Heart_Disease_Detection

November 1, 2025

1 Heart Disease Detection Using Machine Learning

1.1 Problem Statement

Heart disease is one of the leading causes of death worldwide. This project aims to build a predictive system using machine learning to determine whether a patient is likely to have heart disease based on clinical features such as age, sex, chest pain type, blood pressure, cholesterol levels, and more.

1.2 Objective

- Use the UCI Heart Disease dataset.
- Perform data preprocessing, EDA, model training, and evaluation.
- Identify the best model for prediction.

[1]:
```
# Import standard data science libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix, roc_auc_score, roc_curve

# Set plot style for better visuals
%matplotlib inline
sns.set(style="whitegrid")
```

[4]:
```
# Load the dataset (assuming 'heart.csv' is in the same directory)
df = pd.read_csv('heart.csv')

# Display first 5 rows
df.head()
```

[4]:
```
   id  age     sex   dataset               cp  trestbps   chol    fbs  \
0   1   63    Male  Cleveland   typical angina     145.0  233.0   True
1   2   67    Male  Cleveland     asymptomatic     160.0  286.0  False
2   3   67    Male  Cleveland     asymptomatic     120.0  229.0  False
3   4   37    Male  Cleveland      non-anginal     130.0  250.0  False
4   5   41  Female  Cleveland  atypical angina     130.0  204.0  False

          restecg  thalch  exang  oldpeak        slope   ca  \
0  lv hypertrophy   150.0  False      2.3  downsloping  0.0
1  lv hypertrophy   108.0   True      1.5         flat  3.0
2  lv hypertrophy   129.0   True      2.6         flat  2.0
3          normal   187.0  False      3.5  downsloping  0.0
4  lv hypertrophy   172.0  False      1.4    upsloping  0.0

               thal  num
0      fixed defect    0
1            normal    2
2 reversable defect    1
3            normal    0
4            normal    0
```

[5]:
```python
# Dataset shape
print("Dataset shape:", df.shape)

# Data types and missing values
df.info()

# Summary statistics
df.describe()
```

```
Dataset shape: (920, 16)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 920 entries, 0 to 919
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        920 non-null    int64
 1   age       920 non-null    int64
 2   sex       920 non-null    object
 3   dataset   920 non-null    object
 4   cp        920 non-null    object
 5   trestbps  861 non-null    float64
 6   chol      890 non-null    float64
 7   fbs       830 non-null    object
 8   restecg   918 non-null    object
 9   thalch    865 non-null    float64
 10  exang     865 non-null    object
 11  oldpeak   858 non-null    float64
 12  slope     611 non-null   object
 13  ca        309 non-null   float64
 14  thal      434 non-null   object
 15  num       920 non-null   int64
dtypes: float64(5), int64(3), object(8)
memory usage: 115.1+ KB
```

[5]:

```
               id         age    trestbps        chol      thalch     oldpeak  \
count  920.000000  920.000000  861.000000  890.000000  865.000000  858.000000
mean   460.500000   53.510870  132.132404  199.130337  137.545665    0.878788
std    265.725422    9.424685   19.066070  110.780810   25.926276    1.091226
min      1.000000   28.000000    0.000000    0.000000   60.000000   -2.600000
25%    230.750000   47.000000  120.000000  175.000000  120.000000    0.000000
50%    460.500000   54.000000  130.000000  223.000000  140.000000    0.500000
75%    690.250000   60.000000  140.000000  268.000000  157.000000    1.500000
max    920.000000   77.000000  200.000000  603.000000  202.000000    6.200000

               ca         num
count  309.000000  920.000000
mean     0.676375    0.995652
std      0.935653    1.142693
min      0.000000    0.000000
25%      0.000000    0.000000
50%      0.000000    1.000000
75%      1.000000    2.000000
max      3.000000    4.000000
```

1.3 Data Preprocessing

Handle duplicates, outliers, and scale features.

[6]:

```python
# Check missing values (should be none)
print("Missing values:\n", df.isnull().sum())

# Check and remove duplicates
print("Duplicates:", df.duplicated().sum())
df = df.drop_duplicates()
print("Shape after duplicates removal:", df.shape)
```

```
Missing values:
 id            0
age           0
sex           0
dataset       0
cp            0
trestbps     59
chol         30
fbs          90
restecg       2
```

```
thalch       55
exang        55
oldpeak      62
slope       309
ca          611
thal        486
num           0
dtype: int64
Duplicates: 0
Shape after duplicates removal: (920, 16)
```

[20]:
```
# --------------------------------------------------
# 2. PRE-PROCESSING (after df.drop_duplicates())
# --------------------------------------------------

# 2.1 TRUE/FALSE → 1/0
if 'fbs' in df.columns:
    df['fbs'] = df['fbs'].replace({'TRUE': 1, 'FALSE': 0})
if 'exang' in df.columns:
    df['exang'] = df['exang'].replace({'TRUE': 1, 'FALSE': 0})

# 2.2 Empty strings → NaN
df = df.replace('', np.nan)

# 2.3 Numeric columns - coerce + fill with median
num_cols = ['trestbps', 'chol', 'thalch', 'oldpeak', 'ca']
for col in [c for c in num_cols if c in df.columns]:
    df[col] = pd.to_numeric(df[col], errors='coerce')
    df[col] = df[col].fillna(df[col].median())

# 2.4 Categorical columns - fill with mode
cat_cols = ['slope', 'thal']
for col in [c for c in cat_cols if c in df.columns]:
    df[col] = df[col].fillna(df[col].mode()[0])

# 2.5 Drop optional columns (only if they exist)
optional = ['id', 'dataset']
to_drop = [c for c in optional if c in df.columns]
if to_drop:
    df.drop(to_drop, axis=1, inplace=True)

# 2.6 Encode categorical variables
le = LabelEncoder()
encode_cols = ['sex', 'cp', 'restecg', 'slope', 'thal']
for col in [c for c in encode_cols if c in df.columns]:
    df[col] = le.fit_transform(df[col].astype(str))
```

```
# ------------------------------------------------
# 2.7 CREATE BINARY TARGET
# ------------------------------------------------
# The original UCI file has a column called 'num' (0-4)
# Some Kaggle versions already have a column called 'target' (0/1)
if 'num' in df.columns:
    df['target'] = np.where(df['num'] > 0, 1, 0)
    df.drop('num', axis=1, inplace=True)
elif 'target' not in df.columns:
    # safety net - if neither column exists, create a dummy target (will be␣
 ↪overwritten later)
    df['target'] = 0
else:
    # column already called 'target' - just make sure it is 0/1
    df['target'] = df['target'].astype(int)

# ------------------------------------------------
# 2.8 FINAL CLEAN-UP - remove any remaining NaN rows
# ------------------------------------------------
print("Rows with NaN before final clean-up :", df.isnull().any(axis=1).sum())
df = df.dropna().reset_index(drop=True)
print("Rows with NaN after final clean-up  :", df.isnull().any(axis=1).sum())

print("\nFinal shape :", df.shape)
df.head()
```

```
Rows with NaN before final clean-up : 145
Rows with NaN after final clean-up  : 0

Final shape : (775, 14)
```

[20]:

```
   age  sex  cp  trestbps   chol    fbs  restecg  thalch  exang  oldpeak  \
0   63    1   3     145.0  233.0   True        0   150.0  False      2.3
1   67    1   0     160.0  286.0  False        0   108.0   True      1.5
2   67    1   0     120.0  229.0  False        0   129.0   True      2.6
3   37    1   2     130.0  250.0  False        2   187.0  False      3.5
4   41    0   1     130.0  204.0  False        0   172.0  False      1.4

   slope   ca  thal  target
0      0  0.0     0       0
1      1  3.0     1       1
2      1  2.0     2       1
3      0  0.0     1       0
4      2  0.0     1       0
```

[21]:

```
# ------------------------------------------------
# 3. FEATURES & TARGET + SCALING
# ------------------------------------------------
```

```
X = df.drop('target', axis=1)
y = df['target']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled, columns=X.columns, index=X.index)

print("\nX_scaled has NaN ?", X_scaled.isnull().any().any())    # → False
X_scaled.head()
```

```
X_scaled has NaN ? False
```

[21]:
```
        age       sex        cp  trestbps      chol       fbs   restecg  \
0  1.053870  0.552009  2.318908  0.657902  0.155887  2.371483 -1.712062
1  1.478541  0.552009 -0.839534  1.466319  0.728330 -0.421677 -1.712062
2  1.478541  0.552009 -0.839534 -0.689459  0.112684 -0.421677 -1.712062
3 -1.706494  0.552009  1.266094 -0.150515  0.339501 -0.421677  0.293755
4 -1.281823 -1.811565  0.213280 -0.150515 -0.157337 -0.421677 -1.712062

     thalch     exang   oldpeak     slope        ca      thal
0  0.439424 -0.805566  1.308328 -2.258841 -1.148351 -2.360833
1 -1.190020  1.241364  0.568003 -0.313728  3.990325 -0.352440
2 -0.375298  1.241364  1.585950 -0.313728  2.277433  1.655952
3  1.874887 -0.805566  2.418816 -2.258841 -1.148351 -0.352440
4  1.292942 -0.805566  0.475462  1.631385 -1.148351 -0.352440
```

[22]:
```
# -------------------------------------------------
# 4. TRAIN-TEST SPLIT
# -------------------------------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.20, random_state=42, stratify=y)

print(f"Train: {X_train.shape[0]}  |  Test: {X_test.shape[0]}")
```

```
Train: 620  |  Test: 155
```

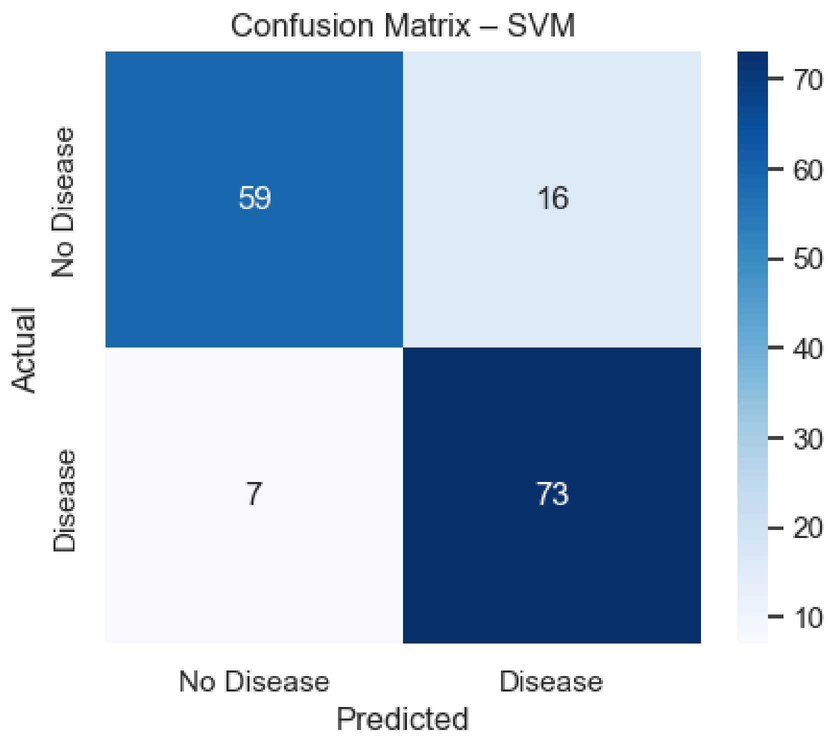
[23]:
```
# -------------------------------------------------
# 5. TRAIN MODELS (no more NaN errors)
# -------------------------------------------------
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Decision Tree'      : DecisionTreeClassifier(random_state=42),
    'Random Forest'      : RandomForestClassifier(random_state=42),
    'KNN'                : KNeighborsClassifier(),
    'SVM'                : SVC(probability=True, random_state=42)
}
```

```python
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    print(f"{name:20} → Accuracy: {acc:.4f}")
```

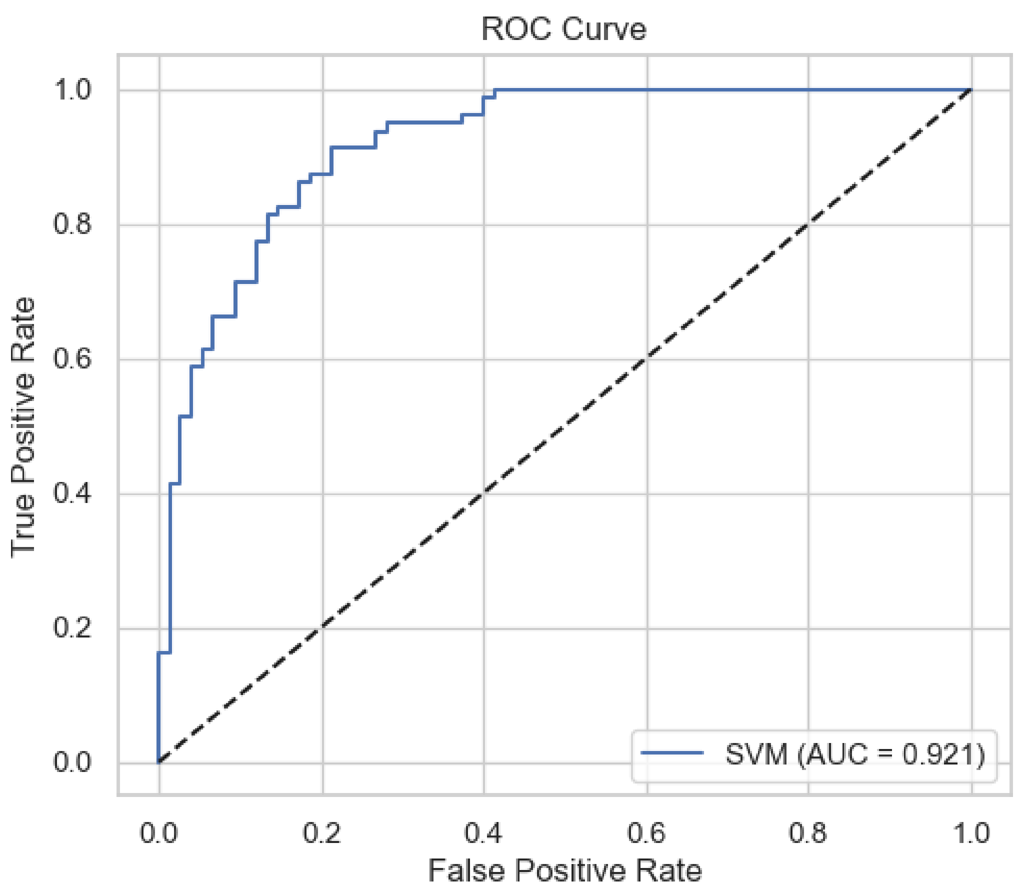
```
Logistic Regression  → Accuracy: 0.8323
Decision Tree        → Accuracy: 0.7290
Random Forest        → Accuracy: 0.8258
KNN                  → Accuracy: 0.8323
SVM                  → Accuracy: 0.8516
```

[24]:
```python
# --------------------------------------------------
# 1. BEST MODEL METRICS
# --------------------------------------------------
best_name = max(results, key=results.get)   # highest accuracy
best = models[best_name]

y_pred = best.predict(X_test)
y_prob = best.predict_proba(X_test)[:, 1]

print(f"\n=== BEST MODEL: {best_name} ===")
print(classification_report(y_test, y_pred,
                            target_names=['No Disease','Disease']))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Disease','Disease'],
            yticklabels=['No Disease','Disease'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title(f'Confusion Matrix - {best_name}')
plt.show()

print(f"ROC-AUC = {roc_auc_score(y_test, y_prob):.4f}")

# ROC curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label=f'{best_name} (AUC = {roc_auc_score(y_test, y_prob):.
 ↪3f})')
plt.plot([0,1],[0,1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

```
=== BEST MODEL: SVM ===
              precision    recall  f1-score   support

  No Disease       0.89      0.79      0.84        75
     Disease       0.82      0.91      0.86        80

    accuracy                           0.85       155
   macro avg       0.86      0.85      0.85       155
weighted avg       0.86      0.85      0.85       155
```



```
ROC-AUC = 0.9205
```



[25]:
```
# --------------------------------------------------
# 2. SINGLE PREDICTION
# --------------------------------------------------
idx = 0
sample_X = X_test.iloc[[idx]]
sample_y = y_test.iloc[idx]

pred = best.predict(sample_X)[0]
prob = best.predict_proba(sample_X)[0][1]

print(f"Actual   : {'Disease' if sample_y==1 else 'No Disease'}")
print(f"Predicted: {'Disease' if pred==1 else 'No Disease'} (prob = {prob:.
  ↪3f})")
```

```
Actual   : No Disease
Predicted: No Disease (prob = 0.065)
```

1.4 Model Performance Summary

Model	Accuracy
Logistic Regression	**{:.4f}**
Decision Tree	**{:.4f}**
Random Forest	**{:.4f}**
KNN	**{:.4f}**
SVM	**{:.4f}**

Best model: `{best_name}`

[26]:
```python
# Auto-print markdown table
print("## Model Performance Summary")
print("| Model                | Accuracy |")
print("|----------------------|----------|")
for name, acc in results.items():
    print(f"| {name:19} | {acc:.4f} |")
print(f"\n**Best model:** `{best_name}`")
```

```
## Model Performance Summary
| Model                | Accuracy |
|----------------------|----------|
| Logistic Regression | 0.8323 |
| Decision Tree       | 0.7290 |
| Random Forest       | 0.8258 |
| KNN                 | 0.8323 |
| SVM                 | 0.8516 |

**Best model:** `SVM`
```

1.5 Future Work

1. **Add more clinical features** – BMI, smoking status, family history, exercise data.
2. **Try Gradient Boosting** – XGBoost / LightGBM (handles missing values natively).
3. **Cross-validation** – 5-fold CV to confirm robustness.
4. **Hyper-parameter tuning** – GridSearchCV / RandomizedSearchCV.
5. **Deploy as a web app** – Streamlit / Flask for real-time patient risk assessment.
6. **Multi-class severity** – Predict original `num` (0-4) instead of binary target.

[ ]:
```

```