diabetes-prediction-using-decision-tree

October 20, 2024

AIES Mini Project By:-Hrishit Madhavi

[1]:
```
from google.colab import files
import pandas as pd

# Upload the file
uploaded = files.upload()

# Assuming you uploaded 'cleaned_data_logistic_regression.csv'
data = pd.read_csv('cleaned_data_logistic_regression.csv')
```

```
<IPython.core.display.HTML object>
```

```
Saving cleaned_data_logistic_regression.csv to
cleaned_data_logistic_regression.csv
```

Decision Tree Accuracy

[ ]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

# Prepare the data (X for features, y for target/labels)
X = data.drop('Outcome', axis=1)  # Drop 'Outcome' column if it's the target
y = data['Outcome']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Define the Decision Tree model
model = DecisionTreeClassifier(random_state=42)

# Define the hyperparameter grid
param_grid = {
    'max_depth': [2, 3, 4, 5, 6, 7, 8],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 4, 8, 16]
```

```
}

# Use GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,␣
 ↪scoring='accuracy')
grid_search.fit(X_train, y_train)

# Get the best model
best_model = grid_search.best_estimator_

# Predict using the best model
y_pred = best_model.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy and best hyperparameters
print(f"Training score for the best model: {grid_search.best_score_ * 100:.
 ↪2f}%")
print(f"Testing score for the best model: {accuracy * 100:.2f}%")
print(f"Best Hyperparameters: {grid_search.best_params_}")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)

# Classification Report
class_report = classification_report(y_test, y_pred)
print("\nClassification Report:")
print(class_report)
```

```
Training score for the best model: 80.62%
Testing score for the best model: 71.43%
Best Hyperparameters: {'max_depth': 5, 'min_samples_leaf': 4,
'min_samples_split': 20}

Confusion Matrix:
[[82 17]
 [27 28]]

Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.83      0.79        99
           1       0.62      0.51      0.56        55
```

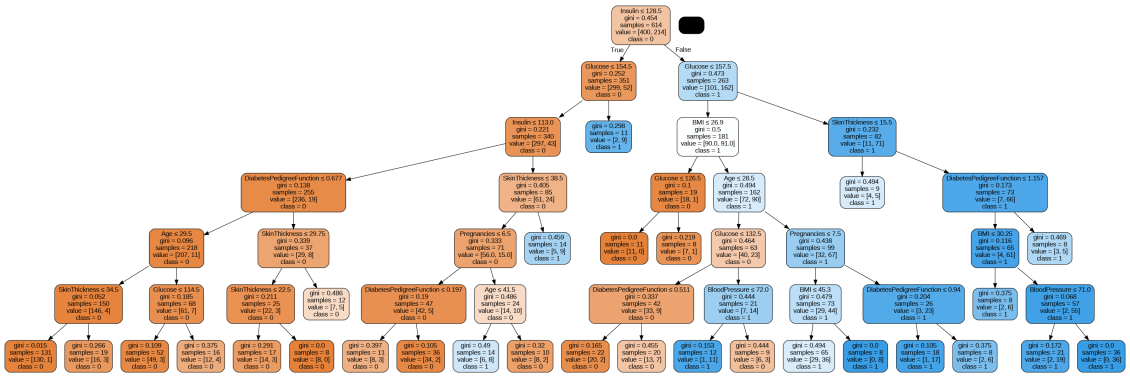
```
    accuracy                           0.71       154
   macro avg       0.69      0.67      0.67       154
weighted avg       0.71      0.71      0.71       154
```

```
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarning:
invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
```

Decision Tree Graph

[ ]:
```
# prompt: make decision tree graph

from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus

# Assuming 'best_model' is your trained Decision Tree model

dot_data = StringIO()
export_graphviz(best_model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,
                feature_names=X.columns,  # Replace with your feature names
                class_names=['0', '1'])  # Replace with your class names

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

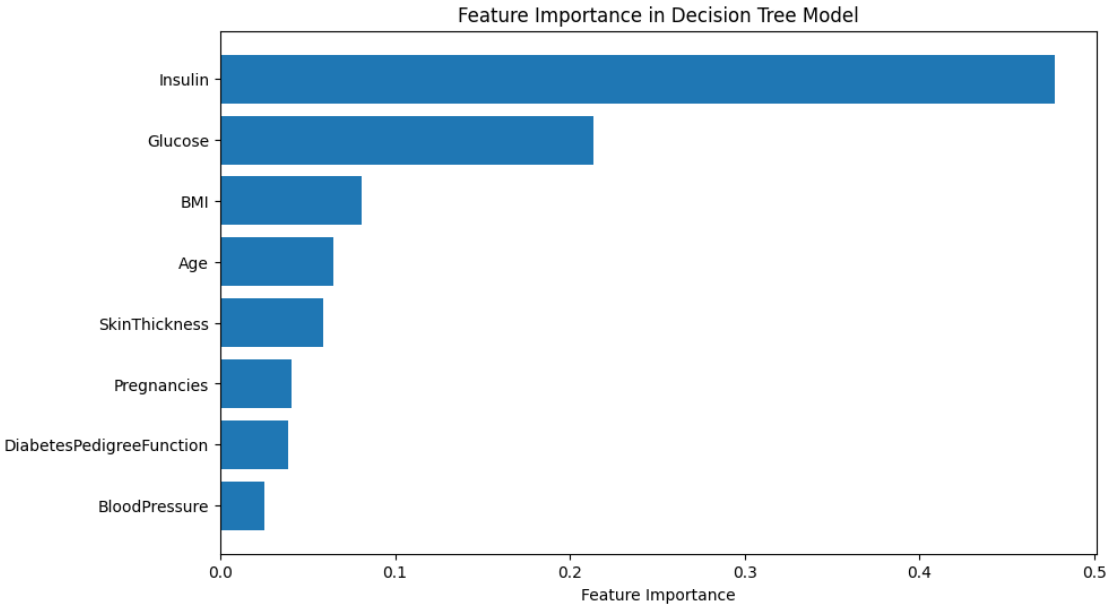
[ ]:



Feature Selection in decision tree

[ ]:
```
# Assuming 'best_model' is your trained Decision Tree model
dot_data = StringIO()
export_graphviz(best_model, out_file=dot_data,
                filled=True, rounded=True,
```

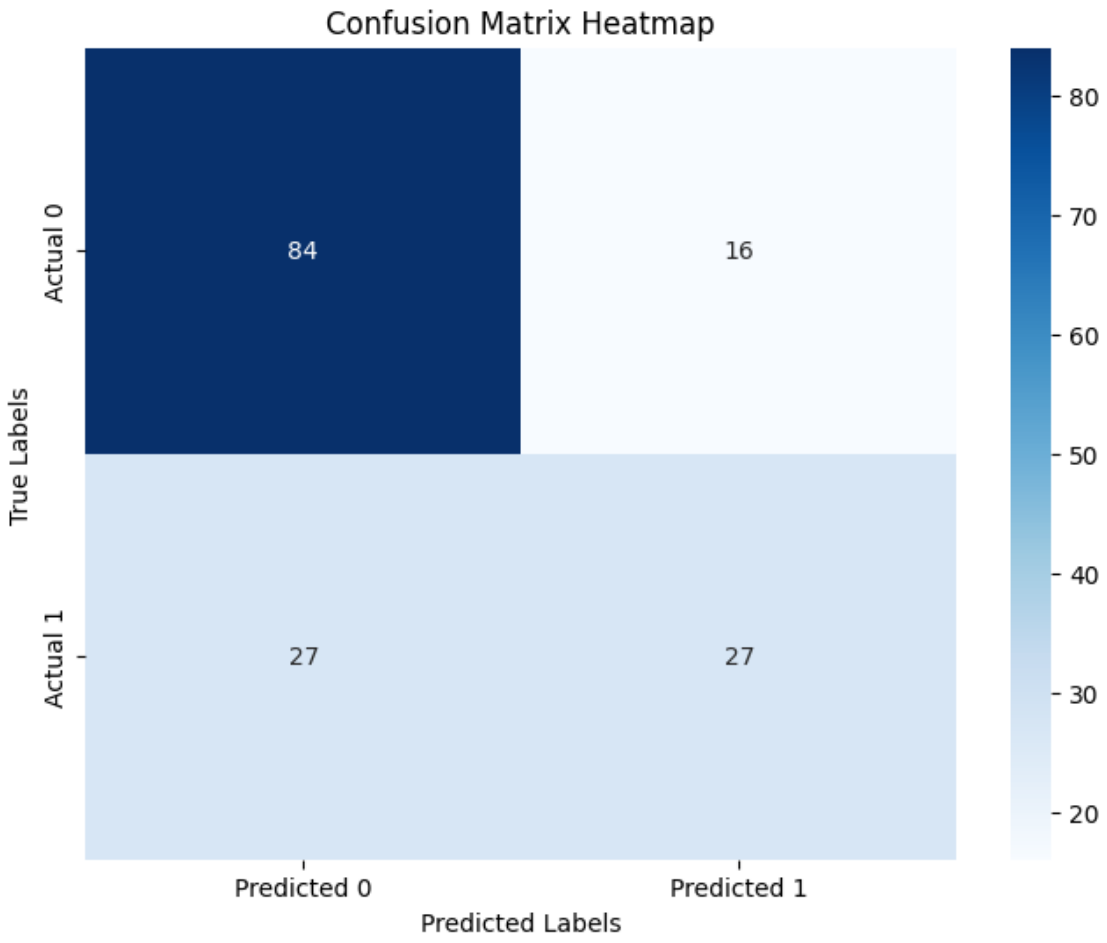
```
                special_characters=True,
                feature_names=X.columns,
                class_names=['0', '1'])

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())

# Feature Importance plot
import matplotlib.pyplot as plt

feature_importances = best_model.feature_importances_
sorted_idx = feature_importances.argsort()

plt.figure(figsize=(10, 6))
plt.barh(X.columns[sorted_idx], feature_importances[sorted_idx])
plt.xlabel("Feature Importance")
plt.title("Feature Importance in Decision Tree Model")
plt.show()

# Tree depth
tree_depth = best_model.tree_.max_depth
print(f"Tree Depth: {tree_depth}")
```




```
Tree Depth: 6
```

Confusion Matrix

[10]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'data' is already loaded and prepared
X = data.drop('Outcome', axis=1)  # Features
y = data['Outcome']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

# Define and train the Decision Tree model
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Create the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Create a heatmap for the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```



Decision Tree Accuracy Increased

[ ]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

# Prepare the data
X = data.drop('Outcome', axis=1)
y = data['Outcome']

# Split the data (stratified)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

# Define the model
model = DecisionTreeClassifier(random_state=42)
```

```
# Hyperparameter grid
param_grid = {
    'max_depth': [None, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 4, 6, 8],
    'criterion': ['gini', 'entropy']
}

# GridSearchCV for tuning
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,␣
 ↪scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Get the best model and evaluate
best_model = grid_search.best_estimator_
train_accuracy = best_model.score(X_train, y_train)
y_pred = best_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)

# Print accuracies and best hyperparameters
print(f"Training Accuracy: {train_accuracy * 100:.2f}%")
print(f"Testing Accuracy: {test_accuracy * 100:.2f}%")
print(f"Best Hyperparameters: {grid_search.best_params_}")

# Confusion Matrix and Classification Report
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)

class_report = classification_report(y_test, y_pred)
print("\nClassification Report:")
print(class_report)
```

```
Training Accuracy: 84.36%
Testing Accuracy: 79.87%
Best Hyperparameters: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf':
8, 'min_samples_split': 2}

Confusion Matrix:
[[83 17]
 [14 40]]

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.83      0.84       100
           1       0.70      0.74      0.72        54
```

```
              precision    recall  f1-score   support

    accuracy                         0.80       154
   macro avg       0.78      0.79      0.78       154
weighted avg       0.80      0.80      0.80       154
```