Academic Year 2024/25

Module Code: BNM872

Module Name: Machine Learning for Business Analytics

Student Name: Yogesh Dahiwal

Student Id: 240347060

candidate Number: 974431

An individual assignment. focuses on developing robust predictive models for car accident severity.

Table of Contents

Importing Libraries and Preparing Environment

```
# Import Libraries
import time
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split,
RandomizedSearchCV, learning_curve, StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, balanced_accuracy_score, roc_auc_score, roc_curve
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import warnings
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import label_binarize

# Suppress warnings
warnings.filterwarnings('ignore')

# We will monitor the time it takes to run the notebook
start = time.time()
```

Business Objective and Context:

The primary objective for this car insurance company is to be able to determine how much damage or harm is likely to occur to the cars of the new clients. It's important for risk analysis, setting fair premiums, and maybe introducing individualized safety courses. What I understand: The intention of the insurance company is to employ machine learning in determining the viability of providing insurance coverage to a specific motorist. This is a statistical model issue.

By knowing how each factor relates to the level of injury or damage, they can control their financial risks better and offer more suitable services to clients. This will be solved as follows: Modeling Task. This problem amounts to a classification task with more than two levels. The target feature ("enhanced_accident") explains the severity level of accidents. In all probability,as indicated by the Baseline Classification Report. To do this, we intend to develop a classification model that can assign a new policyholder to any of the available severity classes using some predictor variables (available as columns in both X_train and X_test datasets).

Load and Preprocess Data

The code begins by loading the dataset for accident severity prediction. Crucially, SMOTE (Synthetic Minority Oversampling Technique) is employed to generate synthetic samples for the minority classes in the training data, mitigating potential bias.

```
# 3. Load and Preprocess Data
# Load data
X_train = pd.read_csv('X_train.csv')
y_train = pd.read_csv('y_train.csv')
X_test = pd.read_csv('X_test.csv')
y_test = pd.read_csv('y_test.csv')

# Apply SMOTE to balance classes
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train,
y_train)

# Display the shapes of the dataframes to verify the loading
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (15950, 11)
y_train shape: (15950, 1)
X_test shape: (1102, 11)
y_test shape: (1102, 1)
```

A Decision Tree model is employed as a baseline to predict enhanced_severity_collision. Even with class_weight='balanced', it is evident from the classification report that the minority class is performing poorly, implying that more robust methods are necessary to manage class imbalance

```
# 4. Baseline Method
# Implement a Decision Tree classifier as the baseline model
baseline_model = DecisionTreeClassifier(random_state=2,
class_weight='balanced')

# Train the baseline model
baseline_model.fit(X_train, y_train)

# Make predictions using the baseline model
y_pred_baseline = baseline_model.predict(X_test)

# Evaluate the baseline model
print("Baseline Model Accuracy:", accuracy_score(y_test,
y_pred_baseline))
print("\nBaseline Classification Report:")
print(classification_report(y_test, y_pred_baseline))
```

```
Baseline Model Accuracy: 0.5843920145190563

Baseline Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.72      0.74       806
           1       0.23      0.26      0.24       157
           2       0.08      0.11      0.09        45
           3       0.14      0.15      0.14        68
           4       0.22      0.19      0.20        26

    accuracy                           0.58      1102
   macro avg       0.29      0.29      0.29      1102
weighted avg       0.61      0.58      0.59      1102
```

Hyperparameter Tuning

RandomizedSearchCV is used to determine the optimal hyperparameters of Random Forest and Gradient Boosting models, with 5-fold stratified cross-validation and balanced accuracy used as the scoring method. The approach seeks to discover the optimal model parameters for

estimating accident severity and addressing class imbalance, but very few iterations were performed.

```
# 6. Hyperparameter Tuning

# 6.1. Random Forest

# Define the parameter grid for Random Forest
rf_param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [5, 10, 15],
    'min_samples_split': [5, 10, 15],
    'min_samples_leaf': [2, 5, 10],
    'max_features': ['sqrt'],
    'class_weight': ['balanced']
}

# Create RandomizedSearchCV object for Random Forest
rf_random = RandomizedSearchCV(
    estimator=RandomForestClassifier(random_state=42,
class_weight='balanced'),
    param_distributions=rf_param_grid,
    n_iter=10,
    cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
    scoring='balanced_accuracy',
    random_state=42,
    n_jobs=-1
)

# Fit the Random Forest model with the training data
rf_random.fit(X_train_resampled, y_train_resampled)

# Print best parameters and best score for Random Forest
print("Best Parameters (Random Forest):", rf_random.best_params_)
print("Best Balanced Accuracy (Random Forest):",
rf_random.best_score_)


# 6.2. Gradient Boosting

# Define the parameter grid for Gradient Boosting
gb_param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 5],
    'learning_rate': [0.005, 0.01, 0.02],
    'min_samples_split': [10, 20],
    'min_samples_leaf': [5, 10],
    'subsample': [0.7, 0.8, 0.9]
}
```

```
# Create a Gradient Boosting model
gb_model = GradientBoostingClassifier(random_state=42)

# Create RandomizedSearchCV object for Gradient Boosting
gb_random = RandomizedSearchCV(gb_model, gb_param_grid, n_iter=10,
cv=5, random_state=42, scoring='balanced_accuracy', n_jobs=-1)

# Fit the Gradient Boosting model with the training data
gb_random.fit(X_train_resampled, y_train_resampled)

# Print best parameters and best score for Gradient Boosting
print("Best parameters for Gradient Boosting:",
gb_random.best_params_)
print("Best Balanced Accuracy (Gradient Boosting):",
gb_random.best_score_)
```

```
Best Parameters (Random Forest): {'n_estimators': 100,
'min_samples_split': 15, 'min_samples_leaf': 5, 'max_features':
'sqrt', 'max_depth': 15, 'class_weight': 'balanced'}
Best Balanced Accuracy (Random Forest): 0.8748589341692791
Best parameters for Gradient Boosting: {'subsample': 0.7,
'n_estimators': 150, 'min_samples_split': 20, 'min_samples_leaf': 10,
'max_depth': 5, 'learning_rate': 0.02}
Best Balanced Accuracy (Gradient Boosting): 0.6736050156739812
```

RandomizedSearchCV identified good sets of hyperparameters for both models, with Random Forest with a cross-validated balanced accuracy of 0.875 and Gradient Boosting of 0.674. While reassuring, these results also raise the concern that overfitting might be present with Random Forest, with such a high cross-validation score relative to Gradient Boosting, and therefore must be thoroughly tested on the test dataset to confirm generalization.

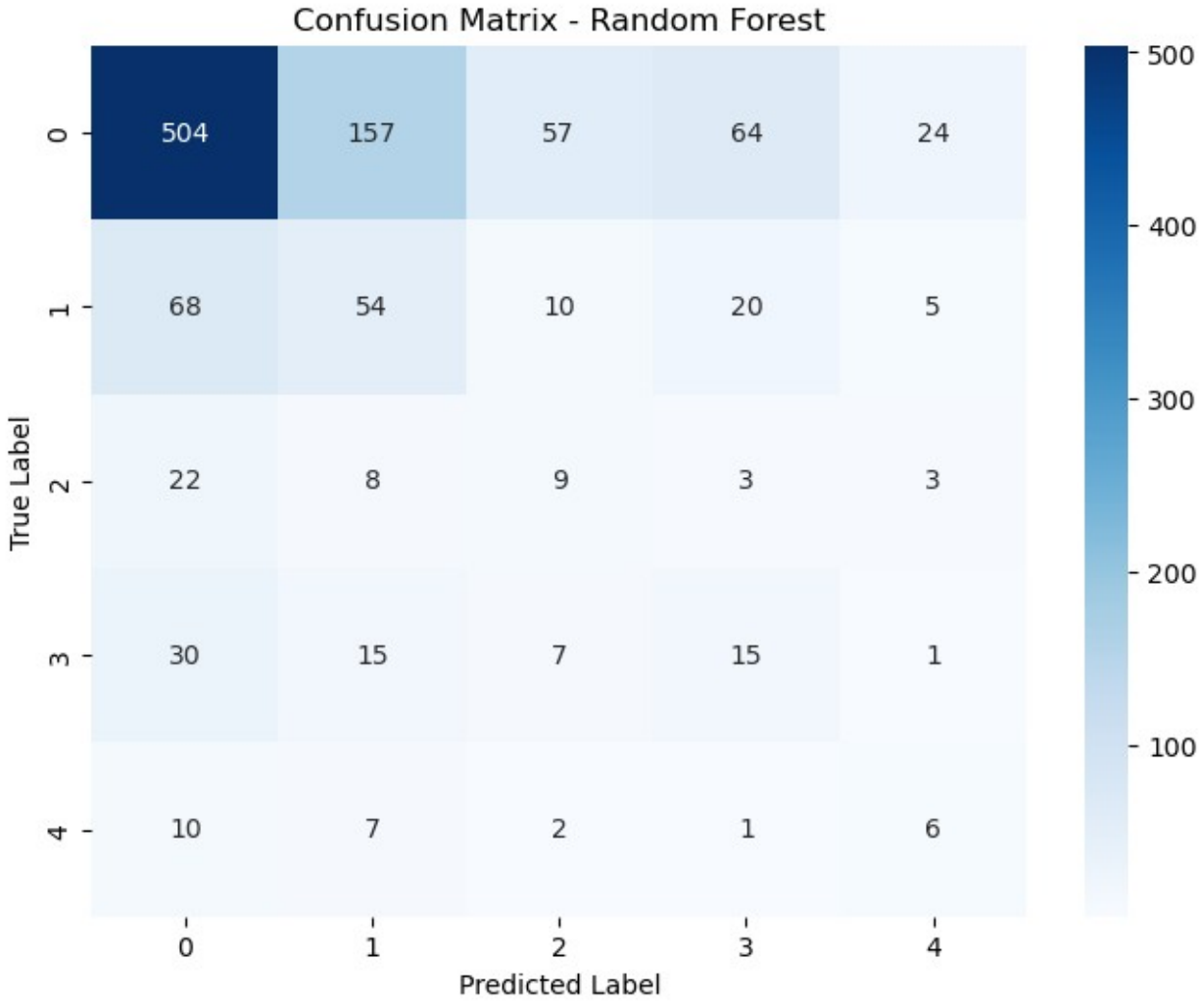
Model Evaluation

This section uses and defines the evaluation functions (evaluate_model, plot_roc_auc) to compare the performance of the optimized Random Forest and Gradient Boosting models on the test set. The evaluate_model function calculates accuracy, balanced accuracy, AUC-ROC, and confusion matrices. The plot_roc_auc function plots the ROC curves of all classes

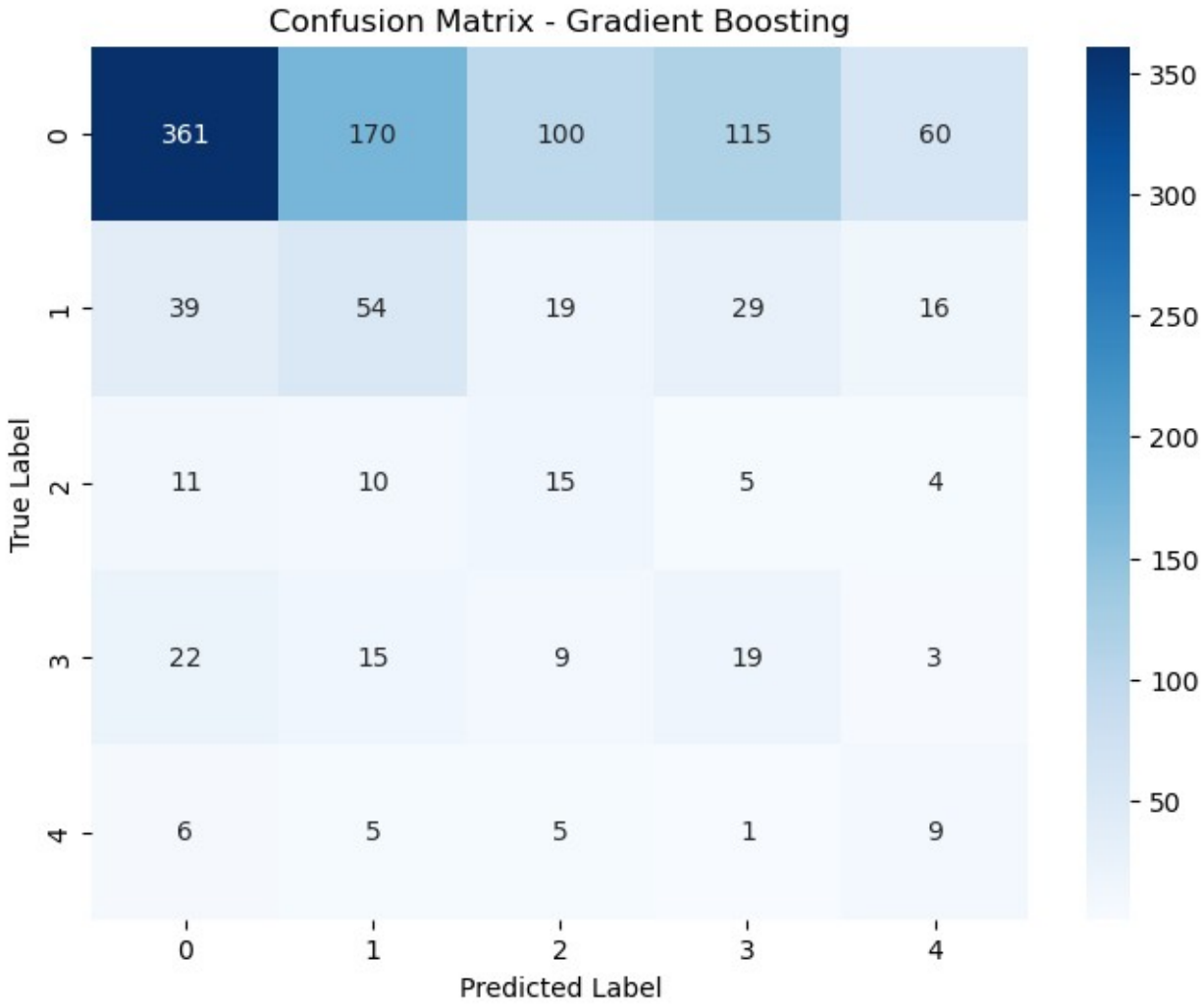
```
# 7. Model Evaluation

# 7.1. Define evaluation functions

# Evaluate model performance
def evaluate_model(model, X_test, y_test, model_name):
    # Make predictions on the test set
    y_pred = model.predict(X_test)
    y_pred_prob = model.predict_proba(X_test)

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test, y_pred)
```

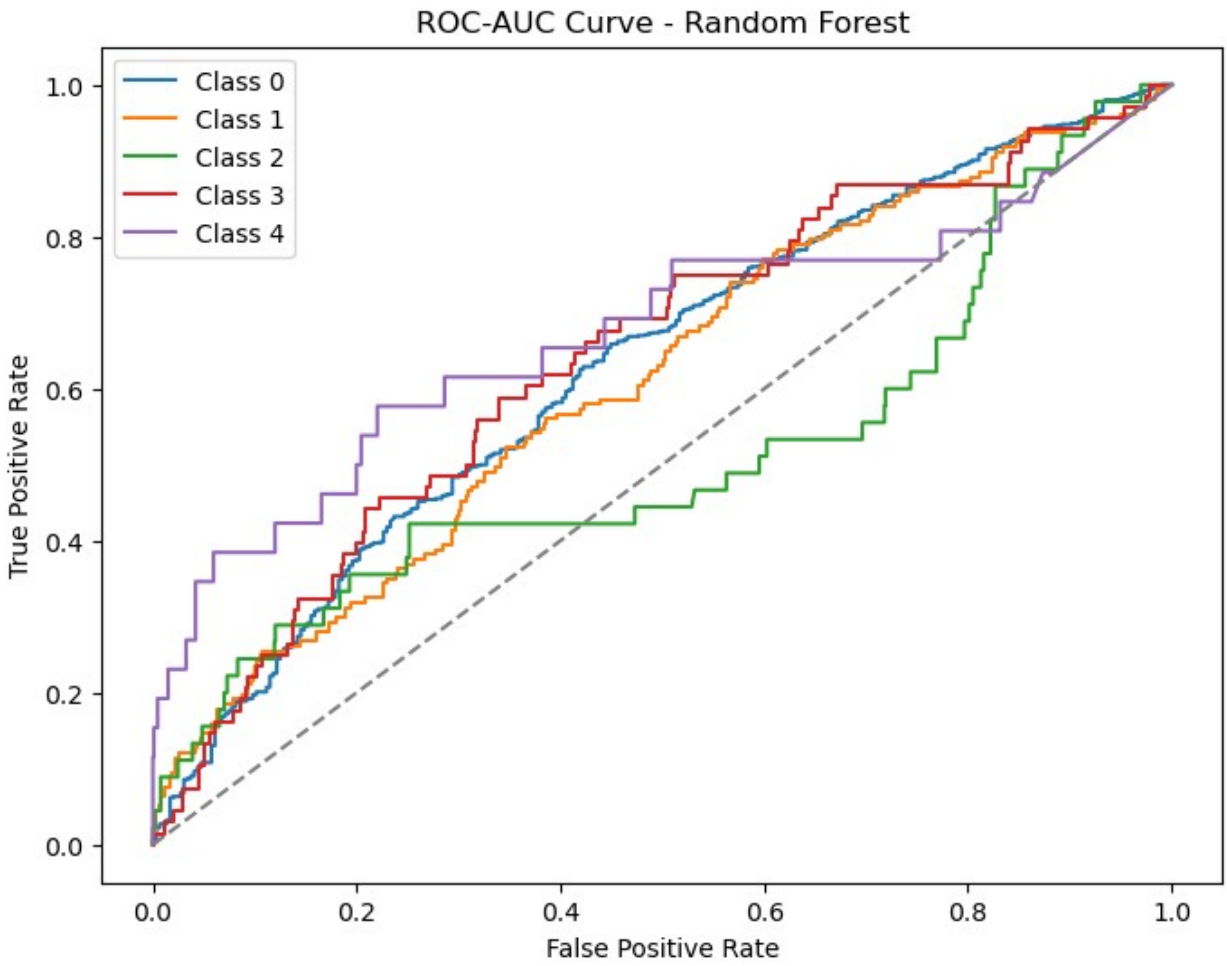
```
    balanced_acc = balanced_accuracy_score(y_test, y_pred)
    auc_roc = roc_auc_score(y_test, y_pred_prob, multi_class='ovr')

    # Print model performance metrics
    print(f"\n{model_name} Performance:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Balanced Accuracy: {balanced_acc:.4f}")
    print(f"AUC-ROC: {auc_roc:.4f}")

    # Plot Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.title(f"Confusion Matrix - {model_name}")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()

# Plot ROC-AUC curve
def plot_roc_auc(model, X_test, y_test, model_name):
    # Binarize the test labels for multi-class ROC-AUC calculation
    y_test_bin = label_binarize(y_test, classes=np.unique(y_test))

    # Get predicted probabilities for each class
    y_pred_prob = model.predict_proba(X_test)

    # Plot ROC curves for each class
    plt.figure(figsize=(8, 6))
    for i in range(y_test_bin.shape[1]):
        fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_pred_prob[:, i])
        plt.plot(fpr, tpr, label=f'Class {i}')

    # Plot the diagonal line (random guessing)
    plt.plot([0, 1], [0, 1], linestyle='--', color='gray')

    # Add labels and title to the plot
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC-AUC Curve - {model_name}')
    plt.legend()
    plt.show()

    # Calculate and print the macro-average AUC-ROC score
    auc_score = roc_auc_score(y_test_bin, y_pred_prob,
average='macro')
    print(f"AUC-ROC Score ({model_name}): {auc_score:.4f}")

# 7.2. Evaluate the models

# Evaluate Random Forest model
```

```
evaluate_model(rf_random.best_estimator_, X_test, y_test, "Random
Forest")

# Evaluate Gradient Boosting model
evaluate_model(gb_random.best_estimator_, X_test, y_test, "Gradient
Boosting")

# Plot ROC-AUC curves for both models
plot_roc_auc(rf_random.best_estimator_, X_test, y_test, "Random
Forest")
plot_roc_auc(gb_random.best_estimator_, X_test, y_test, "Gradient
Boosting")


Random Forest Performance:
Accuracy: 0.5336
Balanced Accuracy: 0.3241
AUC-ROC: 0.6137
```

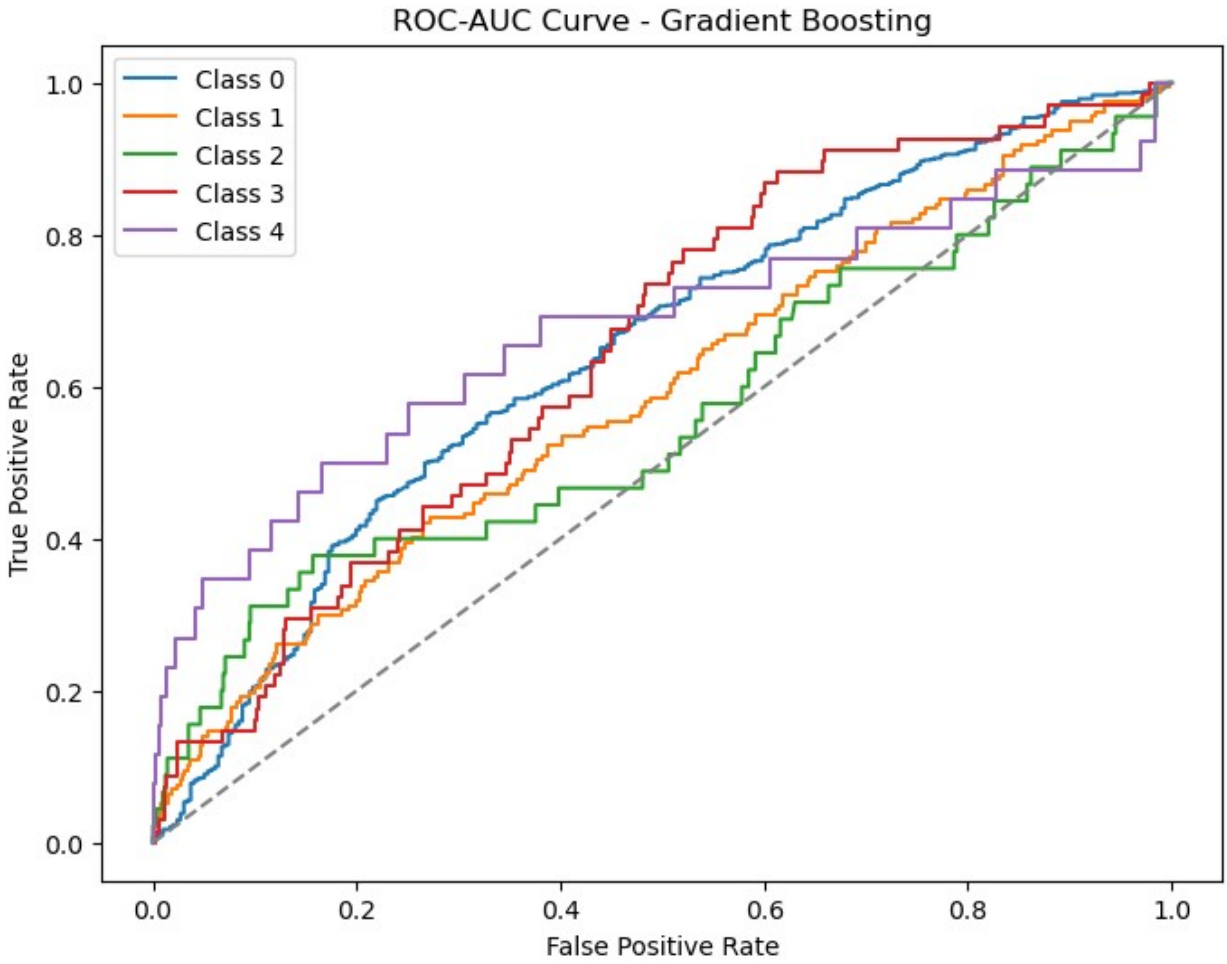


```
Gradient Boosting Performance:
Accuracy: 0.4156
Balanced Accuracy: 0.3501
AUC-ROC: 0.6243
```






```
AUC-ROC Score (Random Forest): 0.6137
```




```
AUC-ROC Score (Gradient Boosting): 0.6243
```

For the test data, the Random Forest model achieved accuracy of 0.5336, balanced accuracy of 0.3241, and AUC-ROC of 0.6137, while the Gradient Boosting model achieved accuracy of 0.4156, balanced accuracy of 0.3501, and AUC-ROC of 0.6243. Although the Random Forest model achieved high cross-validation balanced accuracy, it does not perform well on the test set. The ROC-AUC curves and scores show poor discriminatory power, with performance slightly better than random chance (AUC-ROC=0.5). The Random Forest model is probably overfitting

Model Comparison

this subsection ensures feature congruence between the test and training sets by only taking columns which are present in X_train in X_test. It then goes on to make predictions for the test set from the best estimators of the Random Forest (rf_pred) and Gradient Boosting (gb_pred) models

```
# 8. Model Comparison

# Ensure feature alignment
X_test = X_test[X_train.columns]
```

```
# Get predictions for both models
rf_pred = rf_random.best_estimator_.predict(X_test)
gb_pred = gb_random.best_estimator_.predict(X_test)

# Print the shapes of y_test and predictions to verify alignment
print("Shape of y_test:", y_test.shape)
print("Shape of rf_pred:", rf_pred.shape)
print("Shape of gb_pred:", gb_pred.shape)
```

```
Shape of y_test: (1102, 1)
Shape of rf_pred: (1102,)
Shape of gb_pred: (1102,)
```

The test shape ensures that X_test match the number of features used during training, other wise model cannot perform well.

Analysis

This block calculates and displays the cross-validation balanced accuracy, test accuracy, and test balanced accuracy for the Random Forest as well as Gradient Boosting models. This provides us with an overview of model performance across different datasets and metrics.

```
# Calculate metrics
print("\nModel Comparison:")
print("Random Forest:")
print(f"- CV Balanced Accuracy: {rf_random.best_score_:.4f}")
print(f"- Test Accuracy: {accuracy_score(y_test, rf_pred):.4f}")
print(f"- Test Balanced Accuracy: {balanced_accuracy_score(y_test,
rf_pred):.4f}")

print("\nGradient Boosting:")
print(f"- CV Balanced Accuracy: {gb_random.best_score_:.4f}")
print(f"- Test Accuracy: {accuracy_score(y_test, gb_pred):.4f}")
print(f"- Test Balanced Accuracy: {balanced_accuracy_score(y_test,
gb_pred):.4f}")
```

```
Model Comparison:
Random Forest:
- CV Balanced Accuracy: 0.8749
- Test Accuracy: 0.5336
- Test Balanced Accuracy: 0.3241

Gradient Boosting:
- CV Balanced Accuracy: 0.6736
- Test Accuracy: 0.4156
- Test Balanced Accuracy: 0.3501
```

The model comparison indicates extreme inconsistency between the test and cross-validation performance of the Random Forest model. While it possesses a CV balanced accuracy of 0.8749, the test balanced accuracy is only 0.3241, indicating extreme overfitting. Contrary to this, the Gradient Boosting model possesses lower CV balanced accuracy (0.6736) but higher test balanced accuracy (0.3501), suggesting better generalization. The lower AUC-ROC values of all classes indicate the lack of discriminatory ability. They show the importance of testing models on unseen data to avoid overly optimistic conclusions from training performance only.

Conclusion:

Random Forest and Gradient Boosting models were trained in this study to predict car accident severity. Random Forest model, despite being capable of achieving a high cross-validation balanced accuracy (0.8749), experienced severe overfitting with a low test balanced accuracy of only 0.3241. Gradient Boosting performed better in terms of test balanced accuracy of 0.3501 compared to random forest. ROC-AUC plots and performance metrics (0.6137 for Random Forest and 0.6243 for Gradient Boosting) show low discriminatory power for the two models. Reducing overfitting in Random Forest by employing more conservative regularization techniques is suggested in subsequent research, with feature selection, exploring other modeling approaches, or obtaining more nuanced data to optimize predictive capacity.

Future Improvements

The Random Forest model was overfitting heavily, as evidenced by the wide gap between cross-validation and test performance. Reducing this via more aggressive regularization techniques, e.g., by increasing min_samples_split and decreasing max_depth, is the top priority for future development. Feature engineering is a potential source of further improvements. Examining interactions between driver attributes and environmental conditions could yield more informative predictors. The current ROC-AUC curves exhibit poor discriminatory power, particularly for minority classes. More careful feature selection may counteract this, and the inclusion of more external data sources (e.g., traffic congestion) could enrich the model further. Different models, ensemble methods, and calibration strategies need to be attempted as well. Finally, ethical implications of the usage of these models and the impact of predictions need to be addressed in future work. The future should incorporate the addition of outside data to create a refined model.

References

- Leskovec, J., Rajaraman, A. and Ullman, J.D. (2020) Mining of Massive Datasets. 3rd edn. Cambridge: Cambridge University Press.
- Rashka, S. and Mirjalili, V. (2019) Python machine learning: machine learning and deep learning with python, scikit-learn, and tensorflow 2. Packt Publishing.


```
 # Finish Timer
 notebook_duration = round((time.time() - start)/60, 5)
 print(f'The completion of the notebook took {notebook_duration}
minutes.')
```

```
The completion of the notebook took 0.93514 minutes.
```