Sensors measurement analysis for Transport Mode Detection

Alessandro Giacchè [953995] Paola Persico [929044]

May 8, 2021

1 Introduction

Transport Mode Detection is a type of Human Activity Recognition (HAR), which plays an important role in several Context-Aware Systems. One possible technique to solve this task is to exploit a GPS sensor, but this has two major disadvantages: energy consumption and privacy [BDFB16]. Another technique entails exploiting other sensors available on smartphones, such as accelerometer and gyroscope, instead of the GPS. To accomplish this multi-label classification task, several models can be trained on a dataset which contains a great number of observations: from a Data Analytics perspective, the basic statistical features of acquired sensors' values are the variables, while the transport mode is the target.

Concretely, this project follows this latter path and aims to analyze the provided Transportation Mode Detection Dataset (TSD) in order to find a reasonable solution to the transport mode detection problem, by training and comparing models on the basis of several evaluation metrics such as accuracy and fitting time.

2 Methodology

The project follows the standard Data Analytics pipeline, each step is explained in the related subsection. The visualization is used in various steps of the pipeline for different purposes as explained below.

2.1 Data Acquisition

The used dataset, statically acquired from a CSV file, contains 5,893 observations and each one of them is described by 64 statistical values which are obtained computing on a 5 seconds window the maximum, minimum, average and standard deviation of the measurement of the following sensors: accelerometer, game rotation vector, gravity, gyroscope (calibrated and uncalibrated), light, linear acceleration, magnetic field (calibrated and uncalibrated), orientation, pressure, proximity, rotation vector, step counter and sound.
Each observation has a "target" field that can be one of the following transport modes: "Still", "Car", "Train", "Bus", "Walking".

2.2 Priori visualization

In order to get a grasp of the dataset's attributes, several characteristics are plotted:

- the class distribution, which reveals that the dataset is balanced (Figure 1)

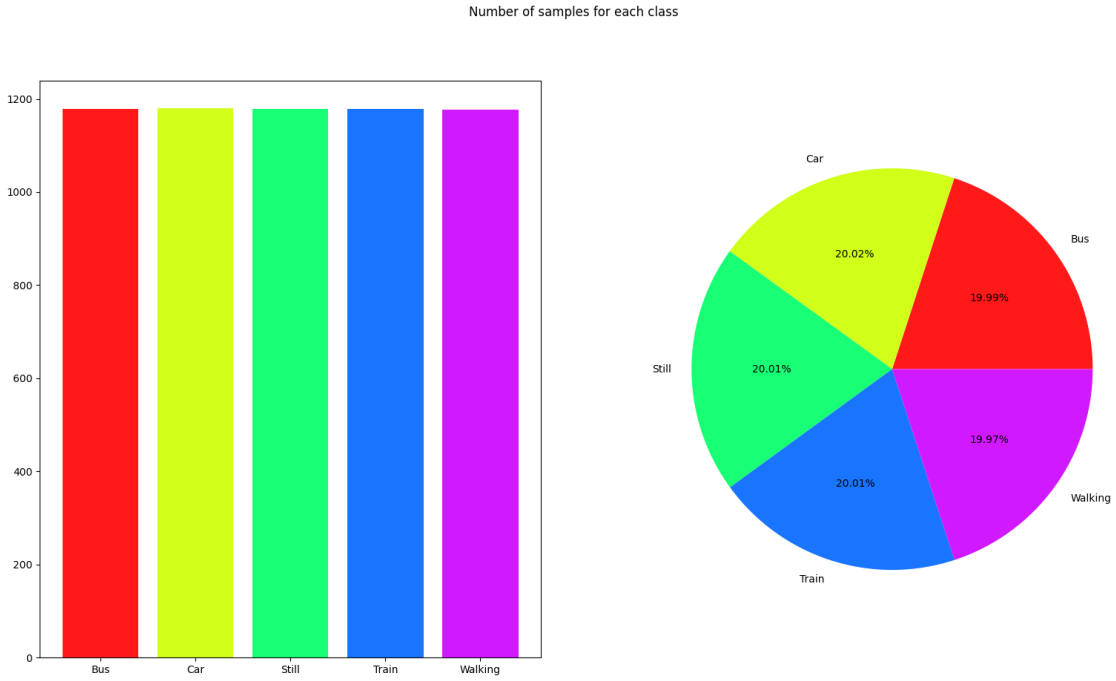


Figure 1: Class distribution

- the features' values distribution, which reveals that the majority of the features follows a skew normal distribution (Figure 2)

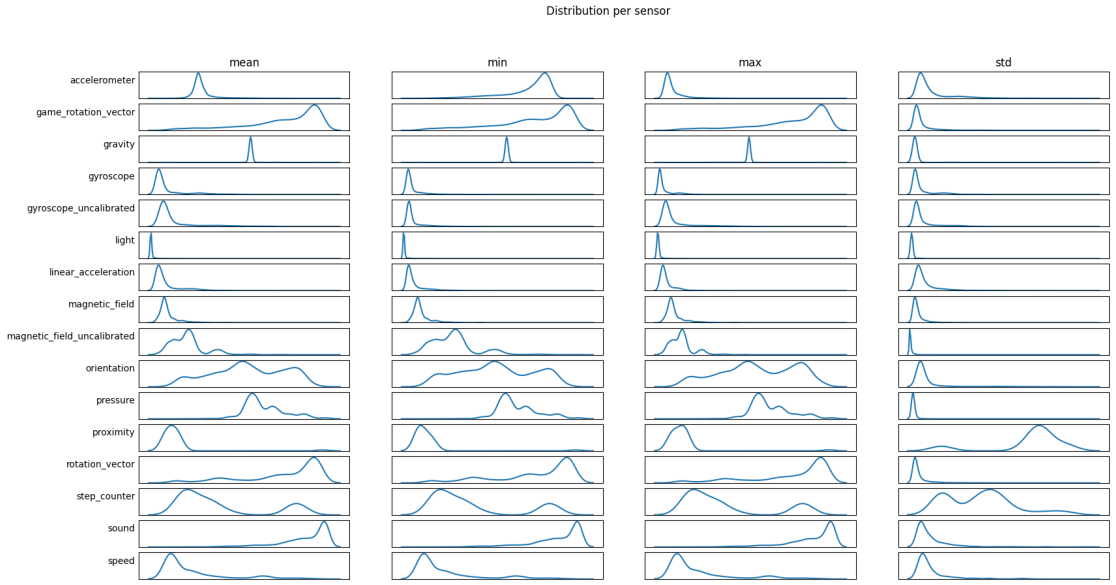


Figure 2: Features values distribution

- the percentage of missing values for each feature (with the average ratio of missing values), which reveals that the majority of features has many missing values (Figure 3)

