HI LR.

October 8, 2024

1 Logistic Regression: Health Insurance

1.1 Load Data

[5]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix, f1_score,␣
 ↪classification_report, roc_curve, roc_auc_score
from sklearn.model_selection import StratifiedKFold

numeric_df = pd.read_csv('cleaned_data.csv')
```

1.2 Training Phase

[7]:
```
X = numeric_df.drop('DentalPlan', axis=1)
y = numeric_df['DentalPlan']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
log_reg = LogisticRegression()


param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'solver': ['liblinear', 'lbfgs'],
}

skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

grid_search = GridSearchCV(log_reg, param_grid, cv=skf, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)

best_params = grid_search.best_params_
```

```
print("Best Hyperparameters:", best_params)
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print("Cross-validated accuracy on test set:", accuracy)

accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
error_rate = 1 - accuracy

print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Error Rate:", error_rate)

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
```

```
Best Hyperparameters: {'C': 0.01, 'solver': 'liblinear'}
Cross-validated accuracy on test set: 0.7371042602165956
Accuracy: 0.7371042602165956
F1 Score: 0.5253117122027662
Error Rate: 0.2628957397834044
Confusion Matrix:
 [[15679   746]
 [ 6221  3855]]
```

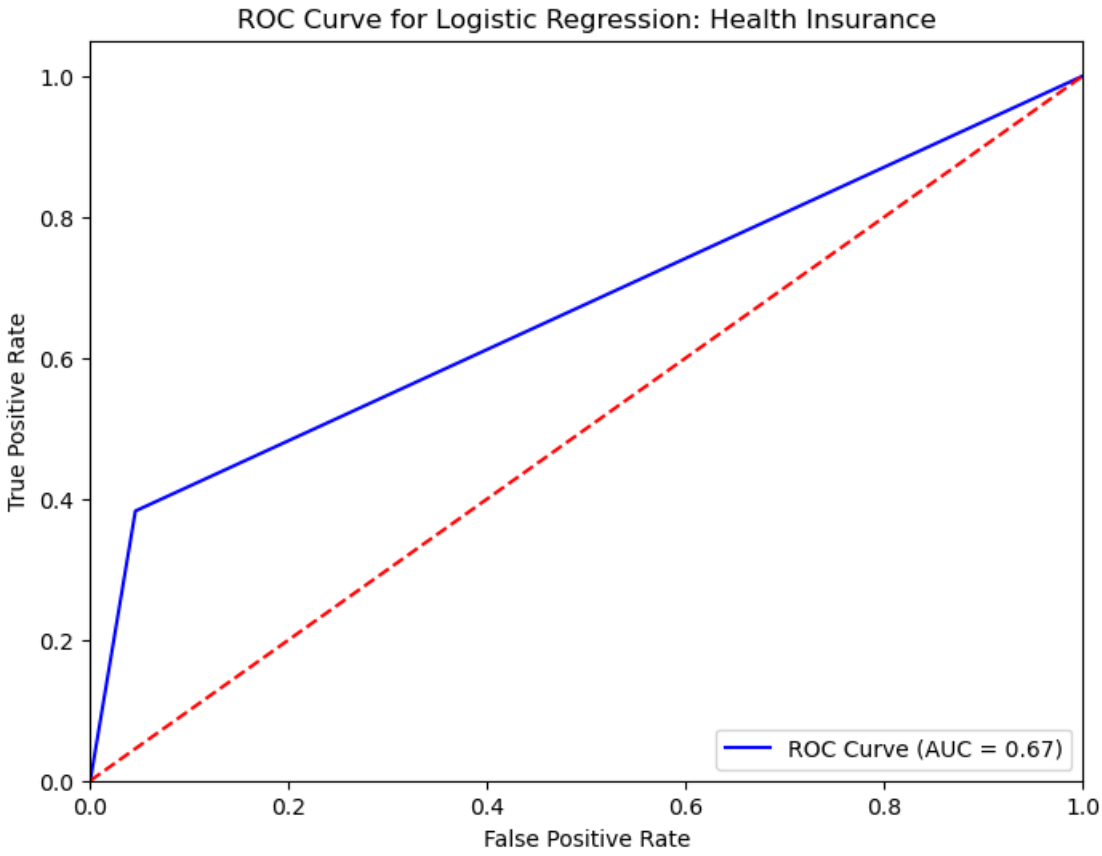
[16]:
```
print("Classification Report:\n", classification_report(y_test, y_pred))
```

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

           0       0.72      0.95      0.82     16425
           1       0.84      0.38      0.53     10076

    accuracy                           0.74     26501
   macro avg       0.78      0.67      0.67     26501
weighted avg       0.76      0.74      0.71     26501
```

1.3 ROC Curve

[17]:
```
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

plt.figure(figsize=(8, 6))
```

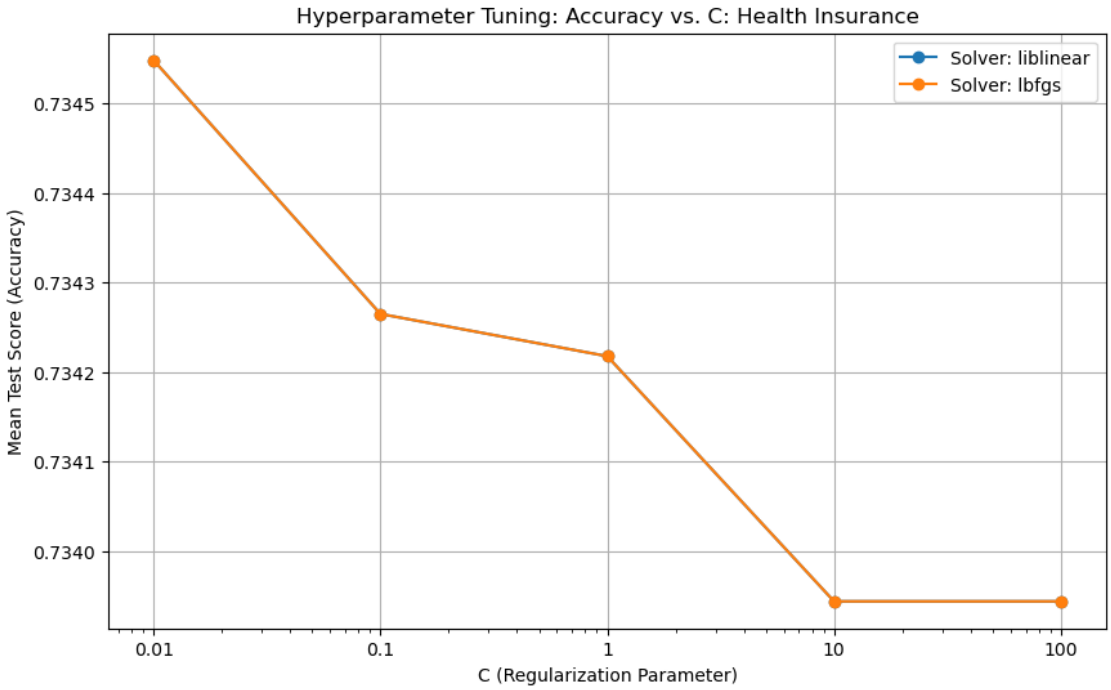
```
plt.plot(fpr, tpr, color='blue', label='ROC Curve (AUC = {:.2f})'.
  ↪format(roc_auc))
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Logistic Regression: Health Insurance')
plt.legend(loc='lower right')
plt.show()
```



1.4 Hyperparameter Tuning Curve: solver and Regularization

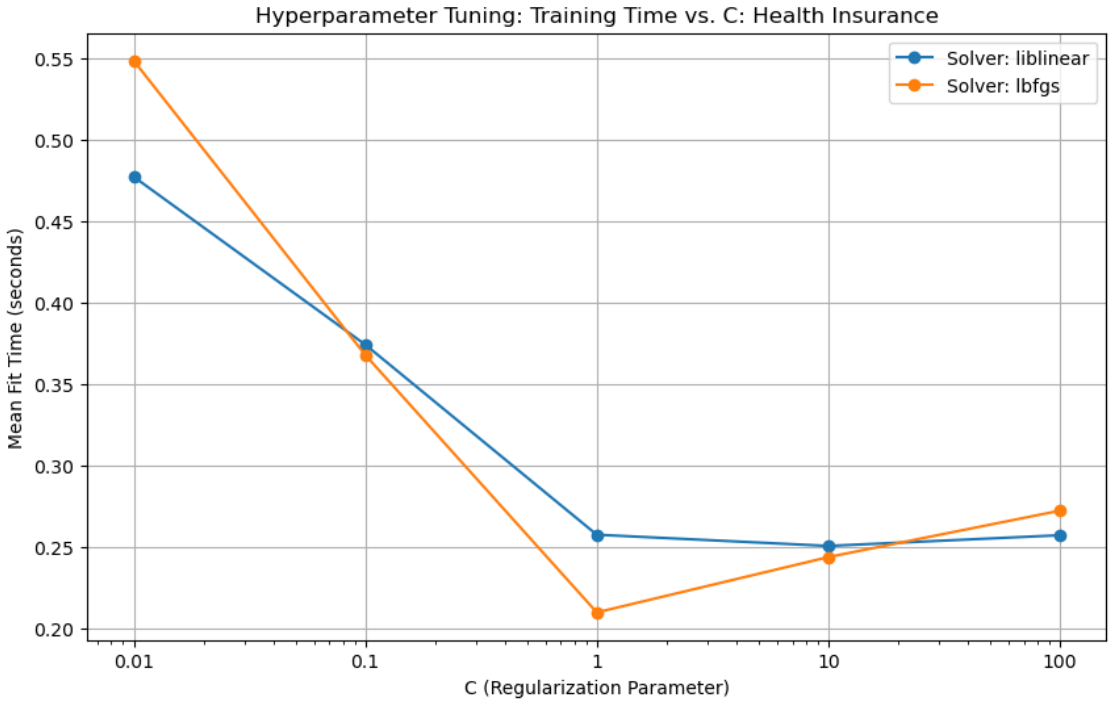
[18]:
```
import matplotlib.pyplot as plt
import pandas as pd

# Get the results from GridSearchCV
results = grid_search.cv_results_
```

```
results_df = pd.DataFrame(results)

# Plotting C vs. Mean Test Score
plt.figure(figsize=(10, 6))

# Loop through each solver to create separate curves
for solver in param_grid['solver']:
    mask = results_df['param_solver'] == solver
    plt.plot(results_df['param_C'][mask], results_df['mean_test_score'][mask],
             marker='o', label=f'Solver: {solver}')

plt.xscale('log')
plt.title('Hyperparameter Tuning: Accuracy vs. C: Health Insurance')
plt.xlabel('C (Regularization Parameter)')
plt.ylabel('Mean Test Score (Accuracy)')
plt.xticks([0.01, 0.1, 1, 10, 100], [0.01, 0.1, 1, 10, 100])
plt.legend()
plt.grid()
plt.show()
```



1.5 Training Time

[19]:
```
import matplotlib.pyplot as plt
import pandas as pd

# Get the results from GridSearchCV
results = grid_search.cv_results_


results_df = pd.DataFrame(results)

# Plotting C vs. Mean Fit Time
plt.figure(figsize=(10, 6))

# Loop through each solver to create separate curves
for solver in param_grid['solver']:
    mask = results_df['param_solver'] == solver
    plt.plot(results_df['param_C'][mask], results_df['mean_fit_time'][mask],
             marker='o', label=f'Solver: {solver}')

plt.xscale('log')
plt.title('Hyperparameter Tuning: Training Time vs. C: Health Insurance')
plt.xlabel('C (Regularization Parameter)')
plt.ylabel('Mean Fit Time (seconds)')
plt.xticks([0.01, 0.1, 1, 10, 100], [0.01, 0.1, 1, 10, 100])
plt.legend()
plt.grid()
plt.show()
```



1.6 Training and Validation Error

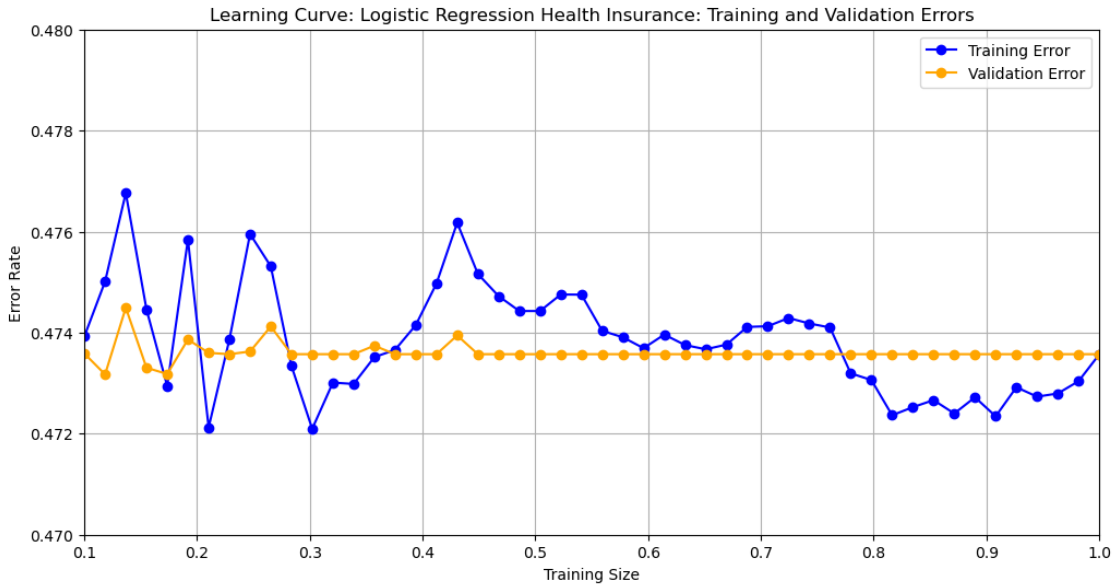
[28]:
```
train_sizes, train_scores, valid_scores = learning_curve(log_reg,
                                                         X_train_scaled,
                                                         y_train,
                                                         train_sizes=np.
 ↪linspace(0.1, 1.0),
                                                         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: [0.52607397 0.52498784 0.52323525 0.52554202 0.52705691
0.5241593
 0.52787628 0.5261414  0.52404765 0.52468714 0.52666008 0.52790342
 0.52698885 0.52701567 0.52648496 0.52634055 0.52585104 0.52502854
 0.52382599 0.5248389  0.52527665 0.52557195 0.52557239 0.52524472
 0.52524797 0.52596381 0.52609176 0.5263078  0.52603827 0.52624525
 0.52632947 0.52623678 0.52588734 0.52587283 0.5257102  0.52581873
 0.52589797 0.5268052  0.52693803 0.52763544 0.52747486 0.52733563
 0.52760039 0.52727971 0.52765623 0.52708068 0.52726753 0.52720355
 0.52696152 0.52643066]
Validation Scores Mean: [0.52641374 0.52682656 0.52550667 0.5266951  0.52681822
0.52613545
 0.52640001 0.52642649 0.52637328 0.52587791 0.52642649 0.52642649
 0.52642649 0.52642649 0.5262582  0.52642649 0.52642649 0.52642649
 0.52605488 0.52642649 0.52642649 0.52642649 0.52642649 0.52642649
 0.52642649 0.52642649 0.52642649 0.52642649 0.52642649 0.52642649
 0.52642649 0.52642649 0.52642649 0.52642649 0.52642649 0.52642649
 0.52642649 0.52642649 0.52642649 0.52642649 0.52642649 0.52642649
 0.52642649 0.52642649]
Shape of X_train: (106004, 6)
Unique classes in y_train: [0 1]
```

[52]:
```
import numpy as np
import matplotlib.pyplot as plt

# Provided training and validation F1 scores
train_scores_mean = np.array([0.52607397, 0.52498784, 0.52323525, 0.52554202,
                              0.52705691, 0.5241593, 0.52787628, 0.5261414,
                              0.52404765, 0.52468714, 0.52666008, 0.52790342,
                              0.52698885, 0.52701567, 0.52648496, 0.52634055,
                              0.52585104, 0.52502854, 0.52382599, 0.5248389,
                              0.52527665, 0.52557195, 0.52557239, 0.52524472,
                              0.52524797, 0.52596381, 0.52609176, 0.5263078,
                              0.52603827, 0.52624525, 0.52632947, 0.52623678,
                              0.52588734, 0.52587283, 0.5257102, 0.52581873,
                              0.52589797, 0.5268052, 0.52693803, 0.52763544,
                              0.52747486, 0.52733563, 0.52760039, 0.52727971,
                              0.52765623, 0.52708068, 0.52726753, 0.52720355,
                              0.52696152, 0.52643066])

valid_scores_mean = np.array([0.52641374, 0.52682656, 0.52550667, 0.5266951,
                              0.52681822, 0.52613545, 0.52640001, 0.52642649,
                              0.52637328, 0.52587791, 0.52642649, 0.52642649,
                              0.52642649, 0.52642649, 0.5262582, 0.52642649,
```

```
                                0.52642649, 0.52642649, 0.52605488, 0.52642649,
                                0.52642649, 0.52642649, 0.52642649, 0.52642649,
                                0.52642649, 0.52642649, 0.52642649, 0.52642649,
                                0.52642649, 0.52642649, 0.52642649, 0.52642649,
                                0.52642649, 0.52642649, 0.52642649, 0.52642649,
                                0.52642649, 0.52642649, 0.52642649, 0.52642649,
                                0.52642649, 0.52642649, 0.52642649, 0.52642649,
                                0.52642649, 0.52642649, 0.52642649, 0.52642649,
                                0.52642649, 0.52642649])

# Simulated training sizes (can be adjusted as needed)
train_sizes = np.linspace(0.1, 1.0, len(train_scores_mean))

# Calculate training and validation errors
train_errors = 1 - train_scores_mean
valid_errors = 1 - valid_scores_mean

# Plotting the Training and Validation Errors
plt.figure(figsize=(12, 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 Health Insurance: Training and␣
 ↪Validation Errors')
plt.xlabel('Training Size')
plt.ylabel('Error Rate')
plt.xlim([0.1, 1.0])
plt.ylim([0.47, 0.48])
plt.grid()
plt.legend(loc='best')
plt.show()
```



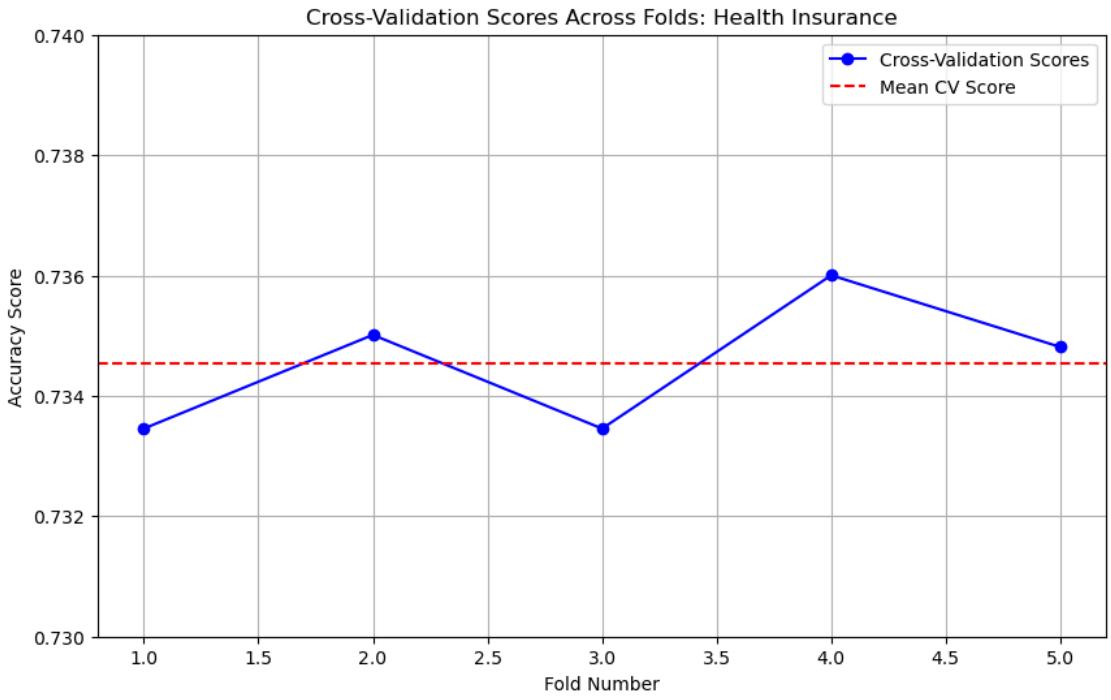
1.7 Cross Validation Scores Across Folds

[9]:
```
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: [0.73345597 0.7350125  0.73345597 0.73600302
0.73481132]
Mean Cross-Validation Score: 0.7345477554013562
```

[27]:
```
cv_scores = [0.73345597, 0.7350125,  0.73345597, 0.73600302, 0.73481132]
mean_cv_score = 0.7345477554013562


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(0.73, 0.74)
plt.legend()
plt.grid(True)
plt.show()
```



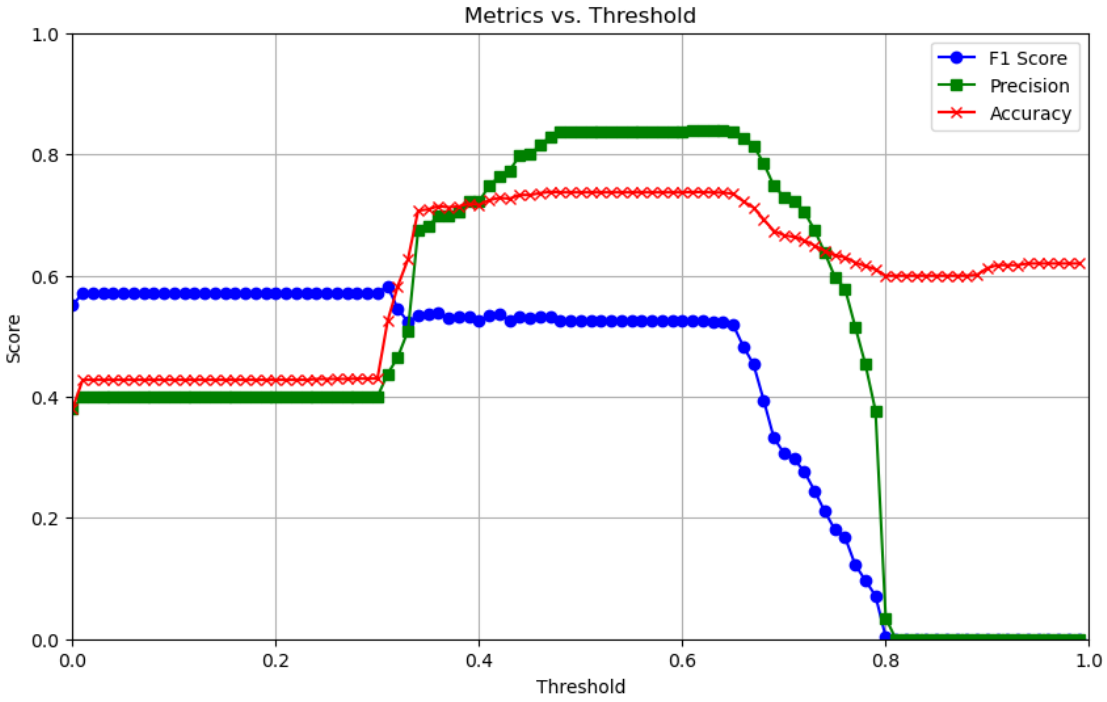
1.8 Additional Visuals

[21]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import precision_score, accuracy_score, f1_score

# Get predicted probabilities for the positive class
y_scores = best_model.predict_proba(X_test_scaled)[:, 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()
```

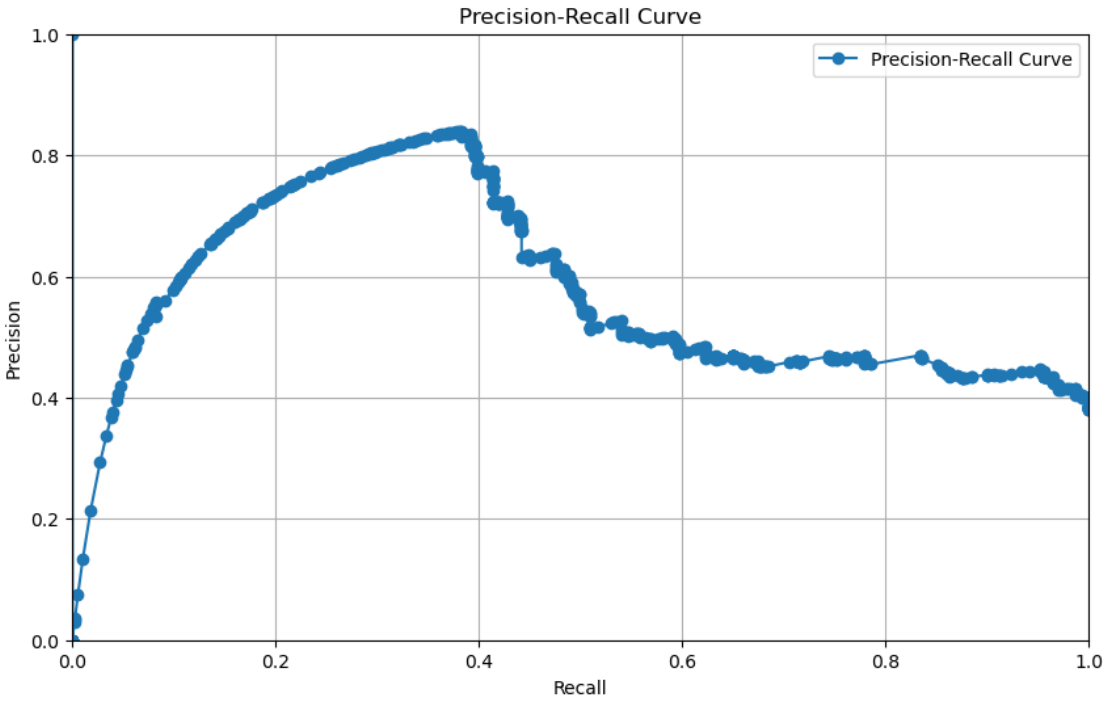
```
/opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-
```

```
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```



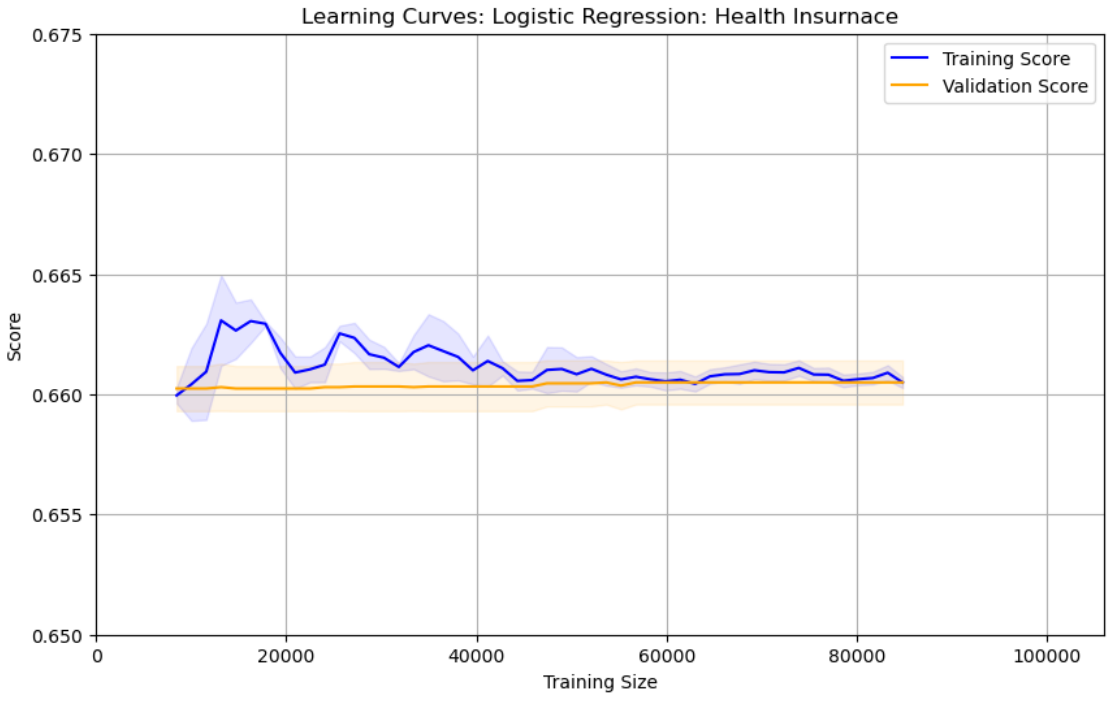
[20]:
```python
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt

# Get predicted probabilities
y_scores = best_model.predict_proba(X_test_scaled)[:, 1]
# Probability of the positive class

# Calculate precision and recall values
precision, recall, thresholds = precision_recall_curve(y_test, y_scores)

# Plotting the Precision-Recall curve
plt.figure(figsize=(10, 6))
plt.plot(recall, precision, marker='o', label='Precision-Recall Curve')
plt.title('Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.grid()
```

```
plt.legend()
plt.show()
```



[22]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve


log_reg = LogisticRegression(**best_params)

# Calculate training and validation scores
train_sizes, train_scores, valid_scores = learning_curve(log_reg,
                                                         X_train,
                                                         y_train,
                                                         train_sizes=np.
 ↪linspace(0.1, 1.0),
                                                         cv=5,
                                                         scoring='accuracy')

# 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)

# Plotting the Learning Curves
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_scores_mean, label='Training Score', color='blue')
plt.plot(train_sizes, valid_scores_mean, label='Validation Score',␣
 ↪color='orange')

# Plot the fill between for the standard deviation
plt.fill_between(train_sizes,
                 train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std,
                 alpha=0.1, color='blue')
plt.fill_between(train_sizes,
                 valid_scores_mean - valid_scores_std,
                 valid_scores_mean + valid_scores_std,
                 alpha=0.1, color='orange')

plt.title('Learning Curves: Logistic Regression: Health Insurnace')
plt.xlabel('Training Size')
plt.ylabel('Score')
plt.xlim([0, X_train.shape[0]])
plt.ylim([0.65, 0.675])
plt.grid()
plt.legend()
plt.show()
```



[ ]: