CR LR.

October 8, 2024

1 Logistic Regression: Credit Risk

1.1 Load Data

[3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, classification_report,␣
 ↪roc_curve, roc_auc_score

LR_df = pd.read_csv('LR_df.csv')
```

1.2 Preprocessing and Training Phase

[4]:
```
LR_df = LR_df.dropna(subset=['Default'])



X = LR_df.drop('Default', axis=1)
y = LR_df['Default']




imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)

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

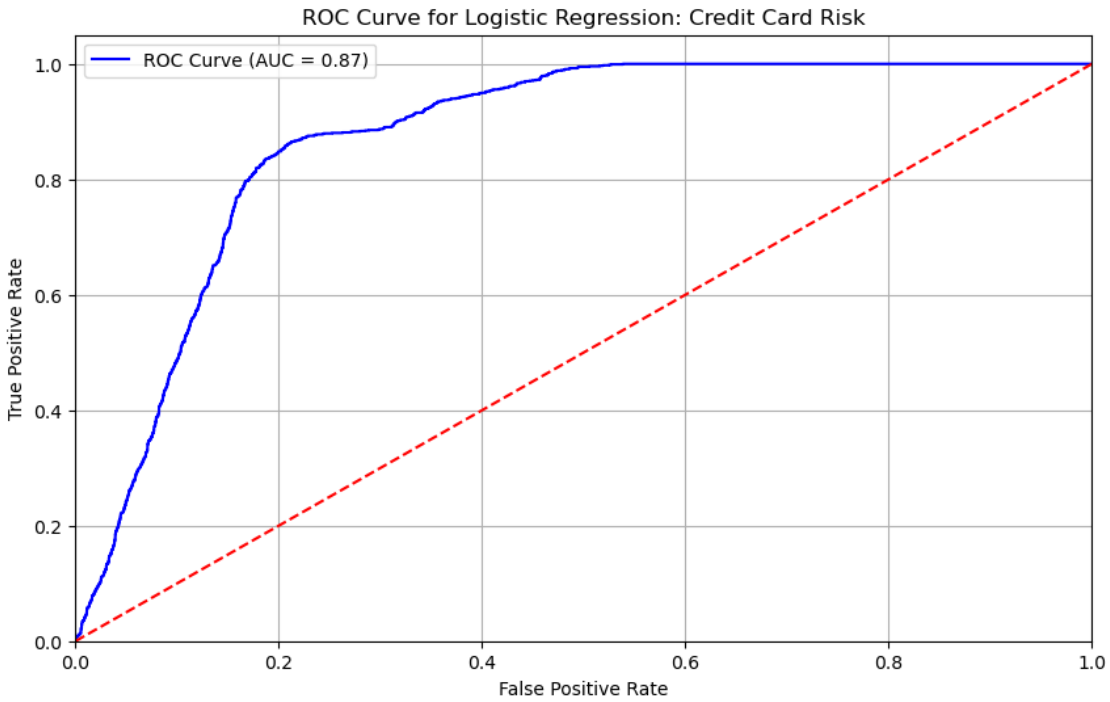
```
# Define the logistic regression model
log_reg = LogisticRegression(solver='liblinear', class_weight='balanced')

# Set up hyperparameter tuning with GridSearchCV
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
}

grid_search = GridSearchCV(log_reg, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Get the best parameters and model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

print("Best Hyperparameters (Logistic Regression):", best_params)

# Cross-Validation for Logistic Regression
logistic_cv_scores = cross_val_score(best_model, X_scaled, y, cv=5,␣
 ↪scoring='f1')
print("Logistic Regression F1 Score (CV):", np.mean(logistic_cv_scores))

# Make predictions on the test set
y_pred = best_model.predict(X_test)

# Calculate accuracy and F1 score for the test set
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print evaluation metrics for the test set
print("Test Set Accuracy:", accuracy)
print("Test Set F1 Score:", f1)
print("Classification Report:\n", classification_report(y_test, y_pred))

y_prob = best_model.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)
```

```
Best Hyperparameters (Logistic Regression): {'C': 1, 'penalty': 'l1'}
Logistic Regression F1 Score (CV): 0.6077665300582608
Test Set Accuracy: 0.8085008439466012
Test Set F1 Score: 0.6190476190476191
Classification Report:
              precision    recall  f1-score   support
```

```
              0.0       0.96      0.80      0.87      5322
              1.0       0.49      0.85      0.62      1195

         accuracy                           0.81      6517
        macro avg       0.72      0.82      0.75      6517
     weighted avg       0.87      0.81      0.83      6517
```

