Building a classifier that classifies whether a school is High- or Low-performing based on certain Test Scores & other features

Arizona State University
Ira A. Fulton Schools of Engineering
The Polytechnic School

Project Step 4 – Model Building and Evaluation

Submitted By: Raja Muhammad Shayan, Shivam Patel,
Anushka Dhananjaya and Ashval Kumar

Professor: Asmaa Elbadrawy, Ph.D., Lecturer
IFT 511: Analyzing Big Data

Introduction

Problem

To build a **classifier** that classifies whether a given school is high- or low-performing based on its ELA (English Language) or Math score, and several other demographic, academic, and socioeconomic features.

Solving the challenge of school categorization based on performance scores has certain far-reaching consequences that could lay the groundwork to build a more inclusive and prosperous environment in the Phoenix metropolitan area holistically. This enables the Phoenix school district to allocate optimal resources with more efficacy by identifying schools with similar needs. This would guarantee that individual schools within the district would obtain resources based on their needs. This would also help parents make better-informed decisions by choosing schools based on performance levels for their children. Moreover, it would also enable the educators to formulate the coursework based on school performance levels.

Data Collection

We used various online sources to collect our data. The final raw dataset had 344 data points and 24 attributes.

The collected attributes are School Name, City, ZIP Code, Student Population, AP Participation, AP Math Participation, Teacher Qualification, Student-Teacher Ratio, Whites, Blacks, Two or More Races, Native Americans, Asians, Native Hawaiians or Pacific Islanders, Hispanics, Female, Male, Students from Low-Income Families, Dropout Rate, Four-year Graduation Rate, Funding per Student, ELA Score, Math Score, and Arizona School Score.

Target Column

Arizona School Score is our Target column. As our problem states the classifier classifies a particular school into two categories i.e. High- or Low-performing, so we had to modify the target column.
Arizona School Score was a column with data points ranging on a scale of 0 to 10. To classify it into High- or Low-performing, we modified the data so that, the data points with values 0 to 5 are considered Low-performing and those with values 6 to 10 are considered High-performing.

Dataset Link

https://docs.google.com/spreadsheets/d/1Ze9lcuXyOzcwdTdQZjjipSoI9Peff-_AHPsALoPNYlk/edit?usp=sharing

Model Building

We worked on building and evaluating four distinct classification models: Decision Trees, K-nearest neighbors (KNN), Support Vector Machines (SVM), and Naive Bayes. Each model is introduced briefly below, highlighting its key characteristics and suitability for the classification task at hand.

Decision Trees

- Decision Trees are intuitive and easy-to-understand models that partition the feature space based on a series of binary decisions.
- They are capable of handling both numerical and categorical data, making them versatile for various types of datasets.
- Decision Trees are prone to overfitting, especially with complex datasets or deep trees. Pruning techniques can help mitigate this issue.

K-Nearest Neighbors (KNN)

- KNN is a non-parametric, instance-based learning algorithm that classifies new data points based on the majority class among their nearest neighbors.
- It does not make any assumptions about the underlying data distribution, making it suitable for nonlinear decision boundaries.
- However, KNN can be computationally expensive, especially with large datasets, as it requires storing all training data and calculating distances for each prediction.

Support Vector Machines (SVM)

- SVM is a powerful supervised learning algorithm that aims to find the optimal hyperplane to separate data points of different classes with the maximum margin.
- It is effective in high-dimensional spaces and is particularly useful when the number of features exceeds the number of samples.
- SVM can handle linear and nonlinear classification tasks using different kernel functions such as linear, polynomial, or radial basis function (RBF) kernels.

Naive Bayes

- Naive Bayes is a probabilistic classifier based on Bayes' theorem with the "naive" assumption of feature independence given the class.
- It is computationally efficient and can handle large datasets with high-dimensional feature spaces.
- Despite its simplicity and assumption of feature independence, Naive Bayes often performs well in practice, especially with text classification tasks.

Before selecting the final model, each of these models were trained and evaluated using appropriate evaluation metrics such as accuracy, precision, recall, F1-score, and ROC AUC. The model that demonstrates superior performance on these metrics was chosen for further process.

Performance Evaluation of Classification Models

In the evaluation of our four classification models—KNN, Decision Tree (DT), Support Vector Machine (SVM), and Naive Bayes—we utilized various metrics such as accuracy, precision, recall, f1 score, and ROC AUC value to assess their performance. Specifically, accuracy, which measures the proportion of correctly predicted observations, indicated how accurately schools were identified as either high or low-performing by our models. The DT and KNN classifiers achieved the highest accuracies at 0.67 and 0.63 respectively, demonstrating their relatively good performance in correctly classifying schools according to performance categories, with correct predictions about 67% of the time. In contrast, SVM and Naive Bayes showed lower accuracies of 0.54 and 0.33 respectively.

