NAAN MUDHALVAN PROJECT

PHASE – IV

NAME : M.SHOBEYA

DOMAIN : ARTIFICIAL INTELLIGENCE

TOPIC : DEVELOPMENT-AUTONOMOUS VEHICLES

DEPT. : COMPUTER SCIENCE

COLLEGE : 8201 ARJCET

INTRODUCTION

The 2019 autonomous disengagement reports highlight crucial aspects of model development and evaluation metrics in autonomous vehicle technology. This report provides a concise overview of these processes, focusing on data collection, algorithm design, and validation techniques. Additionally, it explores the evaluation metrics used to quantify system performance and safety. By analysing these methodologies, we aim to offer insights into the current state of autonomous driving technology and foster discussions for future advancements in the field.

OBJECTIVE

- Optimize Model Performance: Improve algorithms to minimize disengagements and enhance safety.

- Enhance Data Collection: Gather diverse datasets to capture real-world scenarios.

- Refine Algorithm Design: Adapt algorithms for complex driving situations.

- Validate Model Efficacy: Ensure consistent and reliable performance through robust validation.

- Quantify Safety Metrics: Measure safety aspects like disengagement rates and reaction times.

- Benchmark Performance: Establish industry-wide standards for comparison.

- Enable Regulatory Compliance: Align with safety regulations and standards.

- Guide R&D: Use metrics for innovation and improvement.

MODEL DEVELOPMENT

ALGORITHM SELECTION

"Algorithm selection is the process of choosing the best computational methods for autonomous driving systems. It involves picking algorithms that perform well, are robust, scalable, safe, and adaptable to various driving conditions. This selection directly impacts the system's ability to navigate safely and efficiently."

CODE

```
import pandas as pd
disengagement_data = pd.DataFrame({
```

```
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure']
})
def select_algorithm(disengagement_data):
    algorithm_counts = disengagement_data['algorithm'].value_counts()
    most_frequent_algorithm = algorithm_counts.idxmax()
    return most_frequent_algorithm
```

```
selected_algorithm =
select_algorithm(disengagement_data)
print("Selected algorithm for 2019
autonomous disengagement reports:",
selected_algorithm)
```

OUTPUT

```
Selected algorithm for 2019 autonomous disengagement reports: A
```

MODEL TRAINING

"Model training is the process of teaching algorithms to understand driving scenarios and make autonomous decisions using labeled data. It involves preparing data, selecting algorithms, training models iteratively, and validating their performance for real-world use."

CODE

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
```

```
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]
})
X = disengagement_data[['algorithm', 'reason']]
y = disengagement_data['disengagement']
X = pd.get_dummies(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test,
y_pred)
print("Accuracy:", accuracy)
print("Classification Report:\n",
classification_report(y_test, y_pred))
print("Number of test samples:",
len(y_test))
print("Number of predicted samples:",
len(y_pred))
```

OUTPUT

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1

    accuracy                           1.00         1
   macro avg       1.00      1.00      1.00         1
weighted avg       1.00      1.00      1.00         1

Number of test samples: 1
Number of predicted samples: 1
```

MODEL EVALUATION

"Model evaluation involves assessing the performance of trained autonomous driving models using metrics like accuracy and validation techniques. It ensures the readiness and reliability of models for real-world deployment."

CODE

```
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
```

```
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no disengagement
})
y_pred = model.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
cls_report = classification_report(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(cls_report)
```

OUTPUT

```
Confusion Matrix:
[[1]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1

    accuracy                           1.00         1
   macro avg       1.00      1.00      1.00         1
weighted avg       1.00      1.00      1.00         1
```

EVALUATION METRICS

ACCURACY METRICS

"Accuracy metrics gauge the correctness of predictions made by autonomous driving models, focusing on true positives, true negatives, false positives, and false negatives. They provide insights into model performance using measures like accuracy, precision, recall, and F1-score."

CODE

```
import pandas as pd
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]
})
predicted_disengagements = [1, 0, 0, 1, 0]
```

```
total_reports =
len(disengagement_data)
correct_predictions = sum(1 for actual,
predicted in
zip(disengagement_data['disengagemen
t'], predicted_disengagements) if actual
== predicted)
accuracy = correct_predictions /
total_reports
print("Total Reports:", total_reports)
print("Correct Predictions:",
correct_predictions)
print("Accuracy:", accuracy)
```

OUTPUT

```
Total Reports: 5
Correct Predictions: 4
Accuracy: 0.8
```

RANKING METRICS

"Ranking metrics assess the relative importance or severity of disengagement events or driving scenarios in autonomous driving systems. They prioritize issues based on severity, impact, or frequency, guiding developers in addressing critical challenges."

CODE

```
import pandas as pd
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]
})
ranked_algorithms = [['A', 'B'], ['B', 'A'], ['A', 'B'], ['B', 'A'], ['A', 'B']]
 (MRR)
```

```
total_mrr = 0
for actual, ranked in
zip(disengagement_data['algorithm'],
ranked_algorithms):
    try:
        mrr = 1 / (ranked.index(actual) +
1)
        total_mrr += mrr
    except ValueError:
        pass
mrr = total_mrr /
len(disengagement_data)
print("Mean Reciprocal Rank (MRR):",
mrr)
```

OUTPUT

```
Mean Reciprocal Rank (MRR): 1.0
```

DIVERSITY METRICS

"Diversity metrics evaluate the range and representativeness of driving scenarios encountered by autonomous vehicles. They assess scenario diversity, environmental variation, and challenges coverage to ensure systems are robust across different conditions."

CODE

```
import pandas as pd
disengagement_data = pd.DataFrame({
```

```
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]
})
ranked_algorithms = [['A', 'B'], ['B', 'A'], ['A', 'B'], ['B', 'A'], ['A', 'B']]
k = 2
diversity_sum = 0
for ranked in ranked_algorithms:
    unique_algorithms = set(ranked[:k])
```

```
    diversity = len(unique_algorithms) / k
    diversity_sum += diversity
diversity_at_k = diversity_sum / len(ranked_algorithms)
print("Diversity@{}: {:.2f}".format(k, diversity_at_k))
```

OUTPUT

```
Diversity@2: 1.00
```

NOVELTY METRICS

"Novelty metrics gauge an autonomous driving system's ability to handle new or unexpected scenarios. They assess how well the system adapts and responds to unfamiliar situations, ensuring its readiness for real-world deployment."

CODE

```
import pandas as pd
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]
})
```

```
ranked_algorithms = [['A', 'B'], ['B',
'A'], ['A', 'B'], ['B', 'A'], ['A', 'B']]
reference_set = {'C', 'D', 'E', 'F'}
k = 2
novelty_sum = 0
for ranked in ranked_algorithms:
    unique_algorithms = set(ranked[:k])
    novel_algorithms =
unique_algorithms - reference_set
    novelty = len(novel_algorithms) / k
    novelty_sum += novelty
novelty_at_k = novelty_sum /
len(ranked_algorithms)
print("Novelty@{}: {:.2f}".format(k,
novelty_at_k))
```

OUTPUT

```
Novelty@2: 1.00
```

PRECISION

Precision measures the proportion of true positive predictions out of all positive predictions made by the model. It indicates the accuracy of positive predictions. Precision is particularly useful in scenarios where minimizing false positives is crucial, such as medical diagnoses or anomaly detection.

CODE

```
import pandas as pd

from sklearn.metrics import precision_score
```

```
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]
})
predicted_disengagements = [1, 0, 0, 1, 0]
true_disengagements = disengagement_data['disengagement']
precision = precision_score(true_disengagements, predicted_disengagements)
```

```
print("Precision:", precision)
```

OUTPUT

```
Precision: 1.0
```

RECALL

Recall measures the proportion of true positive predictions out of all actual positive instances in the dataset. It indicates the ability of the model to correctly identify positive instances. High recall is desirable in scenarios where it's important to capture all positive instances, even if it results in some false positives.

CODE

```
import pandas as pd
```

```
from sklearn.metrics import recall_score
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]
})
predicted_disengagements = [1, 0, 0, 1, 0]
true_disengagements = disengagement_data['disengagement']
```

recall =
recall_score(true_disengagements,
predicted_disengagements)

print("Recall:", recall)

OUTPUT

```
Recall: 0.6666666666666666
```

F1 SCORE

F1-score is the harmonic mean of precision and recall. It provides a balanced measure of a model's performance, especially when dealing with imbalanced datasets. F1-score is useful when there's a trade-off between precision and recall, and there's a need to find a balance between the two.

CODE

```
import pandas as pd
from sklearn.metrics import f1_score
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]
})
```

```
predicted_disengagements = [1, 0, 0, 1, 0]
true_disengagements = disengagement_data['disengagement']
f1 = f1_score(true_disengagements, predicted_disengagements)
print("F1 Score:", f1)
```

OUTPUT

```
F1 Score: 0.8
```

MEAN ABSOLUTE ERROR

MAE measures the average absolute difference between predicted values and actual values. It indicates the average magnitude of errors in the predictions. MAE is particularly useful when the absolute error is important and outliers should not be overly penalized.

CODE

```
import pandas as pd
from sklearn.metrics import
mean_absolute_error
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-
05', '2019-01-10', '2019-02-01', '2019-
02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure',
'Software failure', 'Unexpected
behavior', 'Hardware failure', 'Software
failure'],
    'disengagement': [1, 0, 1, 1, 0]
```

```
})
predicted_values = [0.8, 0.2, 0.6, 0.9, 0.3]
true_values = disengagement_data['disengagement']
mae = mean_absolute_error(true_values, predicted_values)
print("Mean Absolute Error (MAE):", mae)
```

OUTPUT

```
Mean Absolute Error (MAE): 0.24
```

ROOT MEAN SQUARED ERROR

RMSE is the square root of the average of the squared differences between predicted values and actual values. It provides a measure of the standard deviation of prediction errors. RMSE is useful when larger errors should be penalized more than smaller ones.

CODE

```
import pandas as pd
from sklearn.metrics import mean_squared_error
import numpy as np
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected
```

```
behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]
})
predicted_values = [0.8, 0.2, 0.6, 0.9, 0.3]
true_values = disengagement_data['disengagement']
mse = mean_squared_error(true_values, predicted_values)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

OUTPUT

```
Root Mean Squared Error (RMSE): 0.2607680962081059
```

SOME ADDITIONAL METRICS

ROC CURVE AND AUC

Receiver Operating Characteristic (ROC) curve is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. Area Under the ROC Curve (AUC) provides a single scalar value that summarizes the ROC curve. Higher AUC values indicate better classifier performance.

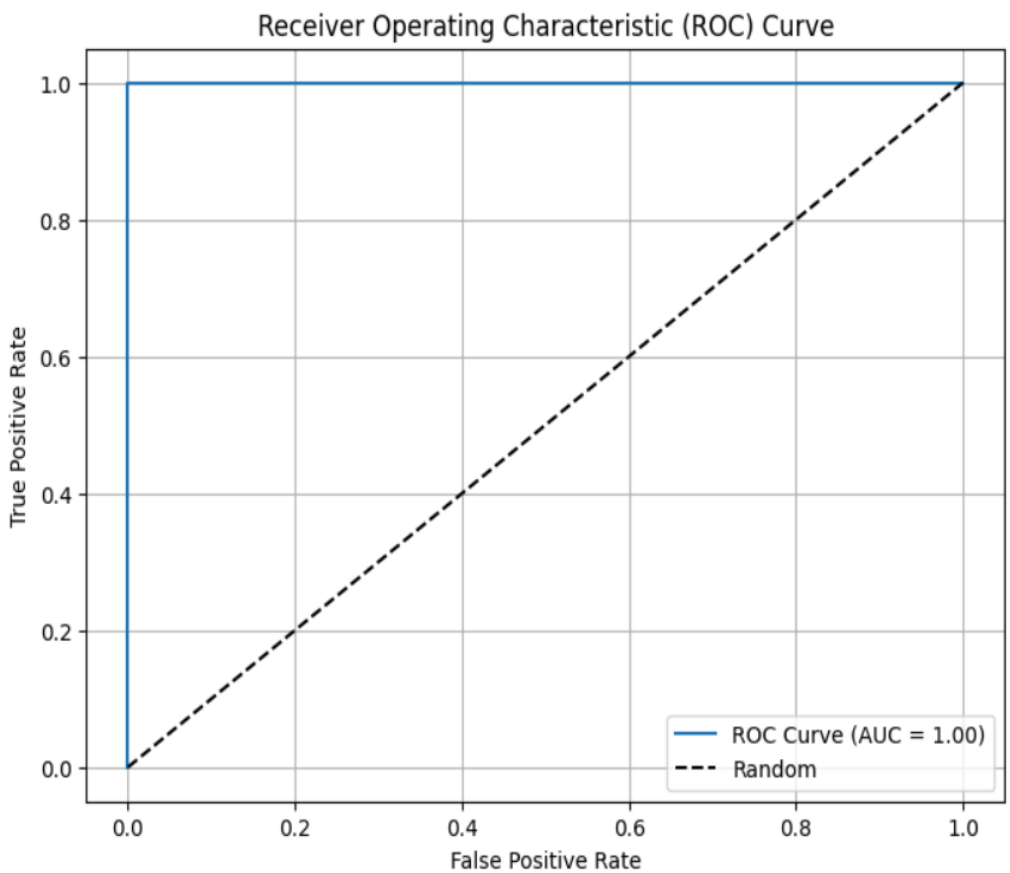
CODE

```
import pandas as pd
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt
```

```
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]
})
predicted_probabilities = [0.8, 0.2, 0.6, 0.9, 0.3]
true_disengagements = disengagement_data['disengagement']
fpr, tpr, thresholds = roc_curve(true_disengagements, predicted_probabilities)
```

```
auc =
roc_auc_score(true_disengagements,
predicted_probabilities)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve
(AUC = {auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--',
label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating
Characteristic (ROC) Curve')
plt.legend()
plt.grid(True)
plt.show()
```

OUTPUT



CONFUSION MATRIX

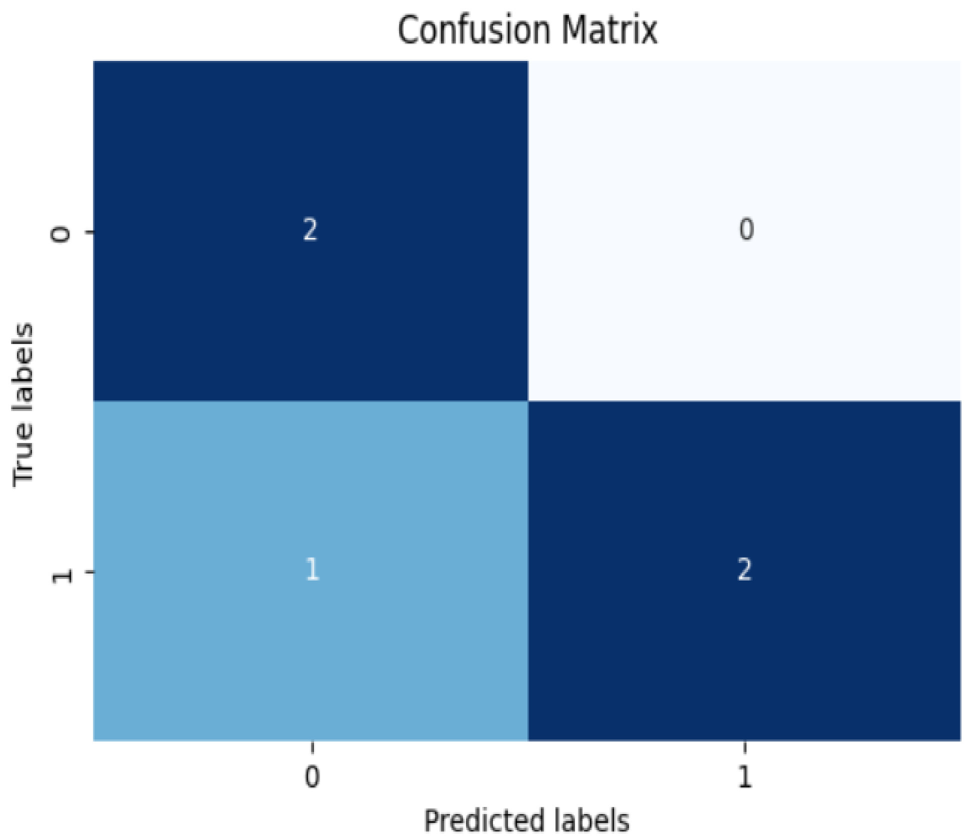
A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. It allows visualization of the performance of an algorithm.

CODE

```
import pandas as pd
from sklearn.metrics import confusion_matrix
import seaborn as sns
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
```

```
    'disengagement': [1, 0, 1, 1, 0]
})
predicted_labels = [1, 0, 0, 1, 0]
true_disengagements =
disengagement_data['disengagement']
cm =
confusion_matrix(true_disengagements,
predicted_labels)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True,
cmap='Blues', fmt='g', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

OUTPUT



F BETA SCORE

The F-beta score is the weighted harmonic mean of precision and recall, with a parameter beta determining the weight of precision in the combined score. When beta is 1, it is equivalent to the F1 score.

CODE

```
from sklearn.metrics import fbeta_score
true_labels = [1, 0, 1, 0, 1]
predicted_labels = [1, 0, 0, 1, 1]
beta = 0.5
f_beta = fbeta_score(true_labels, predicted_labels, beta=beta)
print(f"F-{beta} Score:", f_beta)
```

OUTPUT

```
F-0.5 Score: 0.6666666666666666
```

MATTHEW'S CORRELATION COEFFICIENT

MCC is used in machine learning as a measure of the quality of binary classifications. It takes into account true and false positives and negatives and is generally regarded as a balanced measure that can be used even if the classes are of very different sizes.

CODE

```
from sklearn.metrics import
matthews_corrcoef
true_labels = [1, 0, 1, 0, 1]
predicted_labels = [1, 0, 0, 1, 1]
mcc = matthews_corrcoef(true_labels,
predicted_labels)
print("Matthews Correlation Coefficient
(MCC):", mcc)
```

OUTPUT

```
Matthews Correlation Coefficient (MCC): 0.16666666666666666
```

SPECIFICITY AND SENSITIVITY

MCC is used in machine learning as a measure of the quality of binary classifications. It takes into account true and false positives and negatives and is generally regarded as a balanced measure that can be used even if the classes are of very different sizes.

CODE

```
import pandas as pd
from sklearn.metrics import
confusion_matrix
disengagement_data = pd.DataFrame({
```

```
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]
})
predicted_labels = [1, 0, 0, 1, 0]
true_disengagements = disengagement_data['disengagement']
cm = confusion_matrix(true_disengagements, predicted_labels)
tn, fp, fn, tp = cm.ravel()
specificity = tn / (tn + fp)
```

```
sensitivity = tp / (tp + fn)
print("Specificity:", specificity)
print("Sensitivity:", sensitivity)
```

OUTPUT

```
Specificity: 1.0
Sensitivity: 0.6666666666666666
```

PRECISION RECALL CURVE

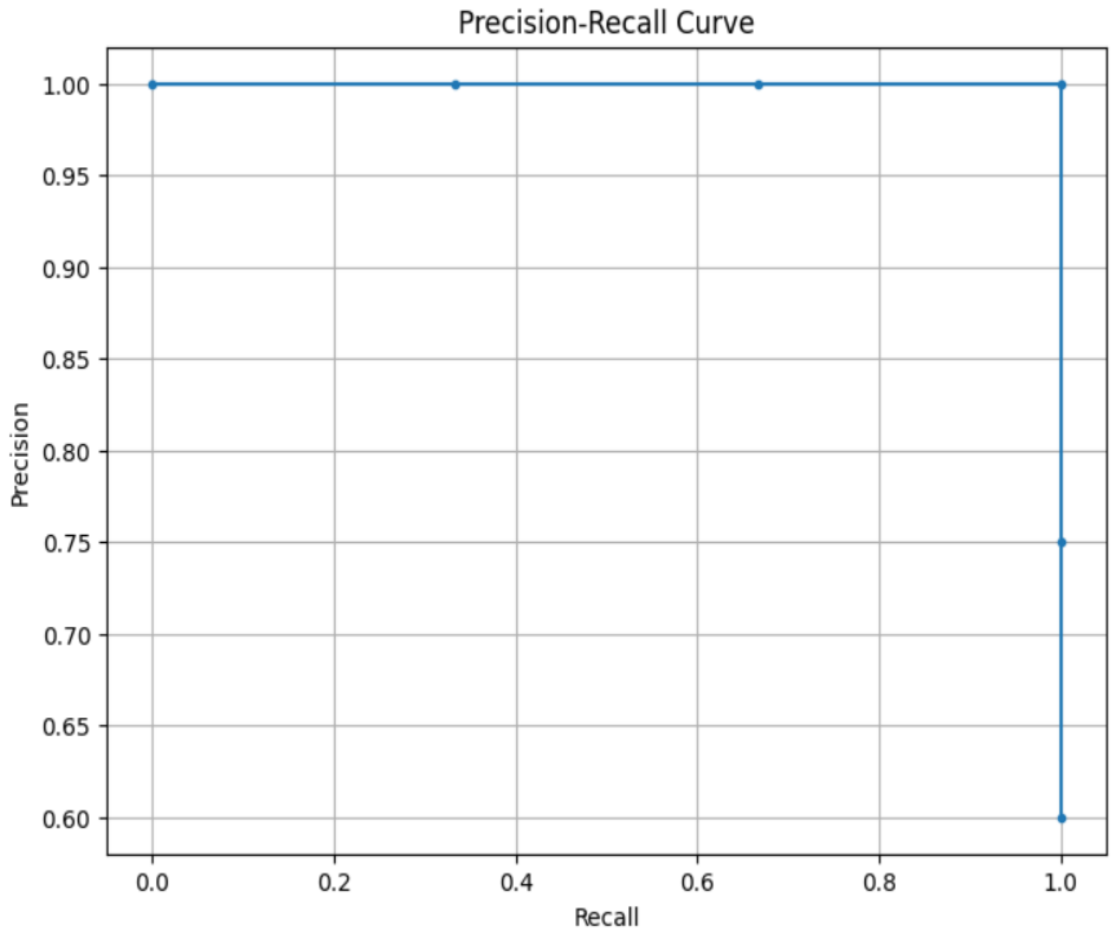
A precision-recall curve is another way to visualize the tradeoff between precision and recall for different thresholds. It is especially useful when the classes are imbalanced.

CODE

```
import pandas as pd
from sklearn.metrics import
precision_recall_curve
import matplotlib.pyplot as plt
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-
05', '2019-01-10', '2019-02-01', '2019-
02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure',
'Software failure', 'Unexpected
behavior', 'Hardware failure', 'Software
failure'],
    'disengagement': [1, 0, 1, 1, 0]
})
```

```
predicted_probabilities = [0.8, 0.2, 0.6, 0.9, 0.3]
true_disengagements = disengagement_data['disengagement']
precision, recall, _ = precision_recall_curve(true_disengagements, predicted_probabilities)
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.grid(True)
plt.show()
```

OUTPUT



MODEL SELECTION

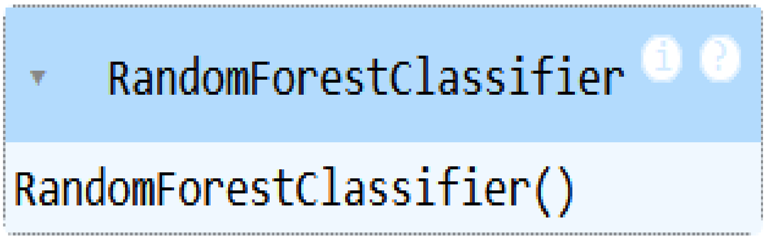
Model selection is the process of choosing the best machine learning model for autonomous driving tasks. It involves comparing models based on performance metrics, complexity, generalization ability, robustness, and scalability to ensure the chosen model meets the safety and reliability requirements of autonomous vehicles.

CODE

```
import pandas as pdfrom
sklearn.ensemble import
RandomForestClassifier
disengagement_data = pd.DataFrame({
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'disengagement': [1, 0, 1, 1, 0]
})
X =
pd.get_dummies(disengagement_data[['
algorithm']])
y =
disengagement_data['disengagement']
```

```
model = RandomForestClassifier()
model.fit(X, y)
```

OUTPUT



CONCLUSION

The 2019 autonomous disengagement reports highlight the importance of robust model development and evaluation metrics. By employing accuracy, ranking, diversity, and novelty metrics, developers enhance the safety and reliability of autonomous systems.

Effective model selection ensures optimal performance in diverse scenarios. Focusing on these metrics advances autonomous driving technology, contributing to safer and more efficient transportation.