HI_Perceptron

October 7, 2024

1 Health Insurance - Perceptron Algorithm

[95]:
```
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
```

[97]:
```
# Load the Dataset
df = pd.read_csv('cleaned_data.csv')
df.head()
```

[97]:

	DentalPlan	IssuerID_2014	MultistatePlan_2014	ChildAdultOnly_2014	\
0	1	21989	0	0	
1	1	21989	0	0	
2	1	21989	0	0	
3	1	21989	0	0	
4	1	21989	0	0	

	IssuerID_2015	MultistatePlan_2015	IssuerID_AgeOff2015
0	21989	0	0
1	21989	0	0
2	21989	0	0
3	21989	0	0
4	21989	0	0

[99]:
```
X = df.drop('DentalPlan', axis=1)
y = df['DentalPlan']
```

[101]:
```
# Check class distribution
print("Class distribution before split:")
print(y.value_counts())
```

```
Class distribution before split:
DentalPlan
0    81629
1    50876
Name: count, dtype: int64
```

[103]:
```
# 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)
```

[105]:
```
# 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:
DentalPlan
0    65303
1    40701
Name: count, dtype: int64
Class distribution in test set:
DentalPlan
0    16326
1    10175
Name: count, dtype: int64
```

[107]:
```
# Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[109]:
```
X_train_scaled
```

[109]:
```
array([[-1.3403366 , -0.21868331, -0.44004832, -1.06821022, -0.38398064,
        -0.34416745],
       [-0.80364036, -0.21868331, -0.44004832, -0.58412942, -0.38398064,
        -0.34416745],
       [-0.50321929, -0.21868331, -0.44004832, -0.31316037, -0.38398064,
        -0.34416745],
       …,
       [ 1.7089619 , -0.21868331, -0.44004832,  1.68214787, -0.38398064,
        -0.34416745],
```

```
        [-0.8144387 , -0.21868331, -0.44004832, -0.59386914, -0.38398064,
         -0.34416745],
        [-1.43467996, -0.21868331, -0.44004832, -1.15330455, -0.38398064,
         -0.34416745]])
```

[111]:
```
# 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)
```

[113]:
```
X_train_scaled
```

[113]:

	IssuerID_2014	MultistatePlan_2014	ChildAdultOnly_2014	\
0	-1.340337	-0.218683	-0.440048	
1	-0.803640	-0.218683	-0.440048	
2	-0.503219	-0.218683	-0.440048	
3	-0.607338	-0.218683	1.804425	
4	0.923487	-0.218683	1.804425	
…	…	…	…	
105999	-0.334841	-0.218683	1.804425	
106000	-1.349657	-0.218683	-0.440048	
106001	1.708962	-0.218683	-0.440048	
106002	-0.814439	-0.218683	-0.440048	
106003	-1.434680	-0.218683	-0.440048	

	IssuerID_2015	MultistatePlan_2015	IssuerID_AgeOff2015
0	-1.068210	-0.383981	-0.344167
1	-0.584129	-0.383981	-0.344167
2	-0.313160	-0.383981	-0.344167
3	-1.546721	3.014916	-0.344167
4	0.973678	-0.383981	3.109263
…	…	…	…
105999	-0.161289	-0.383981	1.554140
106000	-1.546721	3.014916	-0.344167
106001	1.682148	-0.383981	-0.344167
106002	-0.593869	-0.383981	-0.344167
106003	-1.153305	-0.383981	-0.344167

[106004 rows x 6 columns]

[177]:
```
# 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'],
}
```

[179]:
```
# 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), 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(
```

[180]:
```
# 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': 0.001, 'max_iter': 10, 'penalty': 'l1'}
Time taken for grid search: 18.06 seconds
```

[181]:
```
# Train the model with the best parameters
p_model = grid_search.best_estimator_
p_model.fit(X_train_scaled, y_train)
```

[181]:
```
Perceptron(eta0=0.001, max_iter=10, penalty='l1', random_state=42)
```

[182]:
```
# 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)
```

[182]:
```
Perceptron(eta0=0.001, max_iter=10, penalty='l1', random_state=42)
```

[183]:
```
# 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}")
```

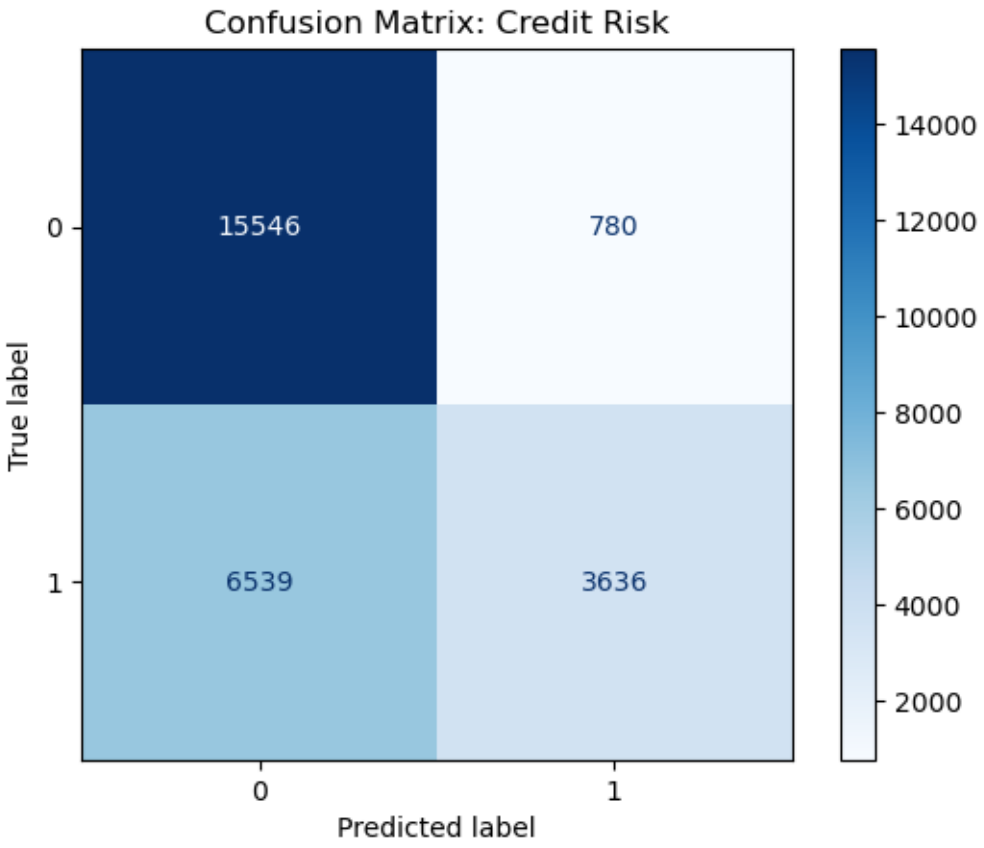
```
Training accuracy: 0.72
Validation accuracy: 0.72
```

[189]:
```
# Evaluating Model Performance - Accuracy Test
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.72
```

[191]:
```
# 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:
[[15546   780]
 [ 6539  3636]]
```



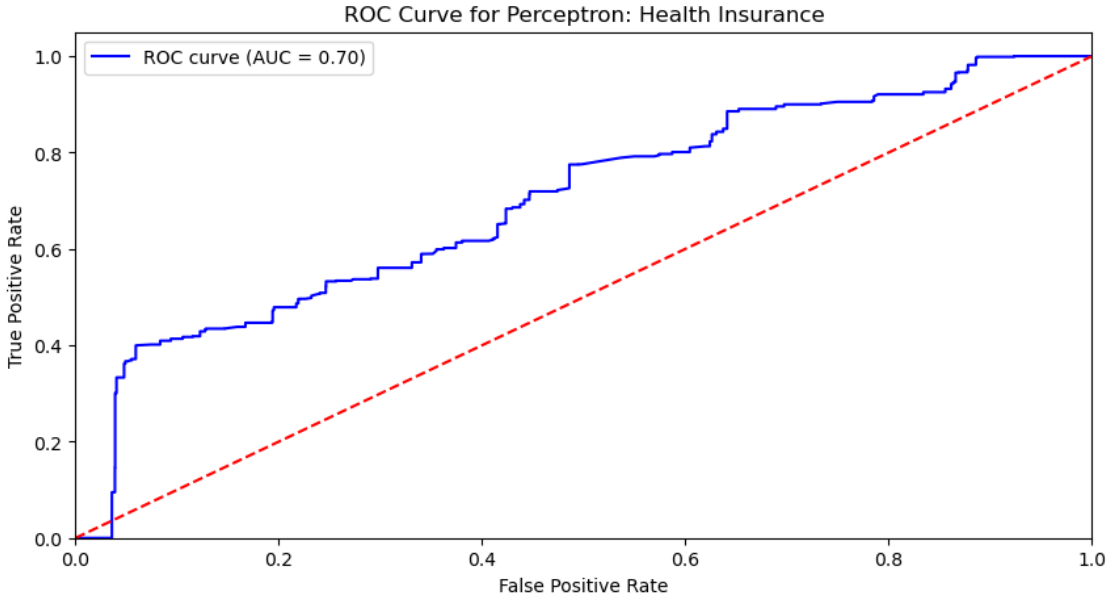
[193]:
```
# Evaluating Model Performance - Classification Report
print("Classification Report: Credit Risk")
print(classification_report(y_test, y_pred))
```

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

           0       0.70      0.95      0.81     16326
           1       0.82      0.36      0.50     10175

    accuracy                           0.72     26501
   macro avg       0.76      0.65      0.65     26501
weighted avg       0.75      0.72      0.69     26501
```

[195]:
```
# Calculate AUC
auc_score = roc_auc_score(y_test, y_scores)
print(f"Best Perceptron Model AUC: {auc_score:.2f}")
```

```
Best Perceptron Model AUC: 0.70
```

[199]:
```
# 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: Health Insurance")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



[201]:
```
# 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({
```

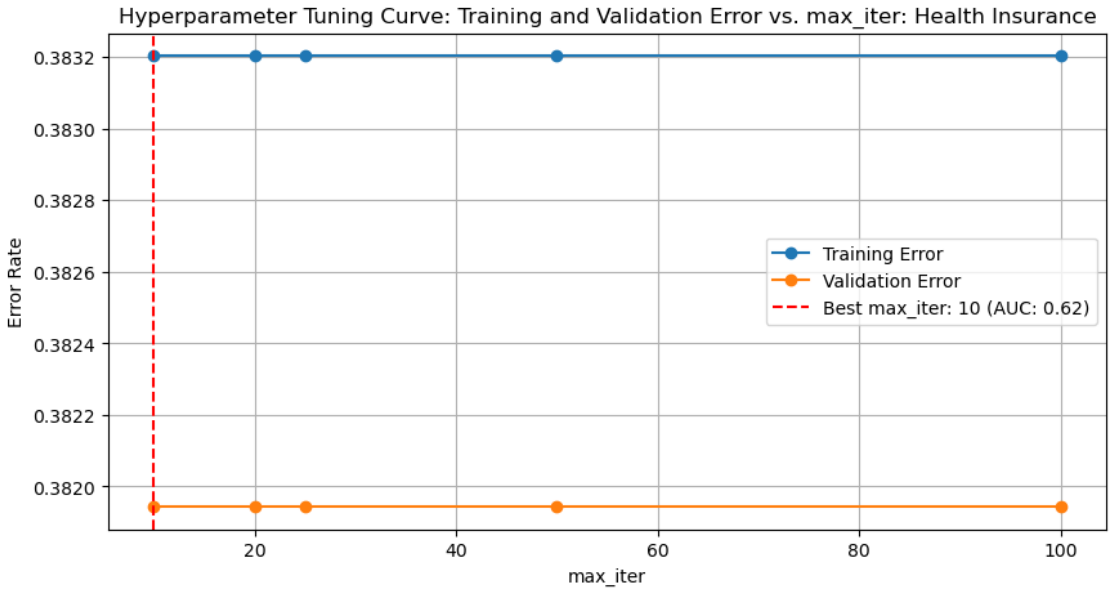
```
    'max_iter': param_values_max_iter,
    'eta0': param_values_eta0,
    'penalty': param_values_penalty,
    'mean_train_score': mean_train_scores,
    'mean_test_score': mean_test_scores
})
```

[203]:
```
# 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')
```

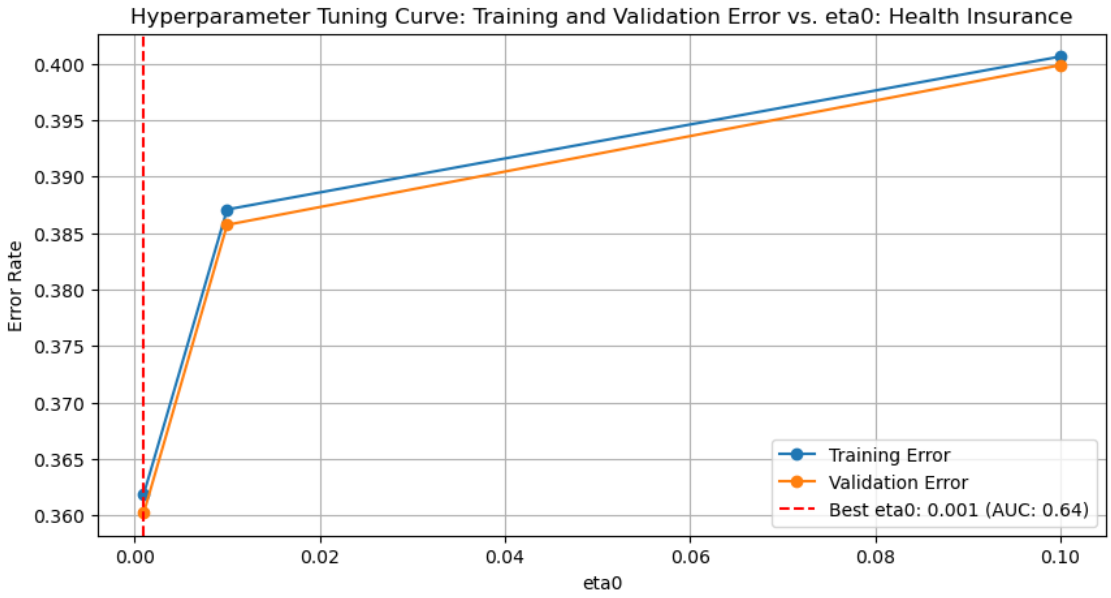
[205]:
```
# Function to plot results
def plot_results(aggregated_results, param_name, best_param):
    best_index = aggregated_results['mean_test_score'].idxmax()
    best_value = aggregated_results[param_name][best_index]
    best_mean_test_score = aggregated_results['mean_test_score'][best_index]

    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}: Health Insurance")
    plt.xlabel(param_name)
```

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

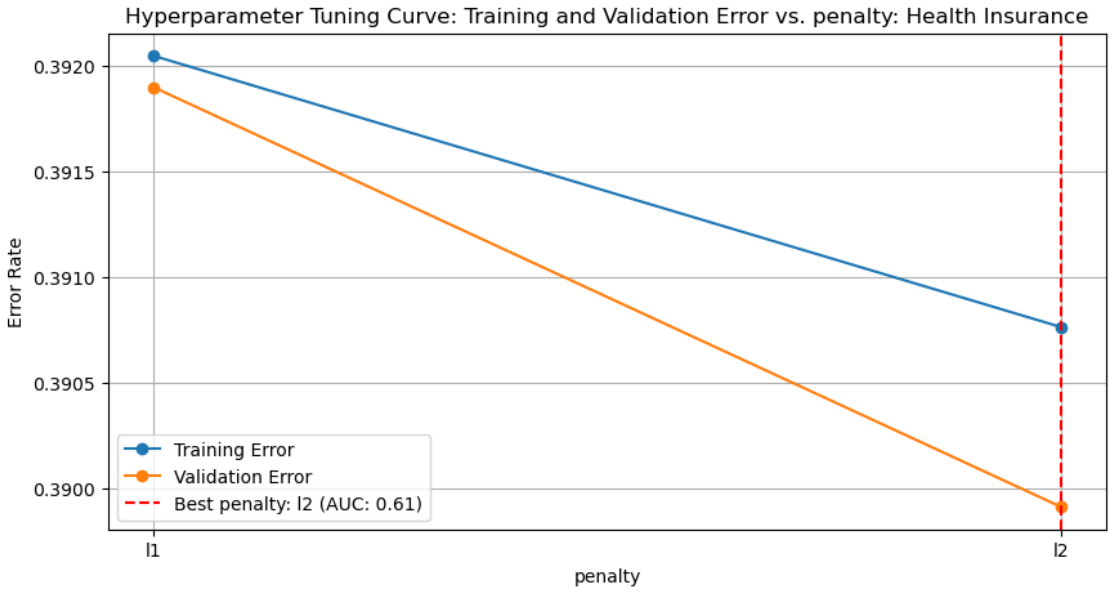
[207]:
```
# Hyperparameter Tuning Curve - max_iter
plot_results(aggregated_results_max_iter, 'max_iter', 'best_max_iter')
```



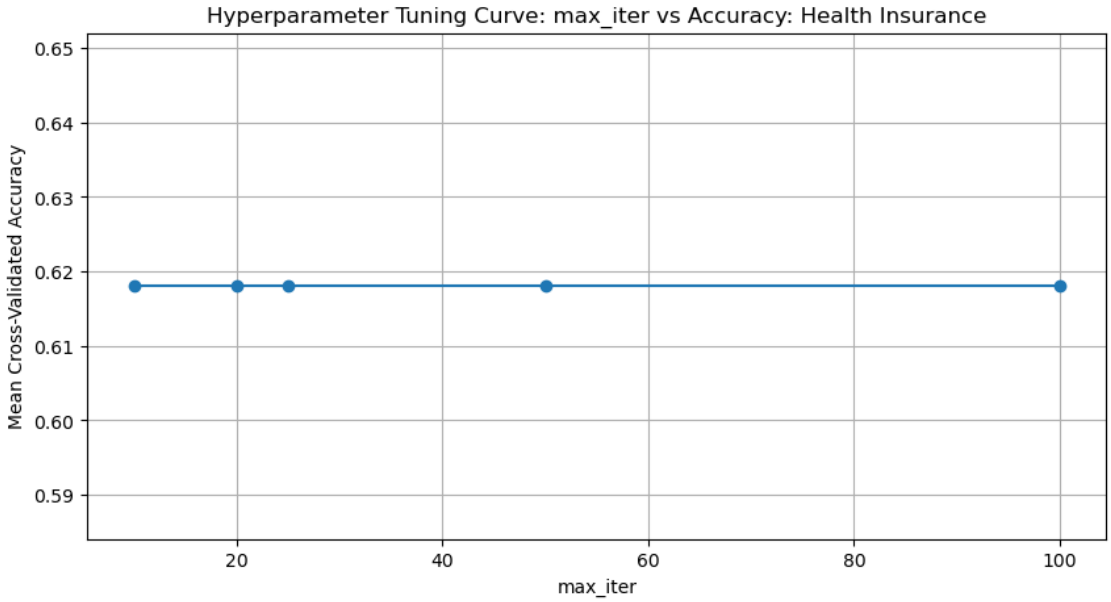
[209]:
```
# Hyperparameter Tuning Curve - eta0
plot_results(aggregated_results_eta0, 'eta0', 'best_eta0')
```



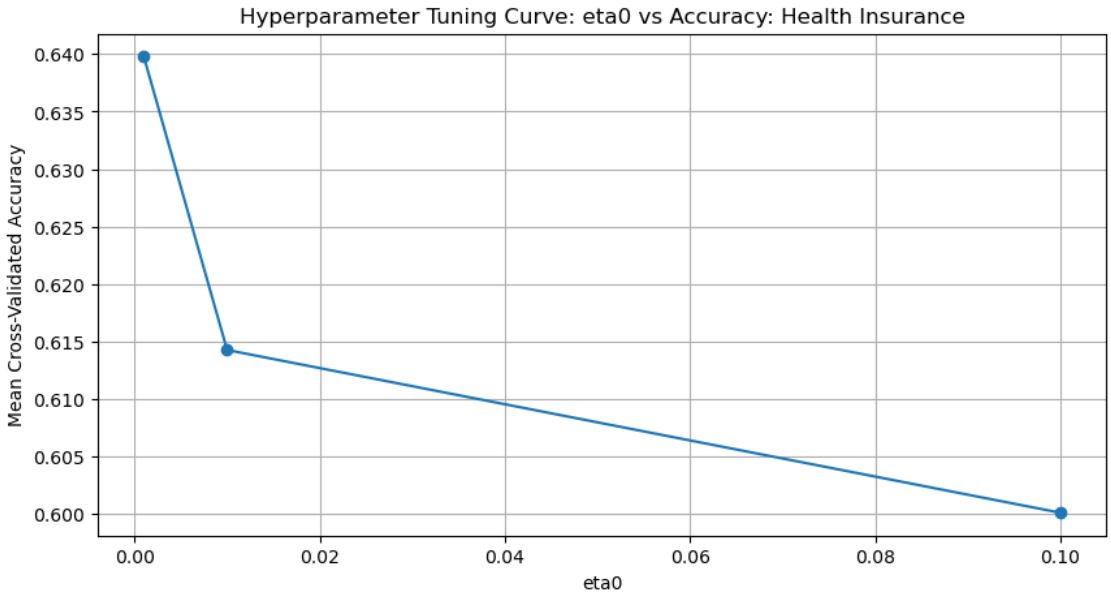
[211]:
```
# Hyperparameter Tuning Curve - penalty
plot_results(aggregated_results_penalty, 'penalty', 'best_penalty')
```



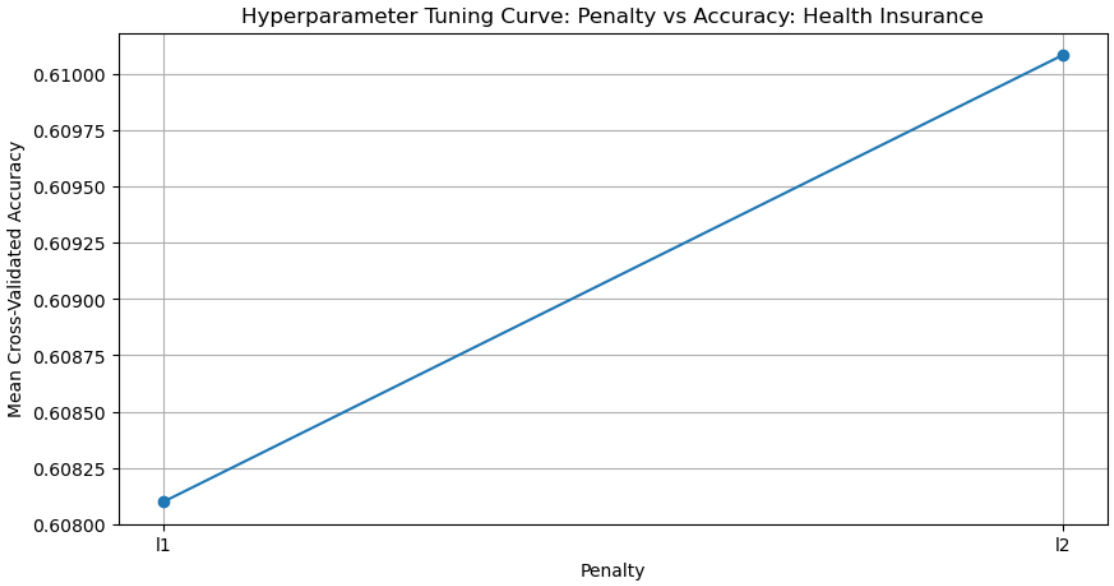
[235]:
```
# Hyperparameter Tuning Curve max_iter vs. Mean Cross-Validated Accuracy
plt.figure(figsize=(10, 5))
plt.plot(aggregated_results_max_iter['max_iter'],␣
 ↪aggregated_results_max_iter['mean_test_score'], marker='o')
plt.title("Hyperparameter Tuning Curve: max_iter vs Accuracy: Health Insurance")
plt.xlabel("max_iter")
plt.ylabel("Mean Cross-Validated Accuracy")
plt.grid()
plt.xticks()
plt.show()
```



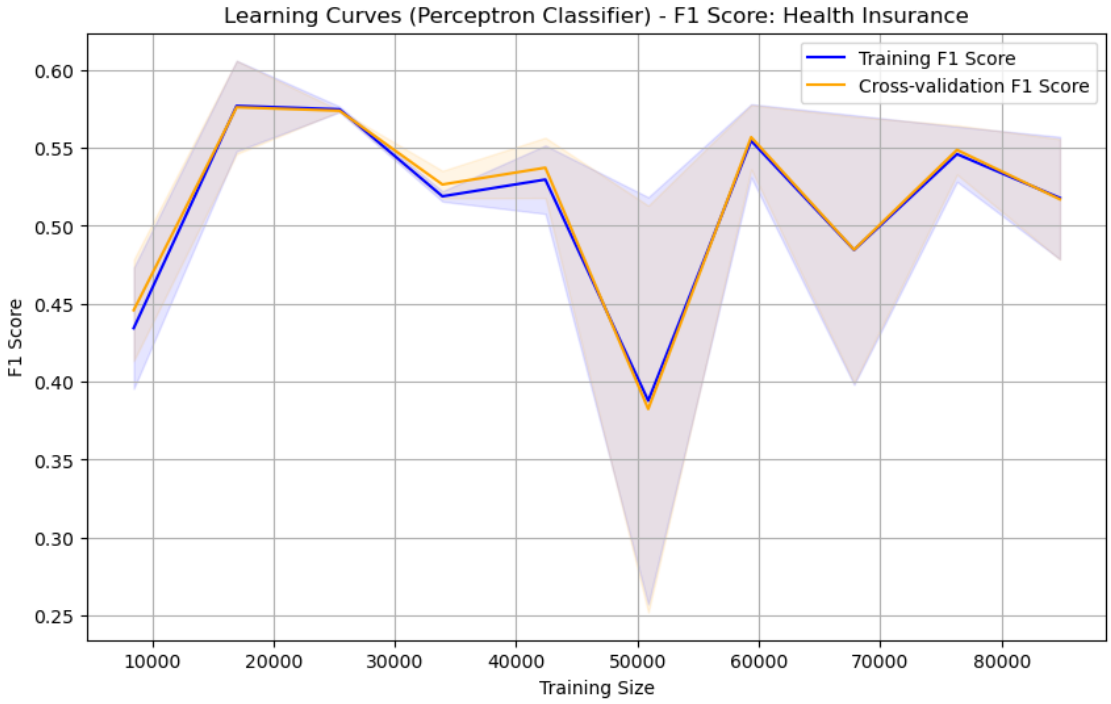
[217]:
```
# Hyperparameter Tuning Curve eta0 vs. Mean Cross-Validated Accuracy
plt.figure(figsize=(10, 5))
plt.plot(aggregated_results_eta0['eta0'],␣
  ↪aggregated_results_eta0['mean_test_score'], marker='o')
plt.title("Hyperparameter Tuning Curve: eta0 vs Accuracy: Health Insurance")
plt.xlabel("eta0")
plt.ylabel("Mean Cross-Validated Accuracy")
plt.grid()
plt.xticks()
plt.show()
```



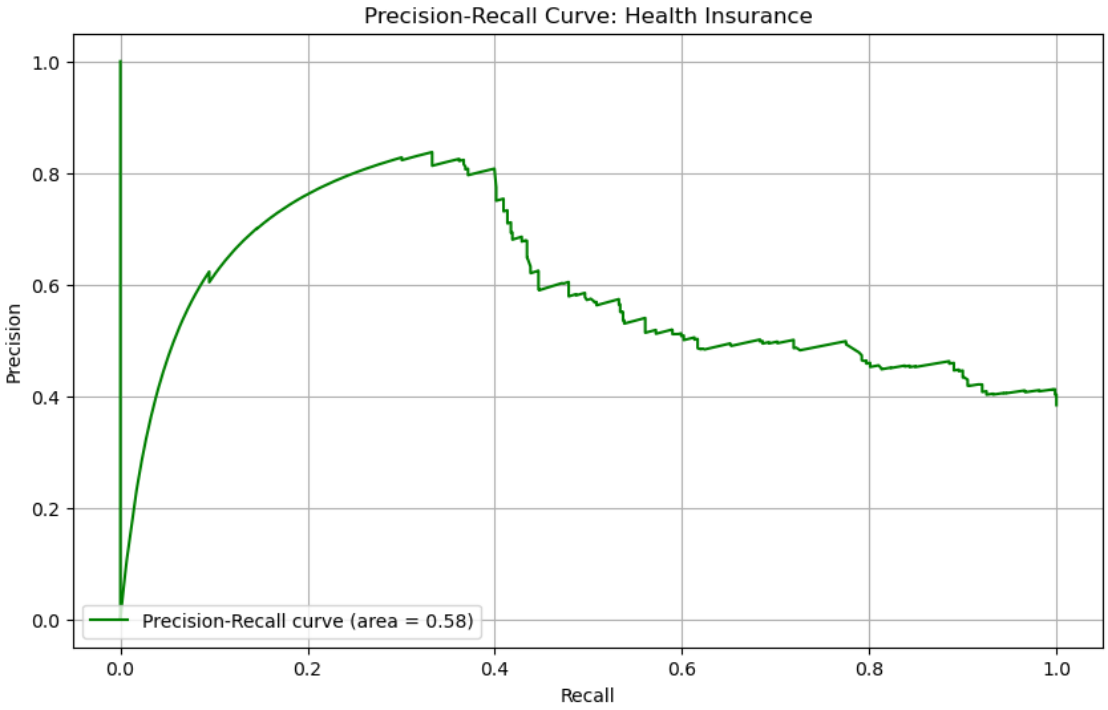
[219]:
```
# Hyperparameter Tuning Curve penalty vs. Mean Cross-Validated Accuracy
plt.figure(figsize=(10, 5))
plt.plot(aggregated_results_penalty['penalty'],␣
 ↪aggregated_results_penalty['mean_test_score'], marker='o')
plt.title("Hyperparameter Tuning Curve: Penalty vs Accuracy: Health Insurance")
plt.xlabel("Penalty")
plt.ylabel("Mean Cross-Validated Accuracy")
plt.grid()
plt.xticks()
plt.show()
```



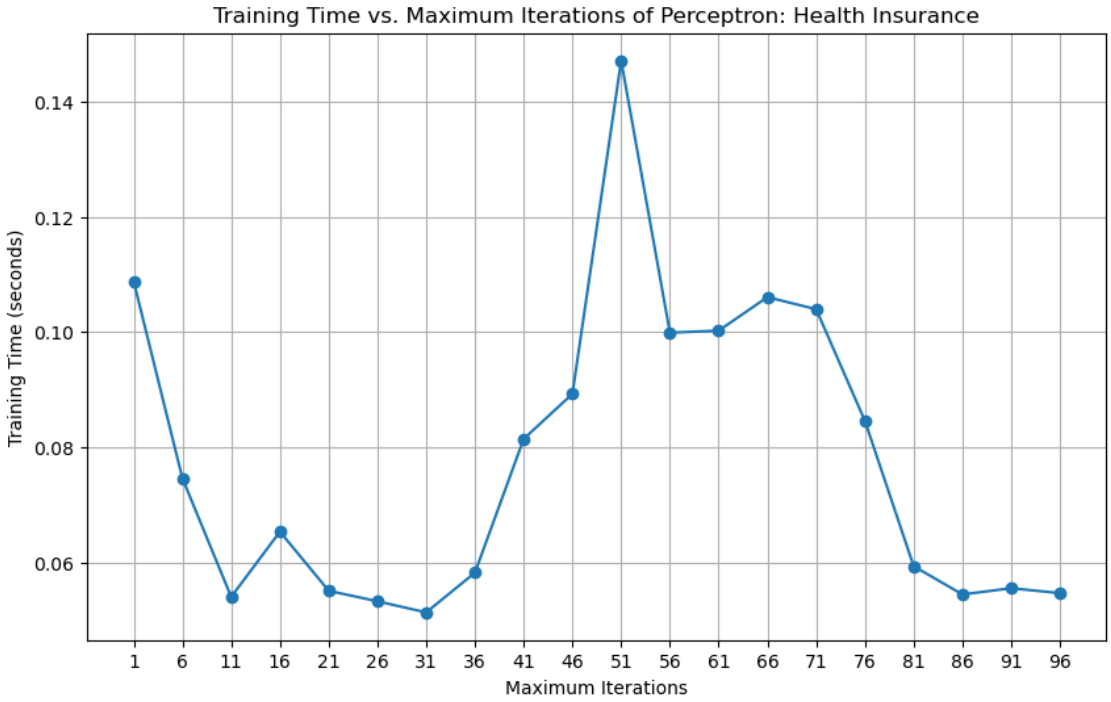
[221]:
```
# Create the F1 score scorer
f1_scorer = make_scorer(f1_score)

# 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)
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: Health␣
 ↪Insurance')
plt.xlabel('Training Size')
plt.ylabel('F1 Score')
plt.legend(loc='best')
plt.grid()
plt.show()
```



[225]:
```
# 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: Health Insurance')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend(loc='lower left')
plt.grid()
plt.show()
```

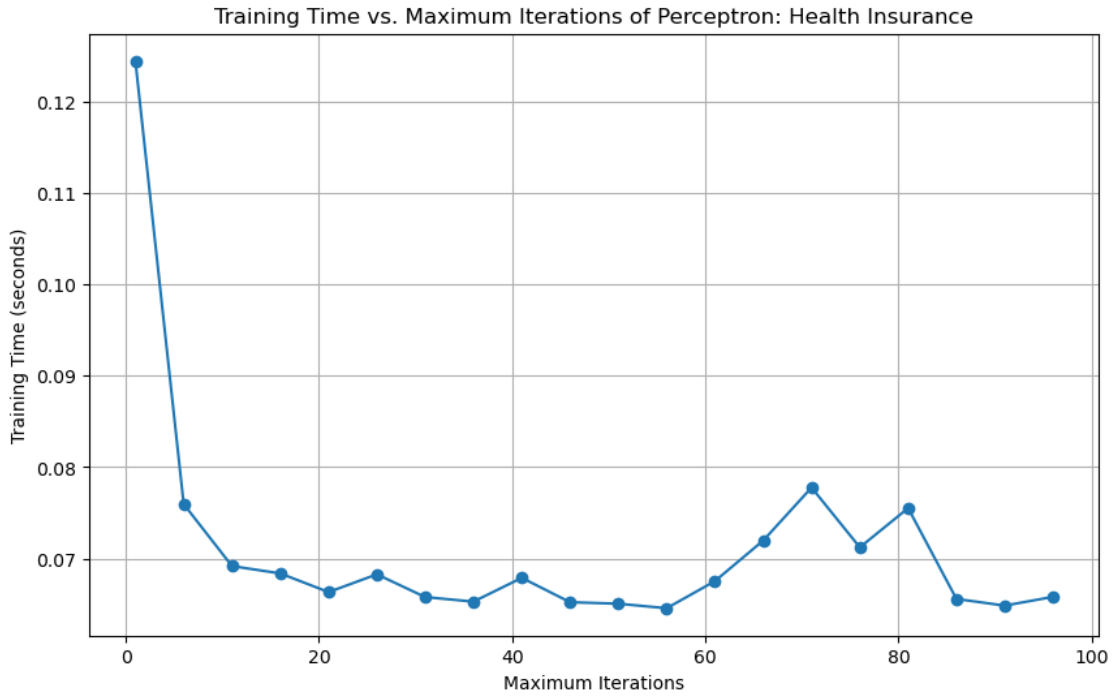


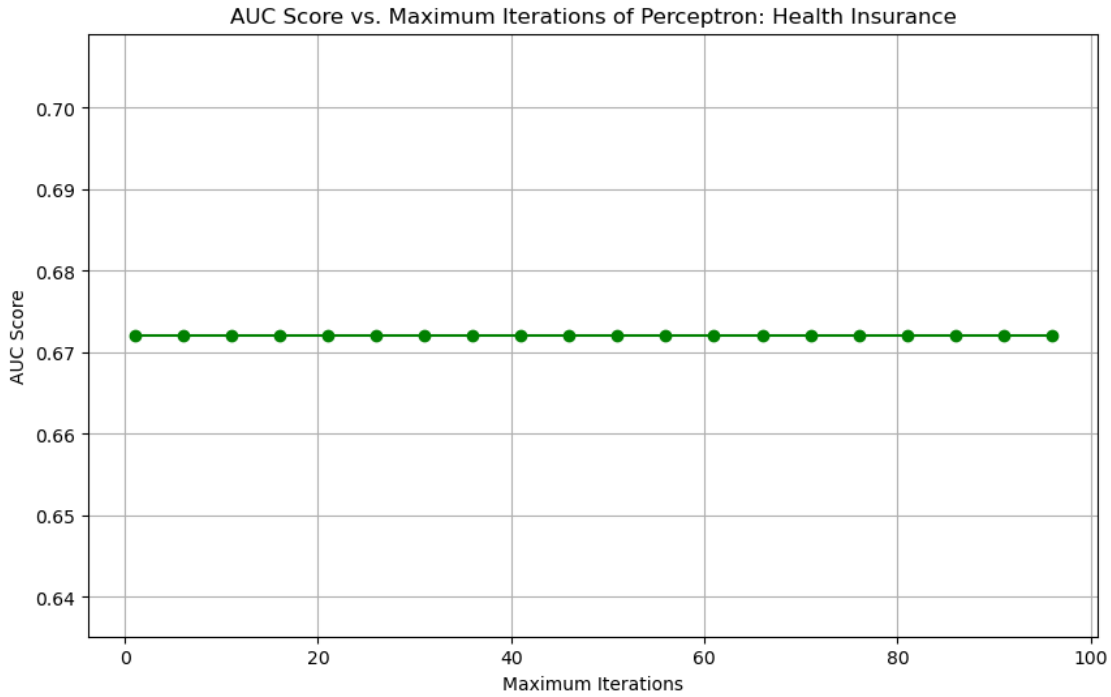
[227]:
```
# Timing Analysis: measuring training time

# 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: Health␣
 ↪Insurance')
plt.xlabel('Maximum Iterations')
plt.ylabel('Training Time (seconds)')
plt.xticks(max_iters)
plt.grid()
plt.show()
```



[239]:
```
max_iters = np.arange(1, 101, 5)
training_times = []
auc_scores = []

for max_iter in max_iters:
    start_time = time.time()
    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: Health␣
 ↪Insurance")
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: Health Insurance")
plt.xlabel("Maximum Iterations")
plt.ylabel("AUC Score")
plt.grid(True)
plt.show()
```





[233]:
```
# 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: Health insurance')
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: [0.55105891 0.71133437 0.73859724 0.61600868
0.62943396]
```

```
Mean Cross-Validation Score: 0.6492866320830363
```