Given that our dataset presented an imbalance in class labels, with 204 high and 109 low instances, accuracy alone was deemed insufficient for a robust assessment. This led us to consider other metrics, recognizing the limitations of accuracy in imbalanced datasets. Precision, which denotes the accuracy of positive predictions, was calculated for each model. It reveals the proportion of schools correctly identified as 'High' performing among those labeled as such by the models. Our Decision Tree classifier exhibited the highest precision at 0.76.

Furthermore, we explored recall—a metric that measures a model's ability to identify all positive cases, also known as the true positive rate. In this project, recall helped us understand how many actual high-performing schools were correctly recognized by our classifiers. The KNN classifier achieved the highest recall at 0.87.

To balance the insights gained from precision and recall, we computed the F1 scores, which combine both metrics into a single indicator. This score is particularly valuable in scenarios with uneven class distributions. The KNN classifier again led with an F1 score of 0.78, suggesting a robust ability to accurately classify schools without mislabeling low performers as high and effectively recognizing true high performers.

Lastly, the performance of our models was also compared using ROC-AUC values, a critical measure for evaluating the efficacy of a binary classification model. This metric illustrates a model's capability to differentiate between the positive and negative classes. While the KNN classifier topped this metric with an ROC-AUC value of 0.56, indicating a superior ability to distinguish between high-performing and low-performing schools, it was followed by the SVM classifier at 0.51. Despite these scores being relatively low, the KNN classifier's lead suggests it was the most effective among the evaluated models in managing the classification challenges presented by our dataset.

Assessing and comparing all these different evaluation metrics, it was apparent that the KNN Classifier was the best model for our classification problem.

Code Explanation

Our code implements a K-Nearest Neighbors (KNN) classifier for the classification task of identifying whether schools are high- or low-performing based on various features.

First, the necessary libraries are imported, including NumPy, pandas, and scikit-learn modules for data manipulation, model building, and evaluation.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score, KFold, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

The cleaned dataset is read into the pandas dataframe, and a new column named 'Target' is created based on the Arizona School Score, where schools with scores greater than 5 are labeled as 'High' and the rest as 'Low'.

```
# Read the cleaned dataset into a pandas dataframe
df = pd.read_csv('Hadoop_Heroes_Cleaned_Dataset.csv')
df
```

	Student Population	AP Participation	AP Math Participation	Teacher Qualification	Student-Teacher Ratio	Whites	Blacks	Two or More Races	Native Americans	Asians	...	85719	85730	85737	85743	85745	8574
0	1000	10	8	80	20	60	20	5	3	5	...	0	0	0	0	0	
1	100	2	1	80	10	50	40	5	3	2	...	0	0	0	0	0	
2	600	5	4	85	18	70	10	5	2	8	...	0	0	0	0	0	
3	800	7	6	88	21	65	12	4	3	8	...	0	0	0	0	0	
4	400	5	4	80	17	50	30	10	5	8	...	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
308	1144	3	15	99	20	45	20	7	4	15	...	0	0	0	0	0	
309	152	8	1	91	35	65	7	6	0	4	...	0	0	0	0	0	
310	59	5	14	93	8	70	5	3	2	20	...	0	0	0	0	0	
311	1739	22	8	98	23	55	15	7	4	15	...	0	0	0	0	0	
312	651	12	12	94	13	55	15	5	3	20	...	0	0	0	0	0	

313 rows × 132 columns

```
# Create 'Target' column based on Arizona School Score
df['Target'] = df['Arizona School Score'].apply(lambda x: 'High' if x > 5 else 'Low')
df
```

AP Math Participation	Teacher Qualification	Student-Teacher Ratio	Whites	Blacks	Two or More Races	Native Americans	Asians	...	85730	85737	85743	85745	85747	85748	85750	85756	Arizona School Score	Target
8	80	20	60	20	5	3	5	...	0	0	0	0	0	0	0	0	5	Low
1	80	10	50	40	5	3	2	...	0	0	0	0	0	0	0	0	6	High
4	85	18	70	10	5	2	8	...	0	0	0	0	0	0	0	0	4	Low
6	88	21	65	12	4	3	8	...	0	0	0	0	0	0	0	0	6	High
4	80	17	50	30	10	5	8	...	0	0	0	0	0	0	0	0	6	High
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15	99	20	45	20	7	4	15	...	0	0	0	0	1	0	0	0	6	High
1	91	35	65	7	6	0	4	...	0	0	0	0	0	1	0	0	5	Low
14	93	8	70	5	3	2	20	...	0	0	0	0	0	1	0	0	6	High
8	98	23	55	15	7	4	15	...	0	0	0	0	0	0	1	0	1	Low
12	94	13	55	15	5	3	20	...	0	0	0	0	0	0	0	1	8	High

Next, the feature matrix (X) and target vector (y) are defined, and the dataset is split into training and testing sets using an 80-20 ratio. Features are then scaled using StandardScaler to ensure all features have the same scale.

```
# Define feature matrix X and target vector y
X = df.drop(['Arizona School Score', 'Target'], axis=1)  # Drop the score and the new target column
y = df['Target']

# Split the dataset into training and testing sets with a 8:2 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

An initial KNN classifier is instantiated with 6 neighbors, and 5-fold cross-validation is performed to evaluate its performance. The mean cross-validation accuracy is printed to assess the classifier's performance.

