CR Decision Tree

October 8, 2024

1 Decision Tree: Credit Risk

1.0.1 Import Data and Preprocessing

[3]:
```
import pandas as pd
```

[5]:
```
df = pd.read_csv('LR_df.csv')
```

[7]:
```
df.head(5)
```

[7]:
```
   Unnamed: 0   Age   Amount   Rate  Status  Percent_income  Default  \
0           0  22.0  35000.0  16.02     1.0            0.59      1.0
1           1  21.0   1000.0  11.14     0.0            0.10      0.0
2           2  25.0   5500.0  12.87     1.0            0.57      0.0
3           3  23.0  35000.0  15.23     1.0            0.53      0.0
4           4  24.0  35000.0  14.27     1.0            0.55      1.0

   Cred_length  Home_MORTGAGE  Home_OTHER  Home_OWN  Home_RENT
0          3.0            0.0         0.0       0.0        1.0
1          2.0            0.0         0.0       1.0        0.0
2          3.0            1.0         0.0       0.0        0.0
3          2.0            0.0         0.0       0.0        1.0
4          4.0            0.0         0.0       0.0        1.0
```

[9]:
```
df.drop('Unnamed: 0', axis=1, inplace=True)
```

[11]:
```
df.head(5)
```

[11]:
```
    Age   Amount   Rate  Status  Percent_income  Default  Cred_length  \
0  22.0  35000.0  16.02     1.0            0.59      1.0          3.0
1  21.0   1000.0  11.14     0.0            0.10      0.0          2.0
2  25.0   5500.0  12.87     1.0            0.57      0.0          3.0
3  23.0  35000.0  15.23     1.0            0.53      0.0          2.0
4  24.0  35000.0  14.27     1.0            0.55      1.0          4.0

   Home_MORTGAGE  Home_OTHER  Home_OWN  Home_RENT
0            0.0         0.0       0.0        1.0
1            0.0         0.0       1.0        0.0
```

```
2              1.0          0.0        0.0         0.0
3              0.0          0.0        0.0         1.0
4              0.0          0.0        0.0         1.0
```

1.1 Training Phase

[13]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
import numpy as np
```

[14]:
```
df.fillna(0, inplace=True)
```

[15]:
```
X = df.drop('Default', axis=1)
y = df['Default']
```

[16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[21]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[23]:
```
dt_classifier = DecisionTreeClassifier(random_state=42,class_weight='balanced')
```

[25]:
```
#Set Hyperparameters

param_grid = {
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

[27]:
```
grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid,
                           cv=5, scoring='accuracy', verbose=1)
grid_search.fit(X_train_scaled, y_train)
```

```
Fitting 5 folds for each of 45 candidates, totalling 225 fits
```

[27]:
```
GridSearchCV(cv=5,
             estimator=DecisionTreeClassifier(class_weight='balanced',
                                              random_state=42),
             param_grid={'max_depth': [None, 5, 10, 15, 20],
```

```
                        'min_samples_leaf': [1, 2, 4],
                        'min_samples_split': [2, 5, 10]},
            scoring='accuracy', verbose=1)
```

[29]:
```
# Get the best parameters
best_params = grid_search.best_params_
print("Best parameters found: ", best_params)
```

```
Best parameters found:  {'max_depth': 5, 'min_samples_leaf': 2,
'min_samples_split': 10}
```

[31]:
```
best_dt_classifier = grid_search.best_estimator_
best_dt_classifier.fit(X_train_scaled, y_train)
```

[31]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=5, min_samples_leaf=2,
                       min_samples_split=10, random_state=42)
```

[33]:
```
#Make Predictions
y_pred = best_dt_classifier.predict(X_test_scaled)

results = grid_search.cv_results_
dt_classifier.fit(X_train, y_train)

train_accuracy = accuracy_score(y_train, dt_classifier.predict(X_train))
test_accuracy = accuracy_score(y_test, dt_classifier.predict(X_test))

print(f"Training accuracy: {train_accuracy:.2f}")
print(f"Validation accuracy: {test_accuracy:.2f}")
```

```
Training accuracy: 1.00
Validation accuracy: 0.82
```

1.2 Get Performance Metrics

[37]:
```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

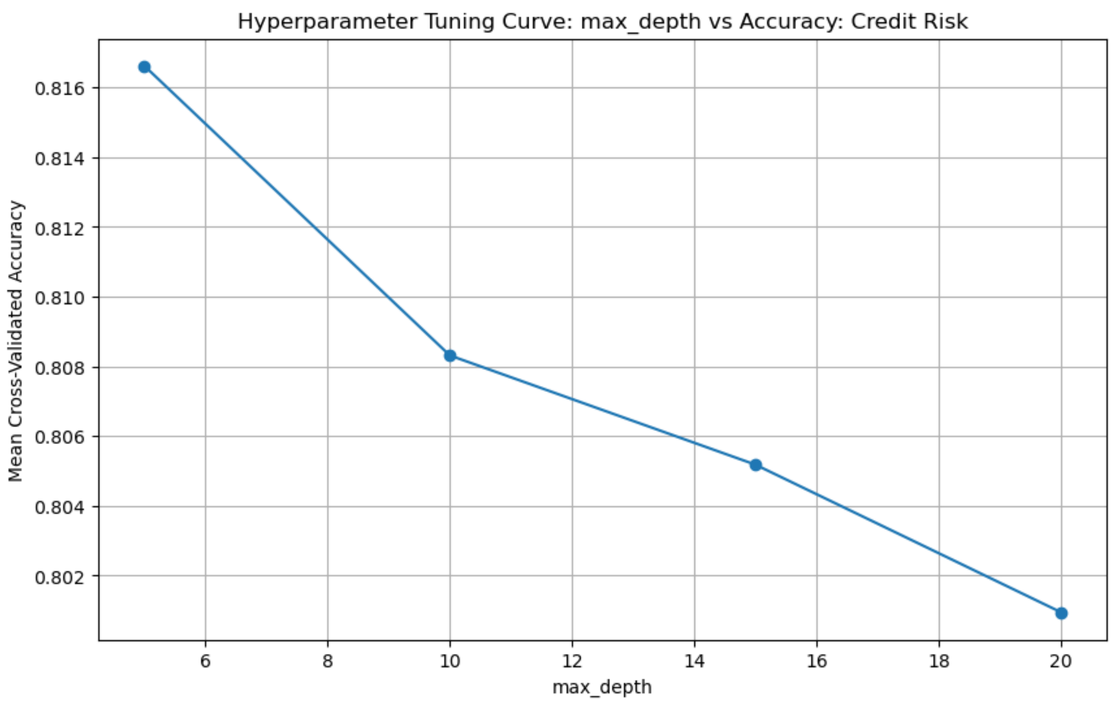
```
Accuracy: 0.83
[[4383 1009]
 [ 107 1018]]
              precision    recall  f1-score   support

         0.0       0.98      0.81      0.89      5392
         1.0       0.50      0.90      0.65      1125
```

```
    accuracy                           0.83      6517
   macro avg       0.74      0.86      0.77      6517
weighted avg       0.89      0.83      0.85      6517
```

