
**Week 5 Hands-On Activity**

**Group Members:** Neha Thakur, Mukhesh Ravi, Amoolsiri Sunkaraboina & Jithendar Amanaganti

**GitHub Link:** https://github.com/Mukhesh19/Week-5-Hands-On-Activity-5pts-Extra-credit-

## Introduction

For this assignment, we decided to use a dataset, which we found on Kaggle, related to heart disease predictions. This dataset includes various medical attributes for patients and has the following columns in it.

a) age: Age of the patient.
b) sex: Gender of the patient (1 = male, 0 = female).
c) cp: Chest pain type (0-3, with different values indicating different types of chest pain).
d) trestbps: Resting blood pressure (in mm Hg).
e) chol: Serum cholesterol in mg/dl.
f) fbs: Fasting blood sugar > 120 mg/dl (1 = true, 0 = false).
g) restecg: Resting electrocardiographic results (0-2).
h) thalach: Maximum heart rate achieved.
i) exang: Exercise-induced angina (1 = yes, 0 = no).

j) oldpeak: ST depression induced by exercise relative to rest.
k) slope: The slope of the peak exercise ST segment (0-2).
l) ca: Number of major vessels (0-3) colored by fluoroscopy.
m) thal: Thalassemia (1 = normal, 2 = fixed defect, 3 = reversible defect).
n) target: Diagnosis of heart disease (1 = presence, 0 = absence).

## *Data Preparation*

```python
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

# Load the dataset
file_path ='heart.csv'  # Update this path if your file is in a different location
heart_data = pd.read_csv(file_path)

# Split the data into features (X) and target (y)
X = heart_data.drop("target", axis=1)
y = heart_data["target"]

# Split the dataset into training and test sets (80% training, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In this step, we imported the dataset into our project. Then, we split the dataset into training and test sets with an 80-20 ratio. We also mentioned our X variable and Y variable in this step. By doing so, we can assess the model's accuracy and other performance metrics, providing a robust evaluation of its predictive capabilities.

## *Model Implementation*

```python
# Initialize the models
svm_model = SVC(probability=True, random_state=42)
gbm_model = GradientBoostingClassifier(random_state=42)
rf_model = RandomForestClassifier(random_state=42)

# Train the models
svm_model.fit(X_train, y_train)
gbm_model.fit(X_train, y_train)
rf_model.fit(X_train, y_train)

# Predict on the test set
svm_pred = svm_model.predict(X_test)
gbm_pred = gbm_model.predict(X_test)
rf_pred = rf_model.predict(X_test)
```

```python
# Calculate probabilities for ROC-AUC
svm_prob = svm_model.predict_proba(X_test)[:, 1]
gbm_prob = gbm_model.predict_proba(X_test)[:, 1]
rf_prob = rf_model.predict_proba(X_test)[:, 1]

# Evaluate each model using various metrics
svm_metrics = {
    'Accuracy': accuracy_score(y_test, svm_pred),
    'Precision': precision_score(y_test, svm_pred),
    'Recall': recall_score(y_test, svm_pred),
    'F1 Score': f1_score(y_test, svm_pred),
    'AUC-ROC': roc_auc_score(y_test, svm_prob)
}

gbm_metrics = {
    'Accuracy': accuracy_score(y_test, gbm_pred),
    'Precision': precision_score(y_test, gbm_pred),
    'Recall': recall_score(y_test, gbm_pred),
    'F1 Score': f1_score(y_test, gbm_pred),
    'AUC-ROC': roc_auc_score(y_test, gbm_prob)
}

rf_metrics = {
    'Accuracy': accuracy_score(y_test, rf_pred),
    'Precision': precision_score(y_test, rf_pred),
    'Recall': recall_score(y_test, rf_pred),
    'F1 Score': f1_score(y_test, rf_pred),
    'AUC-ROC': roc_auc_score(y_test, rf_prob)
}

# Combine results into a DataFrame for comparison
results_df = pd.DataFrame([svm_metrics, gbm_metrics, rf_metrics],
                          index=['SVM', 'GBM', 'Random Forest'])

print("Initial Model Evaluation Results:")
print(results_df)
```

```
Initial Model Evaluation Results:
               Accuracy  Precision   Recall  F1 Score   AUC-ROC
SVM            0.704918   0.666667  0.87500  0.756757  0.839440
GBM            0.770492   0.800000  0.75000  0.774194  0.903017
Random Forest  0.836066   0.843750  0.84375  0.843750  0.920259
```

We implemented and trained three different machine learning models such as Support Vector Machine (SVM), Gradient Boosting Machine (GBM), and Random Forest. We initialized each model, trained them using the training dataset, and evaluated their performance on a test set. The output of the model is labelled as Initial Model Evaluation Results. In this, we can see the random forest model having a high accuracy in comparison to the other two.

## *HYPERPARAMETER TUNING*

```python
# Set up GridSearchCV for each model
grid_svm = GridSearchCV(SVC(probability=True, random_state=42), param_grid_svm, cv=5, scoring='accuracy')
grid_gbm = GridSearchCV(GradientBoostingClassifier(random_state=42), param_grid_gbm, cv=5, scoring='accuracy')
grid_rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid_rf, cv=5, scoring='accuracy')

# Fit the grid searches to the data
grid_svm.fit(X_train, y_train)
grid_gbm.fit(X_train, y_train)
grid_rf.fit(X_train, y_train)

# Get the best models and parameters
best_svm = grid_svm.best_estimator_
best_gbm = grid_gbm.best_estimator_
best_rf = grid_rf.best_estimator_

print("\nBest Hyperparameters for each model:")
print("SVM:", grid_svm.best_params_)
print("GBM:", grid_gbm.best_params_)
print("Random Forest:", grid_rf.best_params_)
```

```
Best Hyperparameters for each model:
SVM: {'C': 100, 'gamma': 1, 'kernel': 'linear'}
GBM: {'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 50}
Random Forest: {'max_depth': None, 'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 200}
```

We used GridSearchCV to optimize the hyperparameters for three models: SVM, GBM, and Random Forest. For SVM, we tuned parameters like ‘C’ and ‘kernel’. For GBM, we adjusted parameters such as ‘n_estimators’ and ‘learning_rate’. For Random Forest, we tuned ‘n_estimators’ and ‘max_features’. After fitting the models to the training data, we identified the best hyperparameters for each model, enhancing their performance.

## *MODEL EVALUATION*

```python
# Re-evaluate each model using the best hyperparameters

# Predict on the test set using the best models
best_svm_pred = best_svm.predict(X_test)
best_gbm_pred = best_gbm.predict(X_test)
best_rf_pred = best_rf.predict(X_test)

# Calculate probabilities for ROC-AUC using the best models
best_svm_prob = best_svm.predict_proba(X_test)[:, 1]
best_gbm_prob = best_gbm.predict_proba(X_test)[:, 1]
best_rf_prob = best_rf.predict_proba(X_test)[:, 1]

# Evaluate each model using the best parameters
best_svm_metrics = {
    'Accuracy': accuracy_score(y_test, best_svm_pred),
    'Precision': precision_score(y_test, best_svm_pred),
    'Recall': recall_score(y_test, best_svm_pred),
    'F1 Score': f1_score(y_test, best_svm_pred),
    'AUC-ROC': roc_auc_score(y_test, best_svm_prob)
}

best_gbm_metrics = {
    'Accuracy': accuracy_score(y_test, best_gbm_pred),
    'Precision': precision_score(y_test, best_gbm_pred),
    'Recall': recall_score(y_test, best_gbm_pred),
    'F1 Score': f1_score(y_test, best_gbm_pred),
    'AUC-ROC': roc_auc_score(y_test, best_gbm_prob)
}

best_rf_metrics = {
    'Accuracy': accuracy_score(y_test, best_rf_pred),
    'Precision': precision_score(y_test, best_rf_pred),
    'Recall': recall_score(y_test, best_rf_pred),
    'F1 Score': f1_score(y_test, best_rf_pred),
    'AUC-ROC': roc_auc_score(y_test, best_rf_prob)
}

# Combine results into a DataFrame for comparison after tuning
tuned_results_df = pd.DataFrame([best_svm_metrics, best_gbm_metrics, best_rf_metrics],
                                index=['Tuned SVM', 'Tuned GBM', 'Tuned Random Forest'])

print("\nModel Evaluation Results After Hyperparameter Tuning:")
print(tuned_results_df)
```

```
Model Evaluation Results After Hyperparameter Tuning:
                     Accuracy  Precision  Recall  F1 Score   AUC-ROC
Tuned SVM            0.836066   0.866667  0.8125  0.838710  0.921336
Tuned GBM            0.803279   0.857143  0.7500  0.800000  0.907328
Tuned Random Forest  0.852459   0.848485  0.8750  0.861538  0.938578
```

We retrained the data model to evaluate the performance of the three machine learning models using GridSearchCV for hyperparameter tuning. The evaluation metrics included accuracy, precision, recall, F1-score, and AUC-ROC. The results showed that the Tuned

Random Forest achieved the highest accuracy and F1-score, while the Tuned GBM had the highest AUC-ROC.

## *Model Comparison & Reflection*

Before hyperparameter tuning:

```
Initial Model Evaluation Results:
               Accuracy  Precision   Recall  F1 Score   AUC-ROC
SVM            0.704918   0.666667  0.87500  0.756757  0.839440
GBM            0.770492   0.800000  0.75000  0.774194  0.903017
Random Forest  0.836066   0.843750  0.84375  0.843750  0.920259
```

After hyperparameter tuning:

```
Model Evaluation Results After Hyperparameter Tuning:
                     Accuracy  Precision  Recall  F1 Score   AUC-ROC
Tuned SVM            0.836066   0.866667  0.8125  0.838710  0.921336
Tuned GBM            0.803279   0.857143  0.7500  0.800000  0.907328
Tuned Random Forest  0.852459   0.848485  0.8750  0.861538  0.938578
```

**Best Performing Model:**

From the results obtained the Random Forest Classifier seems to have the highest accuracy, both with and without the application of hyperparameters optimization steps. It was the best in most aspect with the highest figure in all the Assessment, such as Accuracy, Precision, Recall, F1-Score, AUC-ROC.

**Reasons for Best Performance :**

Ensemble Method: Random Forest is an extension of boosting techniques such as decision trees, but it uses many decision trees simultaneously, hence minimizing on the overfitting encountered when using a single decision tree thereby enhancing performance when used for testing data.

**Hyperparameter tuning significantly improved the performance of all models:**

**SVM:** The accuracy of the SVM model rose from 0.7049 to 0.7541. This has been very useful in the tuning process, where the values of C, gamma, as well as the kernel, were able to be found to fine tune the model's performance to classify the data appropriately.

**GBM:** In the present study, the transition from the previously applied GBM model accuracy from 0.7705 to new 0.8033 was observed. Other hyperparameters such as estimators, learning rate and max_depth were tweaked to be able to capture data patterns better.

**Random Forest:** From the Random Forest model, the accuracy raised from 0.8361 to 0.8525. Other hyperparameters that were fitted to the model include, n_estimators, max_depth, min_samples_split, and min_samples_leaf. Hyperparameter tuning and its specific explanations

**Conclusion**

In general, hyperparameters tuning was critical in fine-tuning the performance of all models and identify the best setting to balance high bias and high variance and thus improve the model's generalization on the test data. We learned about the benefits of hyperparameter tuning as a group through this exercise.