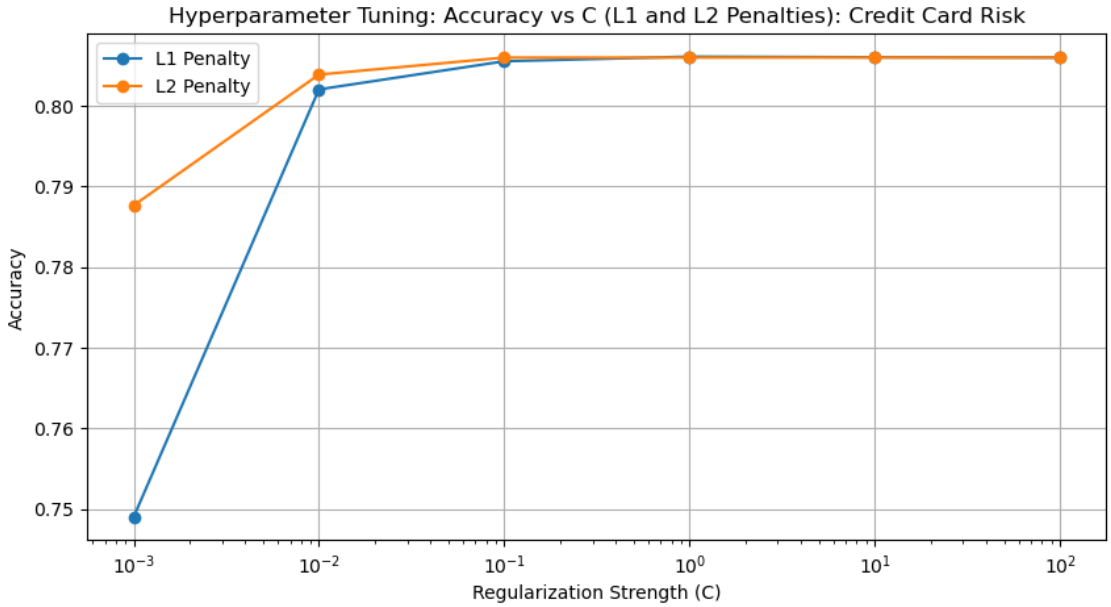
1.3 ROC Curve

[5]:
```
# Plot the ROC curve
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.title("ROC Curve for Logistic Regression: Credit Card Risk")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.grid()
plt.legend()
plt.show()
```



