CR KNN

October 8, 2024

0.1 K-Nearest Neighbor

0.1.1 Import and Preprocessing

0.1.2 Train Model

[3]:
```
from sklearn.model_selection import GridSearchCV  # Import GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import pandas as pd

# Load the dataset
LR_df = pd.read_csv('LR_df.csv')

# Drop rows with missing target values
LR_df = LR_df.dropna(subset=['Default'])

# Prepare feature matrix and target vector
X = LR_df.drop('Default', axis=1)
y = LR_df['Default']

# Impute missing values
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define the parameter grid for GridSearchCV
param_grid = {
```

```
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance']
}

# Initialize GridSearchCV
grid_search = GridSearchCV(
    KNeighborsClassifier(),
    param_grid,
    cv=5,
    return_train_score=True
)

try:
    grid_search.fit(X_train, y_train)
    best_params = grid_search.best_params_
    best_n_neighbors = best_params['n_neighbors']
    best_weights = best_params['weights']
    print(f'Best n_neighbors: {best_n_neighbors}, Best weights: {best_weights}')
except Exception as e:
    print(f"Error during fitting: {e}")

# Train the best model found by GridSearchCV
knn = KNeighborsClassifier(n_neighbors=best_n_neighbors, weights=best_weights)
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

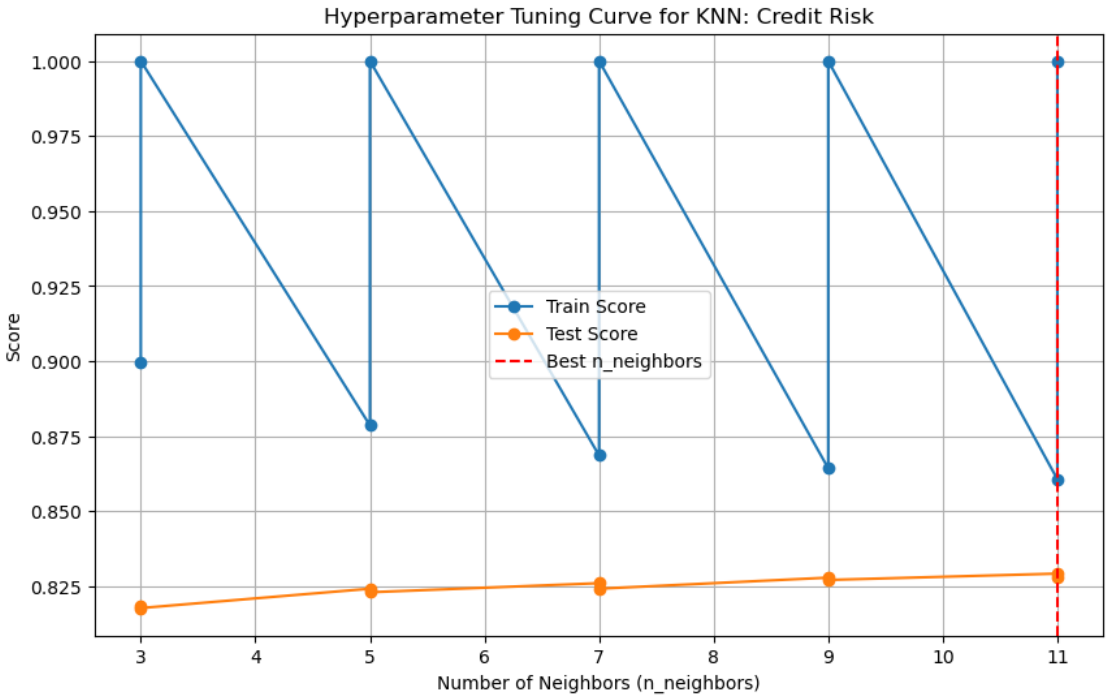
```
Best n_neighbors: 11, Best weights: uniform
Accuracy: 0.82
```

[6]:
```
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

