, config in algos.items():
    gs =  GridSearchCV(config['model'], config['params'], cv=3,
          return_train_score=False, n_jobs=-1, scoring='roc_auc',)
    gs.fit(X_train_rs,y_train_rs)
    scores.append({
        'model': algo_name,
        'best_score': gs.best_score_,
        'best_params': gs.best_params_
    })
    models[algo_name] = gs
pd.DataFrame(scores,columns=['model','best_score','best_params'])
```

	model	best_score	best_params
0	Decision_Tree	0.951680	{'max_features': 'sqrt', 'random_state'...
1	Random_Forest	0.996306	{'max_depth': 50, 'max_features': 'sqrt...
2	Logistic_Regression	0.839347	{'max_iter': 200, 'random_state': 42}
3	Gradient_Boosting	0.848736	{'learning_rate': 0.25, 'max_features':...

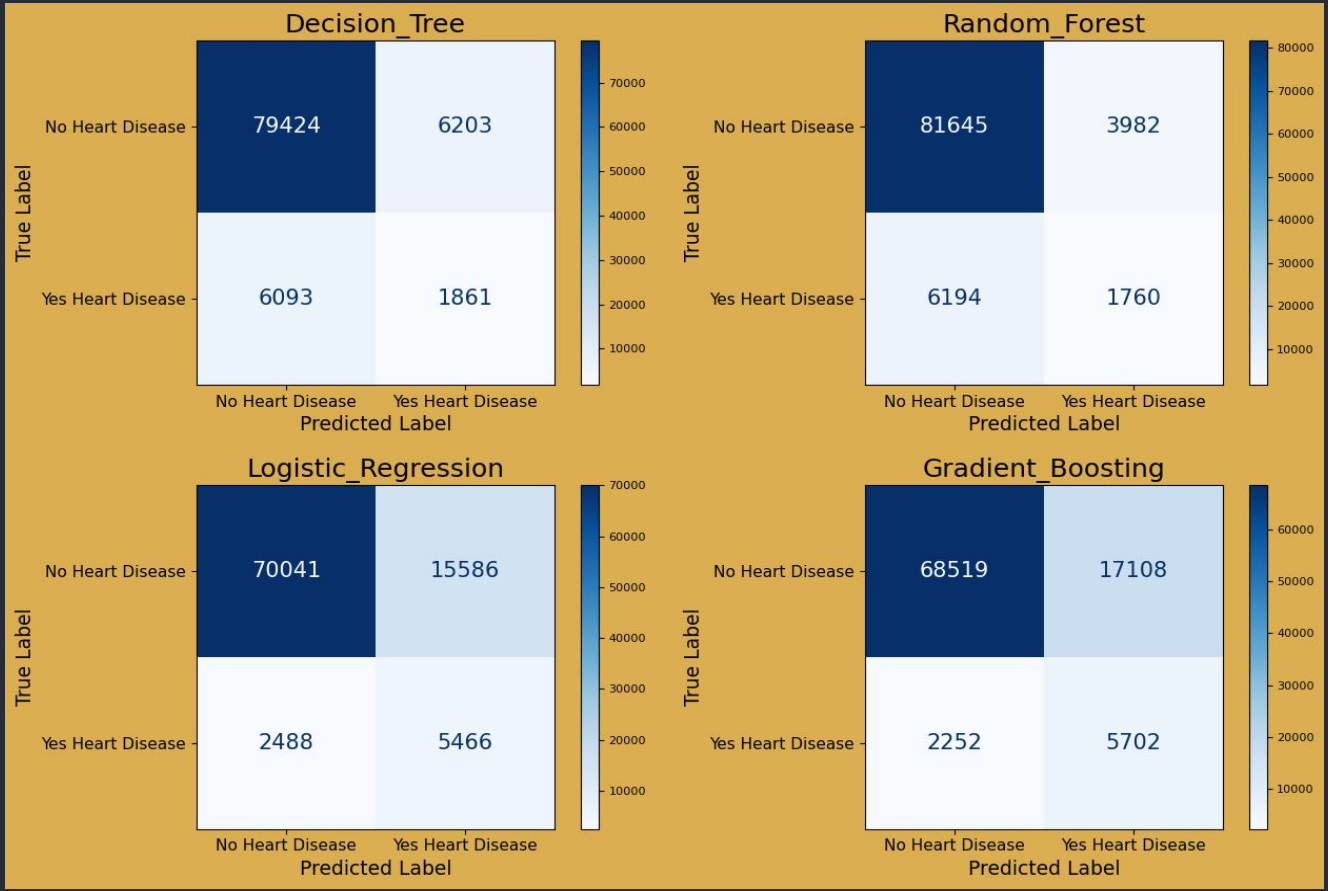
Testing



Although Decision Tree and Random Forest did well on the training data they tend to overfit. Using the test set we can see Logistic Regression and and Gradient Boosting got similar score to their training scores.

Model	Train Score	Test Score
Decision Tree	0.951	0.581
Random Forest	0.996	0.792
Logistic Regression	0.839	0.841
Gradient Boosting	0.848	0.842

Confusion Matrix



The Decision Tree and Random Forest have high accuracy meaning they correctly predicted True Positives and True Negatives more. However, they have low precision meaning they have more False Positives and False Negatives. When there is heart disease the Decision Tree and Random Forest predict “No heart disease” more than Logistic Regression and Gradient Boosting. In health care precision is more important than accuracy. There is a trade off between accuracy and precision.

RESOURCES

- Sklearn
- Roc Curve
- Mean Squared Error
- XGBoost
- Parameter Tuning