```
# Initialize the KNN classifier with 6 neighbors
knn = KNeighborsClassifier(n_neighbors=6)

# Perform 5-fold cross-validation
scores = cross_val_score(knn, X_train_scaled, y_train, cv=5)
```

```
# Print the accuracy for each of the 5 folds
print("Accuracy for each fold: ", scores)

# Print the average accuracy across all 5 folds
print("Mean cross-validation accuracy: ", scores.mean())
```

```
Accuracy for each fold:  [0.68 0.68 0.56 0.5  0.62]
Mean cross-validation accuracy:  0.608
```

To further optimize the KNN classifier, hyperparameter tuning is conducted using GridSearchCV. Since the GridSearchCV expects integer labels in the target variable for classification tasks, our target variable 'y_train' is encoded into binary integer labels before fitting the GridSearchCV object. The parameter grid includes different values for the number of neighbors (k), weights, and distance metrics. The best parameters and cross-validation score are printed to identify the optimal model configuration.

```
# Since the GridSearchCV expects integer labels in the target variable for classification tasks, we are going to
# encode our target variable 'y_train' into binary integer labels before fitting the GridSearchCV object

from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode the target variable 'y_train' into binary format
y_train_encoded = label_encoder.fit_transform(y_train)
```

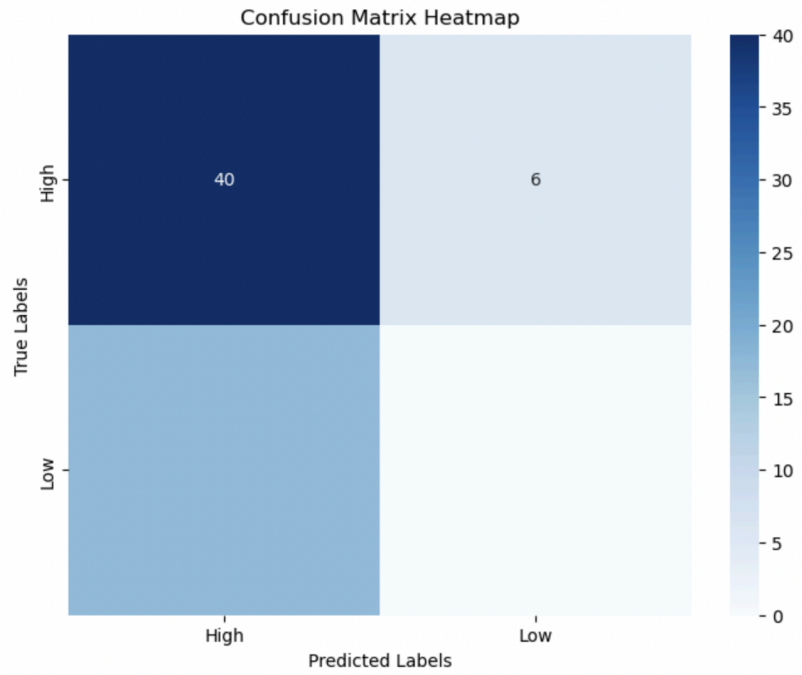
```
# Define the parameter grid for hyperparamter tuning of the KNN Classifier
param_grid = {
    'n_neighbors': range(1, 30),  # Try different values for k from 1 to 30
    'weights': ['uniform', 'distance'],  # Try both uniform and distance weights
    'metric': ['euclidean', 'manhattan']  # Try different distance metrics
}

# Initialize the KNN classifier
knn = KNeighborsClassifier()

# Setup the grid search with 5-fold cross-validation
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')

# Fit the grid search to the training data
grid_search.fit(X_train_scaled, y_train_encoded)

# Print the best parameters found
print("Best parameters:", grid_search.best_params_)

# Print the best cross-validation score
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
```

```
Best parameters: {'metric': 'manhattan', 'n_neighbors': 27, 'weights': 'uniform'}
Best cross-validation score: 0.64
```

The best model obtained from the grid search is then used to make predictions on the test set. A classification report and confusion matrix are generated to evaluate the model's performance, providing insights into precision, recall, and F1 scores for both high and low-performing schools.

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Get the best model from grid search
best_knn_model = grid_search.best_estimator_

# Predict the target values for the test set
y_pred = best_knn_model.predict(X_test_scaled)

# Decode the predicted labels
y_pred_decoded = label_encoder.inverse_transform(y_pred)

# Generate the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred_decoded))

# Generate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_decoded)

# Plot the confusion matrix heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
            xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

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

        High       0.70      0.87      0.78        46
         Low       0.00      0.00      0.00        17

    accuracy                           0.63        63
   macro avg       0.35      0.43      0.39        63
weighted avg       0.51      0.63      0.57        63
```

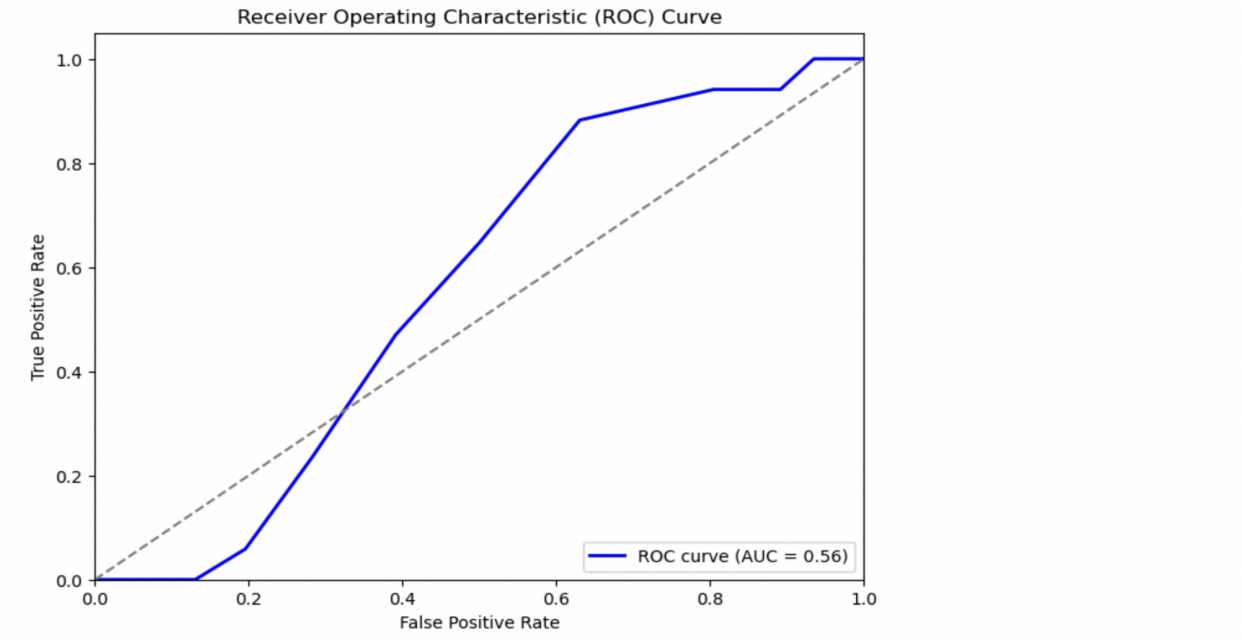


Additionally, the ROC curve and ROC-AUC score are computed to assess the classifier's ability to distinguish between high- and low-performing schools effectively. The ROC curve

visually represents the true positive rate against the false positive rate, while the ROC-AUC score quantifies the classifier's performance, with higher values indicating better performance.

```
# Plot the ROC Curve and calculate the ROC-AUC value for the KNN Classifier

from sklearn.metrics import roc_curve, roc_auc_score

# Encode the true labels 'y_test' into binary format
y_test_encoded = label_encoder.fit_transform(y_test)

# Calculate the predicted probabilities for the positive class ('High')
y_pred_proba = best_knn_model.predict_proba(X_test_scaled)[:, 1]

# Calculate the false positive rate, true positive rate, and threshold values for the ROC curve
fpr, tpr, thresholds = roc_curve(y_test_encoded, y_pred_proba)

# Calculate the ROC-AUC score
roc_auc = roc_auc_score(y_test_encoded, y_pred_proba)

# Print the ROC-AUC score
print("ROC-AUC score:", roc_auc)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='gray', 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 (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
ROC-AUC score: 0.5613810741687979
```



Our code demonstrates the process of building, evaluating, and optimizing a KNN classifier for the classification task of identifying high- and low-performing schools based on relevant features, showcasing the comprehensive analysis required in data-driven decision-making processes.

Code

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score, KFold, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Read the cleaned dataset into a pandas dataframe
df = pd.read_csv('Hadoop_Heroes_Cleaned_Dataset.csv')

# Create 'Target' column based on Arizona School Score
df['Target'] = df['Arizona School Score'].apply(lambda x: 'High' if x > 5 else 'Low')

# Define feature matrix X and target vector y
X = df.drop(['Arizona School Score', 'Target'], axis=1)  # Drop the score and the new target
column
y = df['Target']

# Split the dataset into training and testing sets with a 8:2 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Initialize the KNN classifier with 6 neighbors
knn = KNeighborsClassifier(n_neighbors=6)

# Perform 5-fold cross-validation
scores = cross_val_score(knn, X_train_scaled, y_train, cv=5)

# Print the accuracy for each of the 5 folds
print("Accuracy for each fold: ", scores)

# Print the average accuracy across all 5 folds
print("Mean cross-validation accuracy: ", scores.mean())
```

```
# Since the GridSearchCV expects integer labels in the target variable for classification tasks, we are going to
# encode our target variable 'y_train' into binary integer labels before fitting the GridSearchCV object

from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode the target variable 'y_train' into binary format
y_train_encoded = label_encoder.fit_transform(y_train)

# Define the parameter grid for hyperparamter tuning of the KNN Classifier
param_grid = {
    'n_neighbors': range(1, 30),  # Try different values for k from 1 to 30
    'weights': ['uniform', 'distance'],  # Try both uniform and distance weights
    'metric': ['euclidean', 'manhattan']  # Try different distance metrics
}

# Initialize the KNN classifier
knn = KNeighborsClassifier()

# Setup the grid search with 5-fold cross-validation
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')

# Fit the grid search to the training data
grid_search.fit(X_train_scaled, y_train_encoded)

# Print the best parameters found
print("Best parameters:", grid_search.best_params_)

# Print the best cross-validation score
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))

import seaborn as sns
import matplotlib.pyplot as plt

# Get the best model from grid search
best_knn_model = grid_search.best_estimator_
```

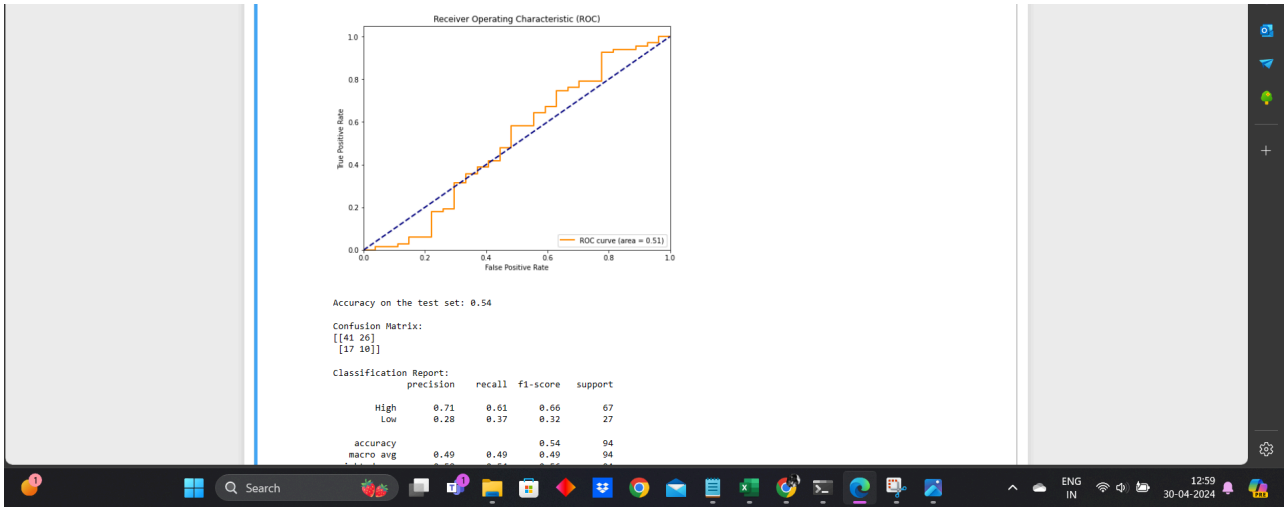
```
# Predict the target values for the test set
y_pred = best_knn_model.predict(X_test_scaled)

# Decode the predicted labels
y_pred_decoded = label_encoder.inverse_transform(y_pred)

# Generate the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred_decoded))

# Generate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_decoded)

# Plot the confusion matrix heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
        xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()

# Plot the ROC Curve and calculate the ROC-AUC value for the KNN Classifier

from sklearn.metrics import roc_curve, roc_auc_score

# Encode the true labels 'y_test' into binary format
y_test_encoded = label_encoder.fit_transform(y_test)

# Calculate the predicted probabilities for the positive class ('High')
y_pred_proba = best_knn_model.predict_proba(X_test_scaled)[:, 1]

# Calculate the false positive rate, true positive rate, and threshold values for the ROC curve
fpr, tpr, thresholds = roc_curve(y_test_encoded, y_pred_proba)