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

         0.0       0.86      0.93      0.90      5322
         1.0       0.52      0.35      0.42      1195

    accuracy                           0.82      6517
   macro avg       0.69      0.64      0.66      6517
weighted avg       0.80      0.82      0.81      6517
```

0.2 Hyperparameter Tuning Curve: Training and Testing Score - n_neighbors

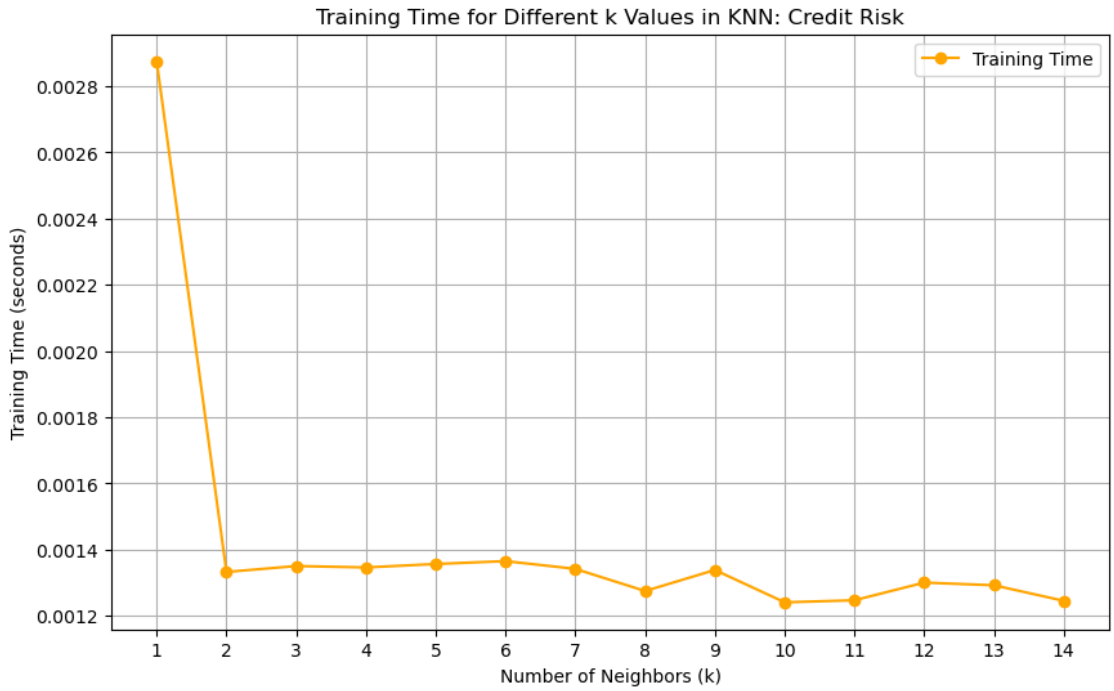
[13]:
```
import matplotlib.pyplot as plt

train_scores = grid_search.cv_results_['mean_train_score']
test_scores = grid_search.cv_results_['mean_test_score']
neighbors = grid_search.cv_results_['param_n_neighbors'].data

plt.figure(figsize=(10, 6))
plt.plot(neighbors, train_scores, label='Train Score', marker='o')
plt.plot(neighbors, test_scores, label='Test Score', marker='o')
plt.axvline(x=best_n_neighbors, color='r', linestyle='--', label='Best␣
 ↪n_neighbors')

plt.title('Hyperparameter Tuning Curve for KNN: Credit Risk')
plt.xlabel('Number of Neighbors (n_neighbors)')
plt.ylabel('Score')
plt.legend()
plt.grid()
plt.show()
```



0.3 Training Time

[16]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import time

def evaluate_knn(X_train, y_train, X_test, y_test, k_values):
    train_accuracies = []
    test_accuracies = []
    training_times = []

    for k in k_values:

        knn = KNeighborsClassifier(n_neighbors=k)


        start_time = time.time()

        # Fit the model on the training data
        knn.fit(X_train, y_train)


        training_time = time.time() - start_time

        # Predict on the training set
        y_train_pred = knn.predict(X_train)

        # Predict on the test set
        y_test_pred = knn.predict(X_test)

        # Calculate accuracy for training and test sets
        train_accuracy = accuracy_score(y_train, y_train_pred)
        test_accuracy = accuracy_score(y_test, y_test_pred)

        # Append accuracies and training time to the lists
        train_accuracies.append(train_accuracy)
        test_accuracies.append(test_accuracy)
        training_times.append(training_time)

    return train_accuracies, test_accuracies, training_times

# Sample Data
X_train_sample = X_train[:1000]
y_train_sample = y_train[:1000]
X_test_sample = X_test[:300]
```

```
y_test_sample = y_test[:300]

# K values to evaluate
k_values = range(1, 15)

# Evaluate KNN
train_accuracies, test_accuracies, training_times = evaluate_knn(
    X_train_sample, y_train_sample, X_test_sample, y_test_sample, k_values
)


# Plotting the training times
plt.figure(figsize=(10, 6))
plt.plot(k_values, training_times, label='Training Time', marker='o',␣
 ↪color='orange')
plt.title('Training Time for Different k Values in KNN: Credit Risk')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Training Time (seconds)')
plt.xticks(k_values)
plt.grid()
plt.legend()
plt.show()
```

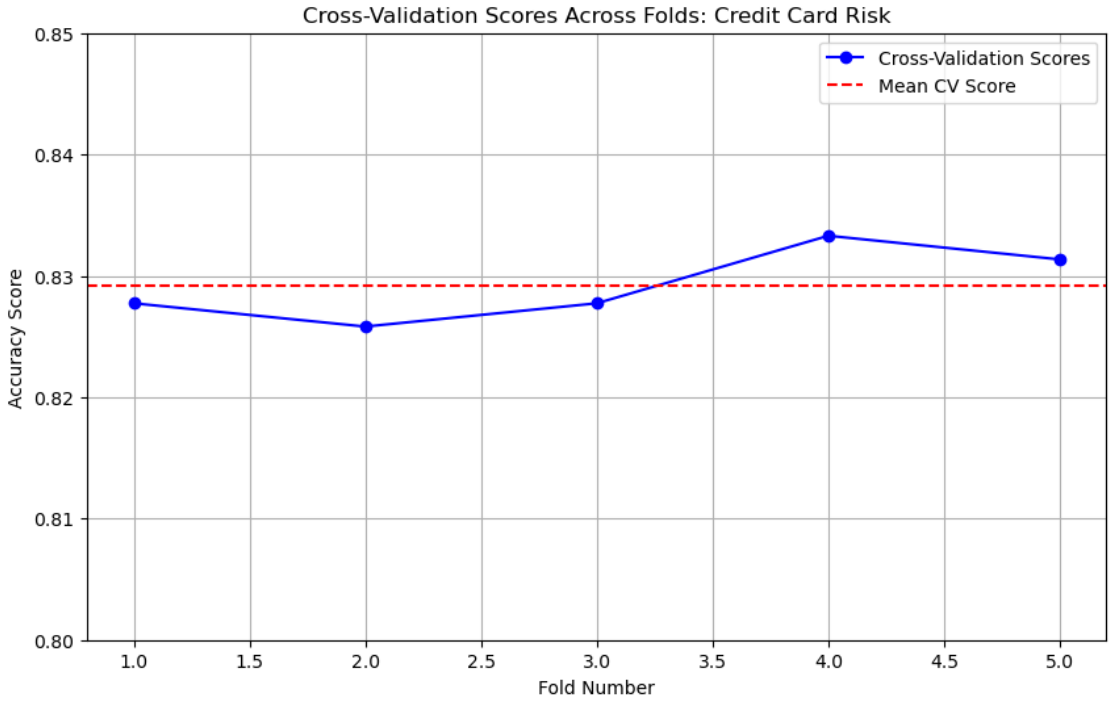


0.4 Cross Validation Scores Across Folds

[26]:
```
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))
```

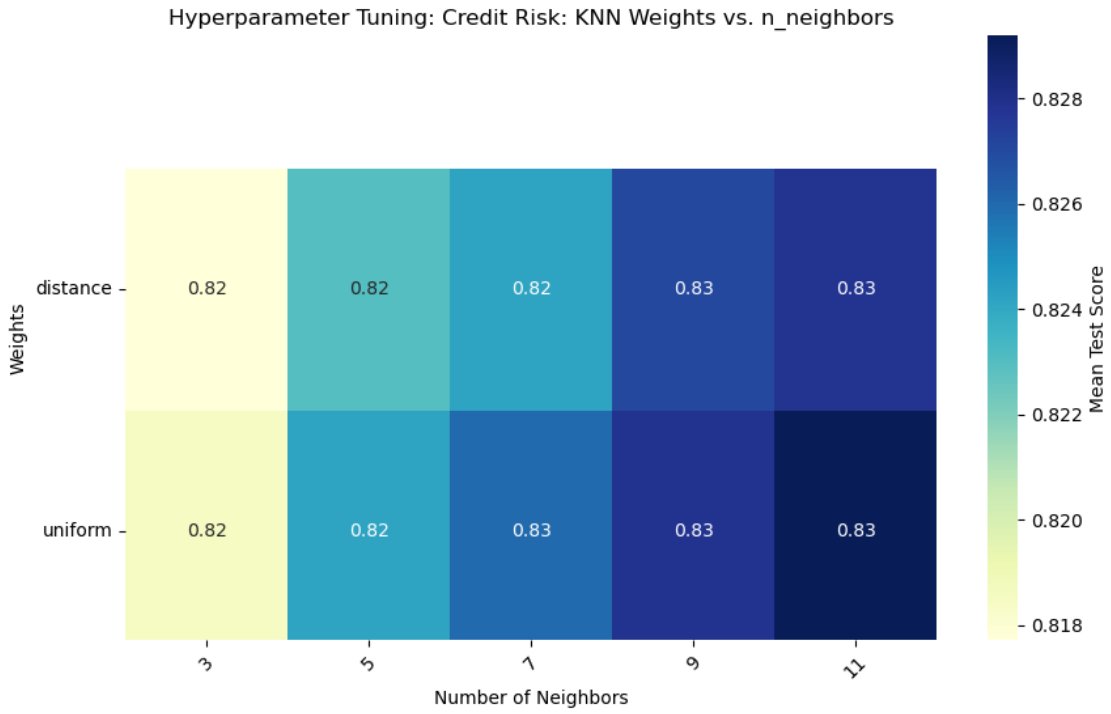
```
Cross-Validation Scores: [0.82773835 0.82582007 0.82773835 0.83330136
0.83135073]
Mean Cross-Validation Score: 0.8291897698342255
```

[19]:
```
cv_scores = [0.82773835, 0.82582007, 0.82773835, 0.83330136, 0.83135073]
mean_cv_score = 0.8291897698342255


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()
```



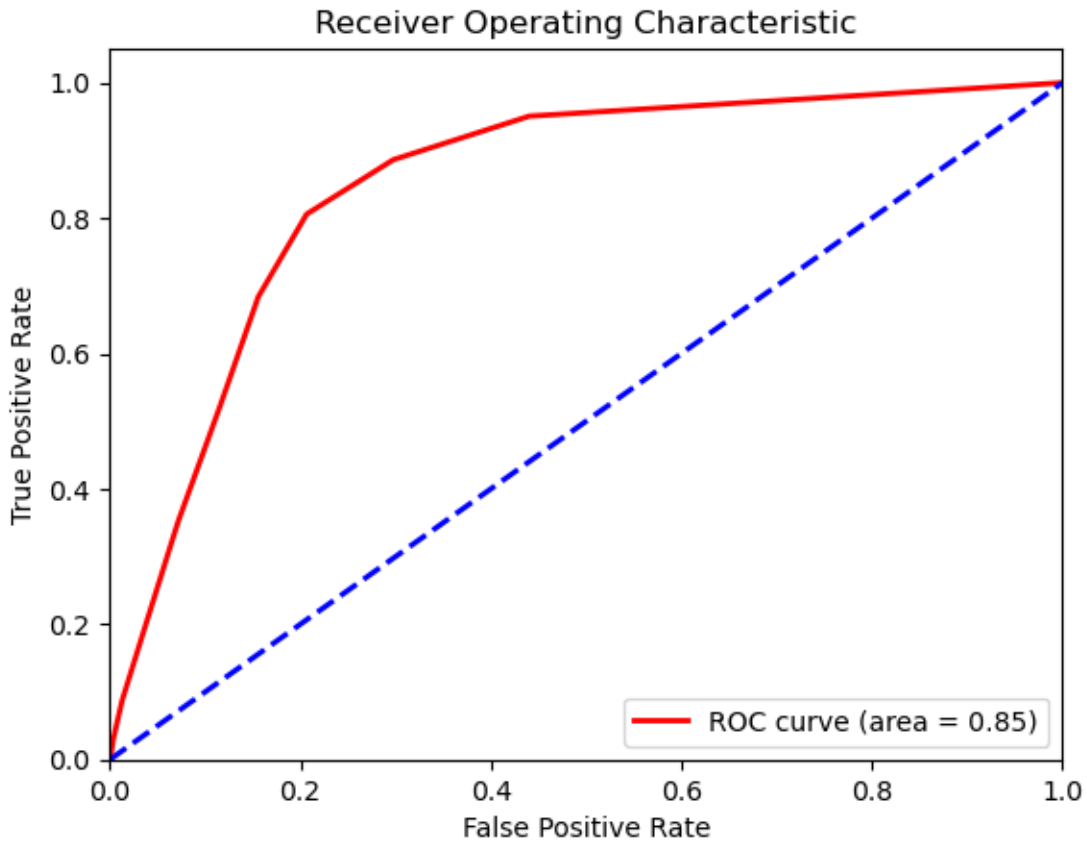
0.5 Hyperparameter Tuning Plot: Weights and N_neighbors Mean Score

[30]:
```
import seaborn as sns
results = pd.DataFrame(grid_search.cv_results_)

# Plotting the results
pivot_table = results.pivot_table(index='param_weights',␣
 ↪columns='param_n_neighbors', values='mean_test_score')

# Plotting
plt.figure(figsize=(10, 6))
sns.heatmap(pivot_table, annot=True, fmt=".2f", cmap="YlGnBu",␣
 ↪cbar_kws={'label': 'Mean Test Score'})
plt.title('Hyperparameter Tuning: Credit Risk: KNN Weights vs. n_neighbors')
plt.xlabel('Number of Neighbors')
plt.ylabel('Weights')
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.ylim(len(pivot_table) - 0.05, -0.55)
plt.show()
```



0.6 ROC Curve

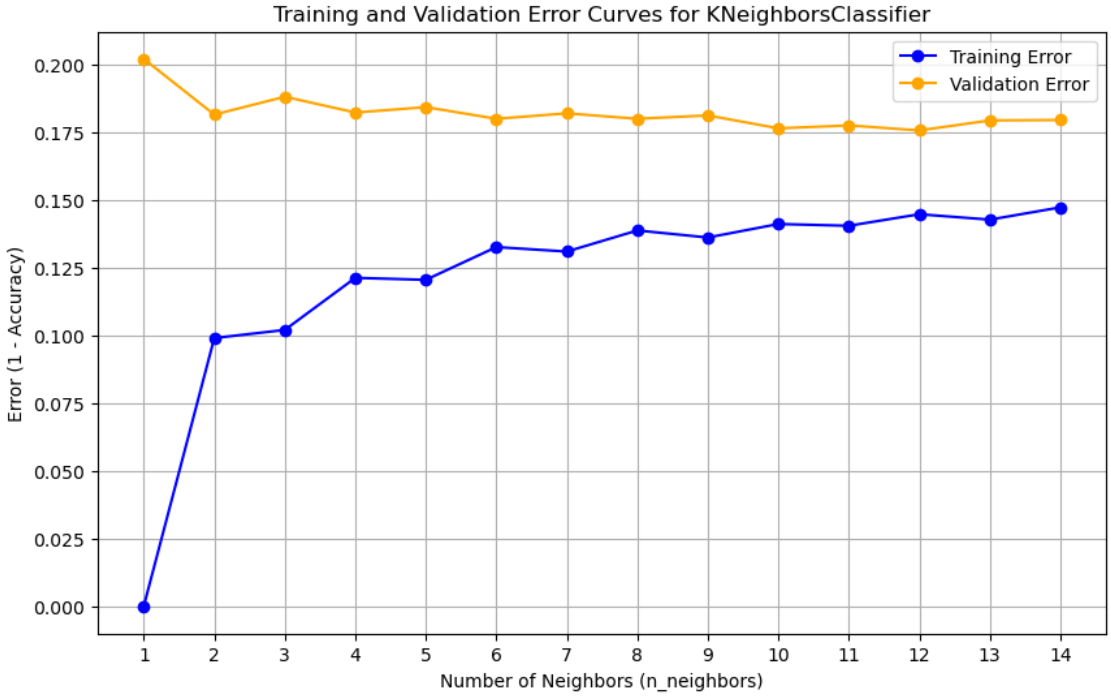
[36]:
```
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report, roc_curve, auc
import matplotlib.pyplot as plt
y_prob = knn.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='red', lw=2, label='ROC curve (area = {:.2f})'.
 ↪format(roc_auc))
plt.plot([0, 1], [0, 1], color='blue', lw=2, 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('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```



0.7 Training and Validation Error

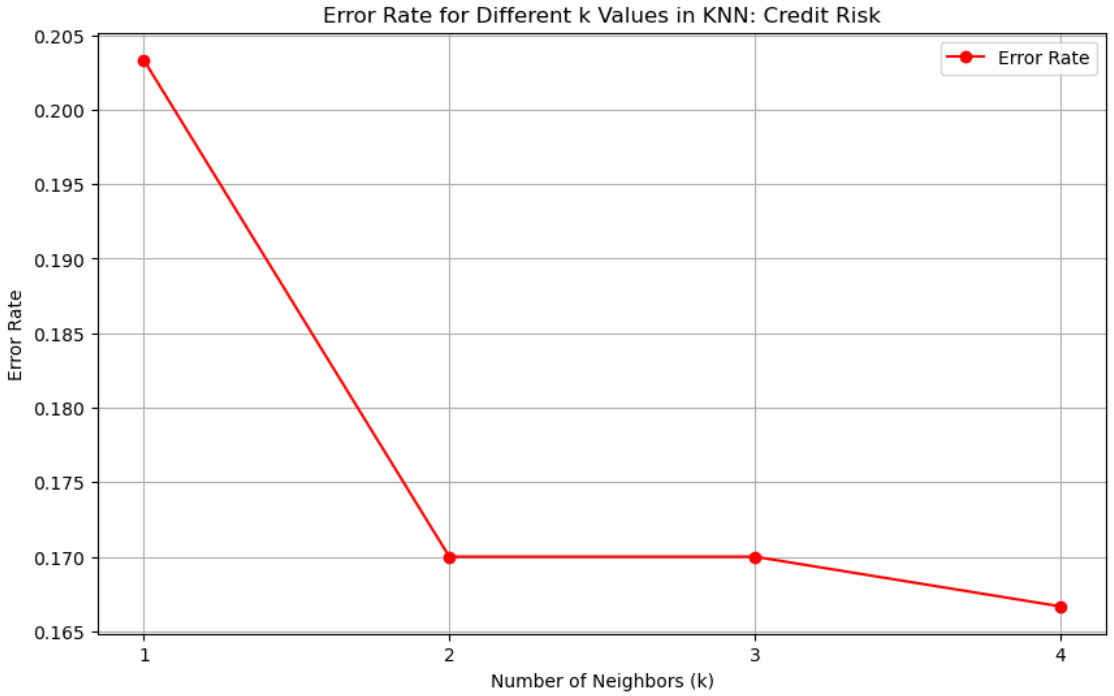
[46]:
```
n_neighbors_range = range(1, 15)
train_scores = []
val_scores = []



