Sensors measurement analysis for Transport Mode Detection

Alessandro Giacchè, Paola Persico

University Alma Mater Studiorum of Bologna

May 1, 2021

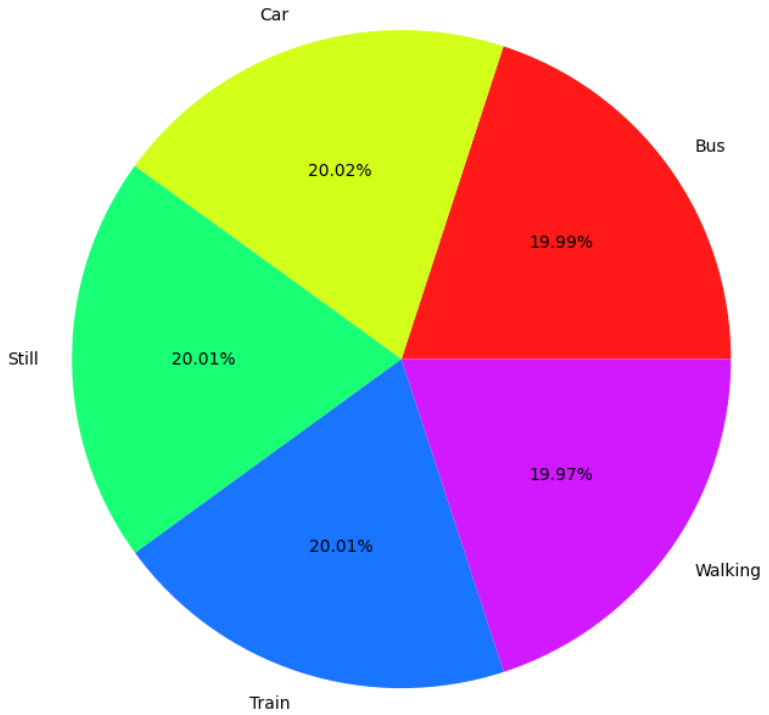
Overview

Introduction

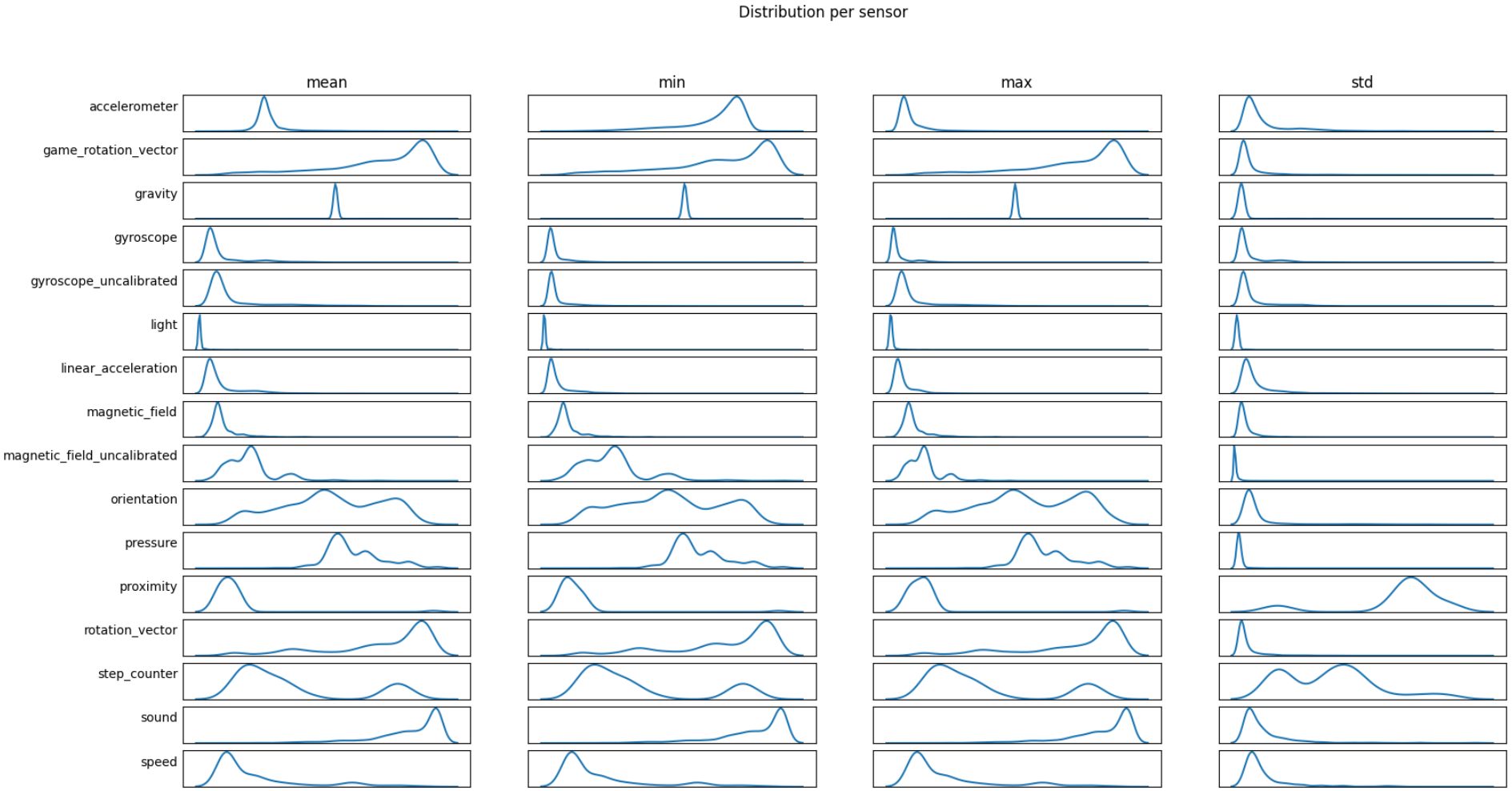
- **Goal**: Transport Mode Detection $\rightarrow$ multi-label classification
  - HAR for Context-Aware Systems
- **Approach**: Smartphone sensors analysis
- **How**: Machine Learning algorithms
  - SVM
  - Gaussian Naive Bayes
  - QDA
  - Random Forest
  - Feedforward Neural Network

Acquisition: Transportation Mode Detection Dataset

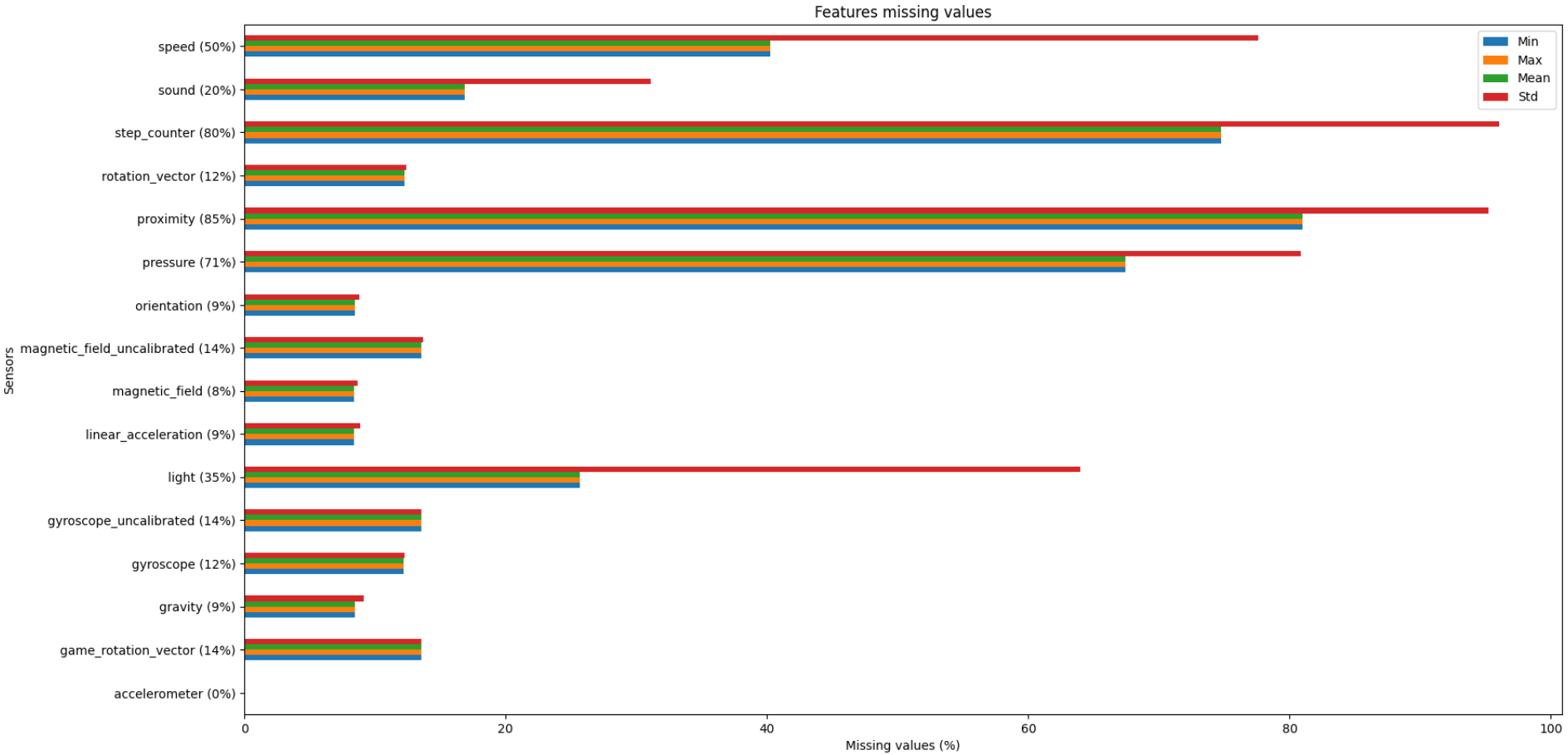
- 5,893 **samples**
- 64 **features**
  - 16 **sensors** sampling
  - 5 seconds window feature extraction
  - 4 **stats** per sensor
    - minimum
    - maximum
    - standard deviation
    - mean
- 5 **classes**



Acquisition: Distribution per feature



Acquisition: Missing values per feature



Pre–processing: Feature Selection

- 4 datasets
  - $D_0 \rightarrow$ **64 features** – full dataset
  - $D_1 \rightarrow$ **46 features** – features with less than 30% of *NaN*
  - $D_2 \rightarrow$ **40 features** – removing:
    - light
    - gravity
    - magnetic field
    - pressure
    - proximity
  - $D_3 \rightarrow$ **16 features** – keeping low-battery sensors:
    - gyroscope (calibrated and uncalibrated)
    - accelerometer
    - sound

Pre–processing

- Missing values replacement $\rightarrow$ Median
- Normalization
  - Min–Max Scaling
  - Standardization
- Train-test splitting

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

Modeling: Classic Models

- Gaussian Naive Bayes
- QDA
- Tuned Models:
  - SVM
    - kernel
    - C, γ, d
  - Random Forest
    - $\#_{TREES}$

Validation

Hyperparameters tuning and validation scoring are performed using a 10–fold cross–validation technique

Modeling: Feedforward Neural Network

- Architecture
  - 3 Hidden Layers
  - Activation function: ReLU + SoftMax
  - Loss function: Multi–class Cross Entropy
  - Optimizer: Stochastic Gradient Descent
  - Learning rate decay
  - Batch normalization
- Hyperparameters: hidden size, epochs, minibatch size and γ (decay rate)
- Hold–out validation

Implementation (1/2)

- Libraries
  - **Pandas**
  - **Numpy**
  - **Scikit–learn**
  - **Pytorch**
  - **Matplotlib**
  - **Seaborn**
  - **Joblib**

```
source
├── main.py
├── data_layer.py
├── preprocessing.py
├── models_config.py
├── model_runner.py
├── evaluation.py
├── visualization.py
├── pytorch
│   ├── dataset.py
│   ├── nn_main.py
│   ├── model.py
│   └── model_runner.py
├── dataset
└── saved_models
```

Implementation: Classic models CV

```
def run_crossvalidation(X_trainval, y_trainval, clf, params, cv=5,
↪  verbose=True):
    params["scaler"] = [StandardScaler(), MinMaxScaler()]
    pipeline = Pipeline([('scaler', StandardScaler()), ('clf', clf)])

    grid_search = GridSearchCV(pipeline, params, cv=cv, verbose=10 if verbose
    ↪  else 0, n_jobs=16, return_train_score=True)
    grid_search.fit(X_trainval, y_trainval)

    return pd.DataFrame(grid_search.cv_results_), grid_search.best_estimator_
```

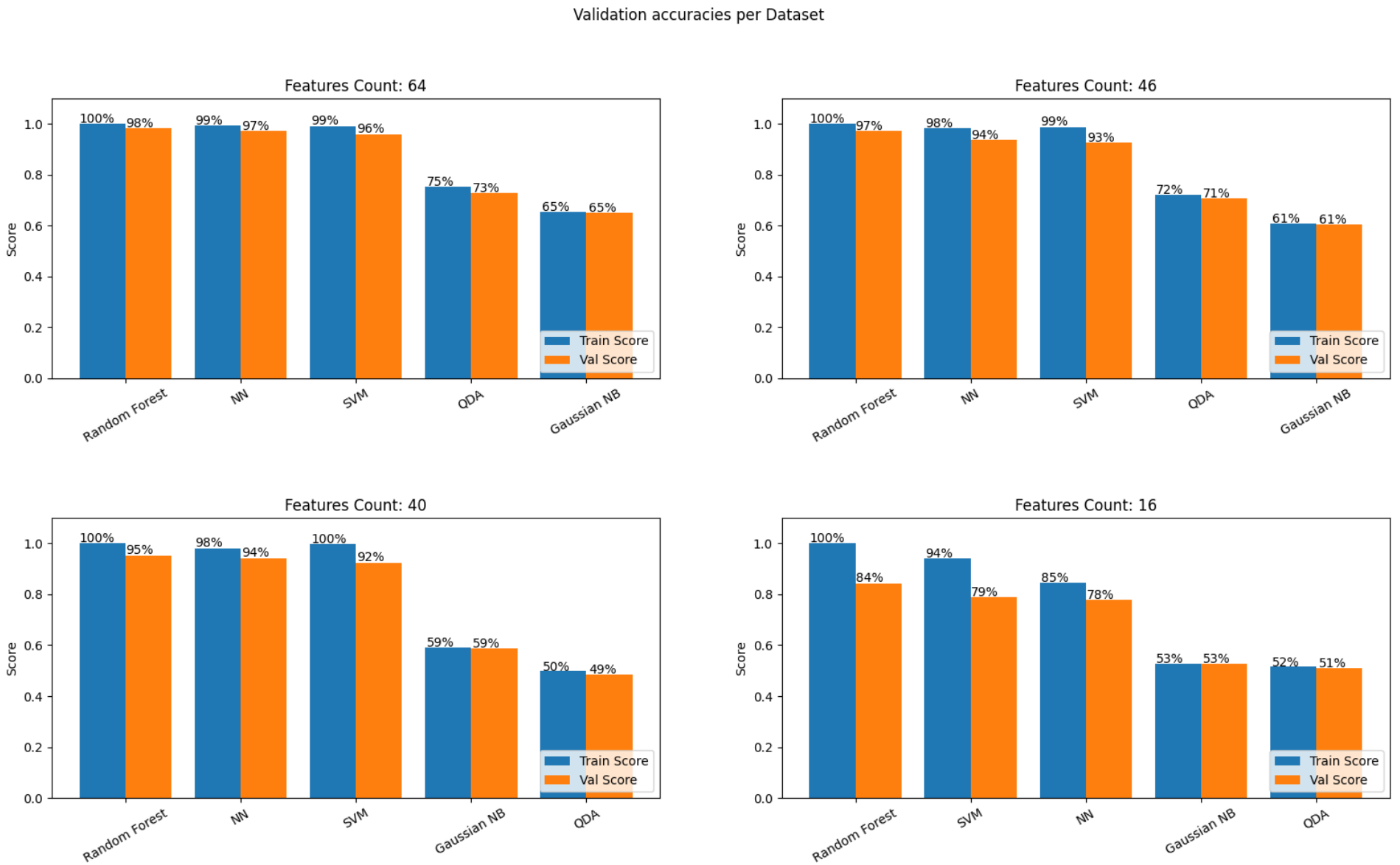
Implementation: MLP Validation

```
# Cartesian product of the sets of hyperparameters
hyperparams = itertools.product(hidden_sizes, nums_epochs, batch_sizes, gamma)
# split the indices (72% train, 8% val, 20% test)
train_idx, test_idx = train_test_split(np.arange(len(y)), test_size=0.2,
↪  stratify=y, random_state=42)
# remove missing values
X[train_idx], X[test_idx] = preprocessing.remove_nan(X[train_idx], X[test_idx])
train_idx, val_idx = train_test_split(train_idx, test_size=0.1,
↪  stratify=y[train_idx], random_state=42)
```

Results: Timing

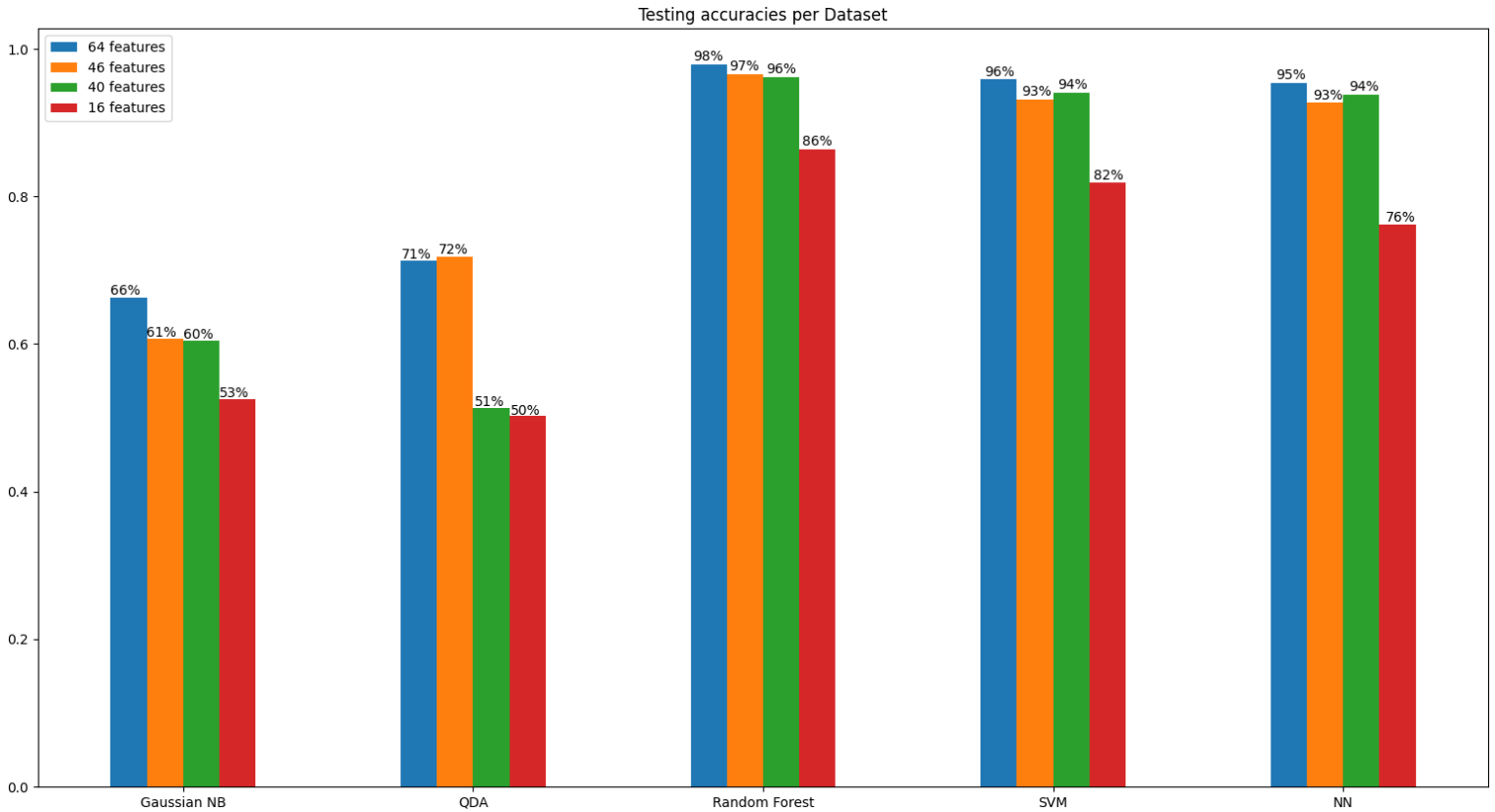
- Fitting time
  1. Gaussian Naive Bayes, QDA $\rightarrow$ $<$0.05s
  2. SVM, Random Forest $\rightarrow$ 1-5s
  3. MLP $\rightarrow$ 12-58s
- Prediction time
  1. Gaussian Naive Bayes, QDA $\rightarrow$ $<$0.01s
  2. SVM $\rightarrow$ 0.05s
  3. Random Forest $\rightarrow$ 0.72-1.07s
  4. MLP $\rightarrow$ 0.53-3.49s

Results: Validation



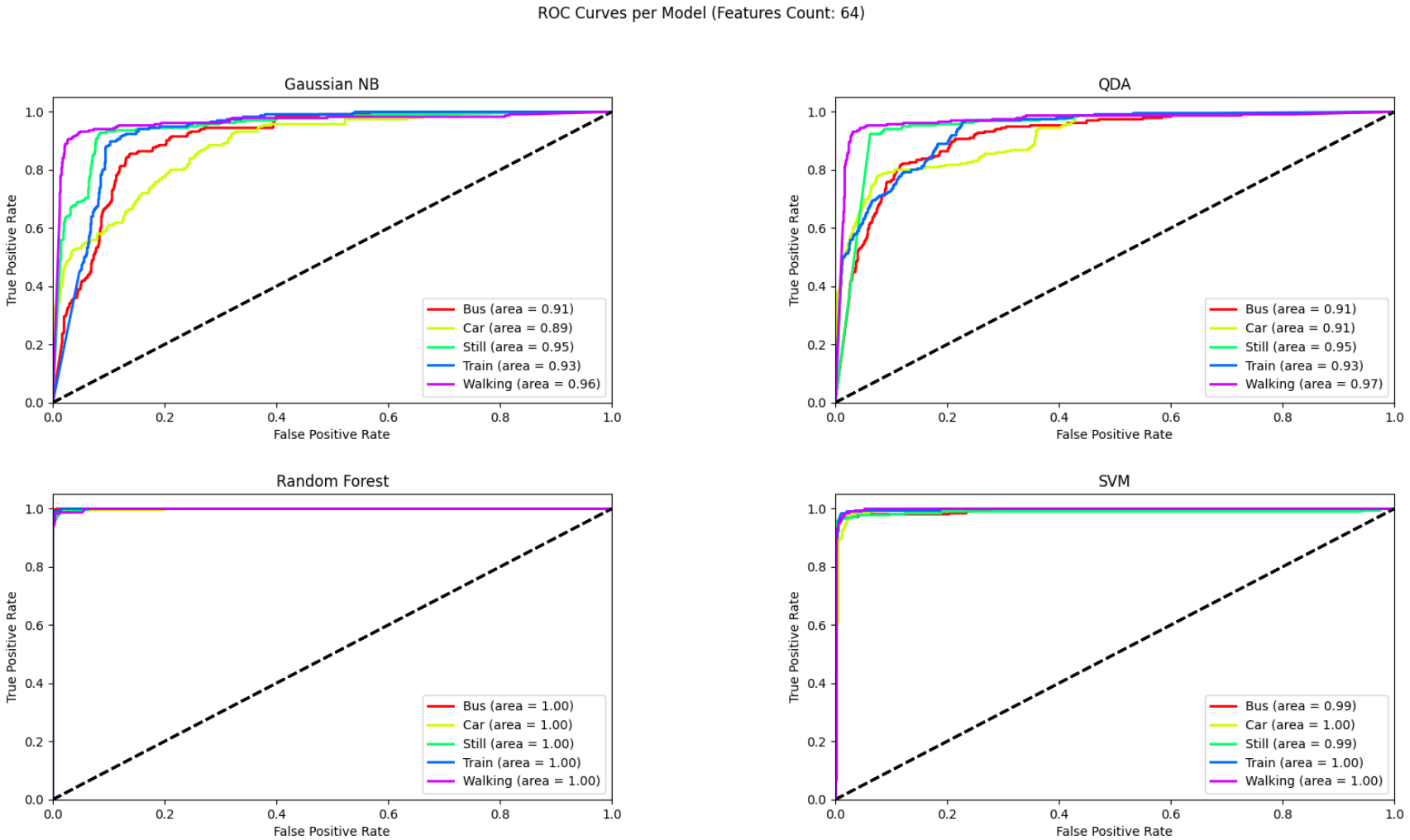
Results: Testing (1/3)

- Best Dataset: D_0
- Best model: Random Forest
- QDA: better performance in D_1

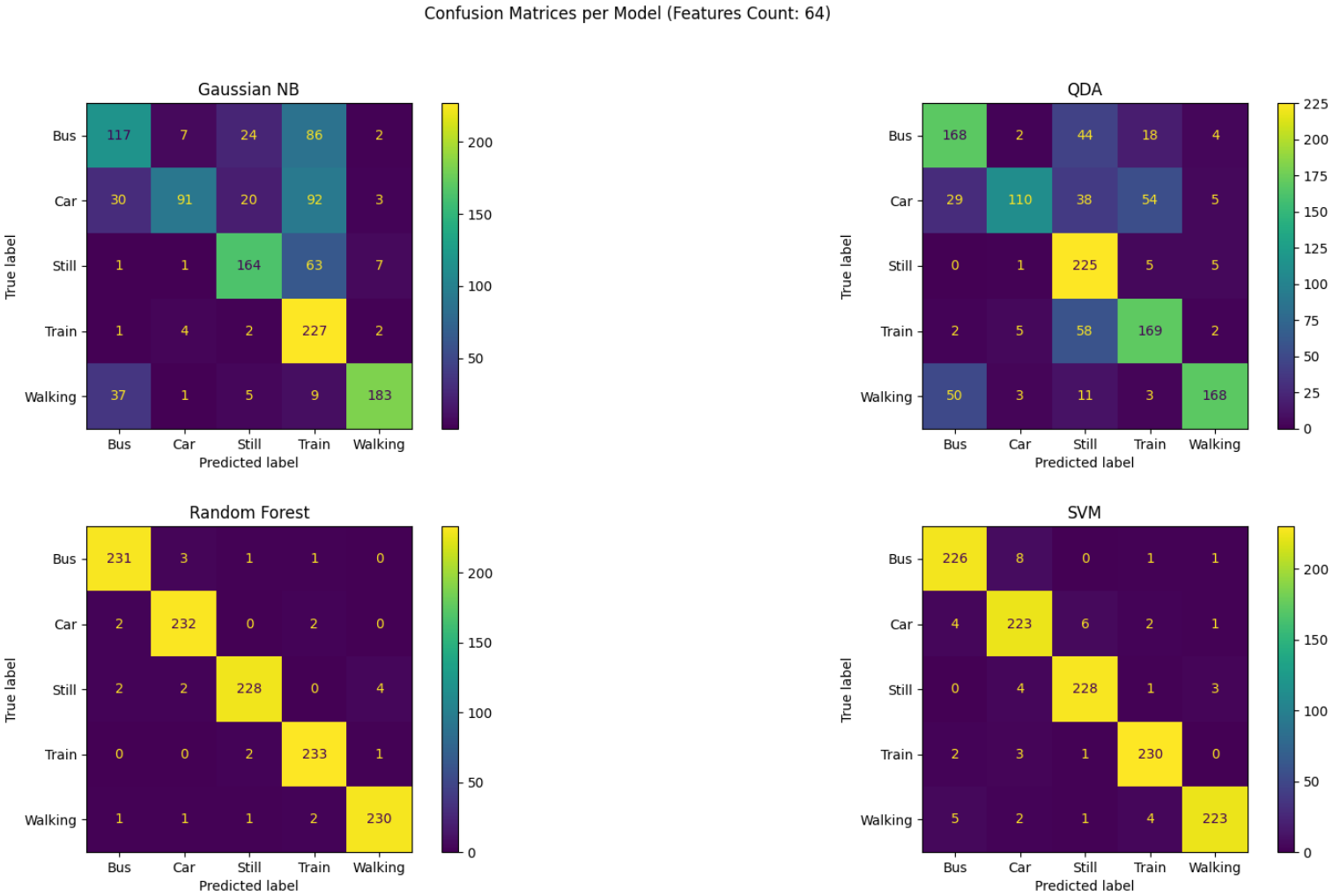


Results: Testing (2/3)

- Large AUC: Walking
- Small AUC: Car

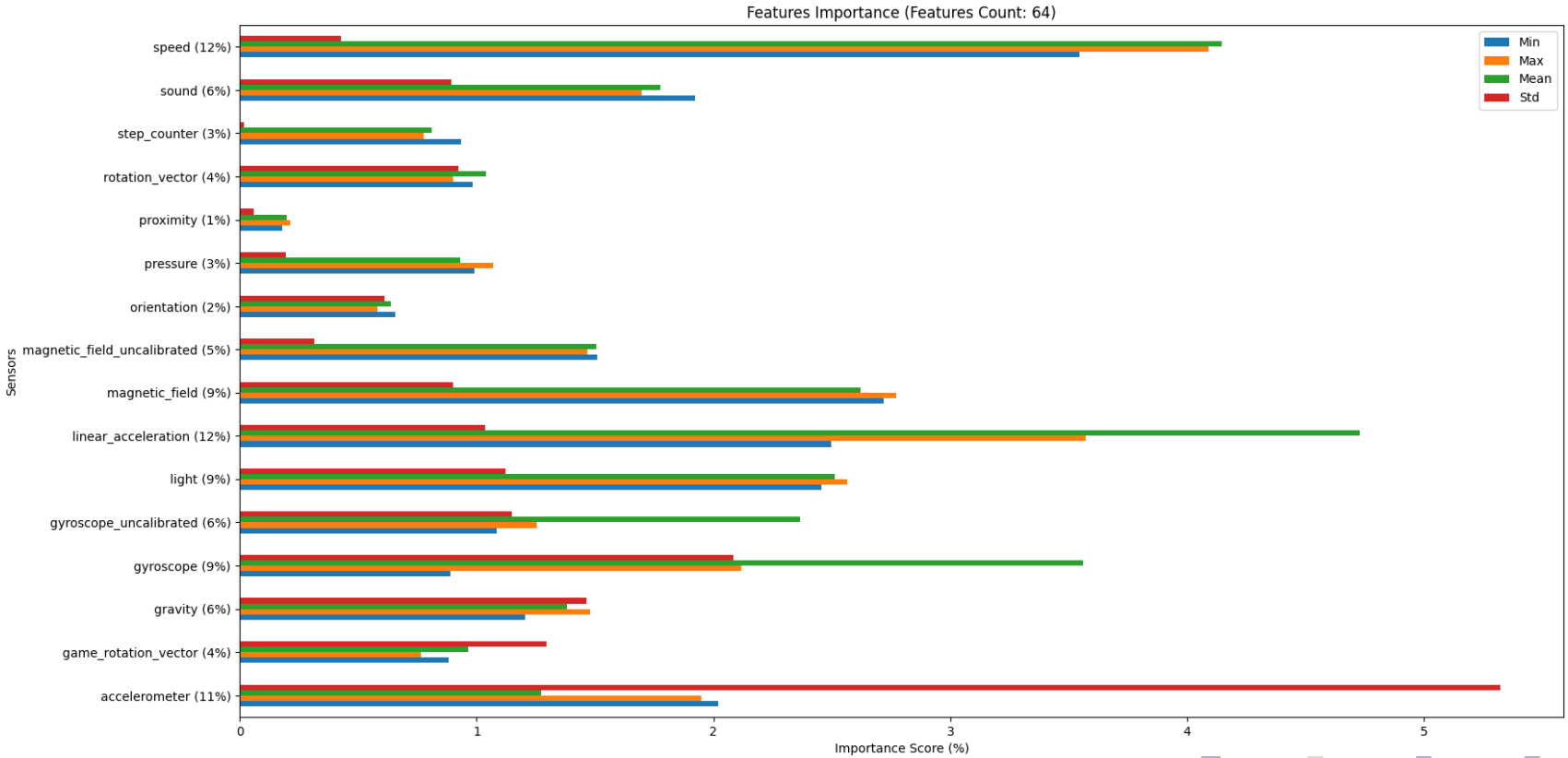


Results: Testing (3/3)



Conclusions

- TMD task solvable through sensor analysis
- Further work: exploit feature importance
  - accelerometer
  - linear acceleration
  - speed



References

Luca Bedogni, Marco Di Felice, Luciano Bononi (2016)

Context-Aware Android Applications through Transportation Mode Detection Techniques

Claudia Carpineti, Vincenzo Lomonaco, Luca Bedogni, Marco Di Felice, Luciano Bononi (2018)

Custom Dual Transportation Mode Detection by Smartphone Devices Exploiting Sensor Diversity