# Calculate the ROC-AUC score
roc_auc = roc_auc_score(y_test_encoded, y_pred_proba)

# Print the ROC-AUC score
```

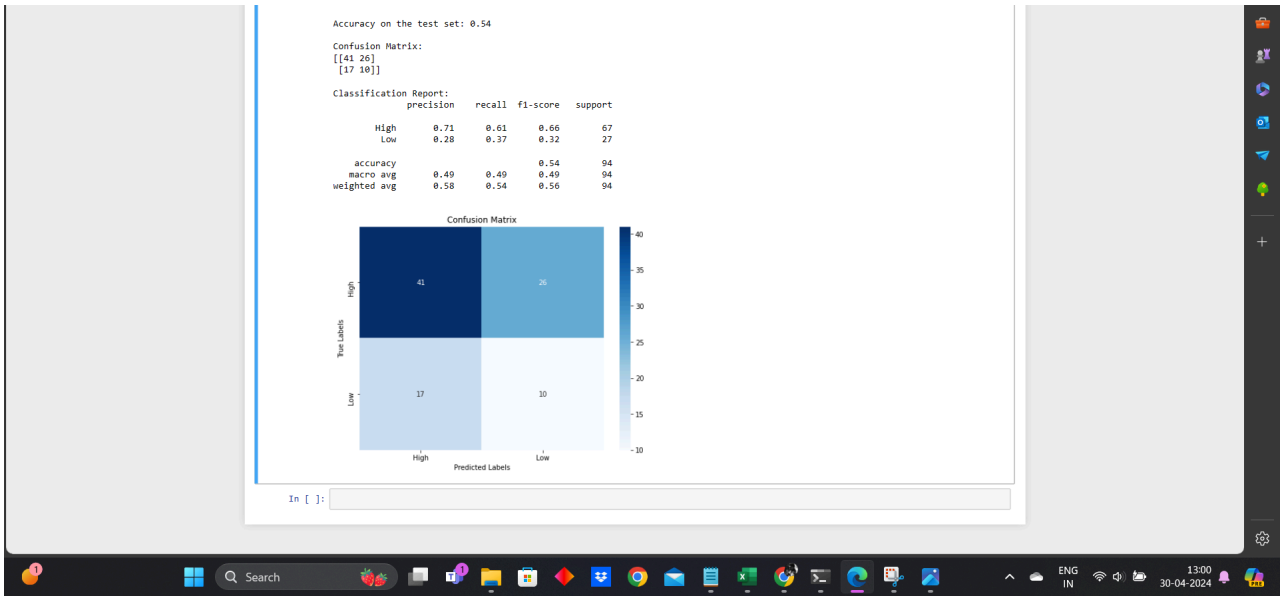
```
print("ROC-AUC score:", roc_auc)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='gray', 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 (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

Evaluation of Other Models Considered

Support Vector Machine (SVM)

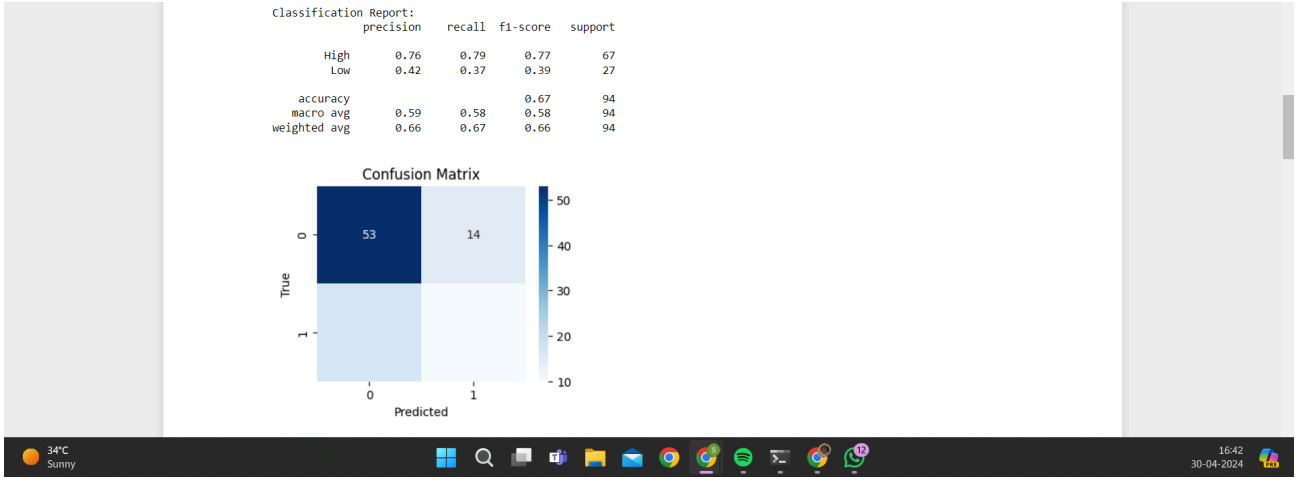
The SVM classifier demonstrated moderate accuracy in cross-validation of 64% but limited ability to distinguish between classes as indicated by the ROC AUC of 0.53. While the model performs reasonably well in predicting 'High' outcomes with precision of 71% and F1-score of 66%, it struggles significantly with predicting 'Low' outcomes, as evidenced by low precision (28%), recall (37%), and F1-score (32%). The confusion matrix further illustrates this challenge, with a considerable number of false negatives and false positives affecting the model's overall performance. This suggested that consideration of different classification algorithms is necessary for predictive performance.

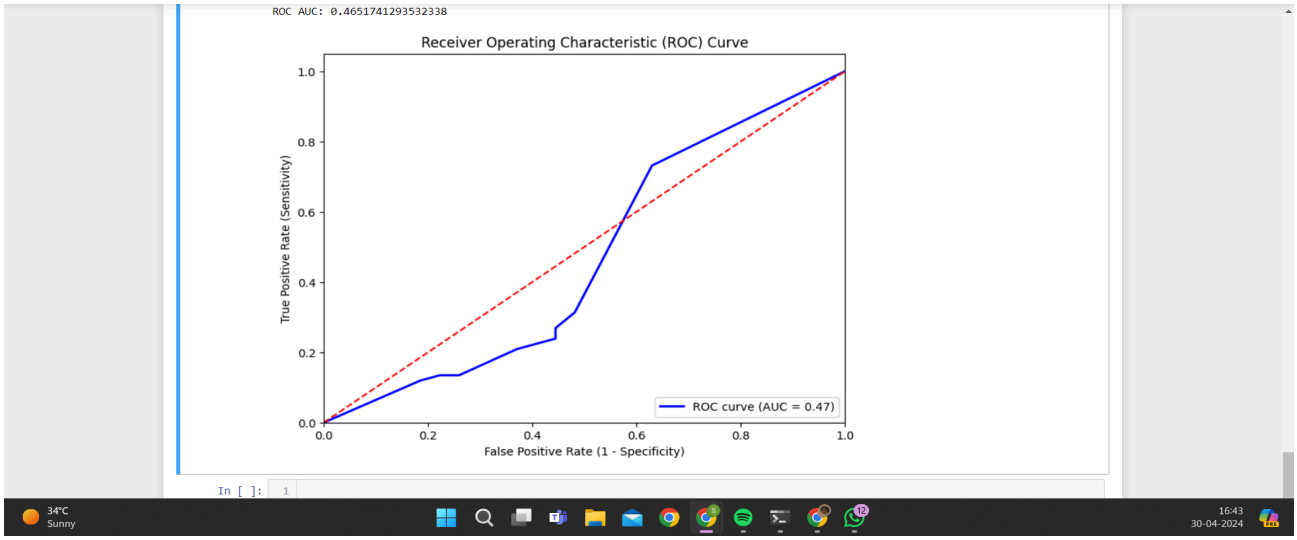




Decision Tree (DT)

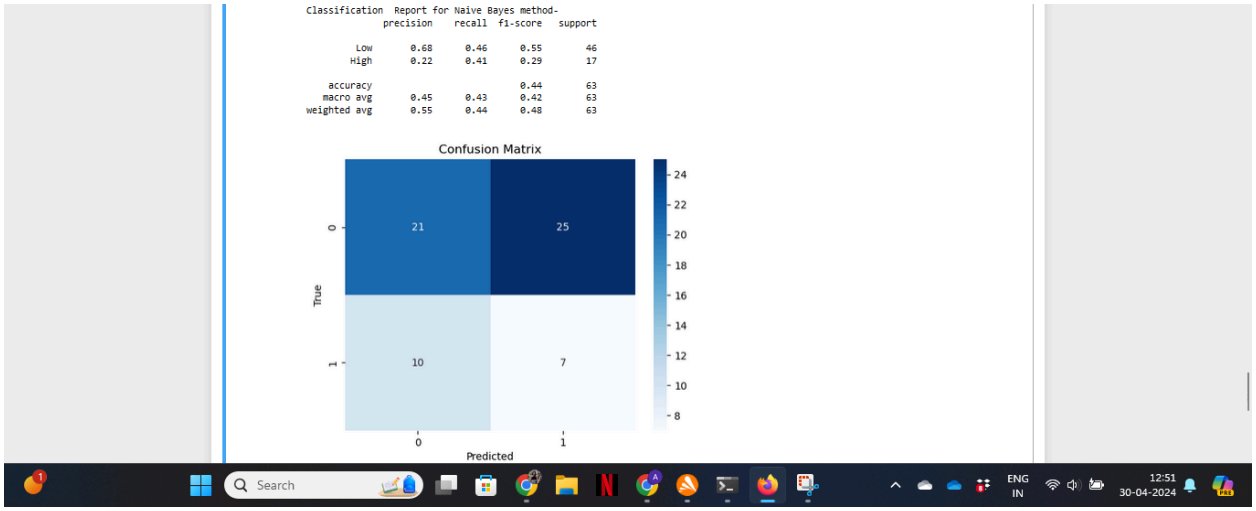
For the Decision Tree classifier, the evaluation metrics indicate moderate performance. The accuracy stands at 0.67, reflecting the proportion of correctly classified instances among the total predictions. Precision, which measures the proportion of true positive predictions among all positive predictions, is relatively high at 0.76, indicating the model's ability to accurately identify true positives. The recall score, indicating the proportion of actual positives that were correctly identified by the model, is also notable at 0.79, suggesting the classifier's capability to capture the most positive instances. The F1-score, which balances precision and recall, is calculated at 0.77, indicating a harmonic mean between precision and recall. However, the ROC AUC score, which measures the area under the receiver operating characteristic curve, stands at 0.465, indicating that the model's ability to distinguish between positive and negative classes is only slightly better than random guessing. Overall, while the Decision Tree classifier shows promise in certain aspects, further optimization may be required to enhance its performance across all evaluation metrics.

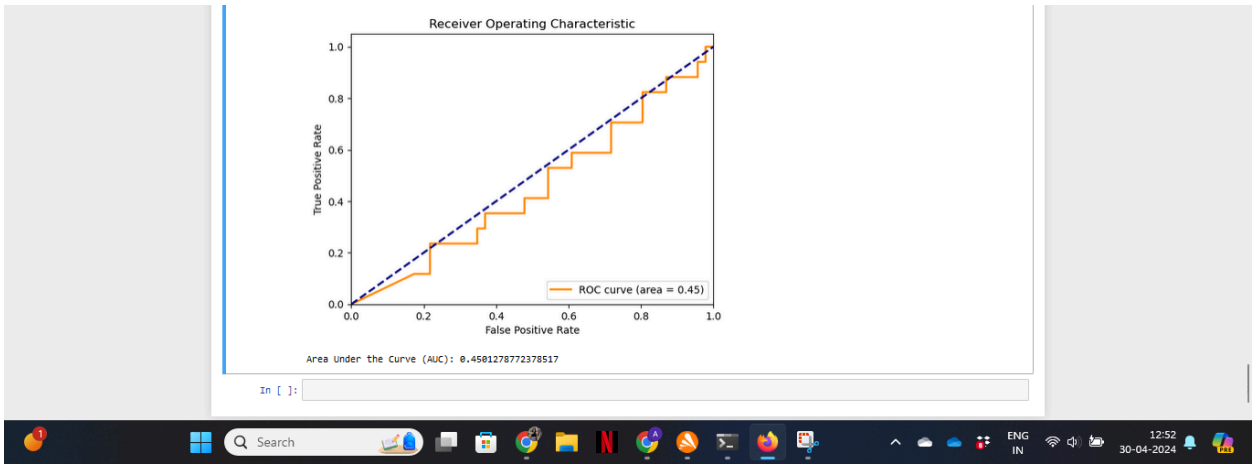




Naive Bayes

The Naive Bayes classifier showed moderate accuracy of 44% but limited discriminative capability, evidenced by an ROC AUC of 0.45. It performed reasonably with a precision of 68% for low-performing schools but struggled with high-performing schools, achieving only 22% precision. The F1 scores further highlighted this issue, scoring 0.55 for low outcomes and 0.29 for high outcomes. The confusion matrix revealed significant misclassifications, particularly false positives and false negatives. This performance suggests Naive Bayes may not suit datasets with complex feature correlations, and exploring alternative models or improving feature engineering could enhance predictive accuracy.





Practical Implications

Targeted Resource Allocation

By accurately identifying high and low-performing schools, the classification models can help education authorities allocate resources more effectively. Low-performing schools can receive targeted interventions, additional resources, and support programs to improve their outcomes.

Policy Enhancement

The insights gained from model predictions can guide the development of nuanced educational policies that address specific needs, helping to elevate educational standards and student achievements across different school demographics.

Proactive Interventions

With reliable predictions on school performance, educational stakeholders can implement proactive measures to support schools at risk of declining performance and potentially prevent deterioration before it becomes entrenched.

Data-driven Decision Making

These models empower educational leaders to make informed decisions based on data-driven insights as well as enhance the strategic planning and operational efficiency of educational programs.

Monitoring and Evaluation

Regular use of these models can help in the ongoing monitoring and evaluation of school performance over time also providing a metric for assessing the impact of educational interventions and reforms.

Future Improvements

Enhanced Data Balancing Techniques

Applying advanced techniques to address data imbalance, such as adaptive synthetic sampling or tailored undersampling, can improve model performance by ensuring that the training data more accurately reflects the diversity of real-world conditions.

Comprehensive Feature Engineering

Investigating additional features or extracting more nuanced information from existing data could reveal deeper insights into factors influencing school performance. This might include more granular socio-economic data, longitudinal study results, or more detailed metrics on school administration effectiveness.

Robust Cross-Validation Strategies

Implementing more sophisticated cross-validation strategies, like stratified or group k-fold validation, can ensure that model evaluation is robust and generalizable across different subsets of the data.

Regular Model Updates and Retraining

Establishing a routine for periodically updating and retraining the models with new data can help capture evolving trends and changes in the educational environment, ensuring that the models remain relevant and effective.

Geographic Adaptation of Models

Customizing models to account for geographic variations in educational practices and standards can enhance their applicability and accuracy in different regions or educational systems.

Model Interpretability and Transparency

Enhancing the interpretability of models can help non-technical stakeholders understand prediction outcomes better, which is crucial for gaining trust and facilitating the adoption of AI-driven decision-making in education.

Ethical Considerations and Bias Mitigation

Continuously assessing and addressing potential biases in the models is crucial. Ensuring that the models do not inadvertently perpetuate existing inequalities or biases is essential for ethical AI practices in education.