heartdiseaseprediction

May 15, 2025

[19]:
```
#KNN
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

# Load the dataset

data = pd.read_csv("C:\\Users\\HP\\Downloads\\heart.csv")

# Step 1: Handle missing values
# Check for missing values
data = data.dropna()

# Step 2: Handle duplicate rows
data = data.drop_duplicates()

# Step 3: Identify categorical and numerical columns
# Separate features and target
X = data.drop('target', axis=1)  # Features
y = data['target']               # Target variable

# If there are categorical columns, encode them
categorical_cols = X.select_dtypes(include=['object']).columns
if len(categorical_cols) > 0:
    X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)

# Step 4: Normalize numerical features
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)

# Step 5: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_normalized, y, test_size=0.3, random_state=42, stratify=y
)
```

```
#Step 6: Initialize and train the KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Step 7: Make predictions on the test data
y_pred = knn.predict(X_test)

# Step 8: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Step 9: Print results
print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
Accuracy: 0.82
Confusion Matrix:
[[34  8]
 [ 8 41]]
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.81      0.81        42
           1       0.84      0.84      0.84        49

    accuracy                           0.82        91
   macro avg       0.82      0.82      0.82        91
weighted avg       0.82      0.82      0.82        91
```

[20]:
```
#SVM
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

# Load the dataset
data = pd.read_csv("C:\\Users\\HP\\Downloads\\heart.csv")

# Step 1: Handle missing values
```

```
# Check for missing values
data = data.dropna()

# Step 2: Handle duplicate rows
data = data.drop_duplicates()

# Step 3: Identify categorical and numerical columns
# Separate features and target
X = data.drop('target', axis=1)  # Features
y = data['target']               # Target variable

# If there are categorical columns, encode them
categorical_cols = X.select_dtypes(include=['object']).columns
if len(categorical_cols) > 0:
    X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)

# Step 4: Normalize numerical features
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)

# Step 5: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_normalized, y, test_size=0.3, random_state=42, stratify=y
)

# Step 6: Initialize and train the SVM model
svm = SVC(kernel='linear', C=1.0, random_state=42)
svm.fit(X_train, y_train)

# Step 7: Make predictions on the test data
y_pred = svm.predict(X_test)

# Step 8: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Step 9: Print results
print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
Accuracy: 0.79
Confusion Matrix:
[[31 11]
```

```
 [ 8 41]]
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.74      0.77        42
           1       0.79      0.84      0.81        49

    accuracy                           0.79        91
   macro avg       0.79      0.79      0.79        91
weighted avg       0.79      0.79      0.79        91
```

[21]:
```python
#RF
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

# Load the dataset
data = pd.read_csv("C:\\Users\\HP\\Downloads\\heart.csv")

# Step 1: Handle missing values
# Check for missing values
data = data.dropna()

# Step 2: Handle duplicate rows
data = data.drop_duplicates()

# Step 3: Identify categorical and numerical columns
# Separate features and target
X = data.drop('target', axis=1)  # Features
y = data['target']               # Target variable

# If there are categorical columns, encode them
categorical_cols = X.select_dtypes(include=['object']).columns
if len(categorical_cols) > 0:
    X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)

# Step 4: Normalize numerical features
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)

# Step 5: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_normalized, y, test_size=0.3, random_state=42, stratify=y
```

```
)

# Step 6: Initialize and train the Random Forest model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Step 7: Make predictions on the test data
y_pred = rf.predict(X_test)

# Step 8: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Step 9: Print results
print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
Accuracy: 0.77
Confusion Matrix:
[[33  9]
 [12 37]]
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.79      0.76        42
           1       0.80      0.76      0.78        49

    accuracy                           0.77        91
   macro avg       0.77      0.77      0.77        91
weighted avg       0.77      0.77      0.77        91
```

[22]:
```
#ANN
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Load the dataset directly
data = pd.read_csv("C:\\Users\\HP\\Downloads\\heart.csv")
```

```
# Step 1: Handle missing values (if any)
data = data.dropna()

# Step 2: Handle duplicate rows
data = data.drop_duplicates()

# Step 3: Separate features and target
X = data.drop('target', axis=1)  # Features
y = data['target']               # Target variable

# Step 4: Normalize numerical features
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)

# Step 5: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_normalized, y, test_size=0.3, random_state=42, stratify=y
)

# Step 6: Build the ANN model
model = Sequential([
    Dense(32, activation='relu', input_shape=(X_train.shape[1],)),  # Input␣
 ↪layer
    Dense(16, activation='relu'),  # Hidden layer 1
    Dense(8, activation='relu'),   # Hidden layer 2
    Dense(1, activation='sigmoid') # Output layer
])

# Step 7: Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Step 8: Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)

# Step 9: Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)

# Print accuracy
print(f"Accuracy of ANN model: {accuracy:.2f}")
```

```
C:\Users\HP\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Accuracy of ANN model: 0.80
```

[23]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset

data = pd.read_csv("C:\\Users\\HP\\Downloads\\heart.csv")

# Split features and target
X = data.drop(columns=['target'])
y = data['target']

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

# Define individual models
knn = KNeighborsClassifier(n_neighbors=5)
svm = SVC(probability=True, random_state=42)  # Enable probability for soft␣
 ↪voting
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Combine models using VotingClassifier (soft voting)
voting_clf = VotingClassifier(
    estimators=[('knn', knn), ('svm', svm), ('rf', rf)],
    voting='soft'
)

# Fit the ensemble model
voting_clf.fit(X_train, y_train)

# Evaluate on the test set
y_pred = voting_clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

# Print results
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
print("\nClassification Report:\n", report)
```

```
Accuracy: 95.61%

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.96      0.96       100
           1       0.96      0.95      0.96       105

    accuracy                           0.96       205
   macro avg       0.96      0.96      0.96       205
weighted avg       0.96      0.96      0.96       205
```

[24]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier  # For ANN
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset
data = pd.read_csv("C:\\Users\\HP\\Downloads\\heart.csv")


# Split features and target
X = data.drop(columns=['target'])
y = data['target']

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

# Define individual models
knn = KNeighborsClassifier(n_neighbors=5)
svm = SVC(probability=True, random_state=42)  # Enable probability for soft␣
 ↪voting
ann = MLPClassifier(hidden_layer_sizes=(100,), max_iter=300, random_state=42)

# Combine models using VotingClassifier (soft voting)
```

```
voting_clf = VotingClassifier(
    estimators=[('knn', knn), ('svm', svm), ('ann', ann)],
    voting='soft'
)

# Fit the ensemble model
voting_clf.fit(X_train, y_train)

# Evaluate on the test set
y_pred = voting_clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

# Print results
print(f"Accuracy: {accuracy * 100:.2f}%")
print("\nClassification Report:\n", report)
```

```
Accuracy: 95.61%

Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.96      0.96       100
           1       0.96      0.95      0.96       105

    accuracy                           0.96       205
   macro avg       0.96      0.96      0.96       205
weighted avg       0.96      0.96      0.96       205


C:\Users\HP\anaconda3\Lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:690:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[1]:
```
# Import necessary 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.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.neural_network import MLPClassifier  # For ANN
```

```
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

# Load the dataset
data = pd.read_csv("C:\\Users\\HP\\Downloads\\heart.csv")

# Split features and target
X = data.drop(columns=['target'])
y = data['target']

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

# Define individual models
knn = KNeighborsClassifier(n_neighbors=5)
svm = SVC(probability=True, random_state=42)  # Enable probability for soft␣
 ↪voting
rf = RandomForestClassifier(n_estimators=100, random_state=42)
ann = MLPClassifier(hidden_layer_sizes=(100,), max_iter=300, random_state=42)

# Combine models using VotingClassifier (soft voting)
voting_clf = VotingClassifier(
    estimators=[('knn', knn), ('svm', svm), ('rf', rf), ('ann', ann)],
    voting='soft'
)

# Fit the ensemble model
voting_clf.fit(X_train, y_train)

# Evaluate on the test set
y_pred = voting_clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

# Print results
print(f"Accuracy: {accuracy * 100:.2f}%")
print("\nClassification Report:\n", report)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot Confusion Matrix
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['No␣
 ↪Disease', 'Disease'], yticklabels=['No Disease', 'Disease'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Plot Feature Importance (for Random Forest)
if hasattr(rf, 'feature_importances_'):
    feature_importances = rf.feature_importances_
    feature_names = data.columns[:-1]
    plt.figure(figsize=(10, 6))
    sns.barplot(x=feature_importances, y=feature_names, palette='viridis')
    plt.title('Feature Importance (Random Forest)')
    plt.xlabel('Importance Score')
    plt.ylabel('Features')
    plt.show()

# Plot Accuracy Comparison of Models
models = ['KNN', 'SVM', 'Random Forest', 'ANN', 'Voting Ensemble']
accuracies = [
    accuracy_score(y_test, knn.fit(X_train, y_train).predict(X_test)),
    accuracy_score(y_test, svm.fit(X_train, y_train).predict(X_test)),
    accuracy_score(y_test, rf.fit(X_train, y_train).predict(X_test)),
    accuracy_score(y_test, ann.fit(X_train, y_train).predict(X_test)),
    accuracy
]
```

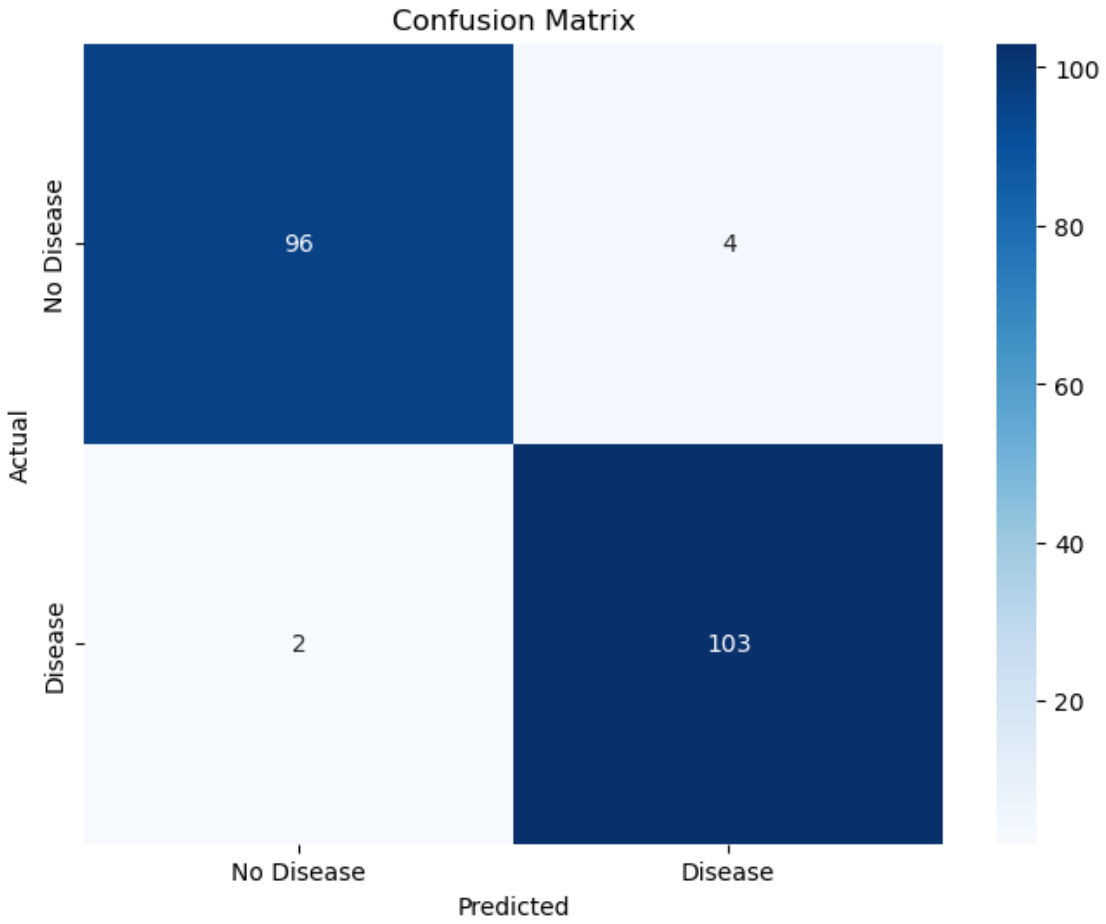
```
C:\Users\HP\anaconda3\Lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:690:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(

Accuracy: 97.07%

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.96      0.97       100
           1       0.96      0.98      0.97       105

    accuracy                           0.97       205
   macro avg       0.97      0.97      0.97       205
weighted avg       0.97      0.97      0.97       205
```




```
C:\Users\HP\anaconda3\Lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:690:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[7]:
```python
# Function to take user input and make predictions
def predict_new_patient(voting_clf, scaler):
    print("\nEnter new patient data:")
    # List of feature names
    feature_names = list(data.columns[:-1])
    # Collect inputs for all features
    patient_data = []
    for feature in feature_names:
        value = float(input(f"Enter value for {feature}: "))
        patient_data.append(value)

    # Convert to numpy array and scale it
```

```
    patient_data = np.array(patient_data).reshape(1, -1)
    patient_data_scaled = scaler.transform(patient_data)

    # Make prediction
    prediction = voting_clf.predict(patient_data_scaled)[0]
    probability = voting_clf.predict_proba(patient_data_scaled)[0]

    # Display results
    if prediction == 0:
        print("\nPrediction: No Disease")
    else:
        print("\nPrediction: Disease")

    print(f"Prediction Probability: {probability}")

# Example usage
predict_new_patient(voting_clf, scaler)
```

```
Enter new patient data:

Enter value for age:  58
Enter value for sex:  0
Enter value for cp:  0
Enter value for trestbps:  100
Enter value for chol:  248
Enter value for fbs:  0
Enter value for restecg:  0
Enter value for thalach:  122
Enter value for exang:  0
Enter value for oldpeak:  1
Enter value for slope:  1
Enter value for ca:  0
Enter value for thal:  2


Prediction: Disease
Prediction Probability: [0.02316842 0.97683158]

C:\Users\HP\anaconda3\Lib\site-packages\sklearn\base.py:493: UserWarning: X does
not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```