1.3 Hyperparameter Tuning Curve - Max_Depth

[39]:
```python
import matplotlib.pyplot as plt

grid_search.fit(X_train_scaled, y_train)

# Get the results
results = grid_search.cv_results_

# Extract mean test scores and parameters
mean_test_scores = results['mean_test_score']
param_values = results['param_max_depth'].data

# Convert to numpy arrays
param_values = np.array(param_values)
mean_test_scores = np.array(mean_test_scores)

# Filter out None values and test scores
valid_indices = ~np.isnan(mean_test_scores) & (param_values != None)
filtered_param_values = param_values[valid_indices]
filtered_mean_test_scores = mean_test_scores[valid_indices]

# Create a DataFrame to group by max_depth and average the scores
results_df = pd.DataFrame({
    'max_depth': filtered_param_values,
    'mean_test_score': filtered_mean_test_scores
})

# Group by max_depth and calculate the mean scores
grouped_results = results_df.groupby('max_depth').mean().reset_index()

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(grouped_results['max_depth'], grouped_results['mean_test_score'],␣
↪marker='o')
plt.title("Hyperparameter Tuning Curve: max_depth vs Accuracy: Credit Risk")
plt.xlabel("max_depth")
plt.ylabel("Mean Cross-Validated Accuracy")
plt.grid()
plt.show()
```

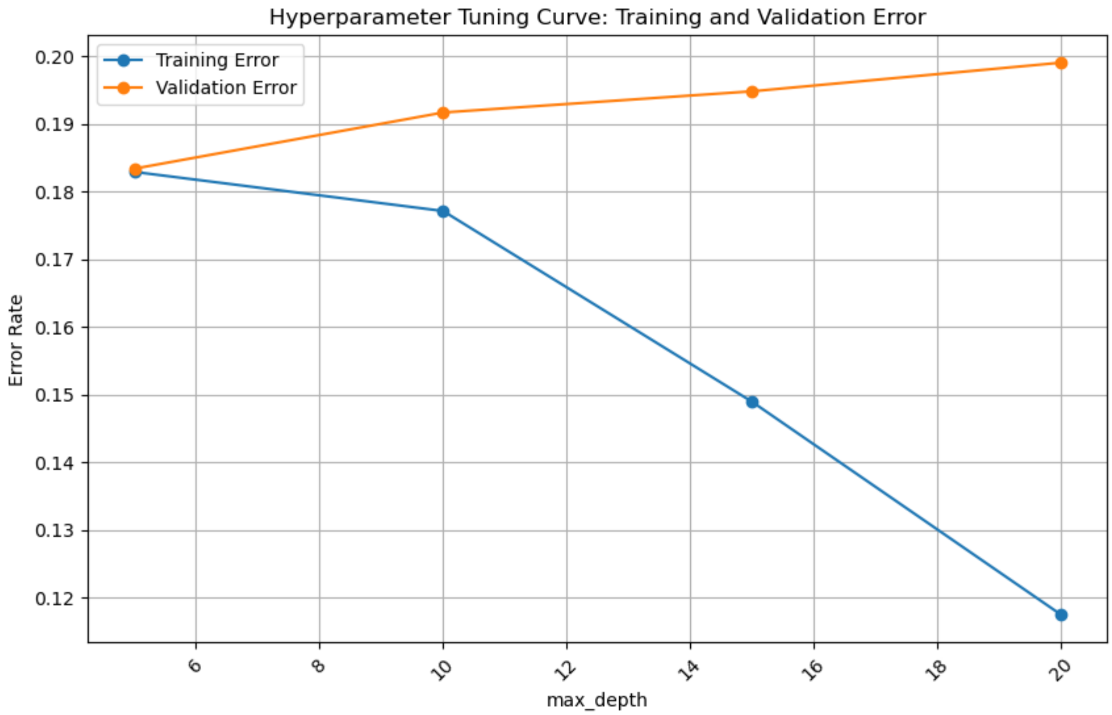
```
Fitting 5 folds for each of 45 candidates, totalling 225 fits
```



1.4 Training vs. Validation Error

[41]:
```python
from sklearn.model_selection import GridSearchCV

grid_search = GridSearchCV(estimator=dt_classifier,
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=5,
                           return_train_score=True)
grid_search.fit(X_train_scaled, y_train)
results = grid_search.cv_results_

# Extract mean training and validation scores
mean_train_scores = results['mean_train_score']
mean_test_scores = results['mean_test_score']
param_values = results['param_max_depth'].data

# Convert to NumPy arrays for easier processing
param_values = np.array(param_values, dtype=object)
mean_train_scores = np.array(mean_train_scores)
mean_test_scores = np.array(mean_test_scores)


results_df = pd.DataFrame({
```

```
    'max_depth': param_values,
    'mean_train_score': mean_train_scores,
    'mean_test_score': mean_test_scores
})

# Group by max_depth and calculate mean scores
aggregated_results = results_df.groupby('max_depth').agg({
    'mean_train_score': 'mean',
    'mean_test_score': 'mean'
}).reset_index()

# Calculate errors
aggregated_results['train_error'] = 1 - aggregated_results['mean_train_score']
aggregated_results['validation_error'] = 1 -
 ↪aggregated_results['mean_test_score']

# Plotting the aggregated errors
plt.figure(figsize=(10, 6))
plt.plot(aggregated_results['max_depth'], aggregated_results['train_error'],
 ↪marker='o', label='Training Error')
plt.plot(aggregated_results['max_depth'],
 ↪aggregated_results['validation_error'], marker='o', label='Validation Error')
plt.title("Hyperparameter Tuning Curve: Training and Validation Error")
plt.xlabel("max_depth")
plt.ylabel("Error Rate")
plt.xticks(rotation=45)
plt.legend()
plt.grid()
plt.show()
```

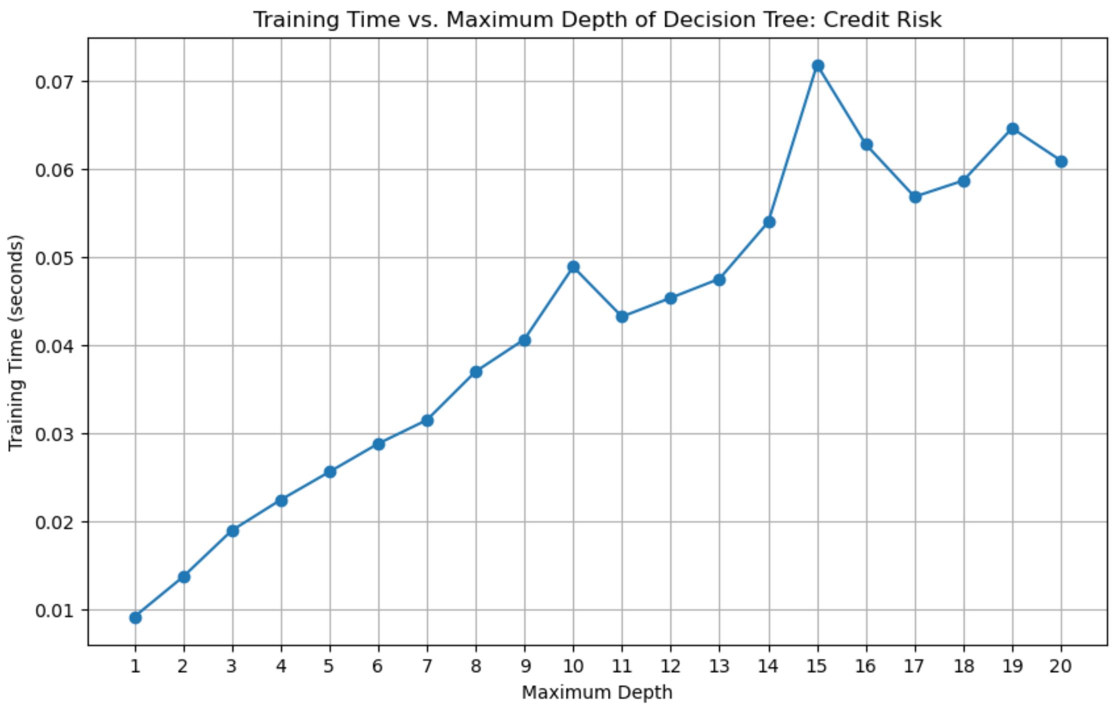


1.5 Training Time

[43]:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import time


# Maximum depths to evaluate
max_depths = np.arange(1, 21)  # from 1 to 20
training_times = []

# Measure training time for different max depths
for max_depth in max_depths:
    clf = DecisionTreeClassifier(max_depth=max_depth)

    start_time = time.time()
    clf.fit(X_train, y_train)
    end_time = time.time()

    training_times.append(end_time - start_time)
```

```
# Plotting the training time vs. max depth
plt.figure(figsize=(10, 6))
plt.plot(max_depths, training_times, marker='o')
plt.title('Training Time vs. Maximum Depth of Decision Tree: Credit Risk')
plt.xlabel('Maximum Depth')
plt.ylabel('Training Time (seconds)')
plt.xticks(max_depths)
plt.grid()
plt.show()
```

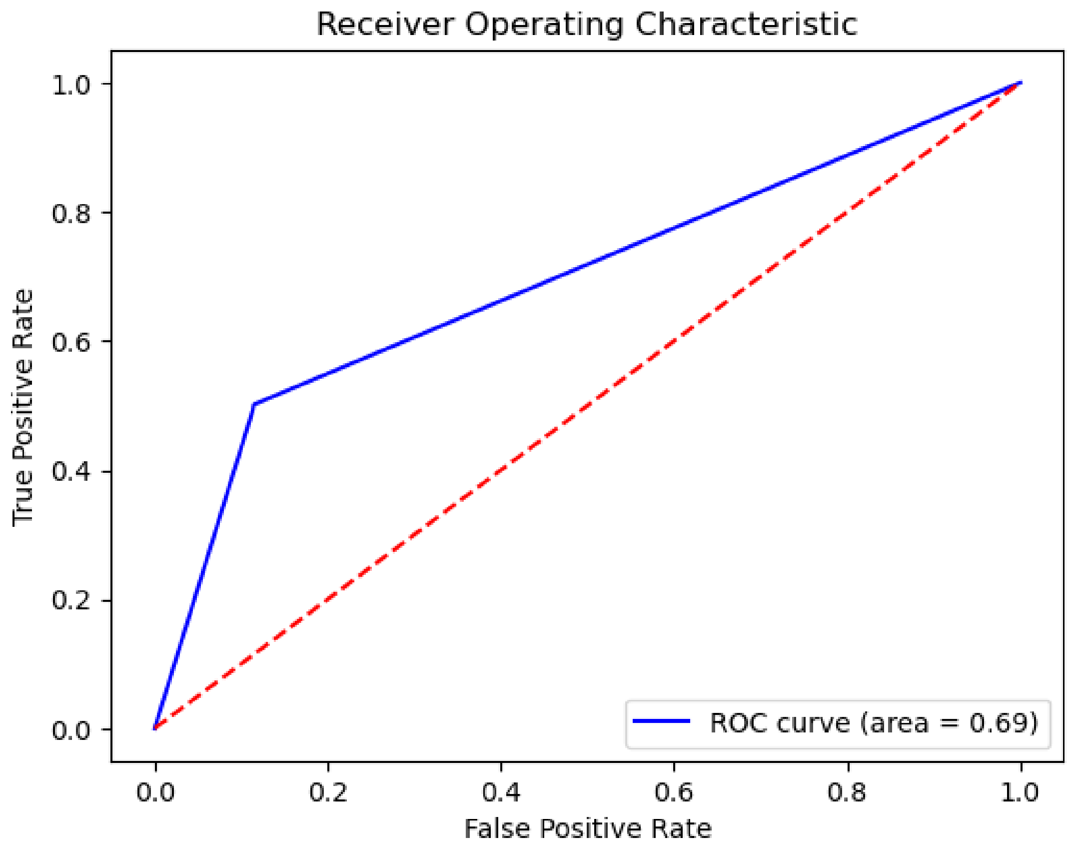


1.6 ROC Curve

[57]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

fpr, tpr, _ = roc_curve(y_test, y_probs)
roc_auc = auc(fpr, tpr)

fpr, tpr, thresholds = roc_curve(y_test, y_probs)

plt.plot(fpr, tpr, color='blue', label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.title('Receiver Operating Characteristic')
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
```

[57]: <matplotlib.legend.Legend at 0x72d65a488f90>



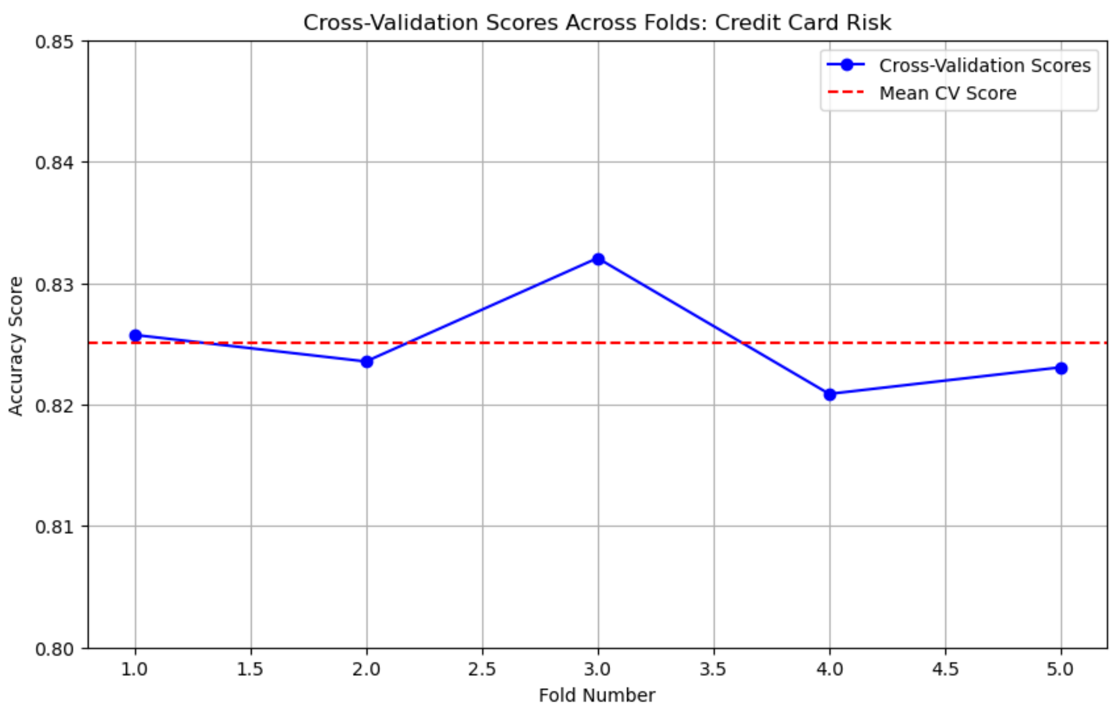
1.7 Cross Validation Scores Across Folds

[59]:
```
import numpy as np
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Evaluate the model with cross-validation
best_model = grid_search.best_estimator_
cv_scores = cross_val_score(best_model, X_train_scaled, y_train, cv=5)
print("Cross-Validation Scores:", cv_scores)
print("Mean Cross-Validation Score:", np.mean(cv_scores))
```

```
Cross-Validation Scores: [0.81910608 0.80663725 0.82179167 0.8185306
0.81737963]
Mean Cross-Validation Score: 0.8166890466142338
```

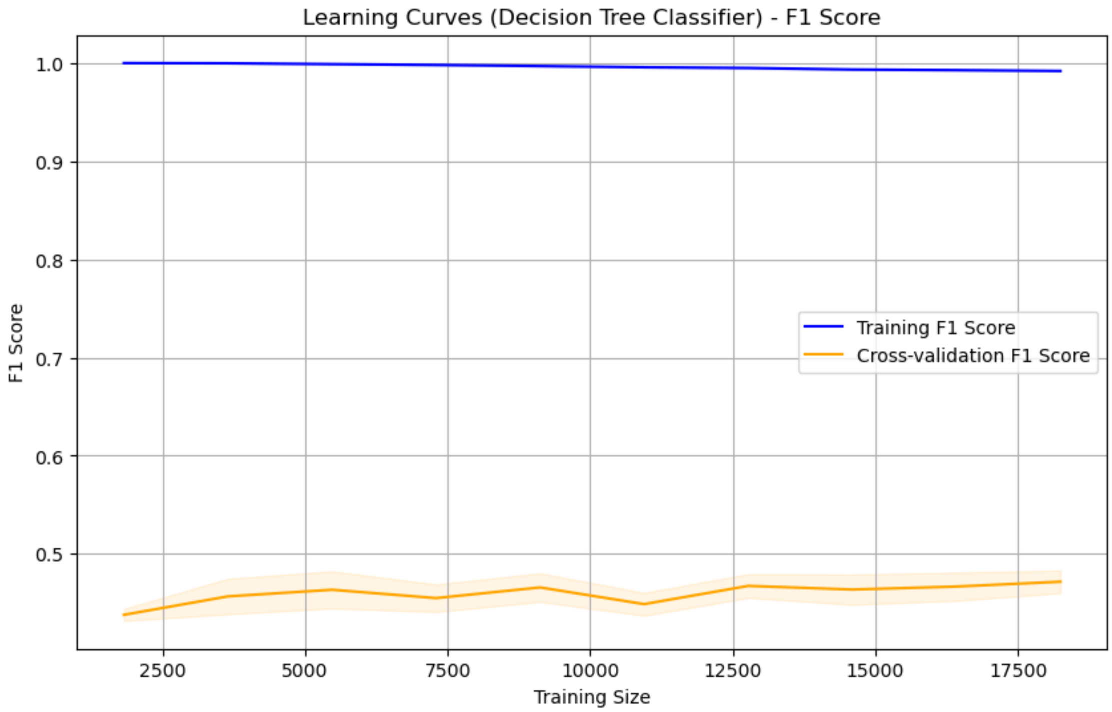
[61]:

```python
cv_scores = [0.82573433, 0.82354231, 0.83205437, 0.82087262, 0.82306512]
mean_cv_score = 0.8250537480099516


plt.figure(figsize=(10, 6))
plt.plot(range(1, len(cv_scores) + 1), cv_scores, marker='o', linestyle='-',␣
 ↪color='blue', label='Cross-Validation Scores')
plt.axhline(y=mean_cv_score, color='red', linestyle='--', label='Mean CV Score')
plt.title('Cross-Validation Scores Across Folds: Credit Card Risk')
plt.xlabel('Fold Number')
plt.ylabel('Accuracy Score')
plt.ylim(0.8, 0.85)
plt.legend()
plt.grid(True)
plt.show()
```



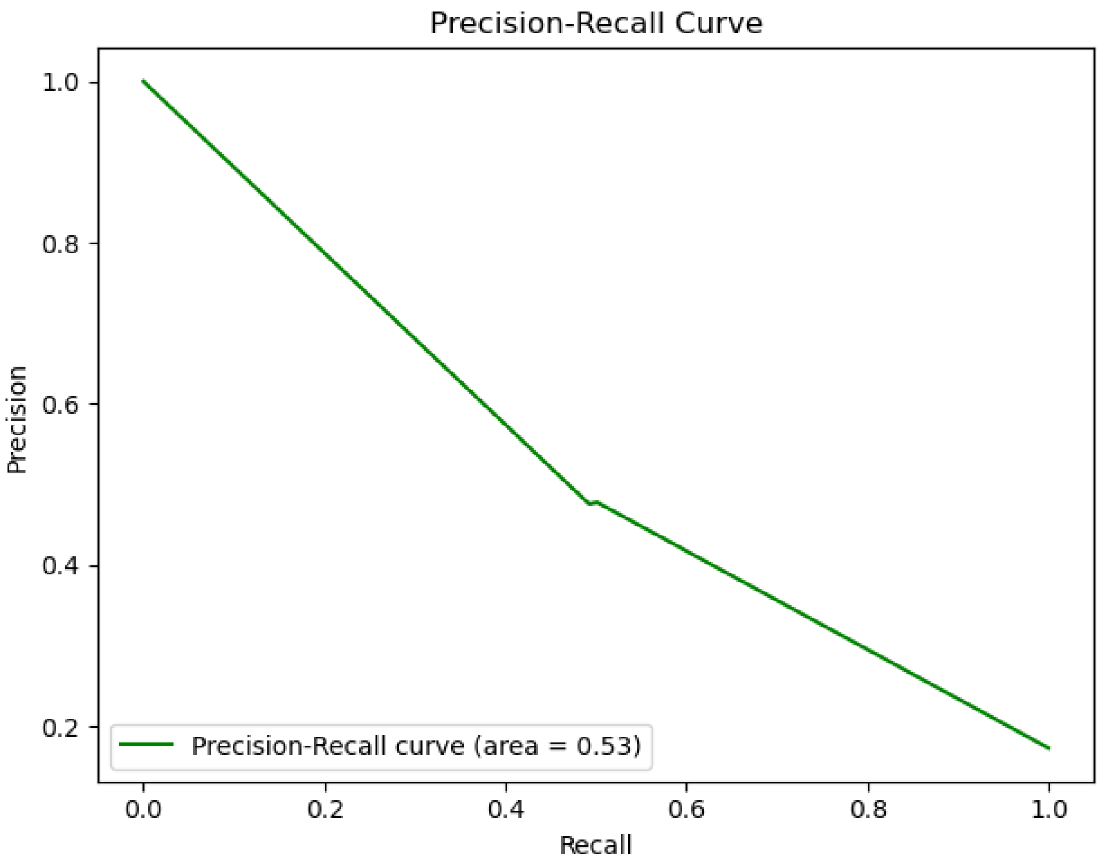
1.8 Additional Visuals

[27]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import learning_curve
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import make_scorer, f1_score


model = dt_classifier
f1_scorer = make_scorer(f1_score)

# Generate learning curves using F1 score
train_sizes, train_scores, test_scores = learning_curve(
    model, X_train, y_train,
    train_sizes=np.linspace(0.1, 1.0, 10),
    cv=5,
    scoring=f1_scorer
)

# Calculate the mean and standard deviation for training and test scores
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot learning curves
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, label='Training F1 Score', color='blue')
plt.plot(train_sizes, test_mean, label='Cross-validation F1 Score',␣
↪color='orange')

# Plot the std deviation as a shaded area
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,␣
↪color='blue', alpha=0.1)
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std,␣
↪color='orange', alpha=0.1)


plt.title('Learning Curves (Decision Tree Classifier) - F1 Score')
plt.xlabel('Training Size')
plt.ylabel('F1 Score')
plt.legend(loc='best')
plt.grid()
plt.show()
```



[55]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve, auc, precision_recall_curve


# Train the Decision Tree classifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# Get predicted probabilities
y_probs = clf.predict_proba(X_test)[:, 1]

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_probs)
roc_auc = auc(fpr, tpr)

# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, y_probs)
pr_auc = auc(recall, precision)
```

```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 2)
plt.plot(recall, precision, color='green', label=f'Precision-Recall curve (area␣
↪= {pr_auc:.2f})')
plt.title('Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend(loc='lower left')

plt.tight_layout()
plt.show()
```




```
[ ]:
```