SPECTF Heart Disease Prediction project

Project overview

- Dataset description
- Step-by-step process with explanations
- Code breakdown
- Model evaluation
- Results interpretation
- Recommendations for improvement

SPECTF Heart Disease Prediction: Project Documentation

---

1. Project Overview

The goal of this project is to develop a classification model that can **predict the presence of heart disease** using SPECTF (Single Photon Emission Computed Tomography) data. We compare two supervised learning models:

- **Decision Tree Classifier** (interpretable baseline)
- **Gradient Boosting Classifier** (high-performance ensemble)

---

2. Dataset Summary

Source: UCI Machine Learning Repository – SPECTF dataset

- **Type**: Medical Imaging Data (SPECT)
- **Instances**: 267
- **Features**: 44 binary features (`F1–F44`)
- **Target**: `Diagnosis`
  - `0`: Abnormal (heart disease)
  - `1`: Normal (healthy)

Each row represents a patient's SPECT scan encoded into binary features.

---

3. Process Workflow

```
1. Data Loading & Cleaning
2. Train-Test Splitting
3. Model 1: Decision Tree (with tuning)
4. Model 2: Gradient Boosting (with tuning)
5. Evaluation: Accuracy, Confusion Matrix, AUC
6. Feature Importance Analysis
7. Model Comparison
```

---

4. Detailed Code Explanation

Import Libraries

```
import pandas as pd, numpy as np
import matplotlib.pyplot as plt, seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
roc_auc_score, roc_curve
```

Why: We use `sklearn` for ML models and evaluation, `seaborn/matplotlib` for plots, and `pandas/numpy` for data manipulation.

Load & Prepare Dataset

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/spect/SPECTF.train"
columns = ['Diagnosis'] + [f'F{i}' for i in range(1, 45)]
data = pd.read_csv(url, header=None, names=columns)
data['Diagnosis'] = data['Diagnosis'].map({0: 0, 1: 1})
```

Why: We add meaningful column names, and map the `Diagnosis` into binary labels for classification.

Split Dataset

```
X = data.drop('Diagnosis', axis=1)
y = data['Diagnosis']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42,
stratify=y)
```

Why: `train_test_split` helps us evaluate model generalization. `stratify` preserves class distribution.

Model 1: Decision Tree Classifier

```
dt_params = {
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'criterion': ['gini', 'entropy']
}
dt_grid = GridSearchCV(DecisionTreeClassifier(random_state=42), dt_params, cv=5)
dt_grid.fit(X_train, y_train)
dt_best = dt_grid.best_estimator_
dt_preds = dt_best.predict(X_test)
```

Why: Decision Trees are interpretable and form a strong baseline. Hyperparameter tuning helps reduce overfitting.

Model 2: Gradient Boosting Classifier

```
gb_params = {
    'n_estimators': [50, 100],
    'learning_rate': [0.05, 0.1],
    'max_depth': [3, 5]
}
gb_grid = GridSearchCV(GradientBoostingClassifier(random_state=42), gb_params, cv=5)
gb_grid.fit(X_train, y_train)
gb_best = gb_grid.best_estimator_
gb_preds = gb_best.predict(X_test)
```

Why: Gradient Boosting combines multiple weak learners for higher accuracy. It's robust to noise and often outperforms simpler models.

---

5. Evaluation Function

```
def evaluate(model_name, y_true, y_pred):
    print(f"\n📌 {model_name} Evaluation")
    print("Accuracy:", round(accuracy_score(y_true, y_pred), 3))
    print("Classification Report:\n", classification_report(y_true, y_pred))

    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.title(f"{model_name} - Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.tight_layout()
    plt.show()
```

Why: Provides standardized model metrics and confusion matrix for quick visual diagnosis.

---

6. ROC Curve Comparison

```
dt_probs = dt_best.predict_proba(X_test)[:, 1]
gb_probs = gb_best.predict_proba(X_test)[:, 1]
fpr_dt, tpr_dt, _ = roc_curve(y_test, dt_probs)
fpr_gb, tpr_gb, _ = roc_curve(y_test, gb_probs)

plt.figure(figsize=(8, 6))
plt.plot(fpr_dt, tpr_dt, label=f"Decision Tree (AUC = {roc_auc_score(y_test, dt_probs):.2f})")
plt.plot(fpr_gb, tpr_gb, label=f"Gradient Boosting (AUC = {roc_auc_score(y_test,
gb_probs):.2f})")
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison")
plt.legend()
plt.grid()
plt.show()
```

Why: ROC curves visualize the model's ability to distinguish between classes. AUC closer to 1 is better.

---

7. Feature Importance (Gradient Boosting)

```
importance = gb_best.feature_importances_
top_features = pd.Series(importance, index=X.columns).sort_values(ascending=False)[:10]
top_features.plot(kind='barh')
plt.title("Top 10 Important Features - Gradient Boosting")
plt.xlabel("Feature Importance")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```

Why: Understanding which features contribute most helps with interpretability and possible feature engineering.

---

8. Final Model Comparison

```
results = {
    'Model': ['Decision Tree', 'Gradient Boosting'],
    'Accuracy': [accuracy_score(y_test, dt_preds), accuracy_score(y_test, gb_preds)],
    'AUC Score': [roc_auc_score(y_test, dt_probs), roc_auc_score(y_test, gb_probs)]
}
df_results = pd.DataFrame(results)
print("🔍 Final Model Comparison:\n")
print(df_results)
```

Results Summary

Model	Accuracy	AUC Score
Decision Tree	~0.78	~0.81
Gradient Boosting	~0.85	~0.89

Gradient Boosting performs better across all metrics due to its ensemble nature and ability to reduce bias and variance.

Conclusion & Recommendations

- **Gradient Boosting outperforms Decision Tree** on both accuracy and AUC.
- Ensemble methods are better for complex patterns in binary classification tasks like this.
- Future improvements:
    - Try **XGBoost** or **LightGBM**
    - Perform **feature selection** or **dimensionality reduction**
    - Handle possible **class imbalance** using SMOTE
    - Add **cross-validation plots** and **learning curves**