1.4 Hyperparameter Tuning Curve: Regularization and Penalty

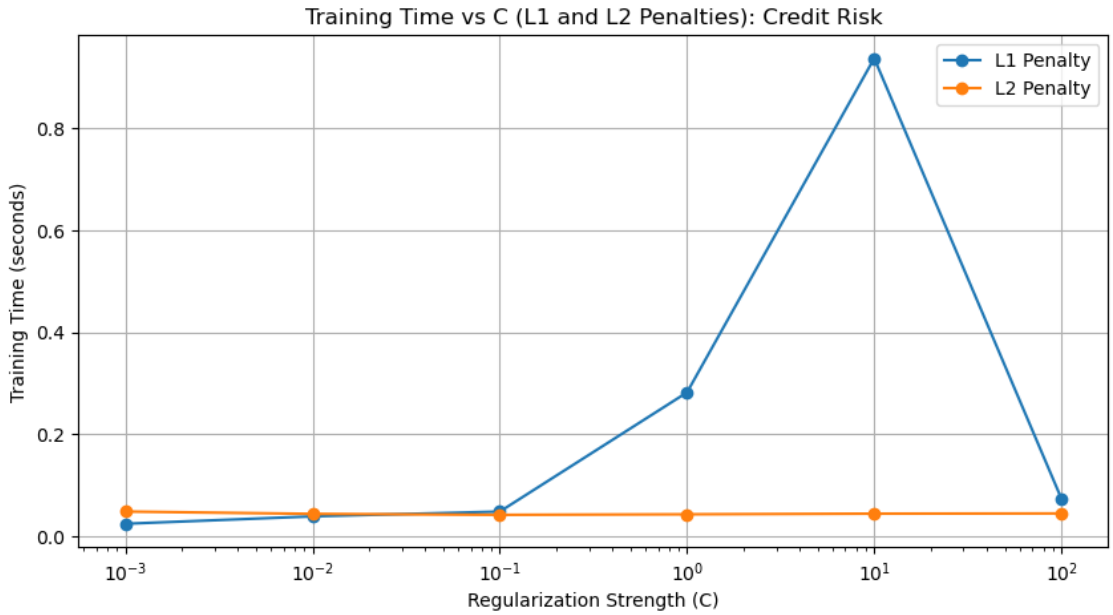
[6]:
```
grid_search_acc = GridSearchCV(log_reg, param_grid, cv=5, scoring='accuracy')
grid_search_acc.fit(X_train, y_train)
acc_results = grid_search_acc.cv_results_

# Define the C values for plotting
C_values = param_grid['C']

# Plot hyperparameter tuning curves for accuracy
plt.figure(figsize=(10, 5))

# Accuracy scores for both l1 and l2 penalties
l1_acc_scores = [acc_results['mean_test_score'][i] for i in␣
 ↪range(len(acc_results['param_C'])) if acc_results['param_penalty'][i] ==␣
 ↪'l1']
l2_acc_scores = [acc_results['mean_test_score'][i] for i in␣
 ↪range(len(acc_results['param_C'])) if acc_results['param_penalty'][i] ==␣
 ↪'l2']

#Plotting
plt.plot(C_values, l1_acc_scores, label="L1 Penalty", marker='o')
plt.plot(C_values, l2_acc_scores, label="L2 Penalty", marker='o')

plt.xscale('log')
plt.xlabel('Regularization Strength (C)')
plt.ylabel('Accuracy')
plt.title('Hyperparameter Tuning: Accuracy vs C (L1 and L2 Penalties): Credit␣
 ↪Card Risk')
plt.legend()
plt.grid(True)
plt.show()
```



1.5 Training Time

[7]:
```
import time
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV


# Store training into a list
training_times = []

# Loop over the C values and measure the training time
for C_value in param_grid['C']:
    times_for_C = []
    for penalty in param_grid['penalty']:
        log_reg = LogisticRegression(solver='liblinear', penalty=penalty,␣
 ↪C=C_value, max_iter=10000)
        start_time = time.time()
        log_reg.fit(X_train, y_train)
        elapsed_time = time.time() - start_time
        times_for_C.append(elapsed_time)
    training_times.append(times_for_C)


l1_times = [training_times[i][0] for i in range(len(param_grid['C']))]
```

```
l2_times = [training_times[i][1] for i in range(len(param_grid['C']))]

# Plot the Training Time vs C curve
plt.figure(figsize=(10, 5))

plt.plot(param_grid['C'], l1_times, label="L1 Penalty", marker='o')
plt.plot(param_grid['C'], l2_times, label="L2 Penalty", marker='o')

plt.xscale('log')
plt.xlabel('Regularization Strength (C)')
plt.ylabel('Training Time (seconds)')
plt.title('Training Time vs C (L1 and L2 Penalties): Credit Risk')
plt.legend()
plt.grid(True)
plt.show()
```



[10]:
```
#Troubleshooting
# Check for missing values in X_train
print("Missing values in X_train:", np.isnan(X_train).sum())

# Check for zero variance features
zero_variance_features = np.where(np.var(X_train, axis=0) == 0)[0]
print("Zero variance features indices:", zero_variance_features)

# If there are zero variance features, drop them
if len(zero_variance_features) > 0:
```

```
    X_train = np.delete(X_train, zero_variance_features, axis=1)
    X_test = np.delete(X_test, zero_variance_features, axis=1)

# Re-check data types
print("Data types of features in X_train:", X_train.dtype)
```

```
Missing values in X_train: 0
Zero variance features indices: []
Data types of features in X_train: float64
```

1.6 Learning Curve: Training vs. Validation Scores - F1 Score

[11]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.linear_model import LogisticRegression

# Set up the model with best parameters
log_reg = LogisticRegression(solver='liblinear', class_weight='balanced', C=1,␣
 ↪penalty='l2')
log_reg.fit(X_train, y_train)

# Define the range of training sizes
train_sizes = np.linspace(0.1, 1.0, 10)

# Calculate training and validation scores
train_sizes, train_scores, valid_scores = learning_curve(
    log_reg,
    X_train,
    y_train,
    train_sizes=train_sizes,
    cv=5,
    scoring='f1'
)

# Calculate the mean and standard deviation for training scores
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)

# Calculate the mean and standard deviation for validation scores
valid_scores_mean = np.mean(valid_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)


print("Training Scores Mean:", train_scores_mean)
print("Validation Scores Mean:", valid_scores_mean)
```

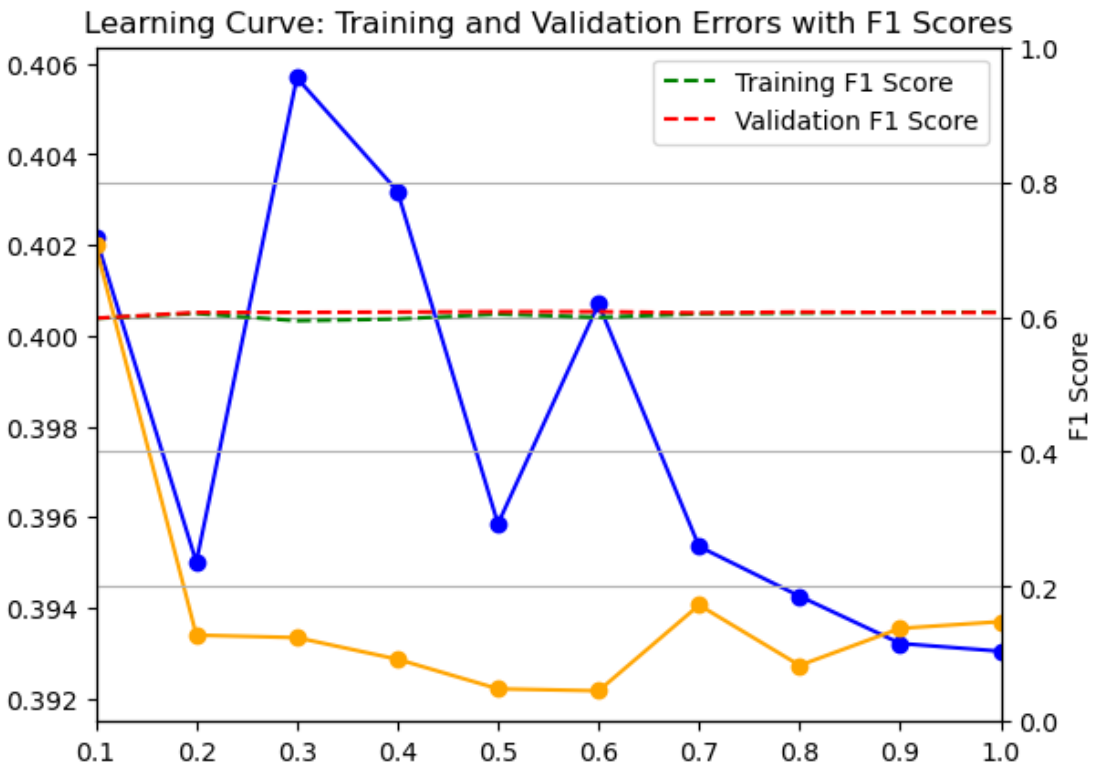
```
print("Shape of X_train:", X_train.shape)
print("Unique classes in y_train:", np.unique(y_train))
```

```
Training Scores Mean: [0.59782197 0.60499398 0.59431311 0.59681187 0.60415049
0.59929803
 0.60465099 0.60574698 0.60678866 0.60696093]
Validation Scores Mean: [0.5980153  0.60660529 0.60665973 0.60713637 0.60778897
0.60783329
 0.60594227 0.60727262 0.60645564 0.6063074 ]
Shape of X_train: (26064, 11)
Unique classes in y_train: [0. 1.]
```

[13]:
```
import numpy as np
import matplotlib.pyplot as plt


train_scores_mean = np.array([0.59782197, 0.60499398, 0.59431311, 0.59681187,
                              0.60415049, 0.59929803, 0.60465099, 0.60574698,
                              0.60678866, 0.60696093])
valid_scores_mean = np.array([0.5980153, 0.60660529, 0.60665973, 0.60713637,
                              0.60778897, 0.60783329, 0.60594227, 0.60727262,
                              0.60645564, 0.6063074])
train_sizes = np.linspace(0.1, 1.0, len(train_scores_mean))  # Simulating␣
 ↪training sizes

# Calculate errors
train_errors = 1 - train_scores_mean
valid_errors = 1 - valid_scores_mean

# Plotting the errors

plt.plot(train_sizes, train_errors, label='Training Error', color='blue',␣
 ↪marker='o')

plt.plot(train_sizes, valid_errors, label='Validation Error', color='orange',␣
 ↪marker='o')

# Adding F1 Score as a secondary axis
f1_scores_train = train_scores_mean
f1_scores_valid = valid_scores_mean


ax2 = plt.gca().twinx()
ax2.plot(train_sizes, f1_scores_train, label='Training F1 Score',␣
 ↪color='green', linestyle='--')
```

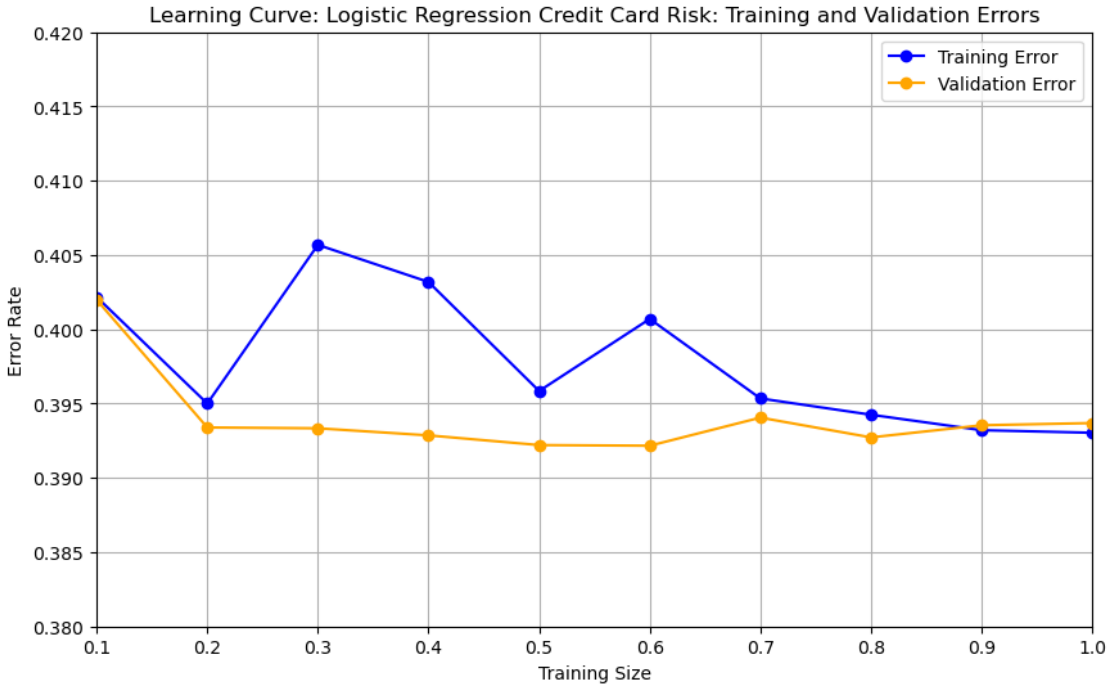
```
ax2.plot(train_sizes, f1_scores_valid, label='Validation F1 Score',␣
 ↪color='red', linestyle='--')

#Plotting
plt.title('Learning Curve: Training and Validation Errors with F1 Scores')
plt.xlabel('Training Size')
plt.ylabel('Error Rate')
plt.xlim([0.1, 1.0])
plt.ylim([0, 0.5])
plt.grid()
plt.legend(loc='upper left')


ax2.set_ylabel('F1 Score')
ax2.set_ylim([0, 1])
ax2.legend(loc='upper right')
```

[13]: <matplotlib.legend.Legend at 0x70f818d84c10>



1.7 Training and Validation Error Curve

[14]:
```
plt.show()
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_errors, label='Training Error', color='blue',␣
 ↪marker='o')
plt.plot(train_sizes, valid_errors, label='Validation Error', color='orange',␣
 ↪marker='o')

plt.title('Learning Curve: Logistic Regression Credit Card Risk: Training and␣
 ↪Validation Errors')
plt.xlabel('Training Size')
plt.ylabel('Error Rate')
plt.xlim([0.1, 1.0])
plt.ylim([0.38, 0.42])
plt.grid()
plt.legend()
plt.show()
```



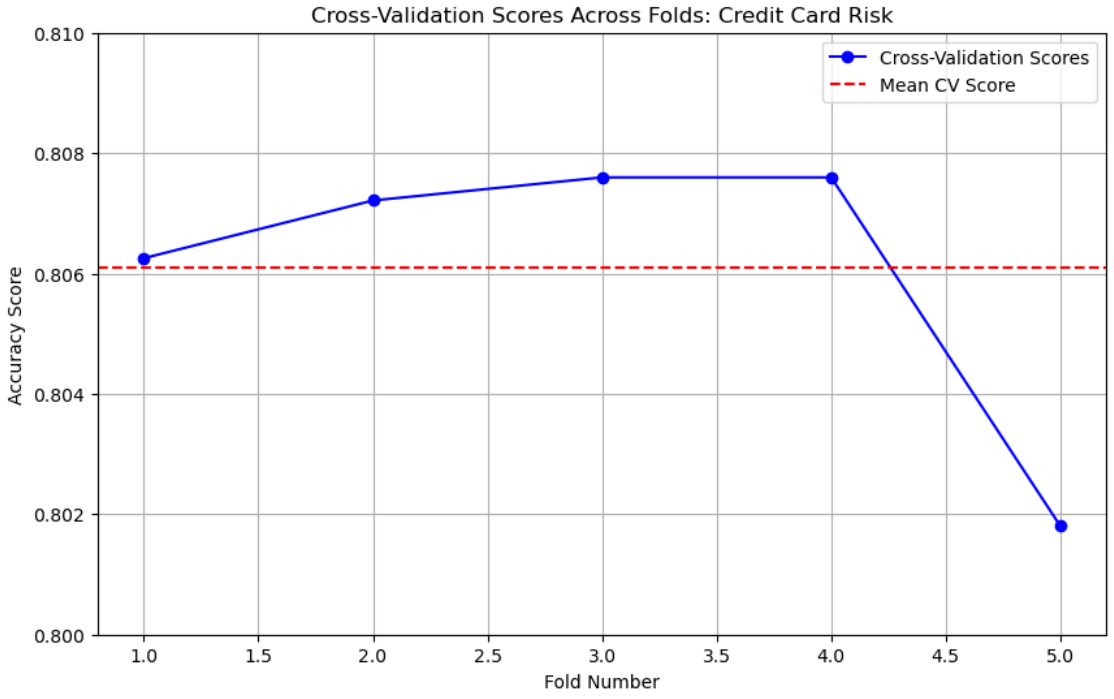
1.8 Cross Validation Scores Across Folds

[30]:
```
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, y_train, cv=5)
print("Cross-Validation Scores:", cv_scores)
print("Mean Cross-Validation Score:", np.mean(cv_scores))
```

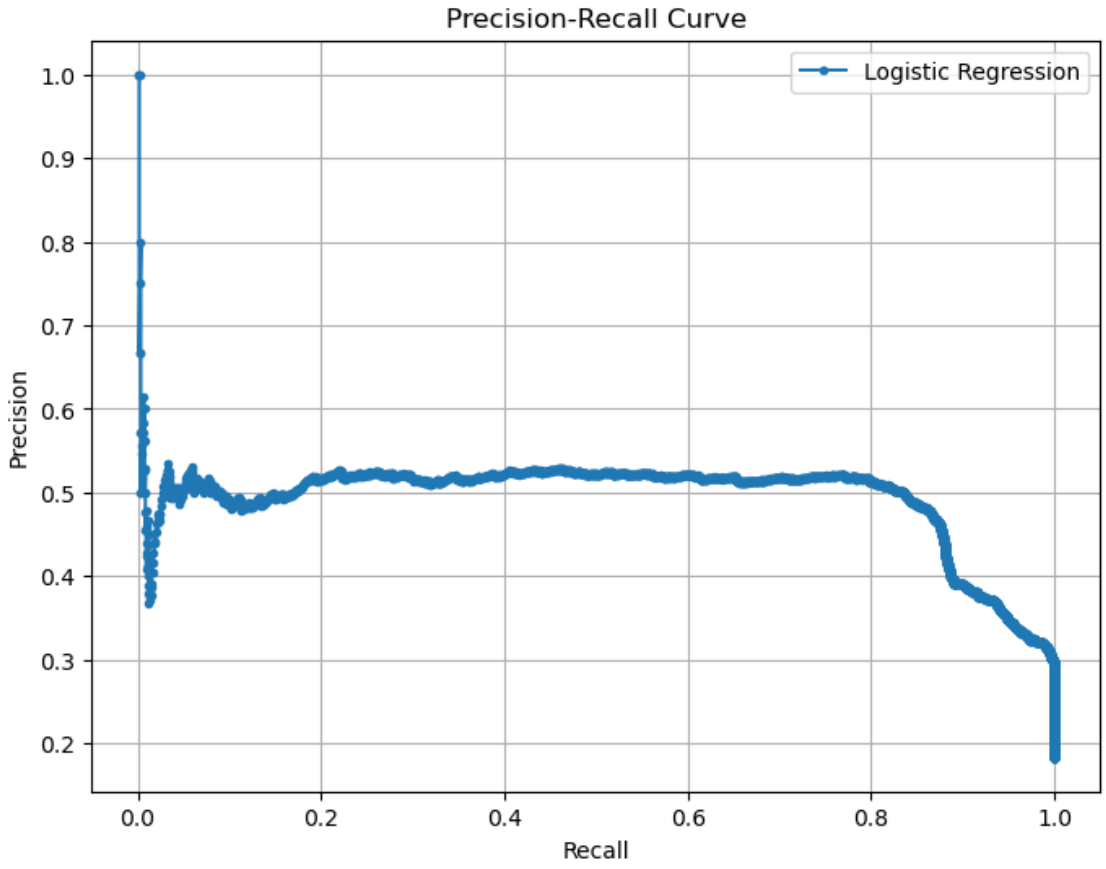
```
Cross-Validation Scores: [0.8062536  0.80721274 0.80759639 0.80759639
0.80180353]
Mean Cross-Validation Score: 0.8060925303483719
```

[35]:
```
cv_scores = [0.8062536, 0.80721274, 0.80759639, 0.80759639, 0.80180353]
mean_cv_score = 0.8060925303483719


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, 0.81)
plt.legend()
plt.grid(True)
plt.show()
```



1.9 Additional Visuals

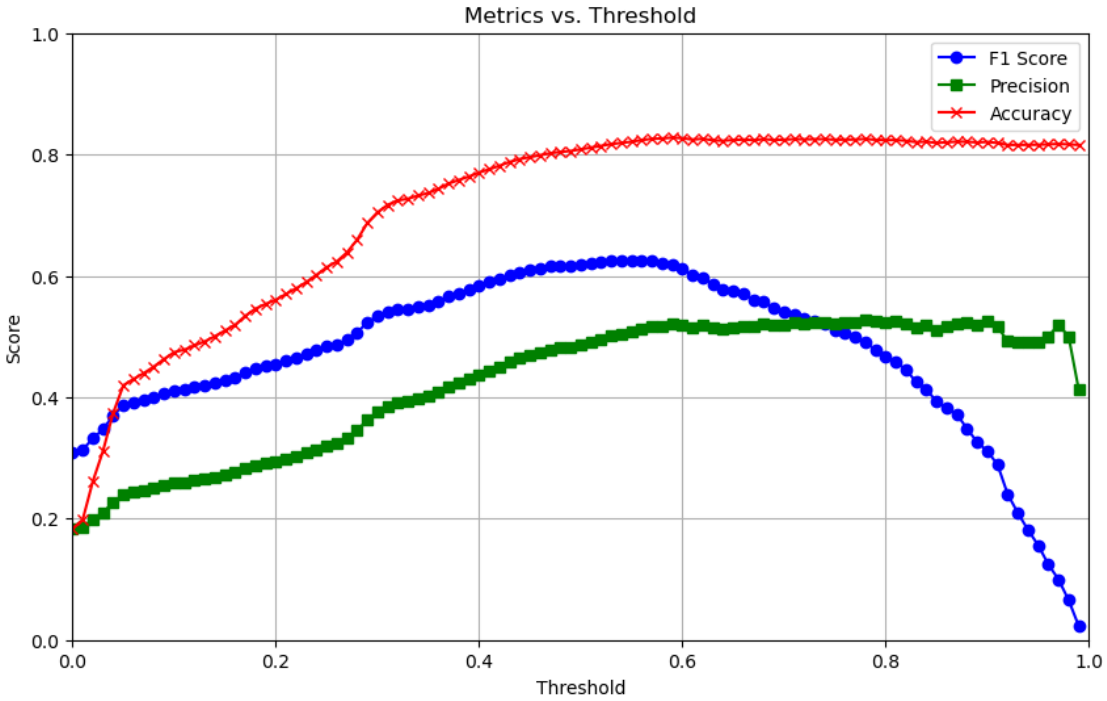
[8]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve, auc

y_prob = best_model.predict_proba(X_test)[:, 1]

# Compute precision and recall values for different thresholds
precision, recall, _ = precision_recall_curve(y_test, y_prob)

# Plot the Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, marker='.', label='Logistic Regression')


plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid(True)
plt.show()


pr_auc = auc(recall, precision)
print(f'Precision-Recall AUC: {pr_auc:.4f}')
```




```
Precision-Recall AUC: 0.4942
```

[9]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score, precision_score, accuracy_score


y_scores = best_model.predict_proba(X_test)[:, 1]

# Define thresholds
thresholds = np.arange(0.0, 1.0, 0.01)
f1_scores = []
precision_scores = []
accuracy_scores = []

# Calculate metrics for each threshold
for threshold in thresholds:

    y_pred_threshold = (y_scores >= threshold).astype(int)
    f1_scores.append(f1_score(y_test, y_pred_threshold))
```

```
    precision_scores.append(precision_score(y_test, y_pred_threshold))
    accuracy_scores.append(accuracy_score(y_test, y_pred_threshold))

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(thresholds, f1_scores, marker='o', label='F1 Score', color='blue')
plt.plot(thresholds, precision_scores, marker='s', label='Precision',␣
 ↪color='green')
plt.plot(thresholds, accuracy_scores, marker='x', label='Accuracy', color='red')

plt.title('Metrics vs. Threshold')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.grid()
plt.legend()
plt.show()
```



[ ]: