MALIS Final Project Report
Driver Classification

Kristian Håland
Elia Faure-Rolland

Abstract

The integration of advanced sensors and machine learning techniques transforms vehicle automation and safety. This project focuses on distinguishing driving behaviors to support applications such as fleet management and safety monitoring. Data from nine scenarios across three drivers and vehicles are processed using a two-phased approach. The first phase involves feature transformation through a sliding window technique and feature reduction using Principal Component Analysis (PCA) to address high dimensionality. The second phase implements and evaluates classification models, including Logistic Regression, Random Forests, Support Vector Machines (SVM), and XGBoost. Logistic Regression and SVM deliver the best accuracy and generalization across unseen data. The findings emphasize the significance of feature engineering and evaluation strategies in improving driver classification performance, highlighting the importance of feature reduction and robust evaluation methods.

I. Introduction

In recent years, advances in vehicle technology and the incorporation of an increasing number of sensors, have enabled a big leap in automation, vehicle safety and data collection capabilities. The data collected by the sensors provides valuable insights, and Machine Learning techniques can be used to analyse environmental factors such as road types, speed bumps, and curves.

In our project, we aim to focus on the behaviour of different drivers on the same roads, as we believe it significantly influences the data collected. This study is valuable for understanding individual driving styles and has numerous applications in autonomous driving, such as predicting rapid acceleration, sharp turns, and hard braking.

These are some of the interesting fields in which this type of analysis can be applied :

- For car rental companies, utilizing the correct use of insurance
- For companies with a fleet of cars, tracking which employee is driving the car, and making sure they drive safely
- Personalizing car setup for specific drivers

The quality of the road and the type of route can significantly affect classification performance, potentially leading to incorrect predictions.

In our work, we aim to classify drivers by considering various driving scenarios, inspired by [1]. We utilize data collected from different sensors placed on a car, as detailed in Section II.

Our approach can be divided into two main phases. In the first phase, we select a subset of features and apply feature transformation using a sliding window technique to extract statistical data from temporal sensor measurements. Additionally, we employ feature reduction techniques to address the high dimensionality of our data.

In the second phase, we implement several classification models and examine the differences between tree-based approaches and other strategies. We compare accuracy values based on varying feature counts, with a primary focus on enhancing the generalization ability of our models on unseen data.

II. Datasets

The datasets used can be found on Kaggle [1] and contains sensor data collected from 9 different scenarios (see Table 1). The data is obtained from tests conducted by 3 drivers, each using a different car, across 3 distinct routes (scenarios). Each scenario, denoted as PVSi, is provided as a separate dataset.

Table 1. Structuring of the datasets. Each PVS-file corresponds to one driver driving a specific route (scenario). Notice how each driver drives a different car [2].

DataSet	Vehicle	Driver	Scenario	Distance
PVS1	Volkswagen Saveiro	Driver 1	Scenario 1	13.81 km
PVS2	Volkswagen Saveiro	Driver 1	Scenario 2	11.62 km
PVS3	Volkswagen Saveiro	Driver 1	Scenario 3	10.72 km
PVS4	Fiat Bravo	Driver 2	Scenario 1	13.81 km
PVS5	Fiat Bravo	Driver 2	Scenario 2	11.63 km
PVS6	Fiat Bravo	Driver 2	Scenario 3	10.73 km
PVS7	Fiat Palio	Driver 3	Scenario 1	13.78 km
PVS8	Fiat Palio	Driver 3	Scenario 2	11.63 km
PVS9	Fiat Palio	Driver 3	Scenario 3	10.74 km

The sensor data, further referred to as the features, is collected using a range of different sensors, namely camera, GPS, accelerometer, gyroscope, magnetometer and temperature. The features are collected both at the left and right side of each car, as well as in the dashboard, depicted in Figure 1. This gives data collection of the same features three times, e.g. acceleration in y-direction.

Due to the large number of sensors mounted on each car, the original datasets contains a total of 64 features (see Appendix B). However, not all of these features are useful or reliable for classification.

On one hand, it must be noted that each driver uses a different car, meaning some features may be biased toward specific cars. For example, temperature readings can be influenced by the vehicle's insulation. On the other hand, certain features, such as GPS-data, may not be suitable for driver classification since all drivers travelled along the same routes.

Going forward, we utilize the features listed in Table 2. It seems reasonable to retain data from sensors located on the dashboard rather than those positioned on the sides of the vehicles. The resulting datasets are reduced to 7 features, referred to as the native features.

Data are sampled with a frequency of one sample every 10 ms, resulting in a total number of samples for each PVSi varying from 91555 to 144036 depending on the scenario and the driver.

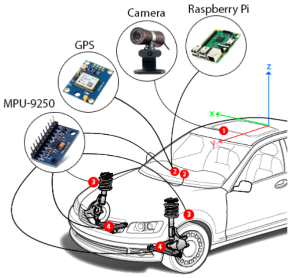

Figure 1. Sensor setup on the cars. Sensors are placed on the left and right sides, as well as in the dashboard. A camera is placed on top of the cars [2].

For the testing phase, in order to test our model on unseen data, we work on three frameworks:

- **Framework 1**: Scenario 2 and Scenario 3 for training, Scenario 1 for testing.
- **Framework 2**: Scenario 1 and Scenario 3 for training, Scenario 2 for testing.
- **Framework 3**: Scenario 1 and Scenario 2 for training, Scenario 3 for testing.

Table 2. Chosen native features to work with.

Native features
acc_x_dashboard_l
acc_y_dashboard_l
acc_z_dashboard_l
speed
gyro_x_dashboard_l
gyro_y_dashboard_l
gyro_z_dashboard_l

III. Feature transformation

Samples represent temporal data, meaning that we cannot draw any conclusive insights about who is driving based on individual time instances. Instead, a feature transformation is required to identify characteristics that describe each driver's driving style.

This involves a three-step process, in which the purpose is the extraction of statistical features from a sliding window and the subsequent reduction of the resulting correlated features.

A. Step 1: Trimming and Alignment

Drivers may not start moving exactly at the beginning of sampling data, and may also continue recording some time after stopping the car. Furthermore, they may drive at different speeds, resulting in a different duration of the recording period. Since sampling is applied on a fixed temporal frequency, these two aspects may result in a different amount of data for each driver on the same scenario, leading to biased predictions.

We address this issue by trimming the PVSi-data based on speed. Specifically, we set a threshold of 0.1 m/s and remove data in the beginning and end of a recording where the speed falls below this threshold.

Subsequently, we equalize the dimensions of the PVSi's in groups based on the shortest record for the specific scenarios by truncating all datasets to the same length.

After this process, each driver ends up with the same number of samples for each scenario.

B. Step 2: Sliding window

In this phase, a sliding window is used to embed temporal information into each row of the dataset by extracting statistical features from N consecutive temporal indices.

Thus, the mean, standard deviation, minimum, and maximum are calculated for each native feature, considering the preceding records within a window of size N.

In addition, we also compute the rate of change of the native features over time, in order to capture the suddenness or smoothness of driving behaviour; we refer to this metric as Jerk and can be interpreted as the derivative of the feature considering N samples. After each iteration the window moves one sample forward.

The end result after windowing is a total of 35 new statistical features.

C. Step 3: Features reduction

Due to the amount of data, our goal is to reduce the number of features in order to better interpret the dataset and ensure that any model can run within a reasonable time. To do so, we focus on the study of feature reduction.

A first approach was based on features correlation, by computing the resulting correlation matrix and fixing a correlation threshold of 0.8 on the absolute value of the features. This resulted in removing at most 8 features.

Ultimately, we opt to use PCA (Principal Component Analysis), which proves to be highly effective for this task. As a consequence, we scale the features and project them onto a new smaller space.

Choosing this strategy allows us to reduce the number of features by nearly half, with an explained variance of 0.95. The explained cumulative variance is shown in Figure 2.

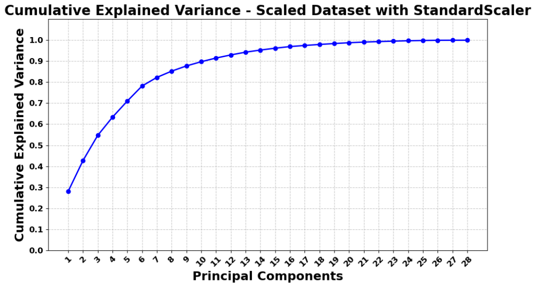

Figure 2. Cumulative explained variance after applying PCA, Framework 3.

The overall workflow for the feature design are shown in Figure 3.

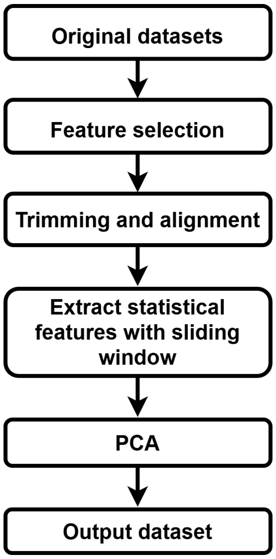

Figure 3. Workflow of the feature design.

IV. Evaluation strategy and hyperparameters tuning

As explained in Section II, we use 3 frameworks to test our models, which allows us to test the final model on unseen data. Since each testing scenario contains data collected from a different route, the characteristics of the track may be very different from the training and testing set.

In a first stage, during the evaluation of our model for the selection of the best hyperparameters, we apply a cross-validation approach in which the two datasets used for training were merged and split in 5-folds. We then train 5 different models for each configuration of hyperparameters, and test each time on a different fold.

Nevertheless, since data from the route used for evaluation is also contained inside the training split, the model tends to overfit and cannot properly generalize on unseen data coming from a different route.

To address this problem, we take 30 % of each route and apply cross validation with a LOO (Leave-one-out) approach: the split of 2 routes are used for training and the last split for evaluation. This results in three different models for each configuration of hyperparameters, each evaluated on data coming from a different scenario.

In the testing phase, subsequently, we use the full training datasets and the remaining 70 % of the third dataset for testing. This lets us improve the generalization behaviour of the model on different routes, while still being able to test on unseen data.

In the next section we focus on the description of the results obtained after tuning different models using the described strategy.

V. Experimental evaluation

For our specific problem, we experiment with four different models and analyze their respective results. Initially, we work with a Logistic Regression model and a Random Forest classifier.

Subsequently, we shift our focus to more advanced and high-performing models, specifically Support Vector Machines (SVM) and Boosting methods (XGBoost).

In the following section, we present the results obtained for each framework, considering various thresholds of explained variance in the feature reduction step. All results are based on a sliding window size of 3000, which is determined to be the optimal value after evaluation.

A. Logistic Regression

As our first model, we select Logistic Regression due to its efficiency and computational speed. Before applying PCA, we scale the data.

Initially, the model does not generalize well, prompting us to adjust the regularization strength parameter C to a value as low as 0.001. (Note that C is the inverse of the regularization parameter λ, meaning a lower value of C corresponds to stronger regularization). Additionally, we apply polynomial feature transformation up to degree 2 and set the maximum number of iterations to 100 to ensure convergence.

Among the models we evaluated, Logistic Regression emerged as the best classifier, results are shown in Table 3.

Table 3. Test accuracies using Linear Regression.

	Scenario		
	1	**2**	**3**
0.7	97.6 %	35.5 %	89.7 %
0.8	97.3 %	47.4 %	94.2 %
0.9	97.9 %	41.9 %	96.4 %

To mitigate overfitting and improve the model's generalization across different scenarios, we experiment with reducing the number of features using PCA.

Specifically, we evaluate the model's performance with lower explained variance values.

As illustrated in Figure 4, the accuracy values for framework 2 with explained variance of 0.5 and 0.6 are significantly better compared to those for higher values. Conversely, for frameworks 1 and 3, the accuracy shows slight improvement as the number of features increases.

These observations lead us to conclude that selecting only a subset of the principal components not only speeds up the training process but also produces a more stable model that generalizes better across different scenarios.

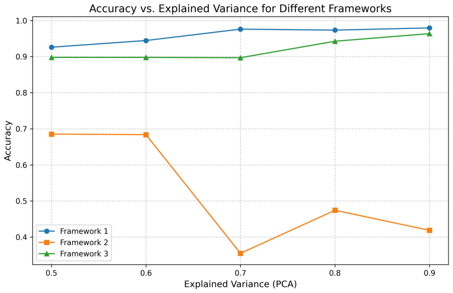

Figure 4. Values of accuracy for different thresholds of explained variance, each line refers to a different framework.

B. Random Forest

Random Forest models are a subset of the bagging strategy, where multiple decision trees are trained on bootstrapped datasets derived from the original data.

After completing the tuning process described in Section IV, we test a model configured with 350 estimators and a maximum of 10 samples per leaf node. The number of features considered for each tree is set to $\sqrt{\text{total\_number\_of\_features}}$, and no maximum tree depth is specified.

The optimal model is tested on all the frameworks, considering an explained variance of 0.7, 0.8 and 0.9. The results are shown in Table 4.

Table 4. Test accuracies using Random Forrest.

	Scenario		
	1	**2**	**3**
0.7	66.8 %	36.4 %	73.2 %
0.8	65.2 %	36.4 %	75.2 %
0.9	73.2 %	36.4 %	75.4 %

As we can observe, the number of features that are used for training does not significantly affect the final accuracy, since each estimator only selects a small subset among all the features. Furthermore, the model is not able to work properly in framework 2, and gives back almost random classification.

C. Support Vector Machine

A Support Vector Machine (SVM) is implemented in order to investigate the use of more complex machine learning models. Specifically, SVM tries to find the optimal decision boundary between classes using support vectors, which are defined by points closest to the decision boundary from each class. The hyperparameters used can be found in Appendix E.

The resulting test accuracies for given PCA thresholds and scenarios used for testing are shown in Table 5.

Table 5. Test accuracies using SVM.

	Scenario		
	1	**2**	**3**
0.7	81.9 %	39.1 %	82.4 %
0.8	85.8 %	46.9 %	84.5 %
0.9	88.0 %	60.1 %	88.6 %

Compared to results for Random Forest in Section V-B, SVM has better test accuracies for all scenarios and PCA thresholds. Again, the model performs suboptimally for framework 2.

For all cases, the model favours a regularization strength parameter C=1 and a linear kernel function, indicating that there exists some linear relationships in our data.

Furthermore, a PCA threshold of 0.9 is favoured in all scenarios, indicating a higher level of explained variance is needed, in this case, to capture the underlying structure of the data.

D. XGBoost

The final model implemented is XGBoost, which combines several weak learners in order to make one strong learner [3]. Decision trees are used as models in the XGBoost algorithm, and used hyperparameters can be found in Appendix F.

The resulting test accuracies for given PCA thresholds and scenarios used for testing are shown in Table 6.

Table 6. Test accuracies using XGBoost.

	Scenario		
	1	**2**	**3**
0.7	58.8 %	41.4 %	74.2 %
0.8	58.8 %	41.3 %	74.3 %
0.9	58.8 %	41.2 %	74.3 %

Evident here is that implemented PCA thresholds have negligible effects for each scenario. Again, the models have the worst performances using framework 2, and also show worse performances than the other implemented models.

This could indicate that classifying the data benefits from simpler models, or that tested hyperparameters are insufficient for the classification task.

VI. Conclusion

In this project, we investigated and implemented machine learning models to classify which driver is operating a car. The preliminary work focused on feature design, where statistical features were extracted from collected sensor data for driver classification.

Significant emphasis was placed on the evaluation process, where various strategies were employed to properly assess model performance. To enhance the classification speed and explore the importance of feature reduction, we applied Principal Component Analysis (PCA).

The results demonstrate that, even with a relatively low value of explained variance, our models perform quite well. In certain cases, a significant reduction in features proved beneficial, enabling the models to better generalize to unseen data.

The best results were achieved using Logistic Regression and Support Vector Machines. While the Decision Trees approaches were less efficient, their performance remained robust, requiring minimal hyperparameter tuning.

VII. Member's contribution

For this part of the project, we worked together on the implementation of the models and on the evaluation technique to adopt. We split the work on different models and then looked and collected the results together. The report was written together. Our code can be found in Appendix (A). We used ChatGPT for the code implementation, but not for determining how to work with our dataset and the logic and reasoning behind each part.

References

[1] J. Menegazzo and A. von Wangenheim, "Multi-Contextual and Multi-Aspect Analysis for Road Surface Type Classification Through Inertial Sensors and Deep Learning," 2020 X Brazilian Symposium on Computing Systems Engineering (SBESC), Florianopolis, 2020, pp. 1-8, doi: 10.1109/SBESC51047.2020.9277846.

[2] Menegazzo, J. (2020) *PVS - Passive Vehicular Sensors Datasets*. Available from: Kaggle (Fetched: November 14, 2024)

[3] XGBoost Contributors. (2022) *XGBoost Documentation*. Available from: https://xgboost.readthedocs.io/en/stable/. (Fetched: January 20, 2025)

Appendix

A. Github

https://github.com/MALIS_PROJECT

B. Native features

Table 7. Native features. All PVSi have the same native features.

Features Left Sensors			
timestamp	acc_x_dashboard_l	acc_y_dashboard_l	acc_z_dashboard_l
acc_x_above_suspension_l	acc_y_above_suspension_l	acc_z_above_suspension_l	acc_x_below_suspension_l
acc_y_below_suspension_l	gyro_x_dashboard_l	gyro_y_dashboard_l	gyro_z_dashboard_l
gyro_x_above_suspension_l	gyro_y_above_suspension_l	gyro_z_above_suspension_l	gyro_x_below_suspension_l
gyro_y_below_suspension_l	acc_z_below_suspension_l	gyro_z_below_suspension_l	mag_x_dashboard_l
mag_y_dashboard_l	mag_z_dashboard_l	mag_x_above_suspension_l	mag_y_above_suspension_l
mag_z_above_suspension_l	temp_dashboard_l	temp_above_suspension_l	temp_below_suspension_l
timestamp_gps	latitude	longitude	speed
Features Right Sensors			
timestamp	acc_x_dashboard_r	acc_y_dashboard_r	acc_z_dashboard_r
acc_x_above_suspension_r	acc_y_above_suspension_r	acc_z_above_suspension_r	acc_x_below_suspension_r
acc_y_below_suspension_r	acc_z_below_suspension_r	gyro_x_dashboard_r	gyro_y_dashboard_r
gyro_z_dashboard_r	gyro_x_above_suspension_r	gyro_y_above_suspension_r	gyro_z_above_suspension_r
gyro_x_below_suspension_r	gyro_y_below_suspension_r	gyro_z_below_suspension_r	mag_x_dashboard_r
mag_y_dashboard_r	mag_z_dashboard_r	mag_x_above_suspension_r	mag_y_above_suspension_r
mag_z_above_suspension_r	temp_dashboard_r	temp_above_suspension_r	temp_below_suspension_r
timestamp_gps	latitude	longitude	speed

C. Hyperparameters used in Linear Regression

Hyperparameter	**Values**
Polynomial degree	[1, 2]
Regularization Strength	[0.001, 0.01, 0.1]
Maximum Iterations	[100, 250, 500]

D. Hyperparameters used in Random Forrest

Hyperparameter	**Values**
Number of Trees in the Forest	[100, 350, 500]
Maximum Depth	[None, 30]
Minimum Samples to Split a Node	[2, 10, 20]
Bootstrap Sampling	[True, False]
Number of features per estimator	[sqrt]

E. Hyperparameters used in SVM

Hyperparameter	**Values**
Regularization Strength	[0.1, 1, 10]
Kernel function	[Radial Basis, Linear]
Scaling factor Radial Basis kernel	[0.1]

F. Hyperparameters used in XGBoost

Hyperparameter	**Values**
#Boosting rounds	[100, 200, 600]
Maximum depth	[3, 6, 9]
Learning rate	[0.01, 0.1, 0.2]
Fraction of training data per estimator	[0.8, 1.0]
Number of features per estimator	[0.8, 1.0]