# Iterate over the range of n_neighbors
for n_neighbors in n_neighbors_range:

    knn = KNeighborsClassifier(n_neighbors=n_neighbors, weights='uniform')
    knn.fit(X_train, y_train)

    # Predict on training and validation sets
    y_train_pred = knn.predict(X_train)
    y_val_pred = knn.predict(X_test)

    # Calculate accuracy scores
    train_accuracy = accuracy_score(y_train, y_train_pred)
```

```
    val_accuracy = accuracy_score(y_test, y_val_pred)

    train_scores.append(train_accuracy)
    val_scores.append(val_accuracy)

# Plotting the training and validation error curves
plt.figure(figsize=(10, 6))
plt.plot(n_neighbors_range, 1 - np.array(train_scores), marker='o',␣
 ↪linestyle='-', color='blue', label='Training Error')
plt.plot(n_neighbors_range, 1 - np.array(val_scores), marker='o',␣
 ↪linestyle='-', color='orange', label='Validation Error')
plt.title('Training and Validation Error Curves for KNeighborsClassifier')
plt.xlabel('Number of Neighbors (n_neighbors)')
plt.ylabel('Error (1 - Accuracy)')
plt.xticks(n_neighbors_range)
plt.legend()
plt.grid(True)
plt.show()
```



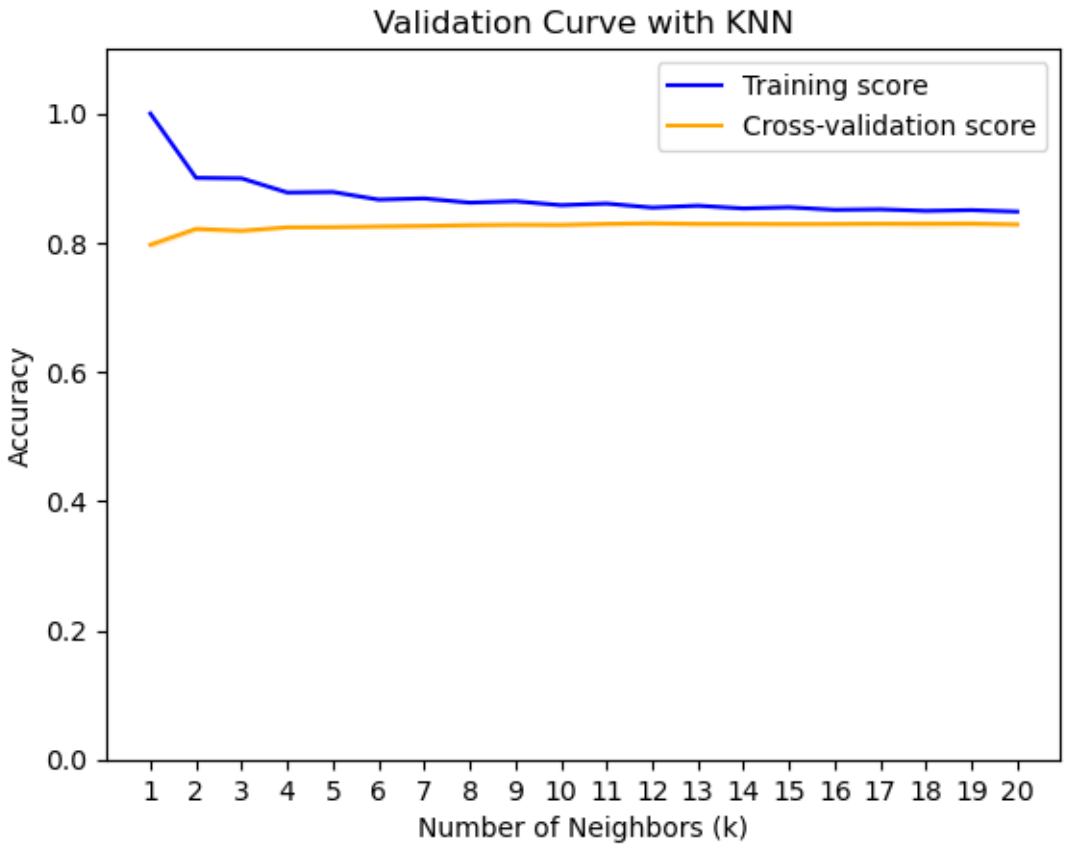
0.8 Additional Visuals

[69]:
```
accuracy_values = test_accuracies
error_values = [1 - accuracy for accuracy in accuracy_values]

plt.figure(figsize=(10, 6))
plt.plot(k_values, error_values, label='Error Rate', marker='o', color='red')
plt.title('Error Rate for Different k Values in KNN: Credit Risk')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Error Rate')
plt.xticks(k_values)
plt.grid()
plt.legend()
plt.show()
```



[18]:
```
from sklearn.model_selection import validation_curve

param_range = np.arange(1, 21)
train_scores, test_scores = validation_curve(
    KNeighborsClassifier(),
    X_train,
    y_train,
    param_name='n_neighbors',
    param_range=param_range,
    cv=5,
```

```
    scoring='accuracy'
)

# Calculate the mean and standard deviation
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
plt.figure()
plt.title("Validation Curve with KNN")
plt.xlabel("Number of Neighbors (k)")
plt.ylabel("Accuracy")
plt.ylim(0.0, 1.1)
plt.xticks(param_range)
plt.plot(param_range, train_mean, label="Training score", color="blue")
plt.fill_between(param_range, train_mean - train_std, train_mean + train_std,␣
 ↪color="blue", alpha=0.1)
plt.plot(param_range, test_mean, label="Cross-validation score", color="orange")
plt.fill_between(param_range, test_mean - test_std, test_mean + test_std,␣
 ↪color="orange", alpha=0.1)
plt.legend(loc="best")
plt.show()
```


Validation Curve with KNN
Training score
Cross-validation score
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Number of Neighbors (k)

[ ]: