CR_Perceptron_Final

October 7, 2024

1 Credit Risk - Perceptron Algorithm

[2]:
```
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪StratifiedKFold, cross_val_score, learning_curve
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪ConfusionMatrixDisplay, accuracy_score, roc_curve, auc, roc_auc_score,␣
 ↪precision_recall_curve, make_scorer, f1_score
from sklearn.preprocessing import StandardScaler
from scipy.special import expit
```

[4]:
```
# Load the Dataset
df = pd.read_csv('LR_df.csv')
df.head()
```

[4]:

	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

[6]:
```
df.drop('Unnamed: 0', axis=1, inplace=True)
df.head()
```

[6]:
```
    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
```

[8]:
```
df.fillna(0, inplace=True)
```

[10]:
```
X = df.drop('Default', axis=1)
y = df['Default']
```

[12]:
```
# Check class distribution
print("Class distribution before split:")
print(y.value_counts())
```

```
Class distribution before split:
Default
0.0    26837
1.0     5745
Name: count, dtype: int64
```

[14]:
```
# 80-Train/20-Test w/ Stratified Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)
```

[16]:
```
# Check class distribution
print("Class distribution in training set:")
print(y_train.value_counts())
print("Class distribution in test set:")
print(y_test.value_counts())
```

```
Class distribution in training set:
Default
0.0    21469
1.0     4596
Name: count, dtype: int64
Class distribution in test set:
Default
0.0    5368
```

```
1.0    1149
Name: count, dtype: int64
```

[18]:
```
# Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[20]:
```
X_train_scaled
```

[20]:
```
array([[-0.91018552, -0.3278337 , -1.13485092, …, -0.05787044,
        -0.29046502, -1.01456997],
       [-0.27230679,  0.70246322, -1.13485092, …, -0.05787044,
        -0.29046502,  0.98563926],
       [ 1.00345066, -0.80335536,  1.62238709, …, -0.05787044,
        -0.29046502,  0.98563926],
       …,
       [-0.59124616,  1.65350652,  0.26482807, …, -0.05787044,
         3.44275531, -1.01456997],
       [-0.27230679,  2.2875354 ,  1.16554742, …, -0.05787044,
        -0.29046502, -1.01456997],
       [-0.75071584, -0.3278337 , -0.00733173, …, -0.05787044,
         3.44275531, -1.01456997]])
```

[22]:
```
# Convert back to DataFrame to retain feature names
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)
```

[23]:
```
X_train_scaled
```

[23]:

	Age	Amount	Rate	Status	Percent_income	Cred_length	\
0	2.105372	0.192352	-1.807870	-0.527111	-0.090112	1.778201	
1	1.470844	-0.568665	-0.547252	-0.527111	-0.654595	2.519002	
2	-0.274105	-0.251574	-1.251905	-0.527111	-0.466434	-0.938071	
3	-0.432737	0.382607	1.608728	1.897133	-0.560515	-0.691138	
4	-0.591369	-1.202846	-0.007449	-0.527111	-1.030917	-0.938071	
…	…	…	…	…	…	…	
22802	-0.115474	-0.378411	-1.138772	1.897133	2.167820	0.790465	
22803	-0.274105	0.699697	-0.505231	1.897133	1.509257	-0.938071	
22804	-0.591369	1.650968	0.264069	-0.527111	0.662532	-0.691138	
22805	-0.274105	2.285149	1.162664	1.897133	0.192129	-0.691138	
22806	-0.750001	-0.330847	-0.007449	-0.527111	0.944773	-0.444204	

	Home_MORTGAGE	Home_OTHER	Home_OWN	Home_RENT
0	-0.837484	-0.058581	-0.289976	0.986542
1	1.194052	-0.058581	-0.289976	-1.013642
2	-0.837484	-0.058581	-0.289976	0.986542

```
3          1.194052   -0.058581 -0.289976  -1.013642
4         -0.837484   -0.058581 -0.289976   0.986542
…               …          …         …          …
22802     -0.837484   -0.058581 -0.289976   0.986542
22803     -0.837484   -0.058581 -0.289976   0.986542
22804     -0.837484   -0.058581  3.448564  -1.013642
22805      1.194052   -0.058581 -0.289976  -1.013642
22806     -0.837484   -0.058581  3.448564  -1.013642

[22807 rows x 10 columns]
```

[24]:
```
# Hyperparameter Grid for the Perceptron
param_grid = {
    'max_iter': [10, 20, 25, 50, 100],
    'eta0': [0.001, 0.01, 0.1],
    'penalty': [None, 'l2', 'l1'],
}
```

[30]:
```
# Stratified K-Fold Cross-Validation
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Start the timer
start_time = time.time()

# Model Training / Grid Search for Hyperparameter Tuning
grid_search = GridSearchCV(Perceptron(random_state=42,␣
 ↪class_weight='balanced'), param_grid, scoring='roc_auc', cv=skf, verbose=1,␣
 ↪return_train_score=True)

# Fit the model
grid_search.fit(X_train_scaled, y_train)

# End the timer
end_time = time.time()
```

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

/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
```

```
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
```

```
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
```

[32]:

```
# Output the best parameters and time taken
best_params = grid_search.best_params_
print(f'Best parameters found: {best_params}')

training_time = end_time - start_time
print(f"Time taken for grid search: {training_time:.2f} seconds")
```

```
Best parameters found: {'eta0': 0.01, 'max_iter': 20, 'penalty': 'l1'}
Time taken for grid search: 7.00 seconds
```

[34]:
```
# Train the model with the best parameters
p_model = grid_search.best_estimator_
p_model.fit(X_train_scaled, y_train)
```

[34]:
```
Perceptron(class_weight='balanced', eta0=0.01, max_iter=20, penalty='l1',
           random_state=42)
```

[36]:
```
# Make Predictions on the Test Dataset
y_pred = p_model.predict(X_test_scaled)
y_scores = p_model.decision_function(X_test_scaled)

results = grid_search.cv_results_
p_model.fit(X_train_scaled, y_train)
```

[36]:
```
Perceptron(class_weight='balanced', eta0=0.01, max_iter=20, penalty='l1',
           random_state=42)
```

[38]:
```
# Training and Test Accuracy
train_accuracy = accuracy_score(y_train, p_model.predict(X_train_scaled))
test_accuracy = accuracy_score(y_test, p_model.predict(X_test_scaled))
print(f"Training accuracy: {train_accuracy:.2f}")
print(f"Validation accuracy: {test_accuracy:.2f}")
```

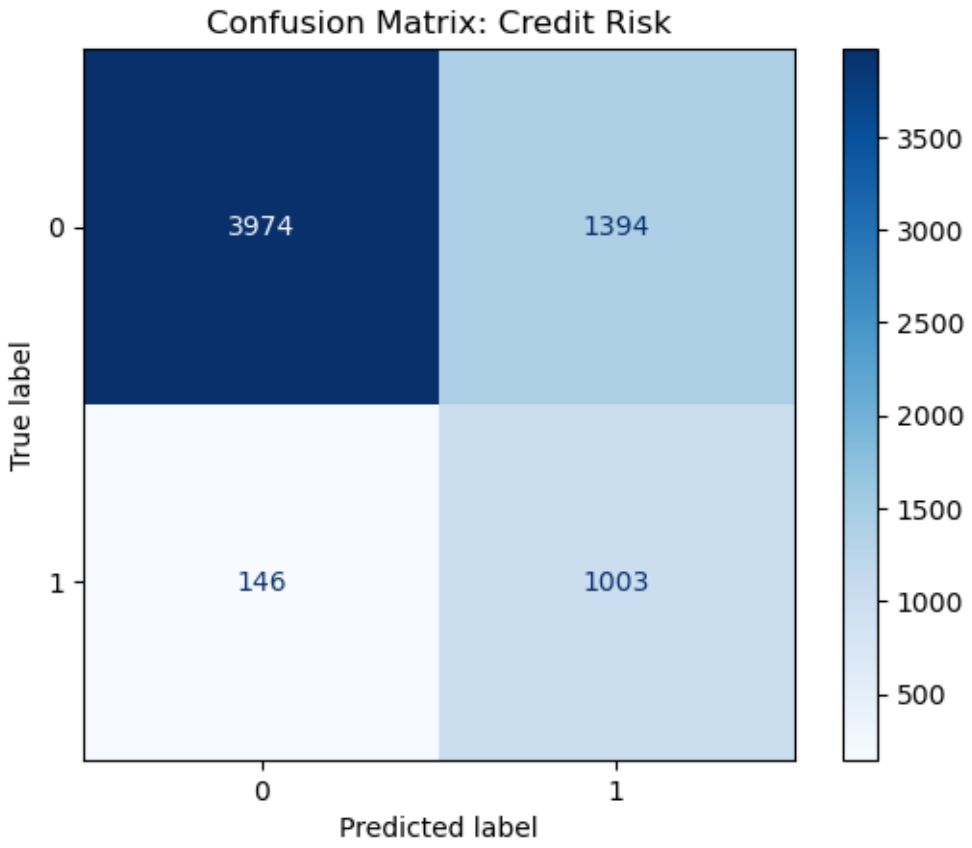
```
Training accuracy: 0.76
Validation accuracy: 0.76
```

[40]:
```
# Evaluating Model Performance - Accuracy Test
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.76
```

[42]:
```
# Evaluating Model Performance - Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix: Credit Risk')
plt.show()
```

```
Confusion Matrix:
[[3974 1394]
 [ 146 1003]]
```



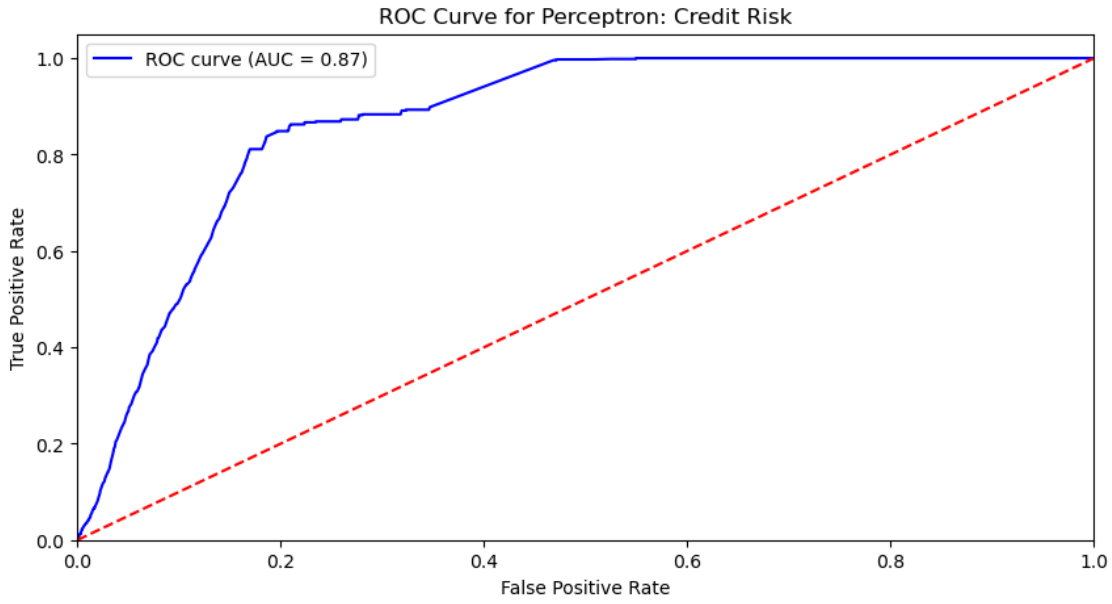
[44]:
```
# Evaluating Model Performance - Classification Report
print("Classification Report: Credit Risk")
print(classification_report(y_test, y_pred))
```

```
Classification Report: Credit Risk
              precision    recall  f1-score   support

         0.0       0.96      0.74      0.84      5368
         1.0       0.42      0.87      0.57      1149

    accuracy                           0.76      6517
   macro avg       0.69      0.81      0.70      6517
weighted avg       0.87      0.76      0.79      6517
```

[46]:
```
# Calculate AUC
auc_score = roc_auc_score(y_test, y_scores)
print(f"Best Perceptron Model AUC: {auc_score:.2f}")
```

```
Best Perceptron Model AUC: 0.87
```

[48]:
```
# ROC Curve
y_scores = p_model.decision_function(X_test_scaled)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)

# Plot ROC Curve
plt.figure(figsize=(10, 5))
plt.plot(fpr, tpr, color='blue', label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='red', linestyle='--')  # Random guessing line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.title("ROC Curve for Perceptron: Credit Risk")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



[50]:
```
# Extract mean training and validation scores
results = grid_search.cv_results_
mean_train_scores = results['mean_train_score']
mean_test_scores = results['mean_test_score']
param_values_max_iter = results['param_max_iter']
param_values_eta0 = results['param_eta0']
param_values_penalty = results['param_penalty']

# DataFrame for easier aggregation
results_df = pd.DataFrame({
```

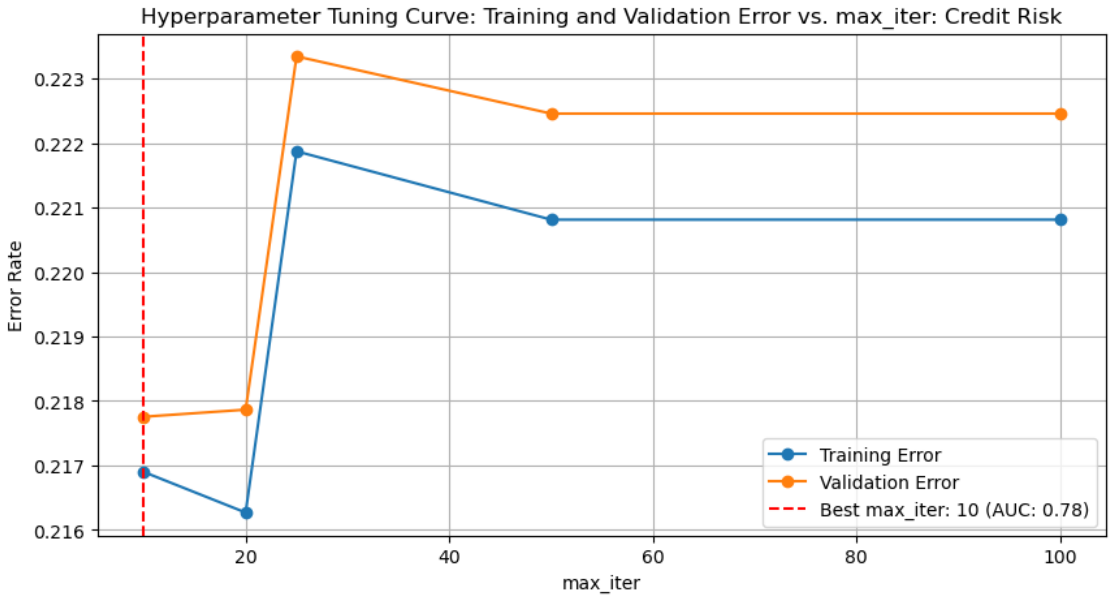
```
    'max_iter': param_values_max_iter,
    'eta0': param_values_eta0,
    'penalty': param_values_penalty,
    'mean_train_score': mean_train_scores,
    'mean_test_score': mean_test_scores
})
```

[52]:
```
# Function to aggregate results and calculate errors
def aggregate_results(param_name):
    aggregated_results = results_df.groupby(param_name).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']

    return aggregated_results

# Aggregate results for max_iter, eta0, and penalty
aggregated_results_max_iter = aggregate_results('max_iter')
aggregated_results_eta0 = aggregate_results('eta0')
aggregated_results_penalty = aggregate_results('penalty')
```

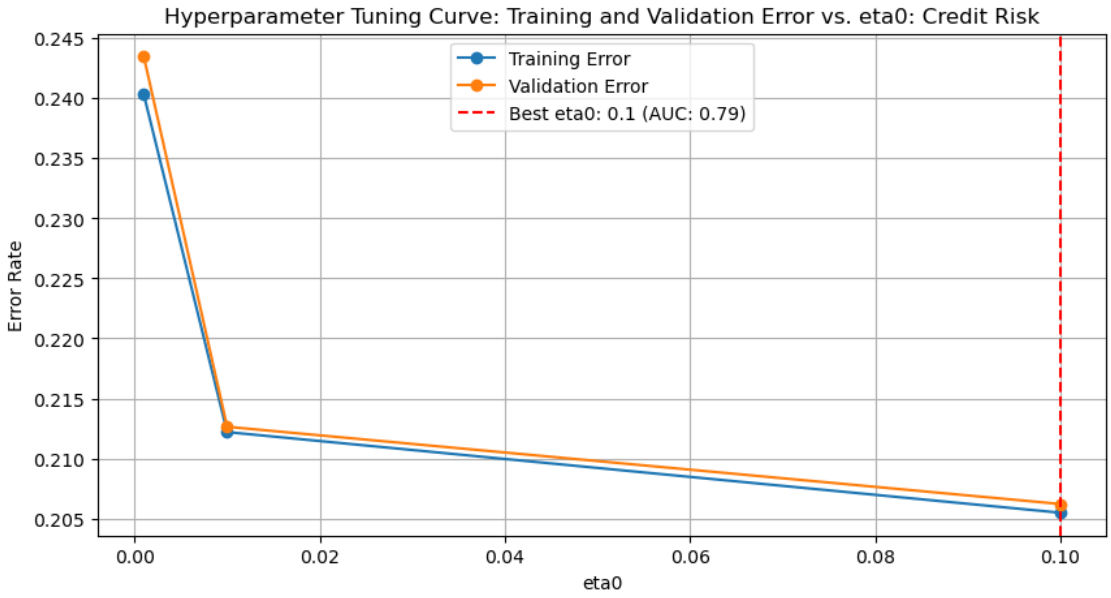
[54]:
```
# Function to plot results
def plot_results(aggregated_results, param_name, best_param):
    best_index = aggregated_results['mean_test_score'].idxmax()
    best_value = aggregated_results[param_name][best_index]
    best_mean_test_score = aggregated_results['mean_test_score'][best_index]

    plt.figure(figsize=(10, 5))
    plt.plot(aggregated_results[param_name], aggregated_results['train_error'],␣
 ↪marker='o', label='Training Error')
    plt.plot(aggregated_results[param_name],␣
 ↪aggregated_results['validation_error'], marker='o', label='Validation Error')


    # Best param line
    plt.axvline(x=best_value, color='r', linestyle='--', label=f'Best␣
 ↪{param_name}: {best_value} (AUC: {best_mean_test_score:.2f})')

    plt.title(f"Hyperparameter Tuning Curve: Training and Validation Error vs.␣
 ↪{param_name}: Credit Risk")
```

```
    plt.xlabel(param_name)
    plt.ylabel("Error Rate")
    plt.legend()
    plt.grid()
    plt.show()
```

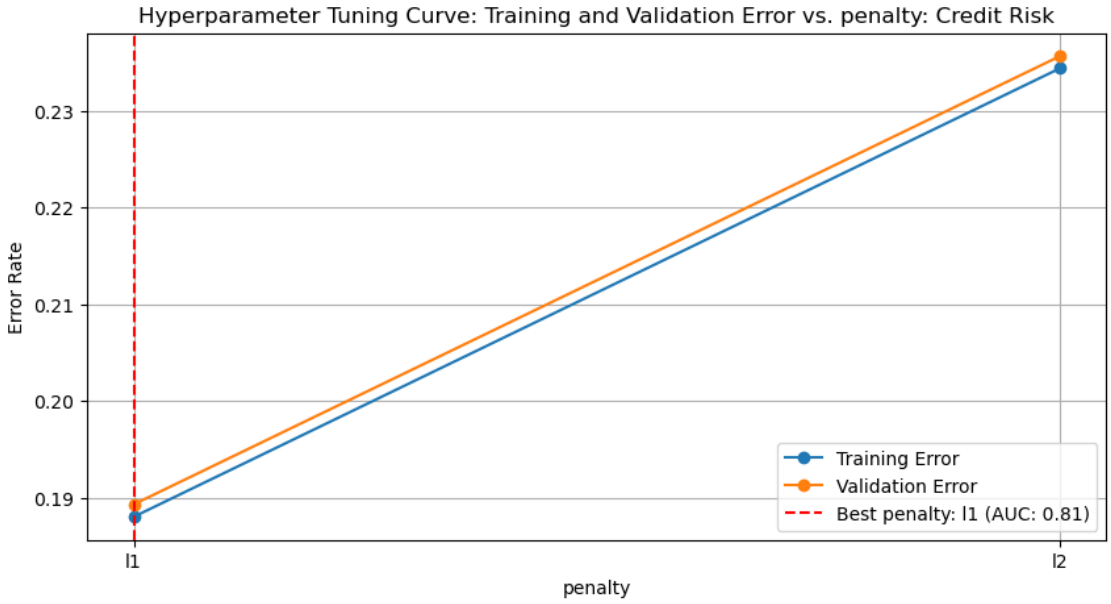
[56]:
```
# Hyperparameter Tuning Curve - max_iter
plot_results(aggregated_results_max_iter, 'max_iter', 'best_max_iter')
```



[58]:
```
# Hyperparameter Tuning Curve - eta0
plot_results(aggregated_results_eta0, 'eta0', 'best_eta0')
```

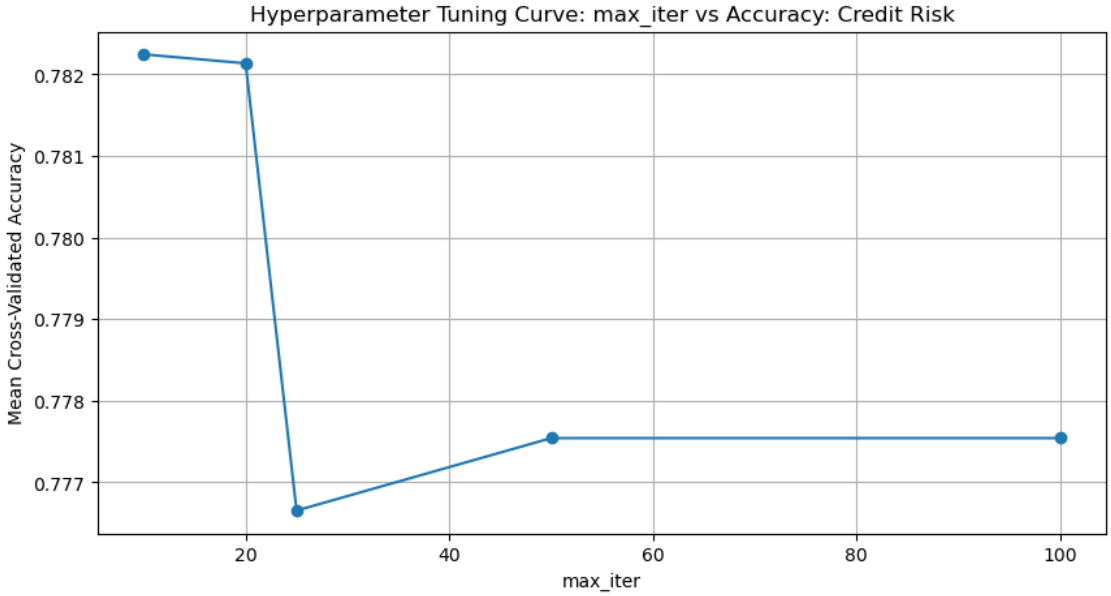


[60]:
```
# Hyperparameter Tuning Curve - penalty
plot_results(aggregated_results_penalty, 'penalty', 'best_penalty')
```

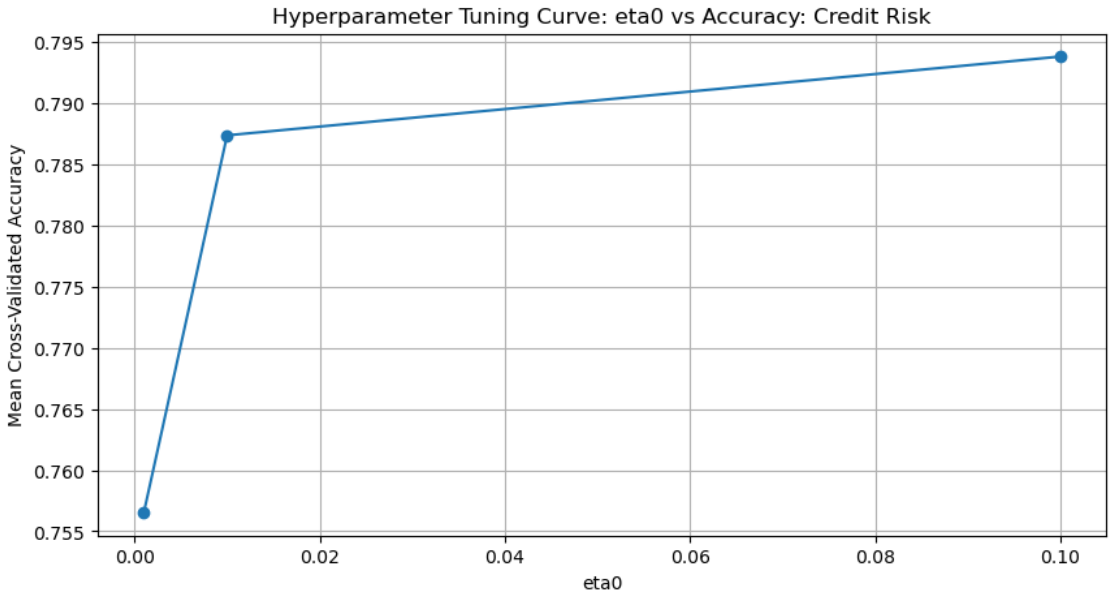


[62]:
```
# Hyperparameter Tuning Curve max_iter vs. Mean Cross-Validated Accuracy
plt.figure(figsize=(10, 5))
plt.plot(aggregated_results_max_iter['max_iter'],␣
 ↪aggregated_results_max_iter['mean_test_score'], marker='o')
```

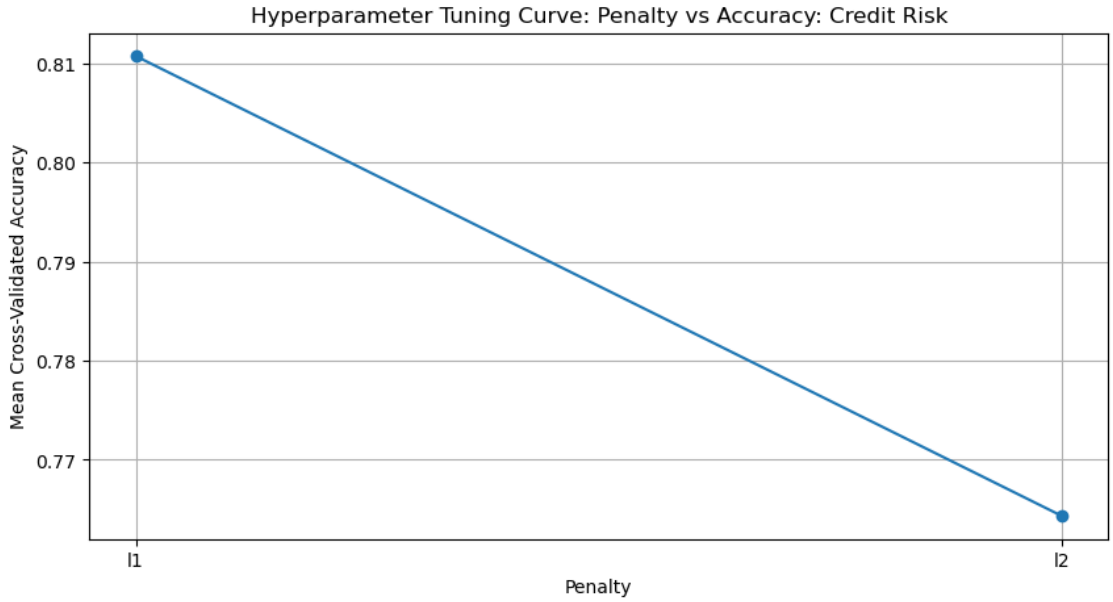
```
plt.title("Hyperparameter Tuning Curve: max_iter vs Accuracy: Credit Risk")
plt.xlabel("max_iter")
plt.ylabel("Mean Cross-Validated Accuracy")
plt.grid()
plt.xticks()
plt.show()
```



[64]:
```
# Hyperparameter Tuning Curve eta0 vs. Mean Cross-Validated Accuracy
plt.figure(figsize=(10, 5))
plt.plot(aggregated_results_eta0['eta0'],␣
 ↪aggregated_results_eta0['mean_test_score'], marker='o')
plt.title("Hyperparameter Tuning Curve: eta0 vs Accuracy: Credit Risk")
plt.xlabel("eta0")
plt.ylabel("Mean Cross-Validated Accuracy")
plt.grid()
plt.xticks()
plt.show()
```



[66]:
```
# Hyperparameter Tuning Curve penalty vs. Mean Cross-Validated Accuracy
plt.figure(figsize=(10, 5))
plt.plot(aggregated_results_penalty['penalty'],␣
 ↪aggregated_results_penalty['mean_test_score'], marker='o')
plt.title("Hyperparameter Tuning Curve: Penalty vs Accuracy: Credit Risk")
plt.xlabel("Penalty")
plt.ylabel("Mean Cross-Validated Accuracy")
plt.grid()
plt.xticks()
plt.show()
```

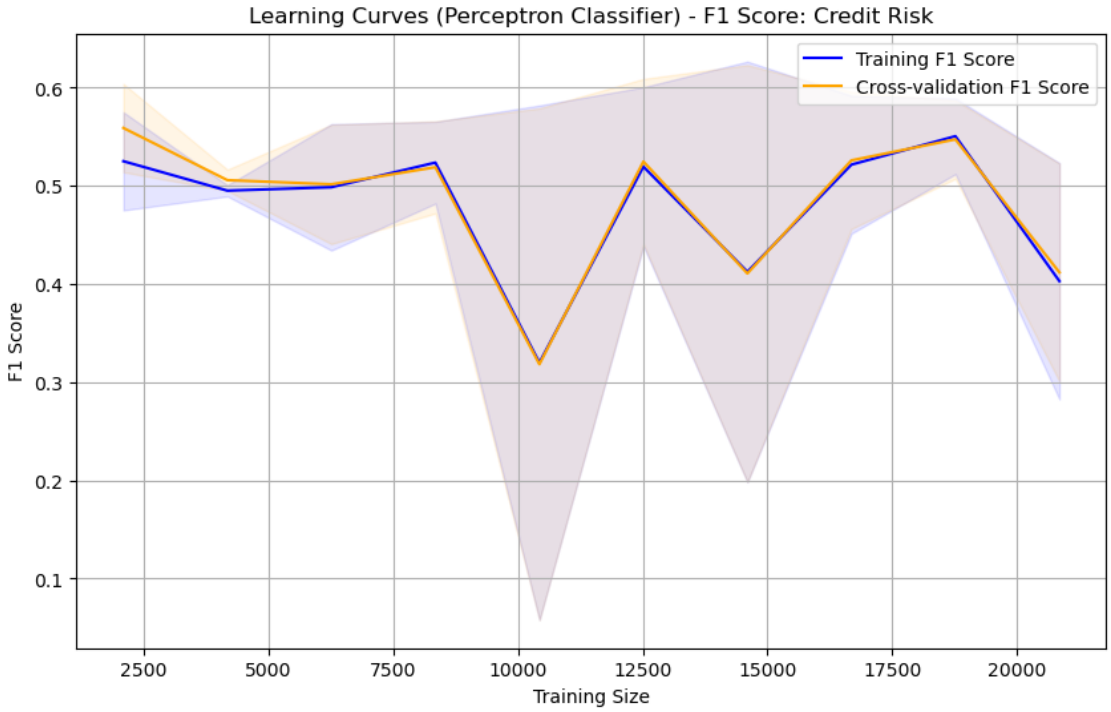


[68]:
```
# Create the F1 score scorer
f1_scorer = make_scorer(f1_score)

# Generate learning curves using F1 score
train_sizes, train_scores, test_scores = learning_curve(
    p_model, X_train_scaled, y_train,  # Use scaled training data
    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)
```

```python
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std,␣
 ↪color='orange', alpha=0.1)

# Add labels and title
plt.title('Learning Curves (Perceptron Classifier) - F1 Score: Credit Risk')
plt.xlabel('Training Size')
plt.ylabel('F1 Score')
plt.legend(loc='best')
plt.grid()
plt.show()
```

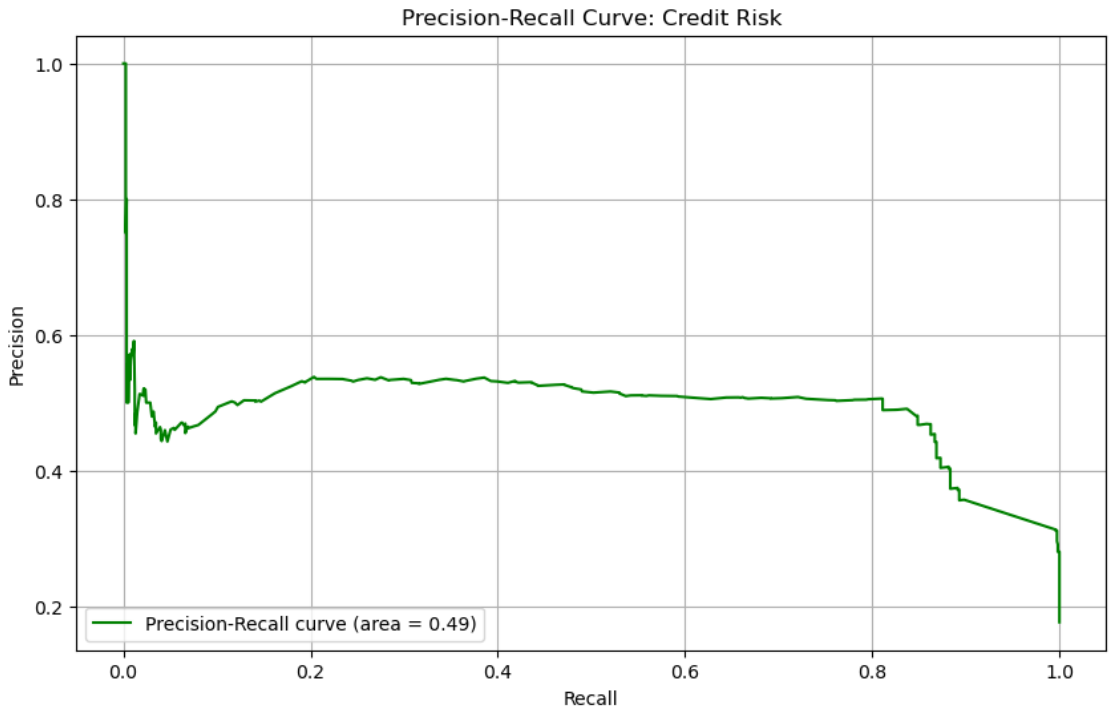
```
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
```

```
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
```



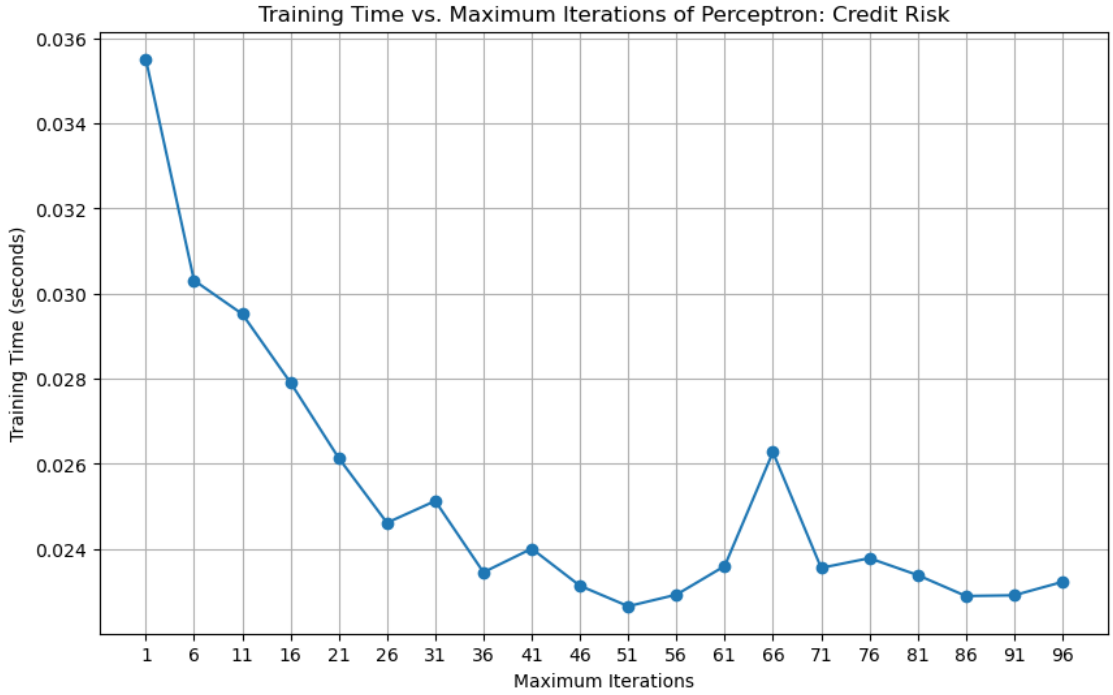
[70]:
```
# Get decision function scores
decision_scores = p_model.decision_function(X_test_scaled)

# Convert decision scores to probabilities
y_probs = expit(decision_scores)

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

# Plot the Precision-Recall Curve
plt.figure(figsize=(10, 6))
plt.plot(recall, precision, color='green', label=f'Precision-Recall curve (area␣
 ↪= {pr_auc:.2f})')
```

```
plt.title('Precision-Recall Curve: Credit Risk')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend(loc='lower left')
plt.grid()
plt.show()
```



[90]:
```
# Timing Analysis: measuring training time

# Maximum iterations to evaluate
max_iters = np.arange(1, 101, 5)  # from 1 to 100, step of 5
training_times = []

# Measure training time for different max iterations
for max_iter in max_iters:
    start_time = time.time()
    p_model.fit(X_train_scaled, y_train)
    end_time = time.time()
    training_times.append(end_time - start_time)

# Plotting the training time vs. max iterations
plt.figure(figsize=(10, 6))
plt.plot(max_iters, training_times, marker='o')
plt.title('Training Time vs. Maximum Iterations of Perceptron: Credit Risk')
```

```
plt.xlabel('Maximum Iterations')
plt.ylabel('Training Time (seconds)')
plt.xticks(max_iters)
plt.grid()
plt.show()
```



[88]:
```
print("Min. Training Time:",min(training_times))
```

```
Min. Training Time: 0.011962890625
```

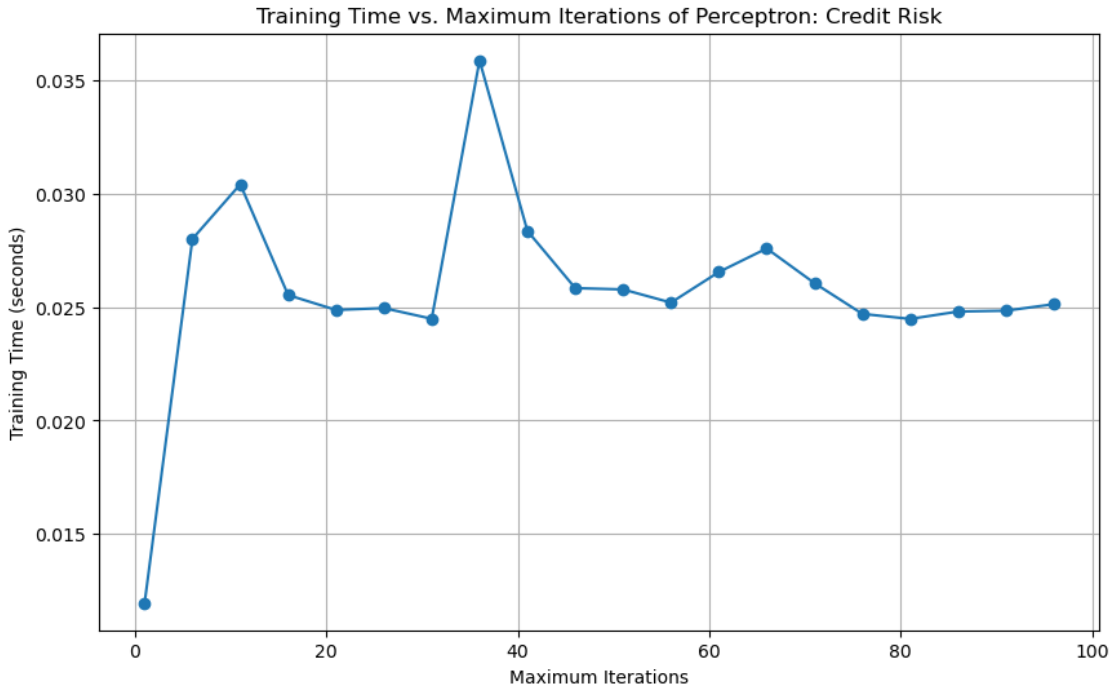
[84]:
```
print("Max. Training Time:",max(training_times))
```

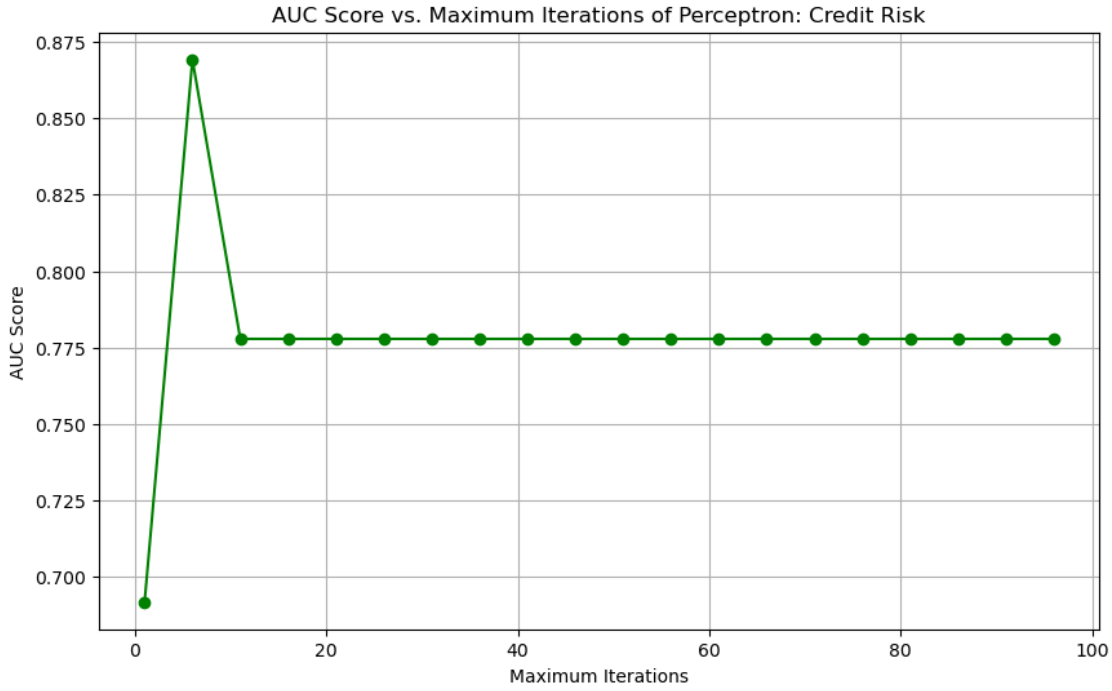
```
Max. Training Time: 0.03585410118103027
```

[74]:
```
max_iters = np.arange(1, 101, 5)
training_times = []
auc_scores = []

for max_iter in max_iters:
    start_time = time.time()
    p_model = Perceptron(penalty='l1', max_iter=max_iter, eta0=0.01,␣
 ↪class_weight='balanced')
    p_model.fit(X_train_scaled, y_train)
    end_time = time.time()
```

```
    # Calculate training time
    training_times.append(end_time - start_time)

    # Predict and calculate AUC score
    y_pred_proba = p_model.decision_function(X_test_scaled)
    auc_scores.append(roc_auc_score(y_test, y_pred_proba))

# Plotting training time
plt.figure(figsize=(10, 6))
plt.plot(max_iters, training_times, marker='o', label="Training Time (seconds)")
plt.title("Training Time vs. Maximum Iterations of Perceptron: Credit Risk")
plt.xlabel("Maximum Iterations")
plt.ylabel("Training Time (seconds)")
plt.grid(True)
plt.show()

# Plotting AUC score vs max_iter
plt.figure(figsize=(10, 6))
plt.plot(max_iters, auc_scores, marker='o', color='green', label="AUC Score")
plt.title("AUC Score vs. Maximum Iterations of Perceptron: Credit Risk")
plt.xlabel("Maximum Iterations")
plt.ylabel("AUC Score")
plt.grid(True)
plt.show()
```

```
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_stochastic_gradient.py:723: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
```





[76]:
```
# Evaluate the model with cross-validation
cv_scores = cross_val_score(p_model, X_train_scaled, y_train, cv=5)
```

```
# Calculate the mean cross-validation score
mean_cv_score = np.mean(cv_scores)

# Output the scores and their mean
print("Cross-Validation Scores:", cv_scores)
print("Mean Cross-Validation Score:", mean_cv_score)

# Plotting the cross-validation scores
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 Risk')
plt.xlabel('Fold Number')
plt.ylabel('Accuracy Score')
plt.ylim(0.0, 1.0)  # Adjusted to include the full range of accuracy scores
plt.legend()
plt.grid(True)
plt.show()
```

```
Cross-Validation Scores: [0.67101477 0.82275082 0.82006522 0.76769614
0.50892001]
Mean Cross-Validation Score: 0.7180893919048532
```