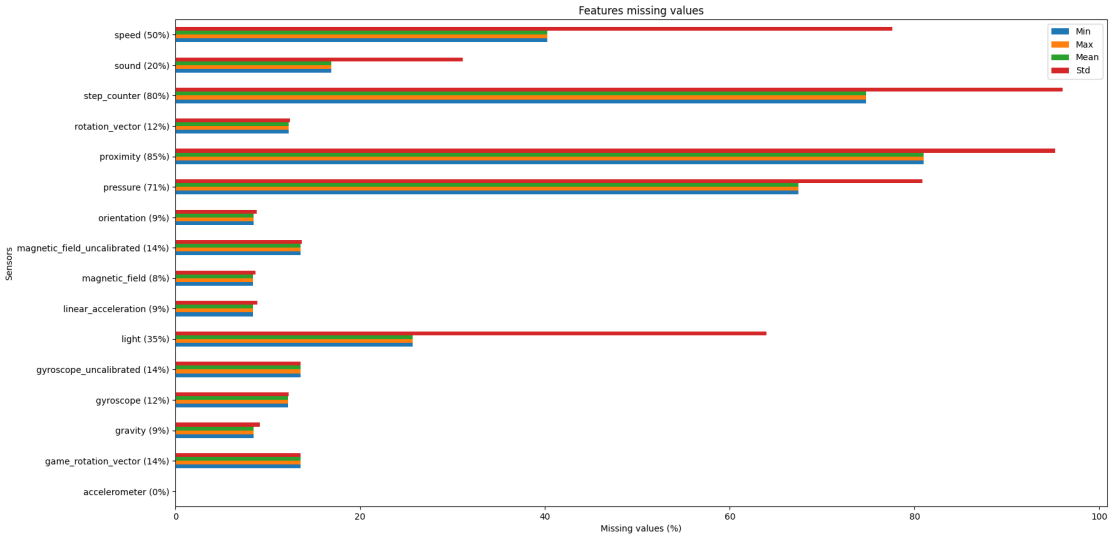


Figure 3: Features missing values

2.3 Data Pre–processing

Since the dataset contains variables which are not related to the sensors' measurements, they are discarded. Moreover, inspired by the strategy proposed by paper [CLB+18], which takes into account the sensors' noise, in addition to the main dataset (D_0), 3 sub–datasets are built with the following procedures:

1. D_1: keeping only features with less than 30% of missing values (46 features)
2. D_2: removing light, gravity, magnetic field, pressure and proximity features (keeping 40 features)
3. D_3: keeping low-battery sensors features only: gyroscope (calibrated and uncalibrated), accelerometer and sound features (16 features)

The other pre–processing operations entail working on the training sets, so a splitting technique is applied. Table 1 shows the splitting ratios adopted.

Train Set Size	Validation Set Size	Test Set Size
72%	8%	20%

Table 1: Datasets splitting ratios

The strategy adopted to remove the missing values in the datasets is to replace them with the median of the variable's values in the training sets (the median is used because it is not greatly affected by outliers). Moreover, since the variables' ranges differ greatly, two different features normalization techniques are proposed: Min–Max Scaling and Standardization, computing the statistics on the training sets and for traditional machine learning algorithms only: for the neural network, only the Min–Max Scaling is applied.

2.4 Modeling

In order to train classic Machine Learning models, a 10–fold cross–validation technique is used to tune hyperparameters instead of a hold–out validation, generating a more reliable accuracy score. On the other hand, for the Multi–Layer Neural Network, a hold–out validation is used to avoid computational overhead.

As required by the assignment, the proposed models are: SVM (with linear, polynomial and radial kernel), Gaussian Naive Bayes, Random Forest and Feedforward Neural Network. Since some features are correlated (eg. the statistics of each sensor), the QDA model has been added to the case of study.

For each model, a set of hyperparameters' values is tried:

- SVM
  - *kernel*: linear, polynomial and radial
  - C: for linear and polynomial kernels:

    $$C = \{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\} \cup \{31.62, 316.23\}$$

    For radial kernel:

    $$C = \{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2\} \cup \{3.17, 31.62\}$$

    In general, hyperparameter's values to test are picked from a log space of 5 elements and successively refining: the first time the first element is 10^{-1} and the last one is 10^3, in the second cycle the elements are taken from 1 to 10^2 for linear and polynomial kernels, and 10 to 10^3 for the radial one. The choice of the values in the second cycle are based on the results reported on the first one: while the first two kernels have a better behaviour when C is higher, the radial kernel is more accurate with a lower C.
  - *gamma* for radial kernel: $\gamma = \{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2\}$
  - *degree* for polynomial kernel: $d = \{2, 3, 4\}$
- Random Forest:
  - *number of trees*: $\{10, 20, 50, 100, 200, 300\}$
- Neural Network
  - *hidden size*: $\{16, 32, 50, 64\}$
  - *epochs*: $\{100, 250, 400, 500\}$
  - *minibatch size*: $\{2^5, 2^6, 2^7, 2^8\}$
  - *gamma* for learning rate decay: $\{0.01, 0.03, 0.05, 0.08\}$
  - *alpha*, the learning rate, is fixed to 0.1

The chosen architecture for the Feedforward Neural Network has the following characteristics:

- Three *hidden layers* and one *output layer*
- Each hidden layer has the same number of *hidden units*
- Each hidden layer has the same *activation function*: ReLU. The chosen activation function of the output layer is SoftMax
- At each hidden layer a *batch normalization* is applied, this improves the performance increasing the accuracy and slightly reducing overfitting
- The selected *loss function* is Multi–class Cross Entropy

- The chosen *optimizer* is Stochastic Gradient Descent with a set of possible minibatch sizes as previously explained
- A *Learning Rate Decay strategy* is applied based on number of epochs with the following formula:

$$\alpha_e = \frac{\alpha_{e-1}}{1 + \gamma e}$$

where e is the current epoch

2.5 Performance Analysis

In the case of study, since the dataset is balanced, the validation accuracy value is the chosen score with which the ranking of models and hyperparameters combinations is performed. For each model, the final measured values include the fitting time, the train, validation and test accuracy.

In addition to this, for every model except Neural Network, the ROC curve and the AUC values are also displayed as well as the confusion matrix. To monitor the training of the neural network models, the loss curve is plotted too.

Last but not least, the features' importances are plotted for each dataset following the Random Forests results.

3 Implementation

The implementation makes use of several libraries related to data analytics, such as Numpy, Scikit-learn, Pytorch and Joblib and plotting libraries such as Matplotlib and Seaborn (this last one used for density plot only).

In order to fully exploit the GPU in the neural network training, CUDA platform is used alongside Pytorch.

The project structure is composed of three folders: one of them contains the dataset only, the "saved_models" folder contains the serialized trained models and their metadata (represented by csv files), the "pytorch" folder contains all the files related to the Neural Network and its implementation.

In the root directory there are several files which are explained below (all files have a "py" extension that is omitted for readability reasons):

- "main": the pipeline coordinator, which manages the other modules and performs the train test splitting operation
- "data_layer": the module in charge of dataset acquisition and of the removal of the first four columns and the last one
- "visualization": a utils module in charge of displaying several information, it shows various plots including the ROC curve which is computed exploiting the `roc_curve` function which requires one hot encoded labels and predictions. This operation is carried out for each model. In those cases where a DataFrame has to be plotted, the pandas `plot()` function is used
- "preprocessing": the module in charge of dataset preprocessing: removal of missing values, creation of the four sub–datasets (feature selection) and "a priori" visualization
- "models_config": contains all the used machine learning algorithms with their sets of hyperparameters. To reduce the computational time, the SVM Classifiers are divided by kernel in order to avoid the cross validation onto parameters that are unused by the function itself (eg. the linear SVM does not use gamma neither degree)

- "models_runner": the kernel of the cross validation strategy, it computes the best hyperparameters for each model and saves all retrieved models with associated metadata (hyperparameters, train and validation accuracies and fitting time), or loads them if they are already saved in the "saved_models" folder

  The cross validation is delegated to the `GridSearchCV` class which takes as input a pipeline (composed of a scaler step and a classifier). To improve performance, the `n_jobs` parameter is set to the number of cores of the host machine, which allows the grid search to be run in a pseudo multi–thread scenario

- "evaluation": the module in charge of coordinating the results display for each sub–dataset and the retrieval and visualization of the tests score for each model

In the "pytorch" folder, "dataset" and "model" contain the classes used in the Pytorch context. The `TMDDataset` class stores the variables grid and the labels vector converting them to `Tensors` and – as child of `Dataset` Pytorch class – allows the iteration through them.
The `Feedforward` class defines the Multilayer Perceptron architecture: the number of inputs and hidden units are variables as well as the number of classes. The output layer's width is equal to the number of classes. The output units' activation function is not explicitly defined because Pytorch framework implicitly uses the SoftMax function when the Multi–class Cross Entropy loss function is adopted.

In the "nn_main" file, the hyperparameters tuning is not performed using `GridSearchCV` because of compatibility. `itertools` is used instead: passing the hyperparameters, the function returns the Cartesian product of the parameters on which training is performed. The dataset is not directly split in train–val–test but through a two-phase index splitting, this behaviour is needed because the batches are provided by a `DataLoader` composed by a `Subset` of indexes. In order to set up the learning rate decay, an object of class `LambdaLR` is initialized passing a lambda function that takes as input the epoch and uses the γ hyperparameter.

While the `GridSearchCV` automatically keeps track of the fitting time, in the Neural Network context it has to be manually tracked using the `time.time()` function.

The validation function also checks if the best model has already been saved. If this happens, the saved hyperparameters are loaded as well as the metadata; furthermore, if it is not required to train the model again, the model state is loaded using the Pytorch function `load`, that configures the internal state of the model from a saved serialized state. However, the model training can be forced in order to plot the loss progression.

The training and testing loops are performed by the "model_runner" module. As far as the training loop is concerned, feedforward and backpropagation steps are performed using each minibatch since the chosen optimizer is SGD. Moreover, the loss value is saved at the end of each epoch (in order to be plotted afterwards), at this moment the learning rate is also decreased. Regarding the testing loop, in order to avoid memory consumption and improve performance, the function `torch.no_grad()` is called to avoid the gradient computation. In order to recognize the output class, the index of the maximum value of each prediction row is taken, this index represents the highest probability class according to the model.

4 Results

In this section the main results of the training and testing phases of each model are presented.

4.1 Timing

The fitting time for the best probabilistic models (QDA and Gaussian Naive Bayes) has turned out to be inferior to 0.05 seconds, while SVM and Random Forest have a longer fitting time, between 1 and 5 seconds, which is nevertheless shorter that the Neural Network training, which requires 12 seconds for the D_3 dataset going up to 58 seconds for the D_0.

As far as the scoring time is concerned, the Neural Network generally requires less than 0.61 seconds, except for D_0 dataset which requires 3.49 seconds. Random Forest requires a time between 0.72 and 1.07 seconds, SVM about 0.05 seconds, while QDA and Gaussian NB are the fastest models with a scoring time inferior to 0.01 seconds.

4.2 Loss function

Regarding the Neural Network model, the loss advancement for each sub–dataset is shown by Figure 4.

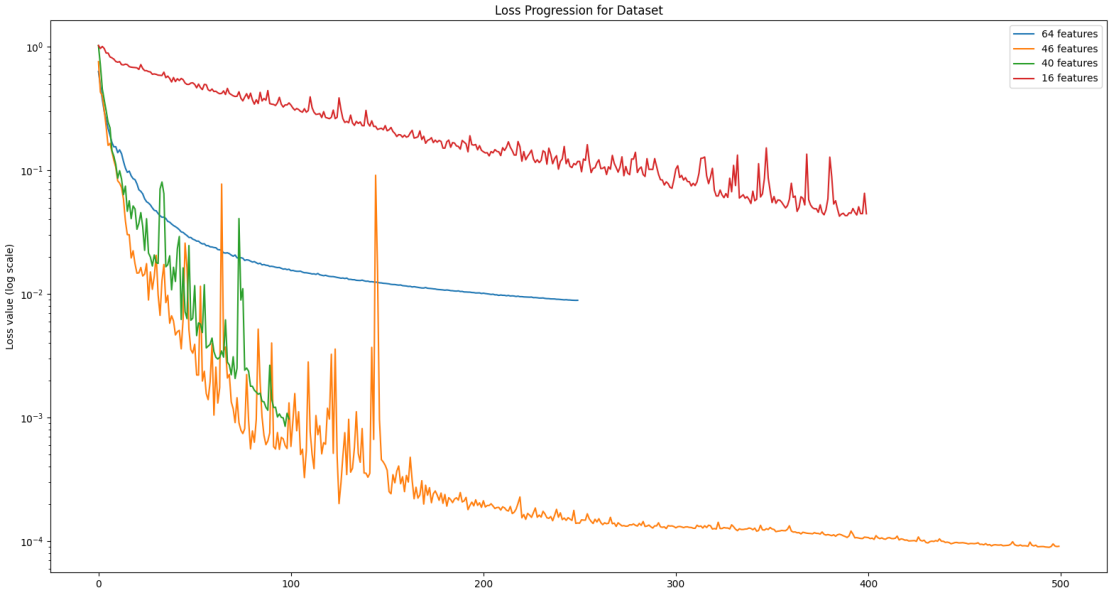


Figure 4: Loss at each epoch

The dataset with the fewest features results in a noisier loss value progression and requires more epochs in order to stabilize. In particular, the loss on the D_3 dataset is noisy even after 400 epochs.

4.3 Validation

The best models selected through validation have different pre–processing techniques and hyperparameters based on the training dataset which is used. Tables 2, 3 and 4 show the best models configurations for Random Forest, SVM and Neural Network, respectively.

Dataset	Pre–processing	N. of trees
D_0	Standardization	300
D_1	Standardization	300
D_2	Min–Max Scaling	200
D_3	Standardization	300

Table 2: Random Forest best configurations

Dataset	Pre–processing	kernel	C	gamma
D_0	Min–Max Scaling	radial	100	1
D_1	Standardization	radial	100	0.1
D_2	Min–Max Scaling	radial	100	10
D_3	Standardization	radial	31.623	1

Table 3: SVM best configurations

Dataset	hidden size	epochs	batch size	decay rate
D_0	64	250	256	0.08
D_1	64	500	128	0.01
D_2	64	100	128	0.01
D_3	64	400	256	0.01

Table 4: Neural Network best configurations

From this results, two observations can be made:

- as far as SVM is concerned, the best kernel is always radial kernel, which suggests that data is not linearly separable
- as far as Neural Network is concerned, the best hidden size is always the greatest one which has been tried

The validation scores for each dataset are shown in Figure 5. As it can be seen, training on the dataset with the fewest features results in a greater overfitting for non–probabilistic models.

