This paper finds Metric Support Vector Machine (SVM) to be the most accurate model, eclipsing Gradient Boosting Machine (GBM) and the Random Forest Classifier.(GBM)

		Random Forest Classifier	
Accuracy	85%	90%	88%
Precision	82%	89%	85%
Recall 80%	91%	87%	
F1-Score	81%	90%	86%
AUC-ROC	0.88	0.93	0.90

Reflection

Best Performing Model: Gradient Boosting Machine is also called as Gradient Boosting, which is a kind of enrichment model.

GBM emerged as the top-performing model across most evaluation metrics:

Highest Accuracy (90%): Says that GBM was accurate in 90 percent of cases for the type of instances that were tested.

Superior Precision (89%) and Recall (91%): They stimulate that as far as the false positive and false negative ratios are concerned, GBM provides satisfactory positive outcomes with less errors.

Best F1-Score (90%): Showed good accuracy in recall as well as good recall of accuracy Hence.

Highest AUC-ROC (0.93): Able to favour one class over another class Paradigm reflects discriminative ability to distinguish between classes.

As the results are quite compelling across all measures considered for evaluation, it is apparent that GBM is a better solution for this prediction of heart diseases.

Why GBM Performed Best

Several factors contribute to GBM's superior performance:

Sequential Learning: The chief innovation of the GBM is that it is a step by step process where after creating one tree, the next tree takes into account the errors of the previous trees. This iteration proves to improve the model.

Handling of Complex Patterns: In contrast to other algorithms GBM widely used to recognize correlation and interactions of features and it is valuable for medical data cause they are interconnected.

Robustness to Overfitting: The last advantage of GBM is that it doesn't overfit and if it does overfit, then there are ways you can control this by for instance set the tree limit or the learning rate.

The Effect of Hyperparameter Tuning

Hyperparameter tuning played a pivotal role in optimizing each model's performance:

SVM:

Kernel Selection: Selection of the right kind of kernel (like RBF or the linear kernel) defined the degree of flexibility of the decision boundary.

Regularization Parameter (C): This led to a small amount of overfitting and underfitting as there was a good and healthy balance between bias and variance.

GBM:

Learning Rate and Number of Estimators: Tuning of these parameters was done in such a way as to allow the model to learn effectively at the same time avoid overfitting.

Tree Depth: Bounded tree size assisted in achieving the objective of learning crucial patterns while also generalizing them.

Random Forest:

Number of Trees (n_estimators): While the number of trees in general enhanced performance up to a certain extent,

Maximum Depth and Features: Optimizing these parameters improved the capacity of resolving feature interactions while averting increased model complexity.

Overall Impact:

Enhanced Performance: Tuning the parameters showed huge increases in all measure of evaluation for all the models.

Balanced Models: Optimization of hyperparameters helped to retain low bias and variance resulting in improved models generalization capacity.

Efficiency: Decreased specificity of parameters lowered complexity by minimizing the inclusion of unhelpful features that affect the time required for training to achieve accurate results and prediction.

Conclusion

All three models provided satisfactory results in heart disease prediction GBDM, however, outperformed all. Optimizing models with the strategic hyperparameter tuning was able to bring out the best of each model showing the significance of carefully fine-tuning models in machine learning processes.

mogutalavignesh/week-5-class-activity (github.com)

```python
# Load necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split

# Load the dataset
file_path = 'Downloads/heart disease.csv'
df = pd.read_csv(file_path, encoding='UTF-8-SIG')

# Display the head of the dataframe to understand its structure
print(df.head())
print(df.info())
# Split the data into training and test sets
X = df.drop('target', axis=1)
y = df['target']

# 80% training and 20% test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

print('Training set size:', X_train.shape)
print('Test set size:', X_test.shape)
```

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

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
<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
None
Training set size: (242, 13)
Test set size: (61, 13)
```