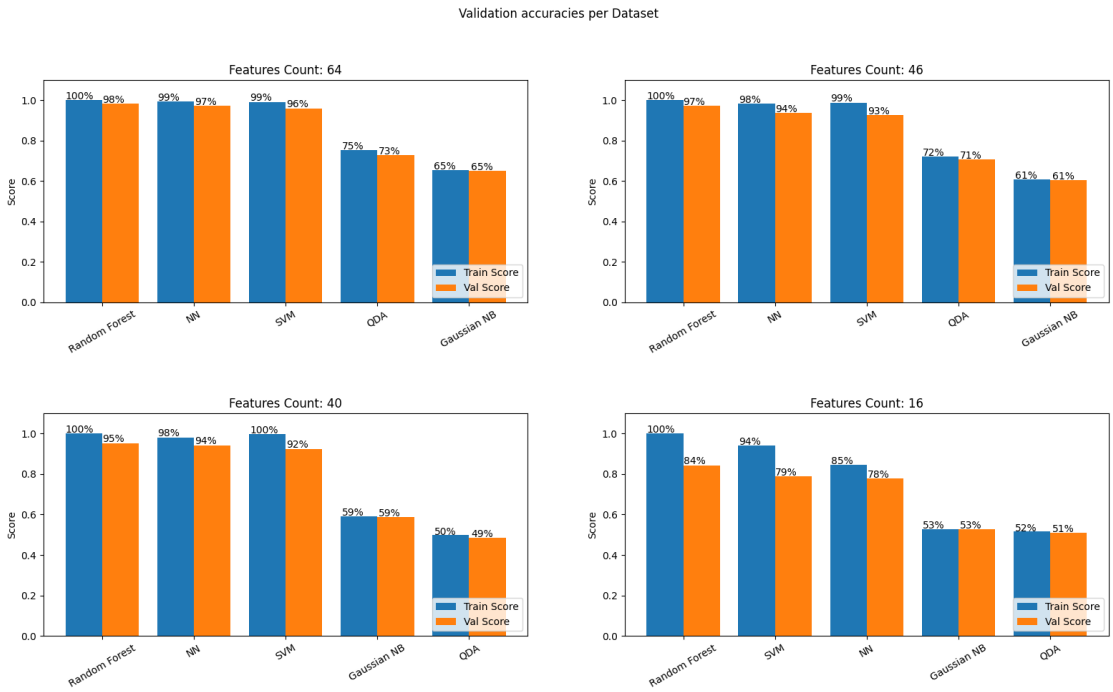


Figure 5: Validation accuracies per Dataset

4.4 Testing

Finally, each model is evaluated on never–seen data in the testing phase. Figure 6 shows a comparison between all the models grouped by sub–dataset.

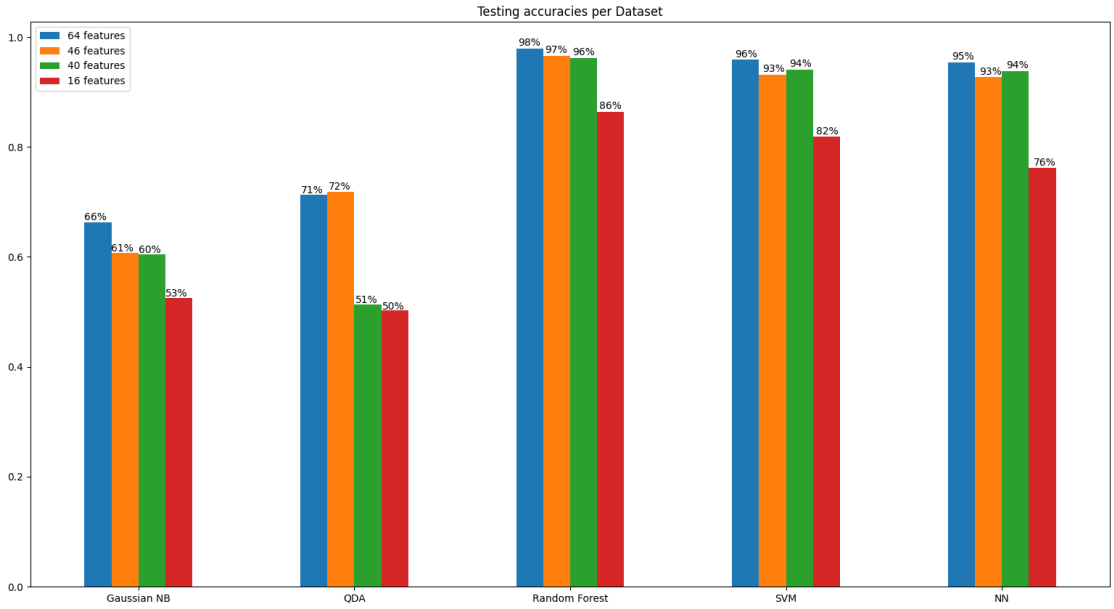


Figure 6: Testing accuracies

As it can be seen:

- the best performance is always achieved using the D_0 dataset
- Random Forest outperforms all the other models for each sub–dataset
- QDA has a better performance removing features with a great number of missing values
- QDA has a worse performance than Gaussian Naive Bayes in some cases because there are too many parameters to estimate and the dataset size is not large enough

In addition to the accuracies, the ROC curve and the confusion matrix are plotted for each sub–dataset and for each classic model. For the sake of completeness, all plotted ROC curves and confusion matrices are added in the Appendix A and Appendix B, respectively. Analysing the plots, several general observations emerge:

- The "Walking" class is the least troublesome with a low False Positive Ratio and a high ratio of True Positives, even in the D_3 dataset
- The "Car" class is the most troublesome, with a small Area Under Curve, except for the D_3 dataset where other classes have a smaller AUC value
- QDA and Gaussian Naive Bayes misclassify "Bus" and "Car" (which are mistaken as "Car" – for the "Bus" class only – "Still" or "Train") and they misclassify "Walking" as "Bus"
- Gaussian Naive Bayes misclassifies "Still" as "Train" in the D_1 and D_2 datasets
- QDA misclassifies "Train" as "Still" or "Car" in the D_2 and D_3 datasets, respectively

4.5 Conclusions

In conclusion, the Random Forest model of each sub–dataset is exploited to obtain an overview of the different sensors' importance. The features' importances of D_0 are shown in Figure 7. The other plots, for the sake of readability, are inserted in the Appendix C.

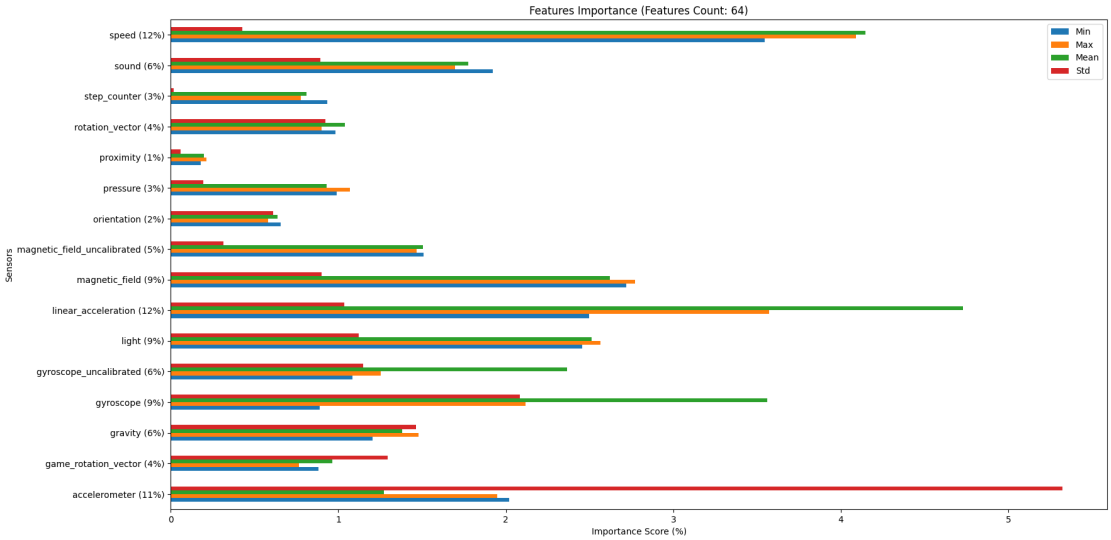


Figure 7: D_0 Features importance

In the plotted dataset, the sensors with a computed importance greater than 10% are: accelerometer (11%), linear acceleration (12%) and speed (12%). Specifically, the accelerometer's standard deviation is the most important feature followed by the linear acceleration's mean.

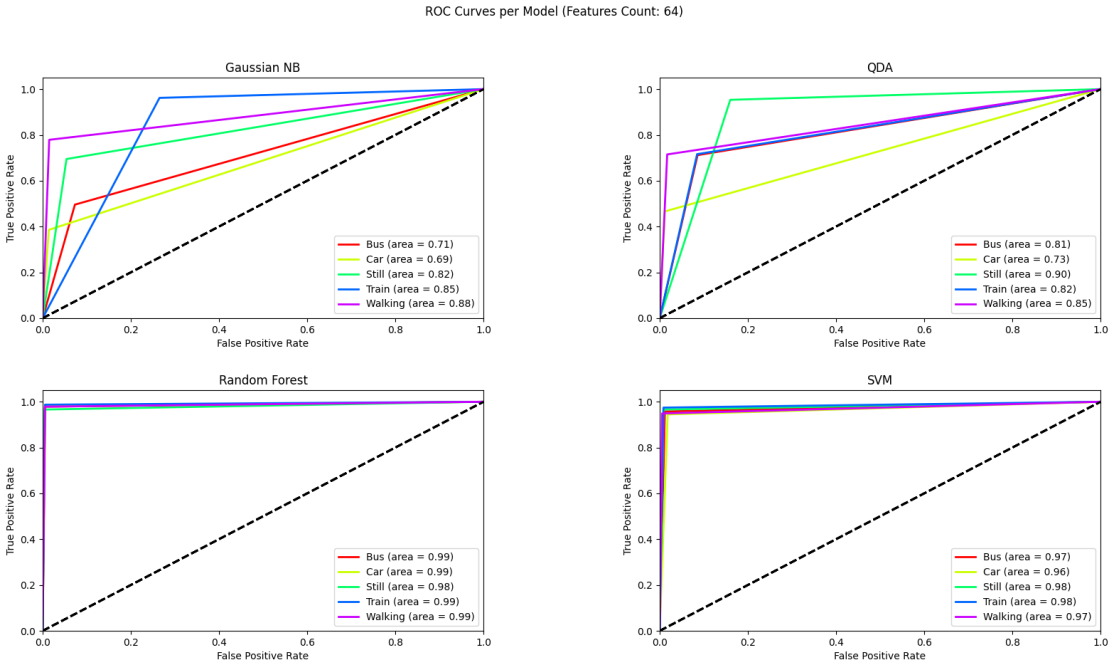
A possible future implementation can implement the models training based on features with an importance greater than a chosen threshold.

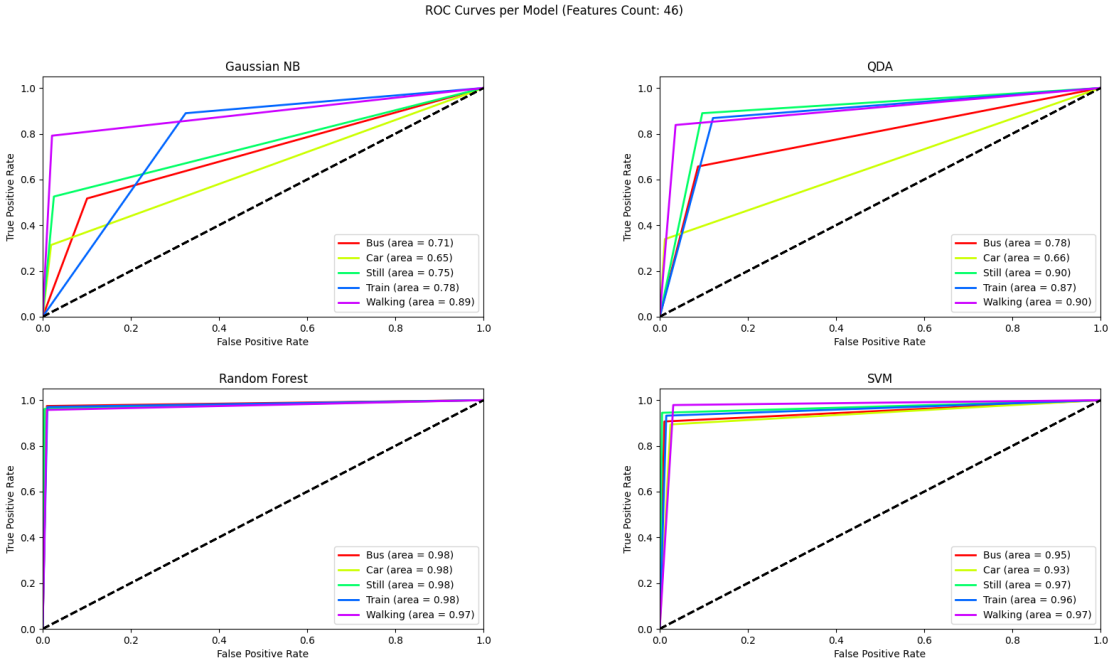
References

[BDFB16] Luca Bedogni, Marco Di Felice, and Luciano Bononi. Context-aware android applications through transportation mode detection techniques, November 2016.

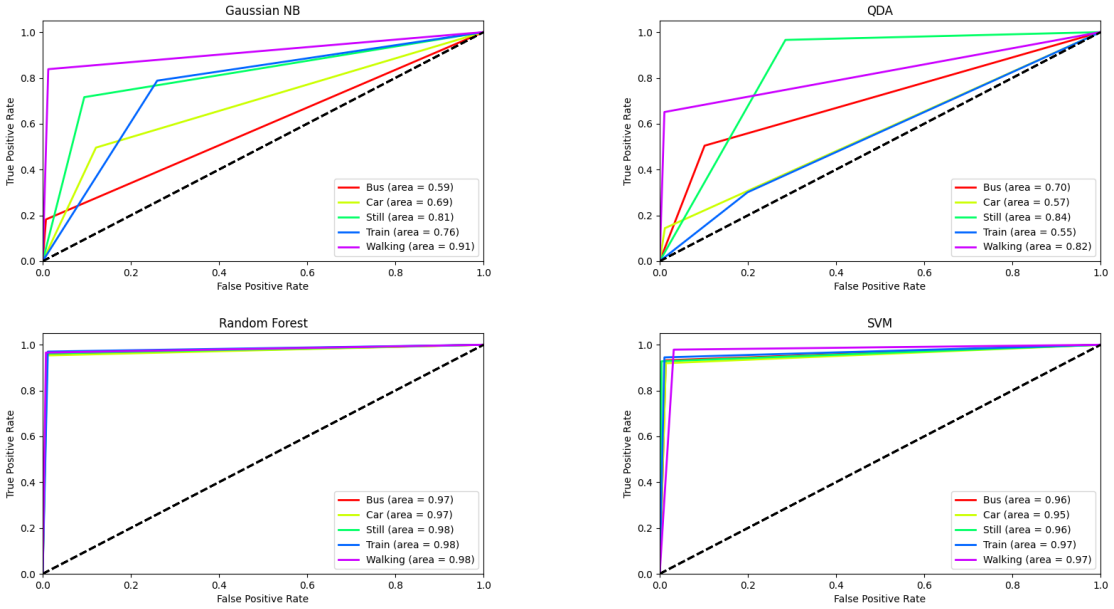
[CLB+18] Claudia Carpineti, Vincenzo Lomonaco, Luca Bedogni, Marco Di Felice, and Luciano Bononi. Custom dual transportation mode detection by smartphone devices exploiting sensor diversity, 2018.

Appendix A ROC curves

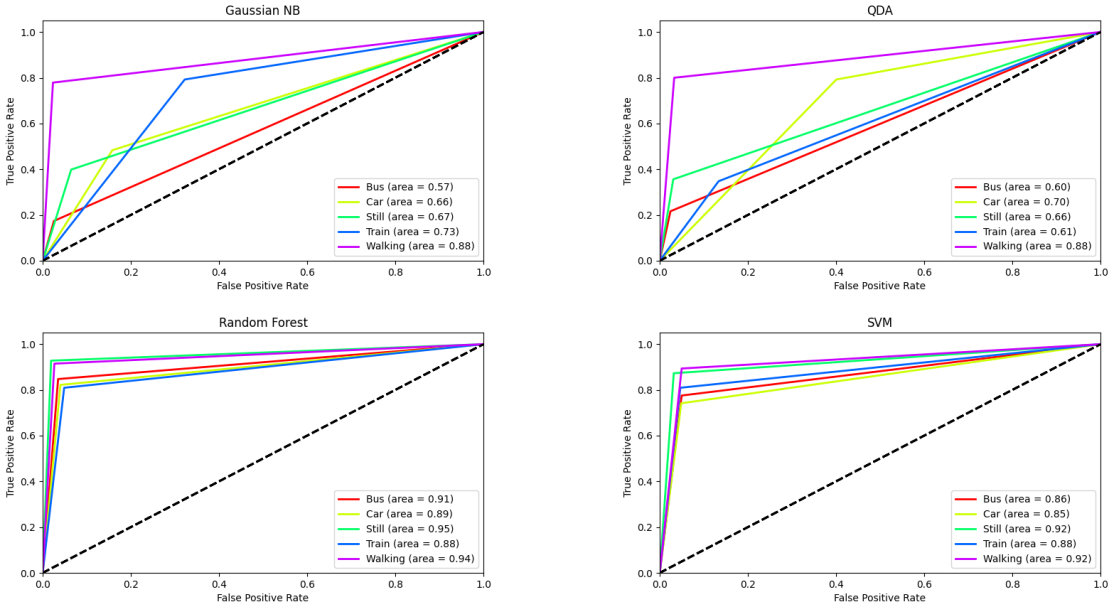




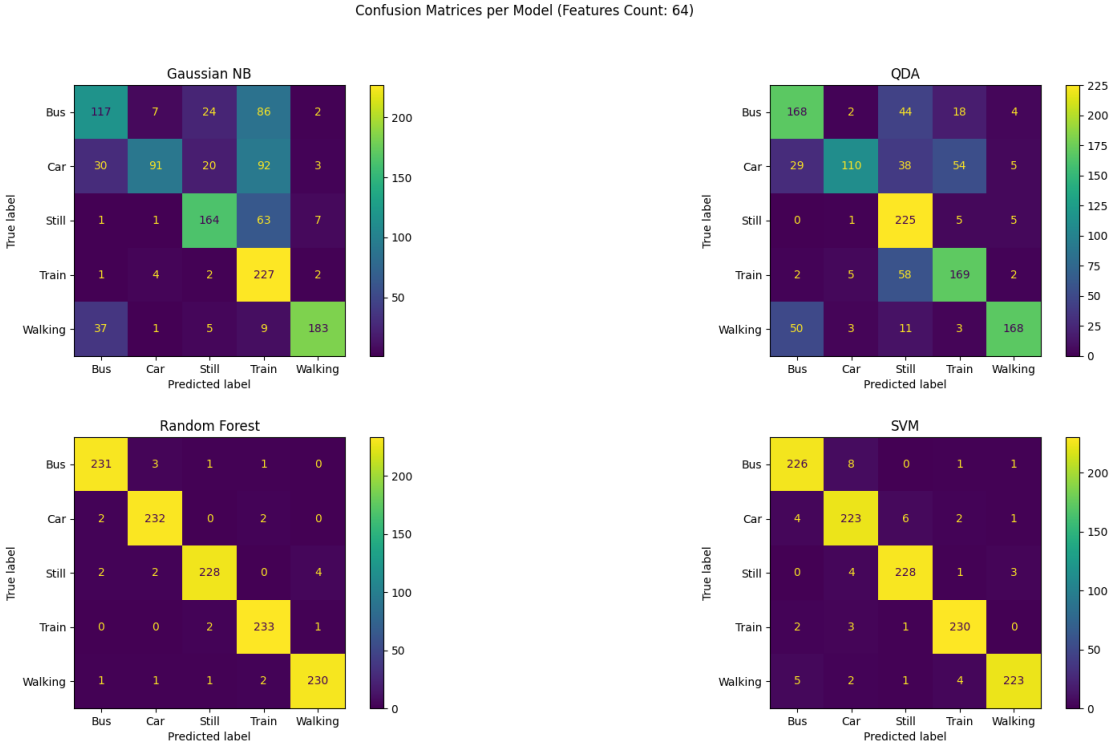
ROC Curves per Model (Features Count: 40)

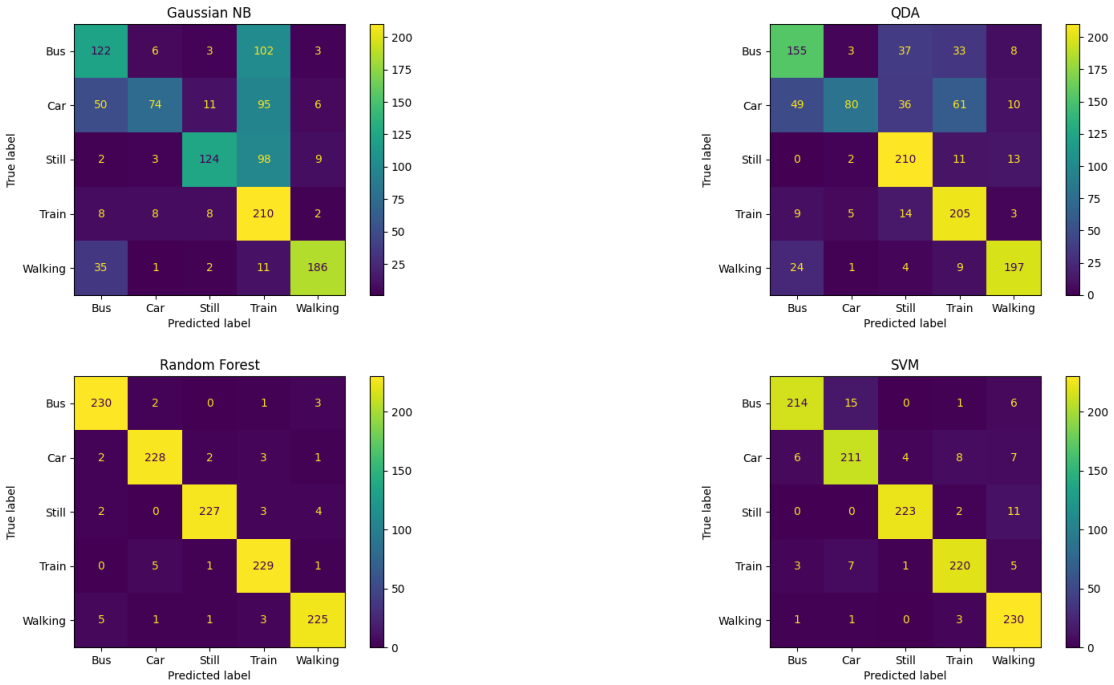


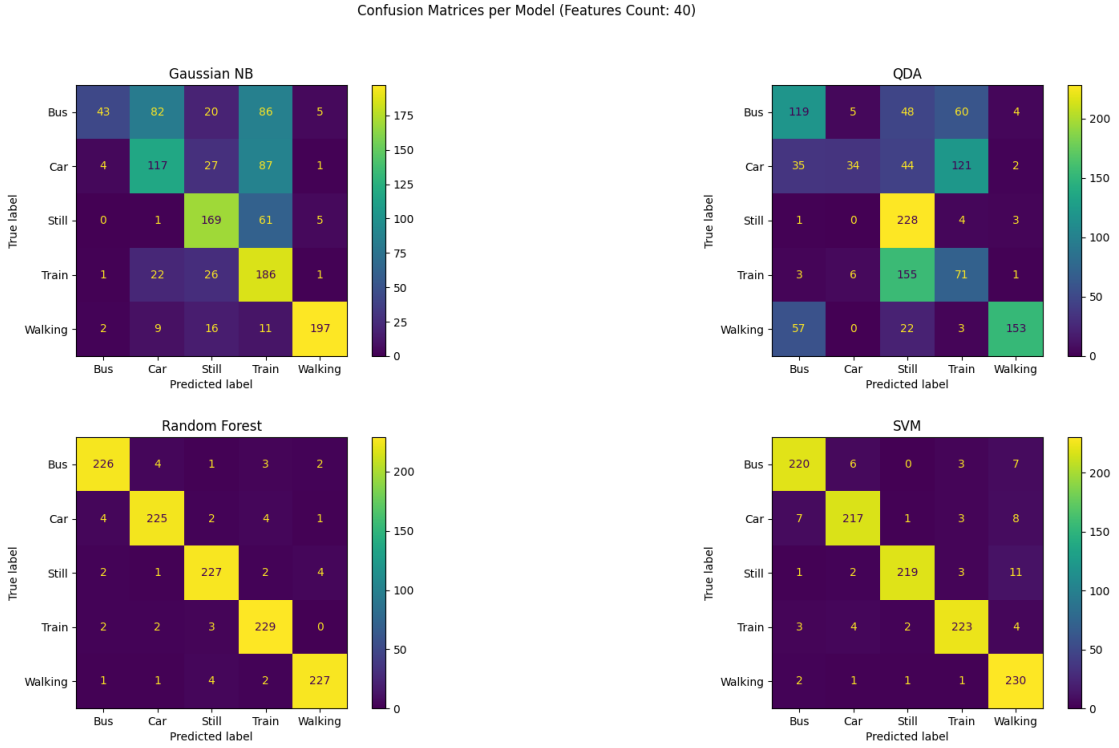
ROC Curves per Model (Features Count: 16)

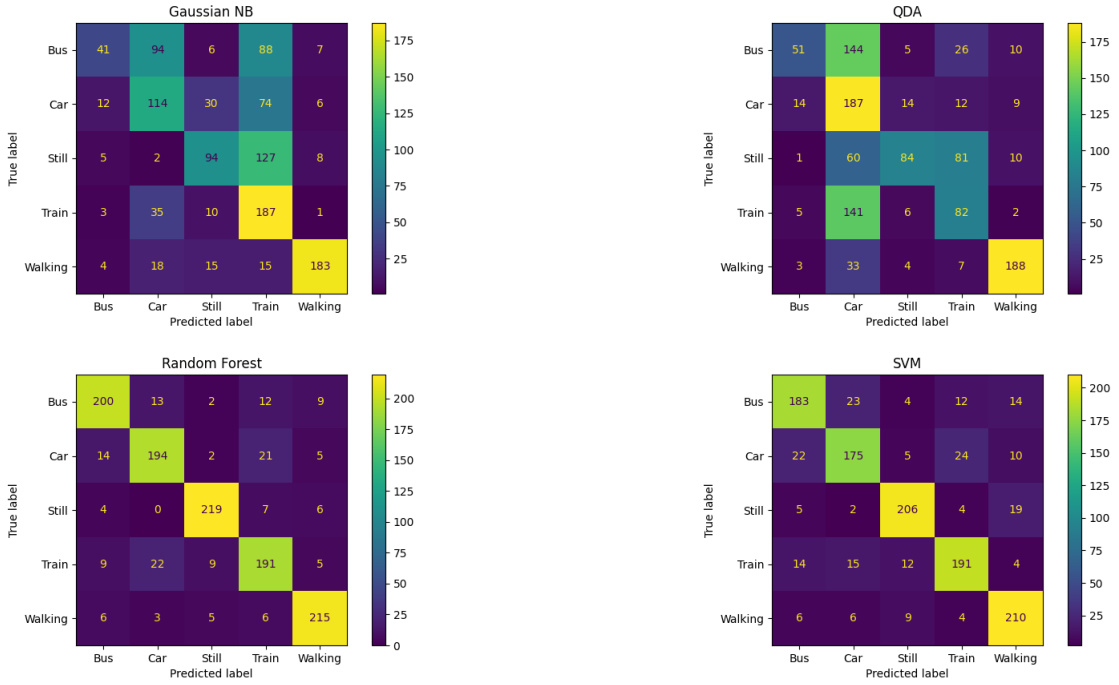


Appendix B Confusion matrices

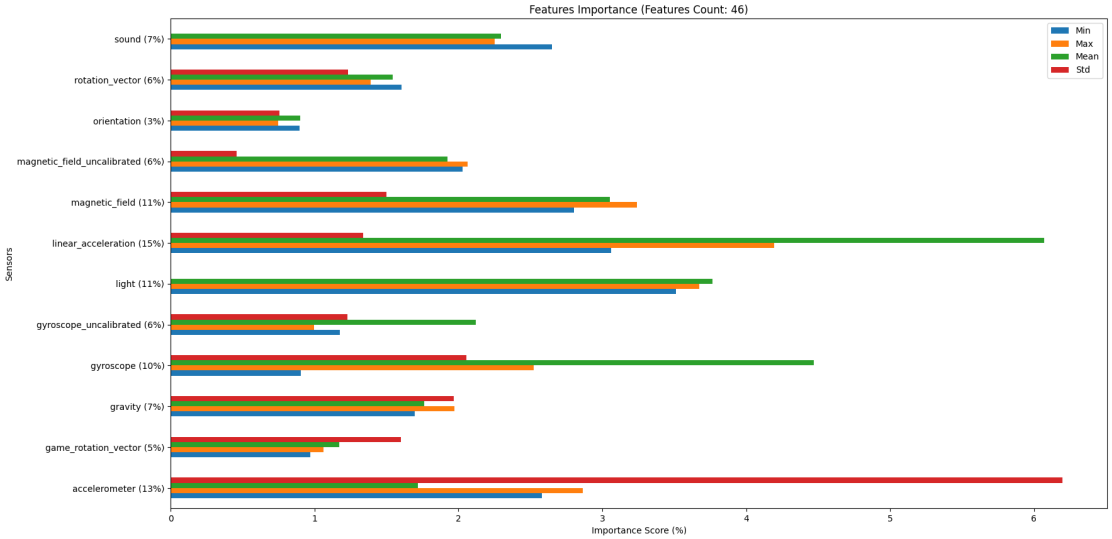


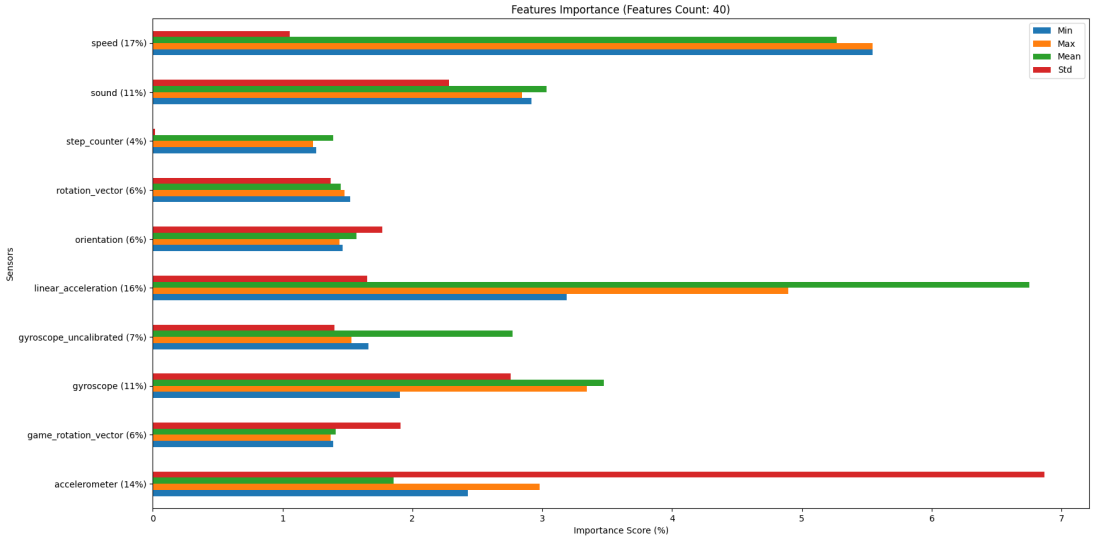


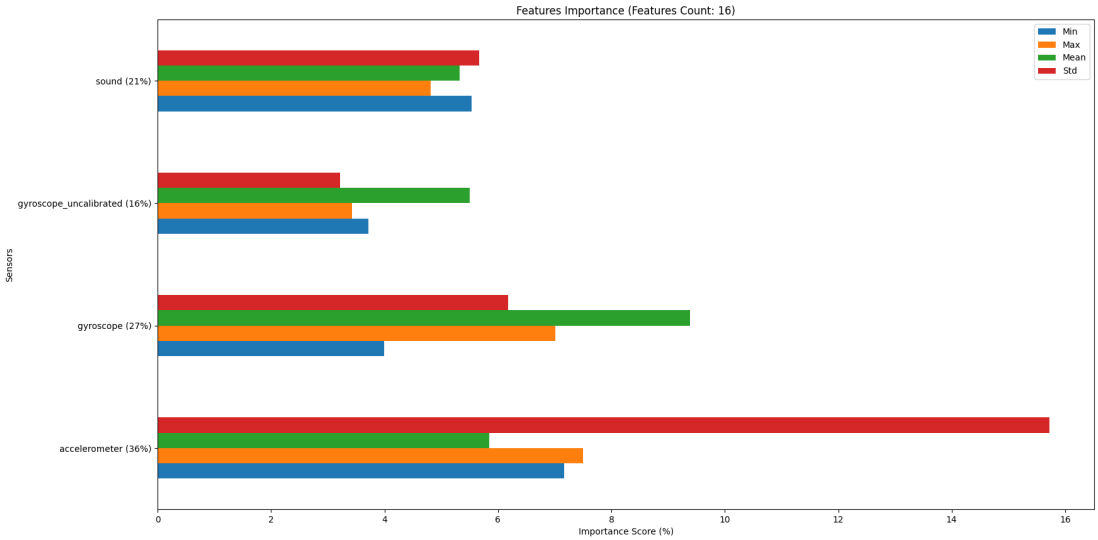

Confusion Matrices per Model (Features Count: 40)
Gaussian NB
Bus 43 82 20 86 5
Car 4 117 27 87 1
Still 0 1 169 61 5
Train 1 22 26 186 1
Walking 2 9 16 11 197
Bus Car Still Train Walking
True label
Predicted label
QDA
Bus 119 5 48 60 4
Car 35 34 44 121 2
Still 1 0 228 4 3
Train 3 6 155 71 1
Walking 57 0 22 3 153
Bus Car Still Train Walking
True label
Predicted label
Random Forest
Bus 226 4 1 3 2
Car 4 225 2 4 1
Still 2 1 227 2 4
Train 2 2 3 229 0
Walking 1 1 4 2 227
Bus Car Still Train Walking
True label
Predicted label
SVM
Bus 220 6 0 3 7
Car 7 217 1 3 8
Still 1 2 219 3 11
Train 3 4 2 223 4
Walking 2 1 1 1 230
Bus Car Still Train Walking
True label
Predicted label


Confusion Matrices per Model (Features Count: 16)
Gaussian NB
Bus 41 94 6 88 7
Car 12 114 30 74 6
Still 5 2 94 127 8
Train 3 35 10 187 1
Walking 4 18 15 15 183
Bus Car Still Train Walking
True label
Predicted label
QDA
Bus 51 144 5 26 10
Car 14 187 14 12 9
Still 1 60 84 81 10
Train 5 141 6 82 2
Walking 3 33 4 7 188
Bus Car Still Train Walking
True label
Predicted label
Random Forest
Bus 200 13 2 12 9
Car 14 194 2 21 5
Still 4 0 219 7 6
Train 9 22 9 191 5
Walking 6 3 5 6 215
Bus Car Still Train Walking
True label
Predicted label
SVM
Bus 183 23 4 12 14
Car 22 175 5 24 10
Still 5 2 206 4 19
Train 14 15 12 191 4
Walking 6 6 9 4 210
Bus Car Still Train Walking
True label
Predicted label

Appendix C Features importance






Features Importance (Features Count: 16)
Min
Max
Mean
Std
sound (21%)
gyroscope_uncalibrated (16%)
gyroscope (27%)
accelerometer (36%)
Sensors
0
2
4
6
8
10
12
14
16
Importance Score (%)