Contrastive Learing

FOR TIME SERIES

Team_26
Mateusz Budzyński
Maciej Ciepiela
Supervisor: Klaudia Blacer

TABLE OF CONTENTS

GOALS AND MOTIVATIONS

Objective n° 1

Comparing classic machine learning algorithms with contrastive learning method

Objective n° 2

Deep understanding, visualization and pre-processing data for better approach to time series

Objective n° 3

Achieving results similar to the ones given in TF-C Paper

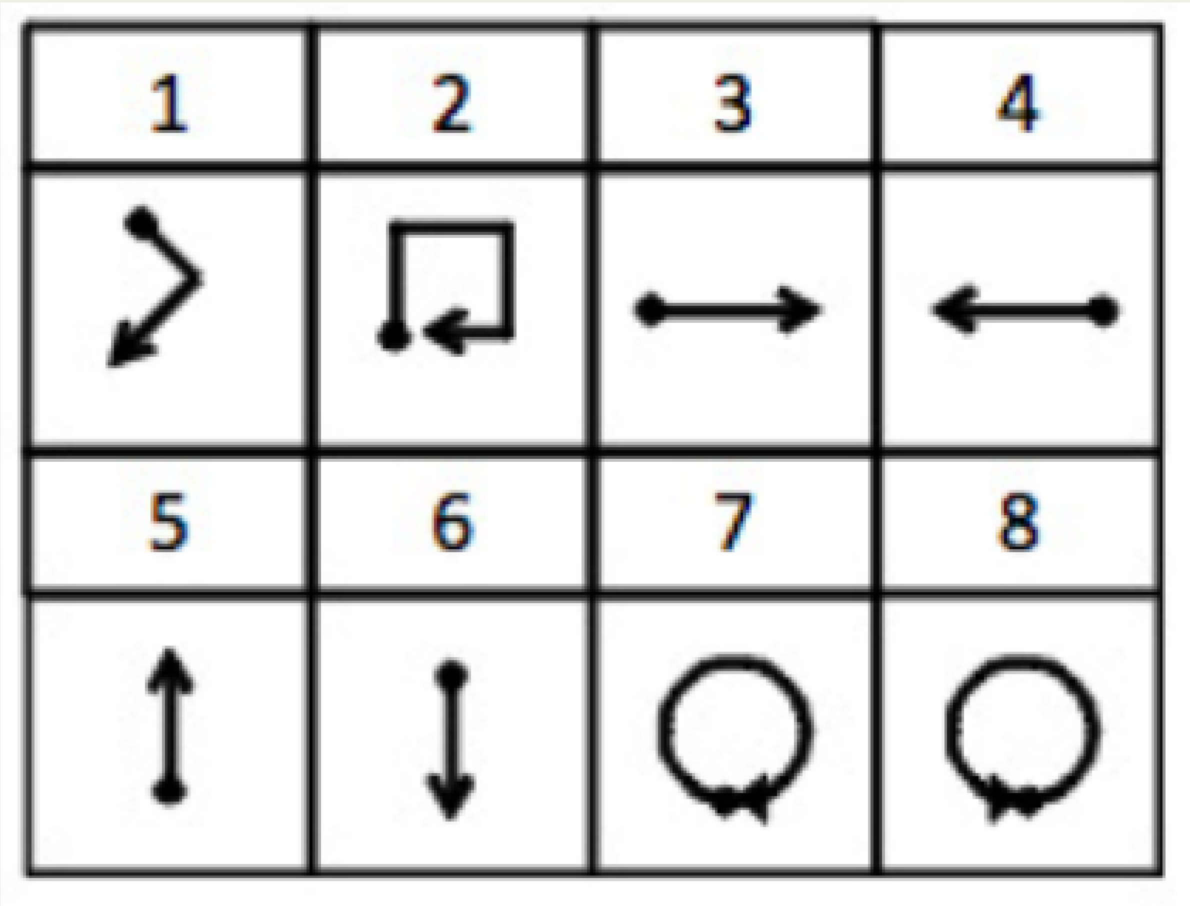
Datasets

1. Pre-training data (only for TF-C): HAR
2. Fine-tuning data: Gesture

DATASET

We downloaded raw gestures dataset. A set of eight simple gestures generated from accelerometers. The data consists of the X,Y,Z coordinates of each motion. Originaly file was in .arff format and we needed to use several function to transform it into numpy array. Then we divided it between training, validation and test sets while keeping classes distribution similar. And also made distinctions for every dimension.

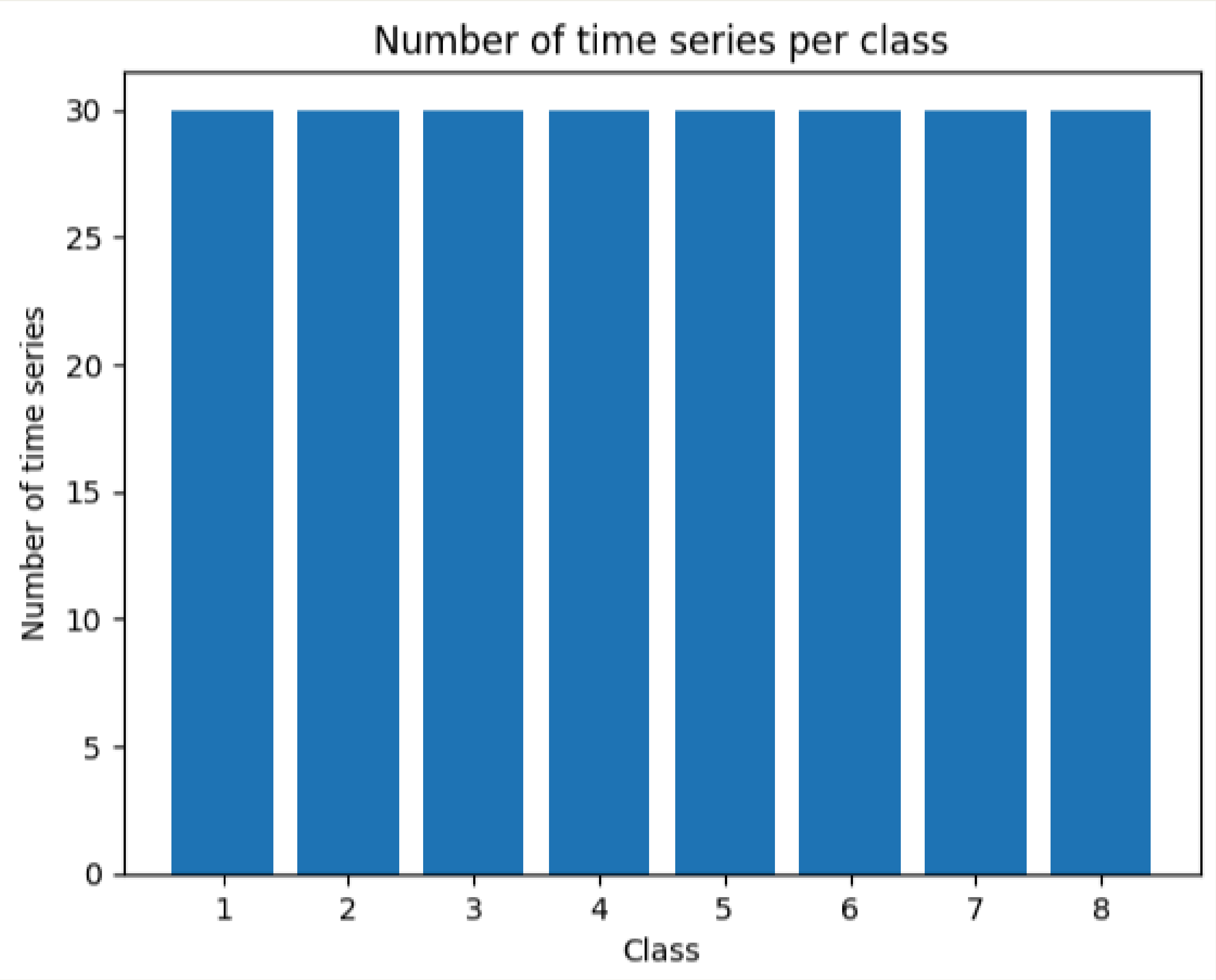
We also downloaded HAR dataset, already processed by TF-C paper authors for future TF-C implementation. Data was already divided between train, validation and test sets, all in pytorch format.



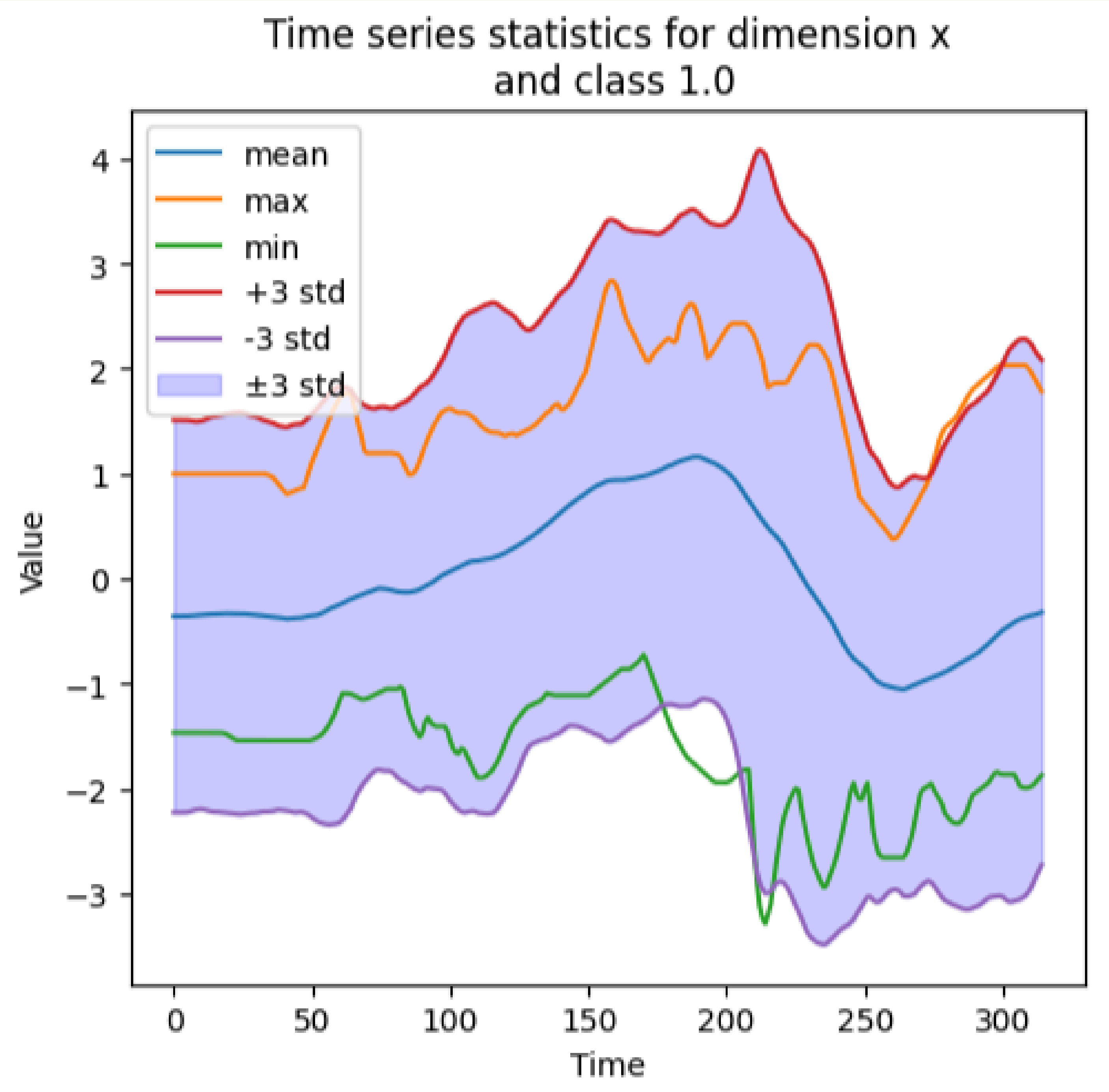
STATISTICS

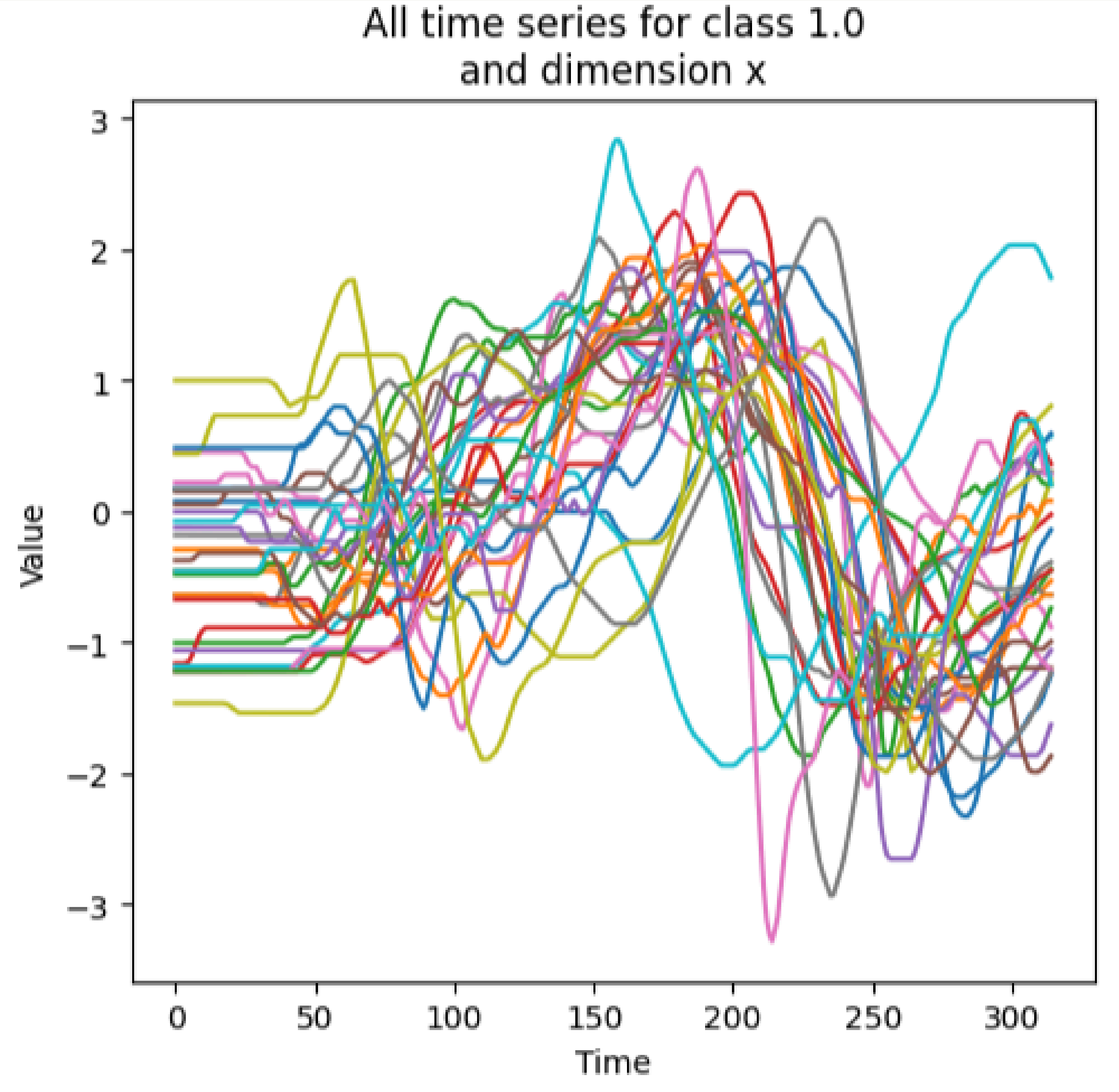
We calculated mean, standard deviation, min and max for timeseries in every singular dimension. Then we plotted the achieved results with distinction for every class and every dimension.

We also made some controlling plots to see data distribution within classes and the difference between specific time series



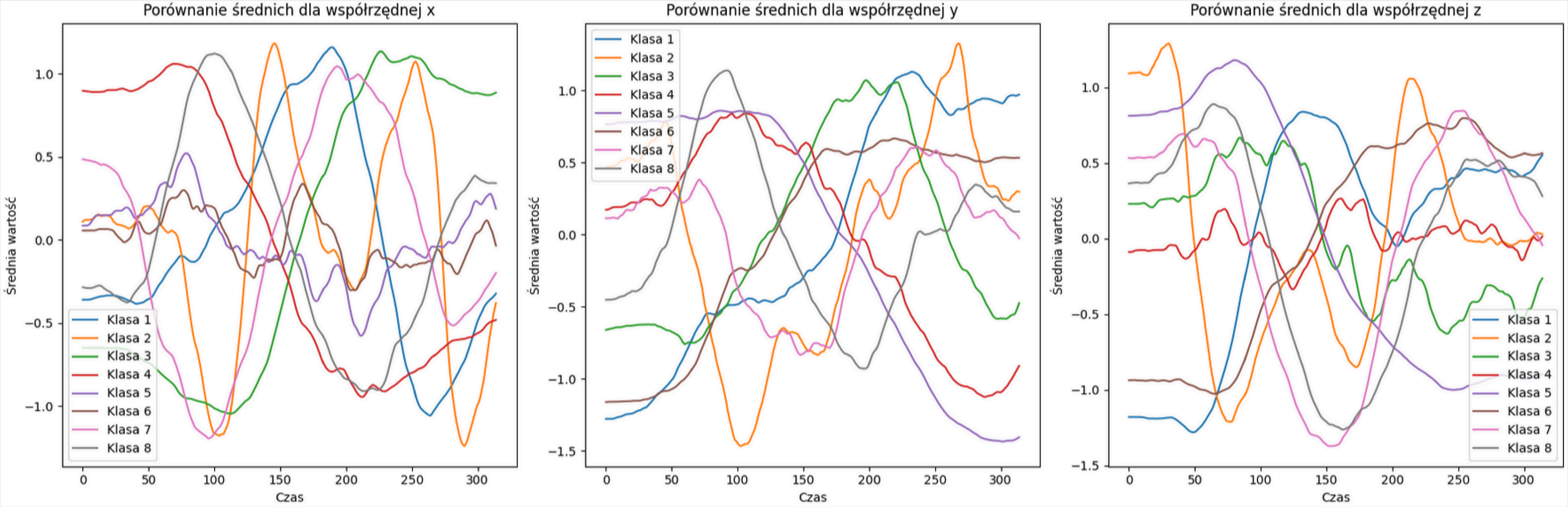
EXAMPLE PLOTS





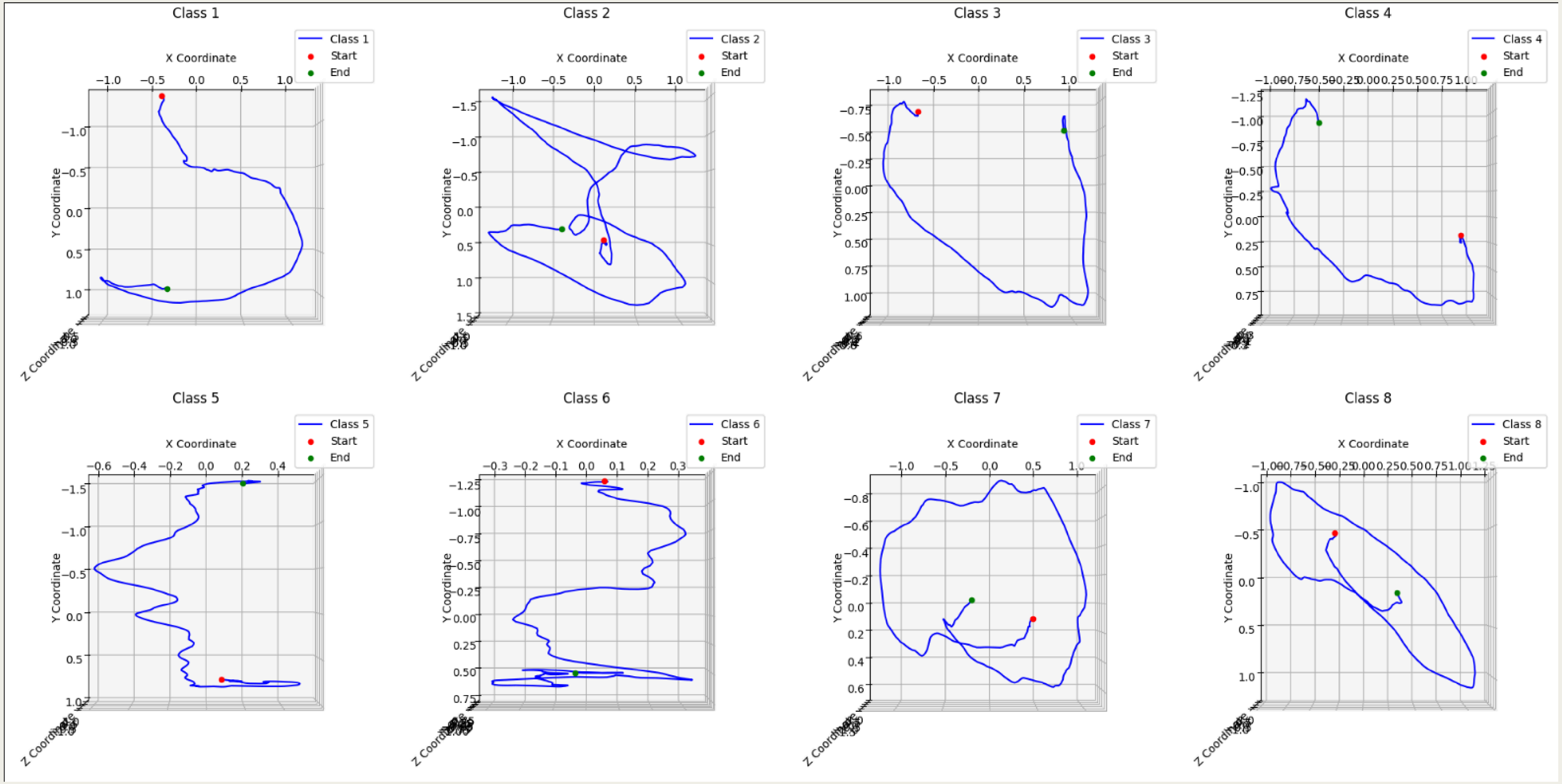
EXAMPLE PLOTS

Means comparition for different classes



VISUALIZATION

Ploted time series means for every class in 3D space to compare results with actual gestures described by dataset authors



MEASURING MODELS PERFORMANCE

Measured models performance using different metrics and plotting the results

Accuracy

Accuracy measures how often the model predicts correctly across all predictions. It's useful when the classes in the dataset are balanced.

$$\frac{TP + TN}{P + N}$$

Precision

Precision evaluates the proportion of correctly predicted positives out of all positive predictions. It's important when minimizing false positives is crucial.

$$\frac{TP}{TP + FP}$$

Recall

Recall calculates how many actual positives the model correctly identified. It's vital when missing positive cases has serious consequences

$$\frac{TP}{P}$$

F1 score

F1-score provides a balance between precision and recall, especially useful when there's an uneven distribution of classes. It gives a single performance metric to evaluate the model.

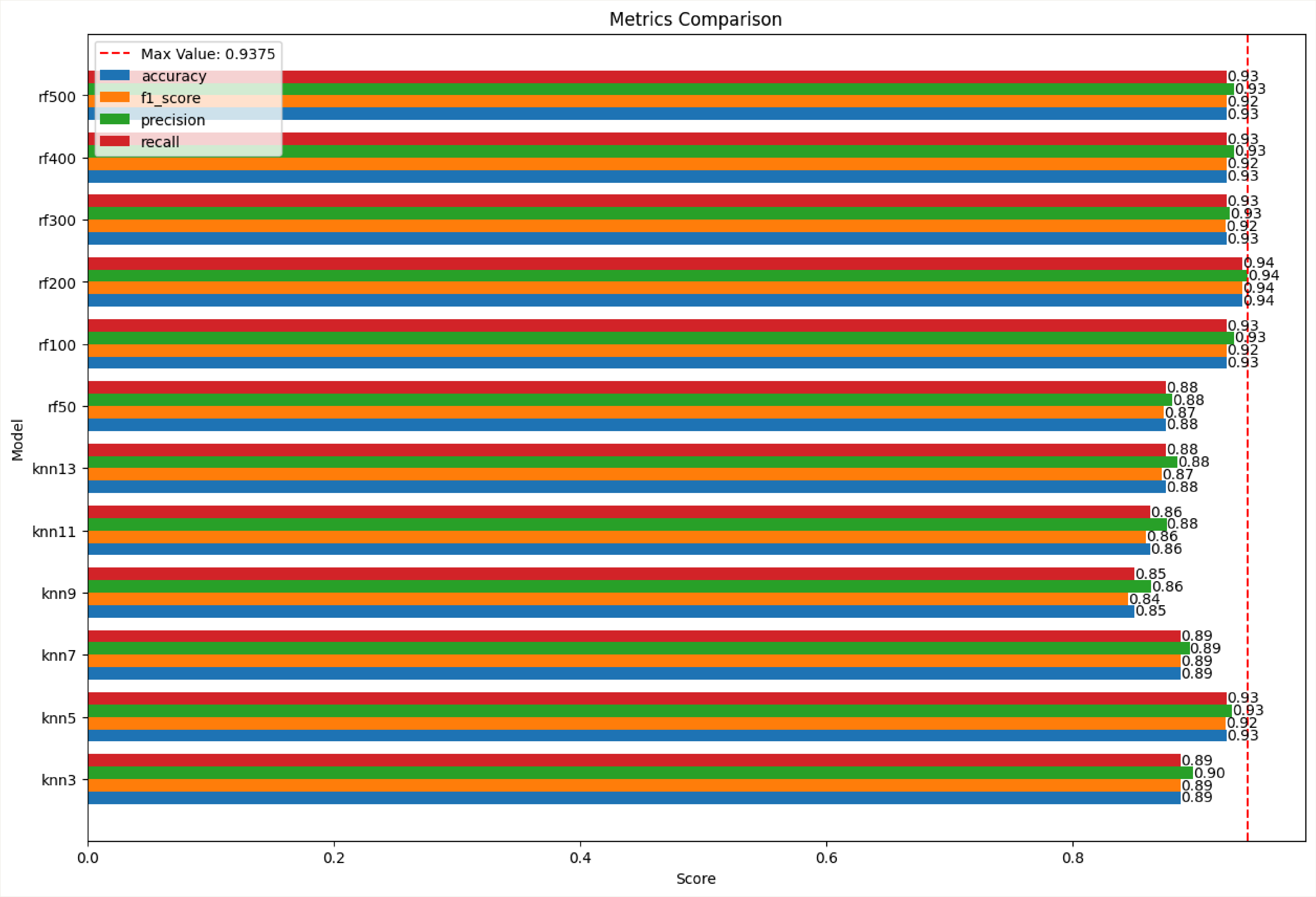
$$\frac{2\,TP}{2\,TP + FP + FN}$$

MODELS TRAINING

Next we trained K-Nearest Neighbours and Random Forests models from sklearn library with different parameters on the gesture normalized data and not normalized data, using training and validation sets

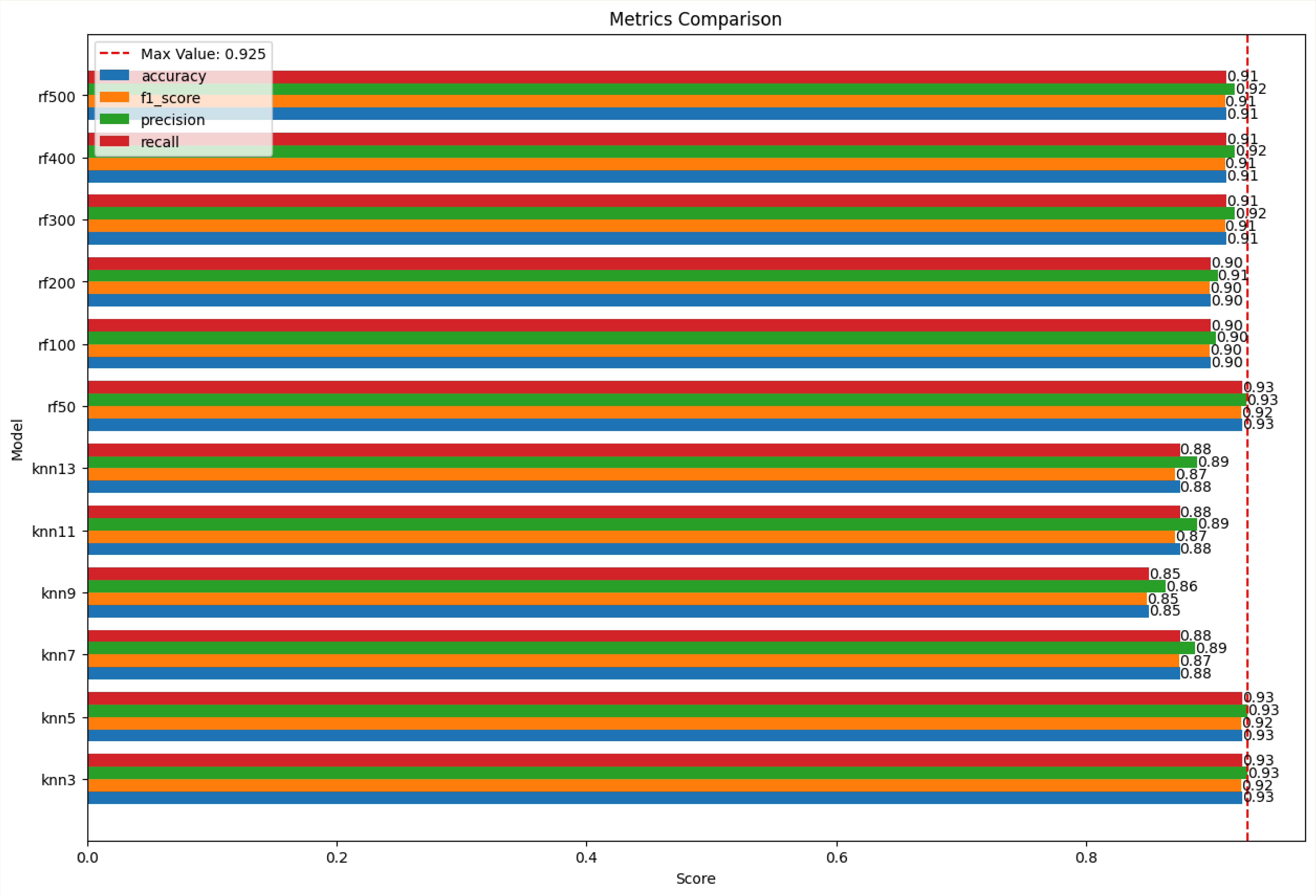
```
{"knn3": KNeighborsClassifier(n_neighbors=3),
 "knn5": KNeighborsClassifier(n_neighbors=5),
 "knn7": KNeighborsClassifier(n_neighbors=7),
 "knn9": KNeighborsClassifier(n_neighbors=9),
 "knn11": KNeighborsClassifier(n_neighbors=11),
 "knn13": KNeighborsClassifier(n_neighbors=13),
 "rf50": RandomForestClassifier(n_estimators=50),
 "rf100": RandomForestClassifier(n_estimators=100),
 "rf200": RandomForestClassifier(n_estimators=200),
 "rf300": RandomForestClassifier(n_estimators=300),
 "rf400": RandomForestClassifier(n_estimators=400),
 "rf500": RandomForestClassifier(n_estimators=500)}
```

MODELS ON RAW DATA




```
best not normalized model on test data
Model: rf200
accuracy: 0.8583333333333333
f1_score: 0.8538676377386054
precision: 0.8593519327894329
recall: 0.8583333333333333
confusion_matrix:
[[14  0  0  0  0  1  0  0]
 [ 0 14  0  0  0  0  0  1]
 [ 0  0 12  0  0  2  1  0]
 [ 0  0  1 12  1  0  0  1]
 [ 0  0  0  0 15  0  0  0]
 [ 2  0  0  0  2  8  0  3]
 [ 0  0  0  0  0  0 15  0]
 [ 0  1  0  1  0  0  0 13]]
```

MODELS ON NORMALIZED DATA




```
best model on normalized test data
Model: rf50
accuracy: 0.8416666666666667
f1_score: 0.8382819737502023
precision: 0.8433608772763184
recall: 0.8416666666666667
confusion_matrix:
[[15  0  0  0  0  0  0  0]
 [ 0 14  0  0  0  0  0  1]
 [ 1  0 11  0  0  2  1  0]
 [ 0  0  1 12  1  0  0  1]
 [ 0  0  1  1 13  0  0  0]
 [ 1  0  0  0  3  9  0  2]
 [ 0  0  0  0  0  0 15  0]
 [ 0  1  0  1  1  0  0 12]]
```

FEATURE ENGINEERING (SLIDING WINDOW)

- We used a sliding window statistics extraction technique, dividing the time series into windows of size 20 with a step of 5. For each window, we computed features like the median, mean, standard deviation, variance, minimum, and maximum, summarizing the data in compact, feature-rich representations.

```
train data windows shape: (240, 3, 60, 6)
validation data windows shape: (80, 3, 60, 6)
test data windows shape: (120, 3, 60, 6)
```

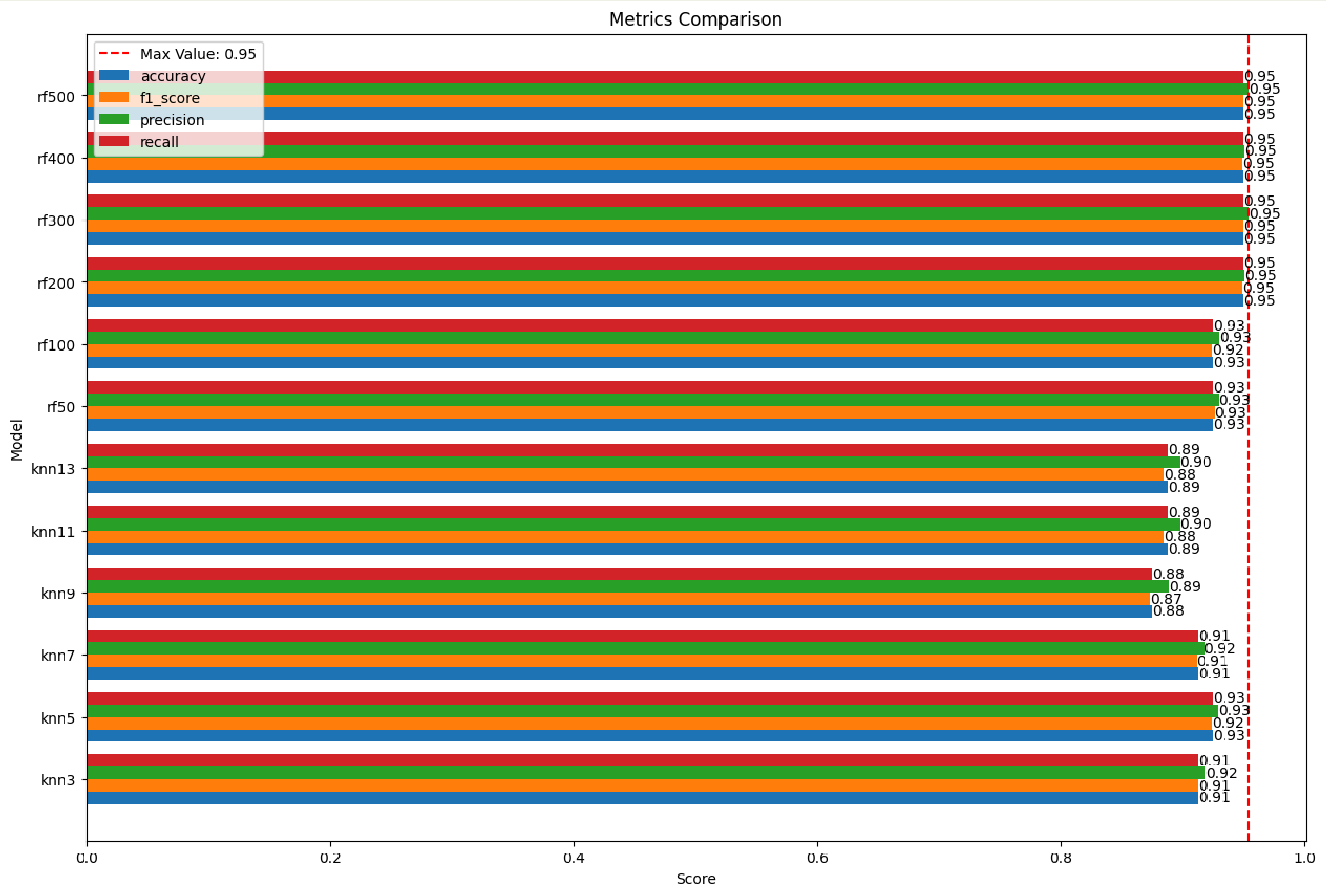
(OBSERVATIONS, DIMENSIONS, NEW "TIME STEPS", STATISTICS)

(DATA PREPARED FOR MODEL TRAINING)

```
sliding window train data, label shape: (240, 1080) , (240,)
sliding window validation data, label shape: (80, 1080) , (80,)
sliding window test data, label shape: (120, 1080) , (120,)

sliding window normalized train data, label shape: (240, 1080) , (240,)
sliding window normalized validation data, label shape: (80, 1080) , (80,)
sliding window normalized test data, label shape: (120, 1080) , (120,)
```

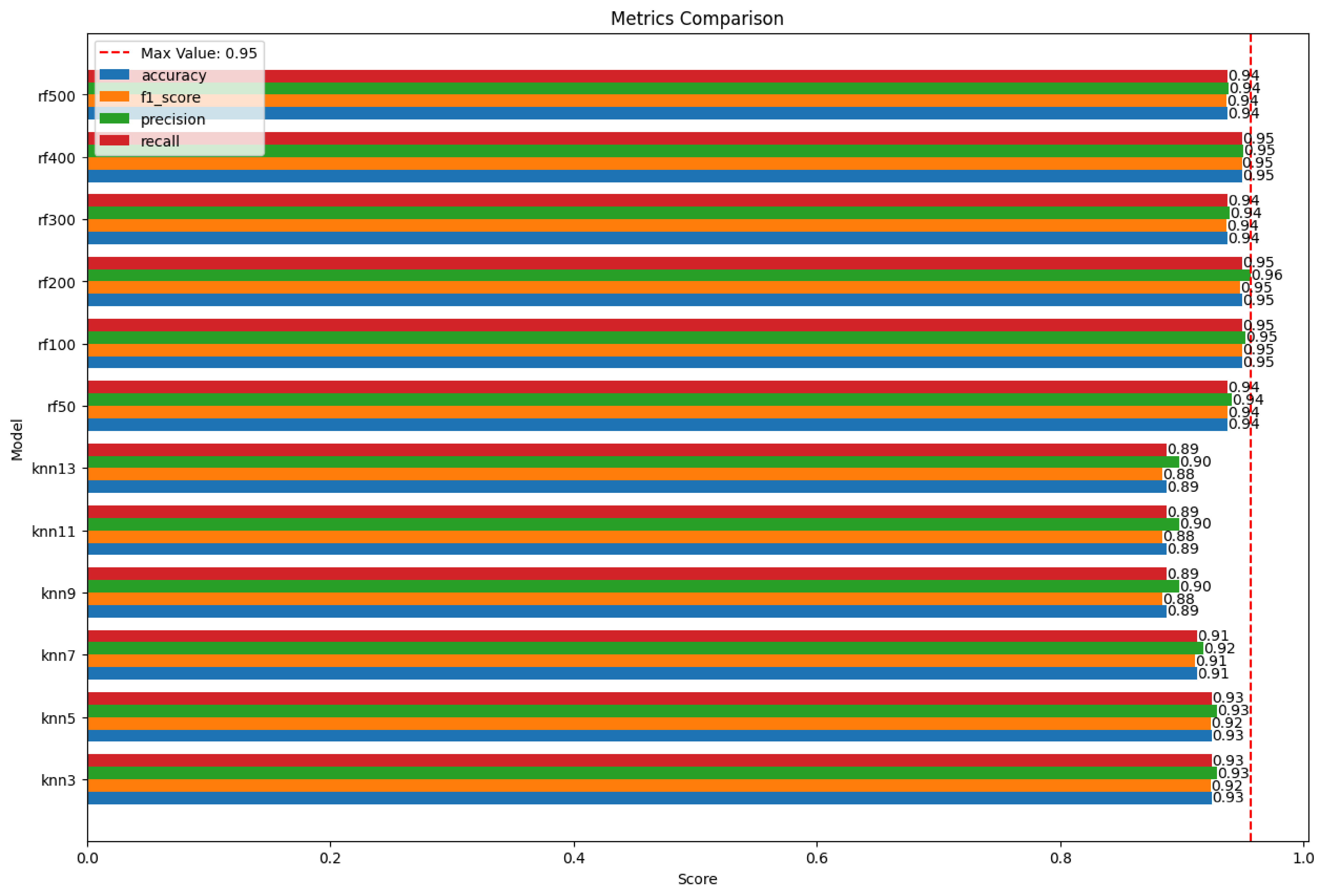
FEATURE ENGINEERING (SLIDING WINDOW)




```
best sliding window model on test data
Model: rf500
accuracy: 0.875
f1_score: 0.8729372080022109
precision: 0.8774396457300869
recall: 0.875
confusion_matrix:
[[15  0  0  0  0  0  0  0]
 [ 0 14  0  0  0  0  0  1]
 [ 1  0 12  0  0  2  0  0]
 [ 0  0  1 12  1  0  0  1]
 [ 0  0  0  1 14  0  0  0]
 [ 1  0  0  0  3 10  0  1]
 [ 0  0  0  0  0  0 15  0]
 [ 0  1  0  1  0  0  0 13]]
```

FEATURE ENGINEERING (SLIDING WINDOW NM)

- Same approach but on normalized data




```
best sliding window model
on normalized test data
Model: rf400
accuracy: 0.875
f1_score: 0.8740521575198995
precision: 0.877230235042735
recall: 0.875
confusion_matrix:
[[15  0  0  0  0  0  0  0]
 [ 0 14  0  0  0  0  0  1]
 [ 0  0 12  0  0  2  1  0]
 [ 0  0  1 12  1  0  0  1]
 [ 0  0  0  1 14  0  0  0]
 [ 0  0  0  0  3 11  0  1]
 [ 0  0  0  0  0  0 15  0]
 [ 0  1  0  2  0  0  0 12]]
```

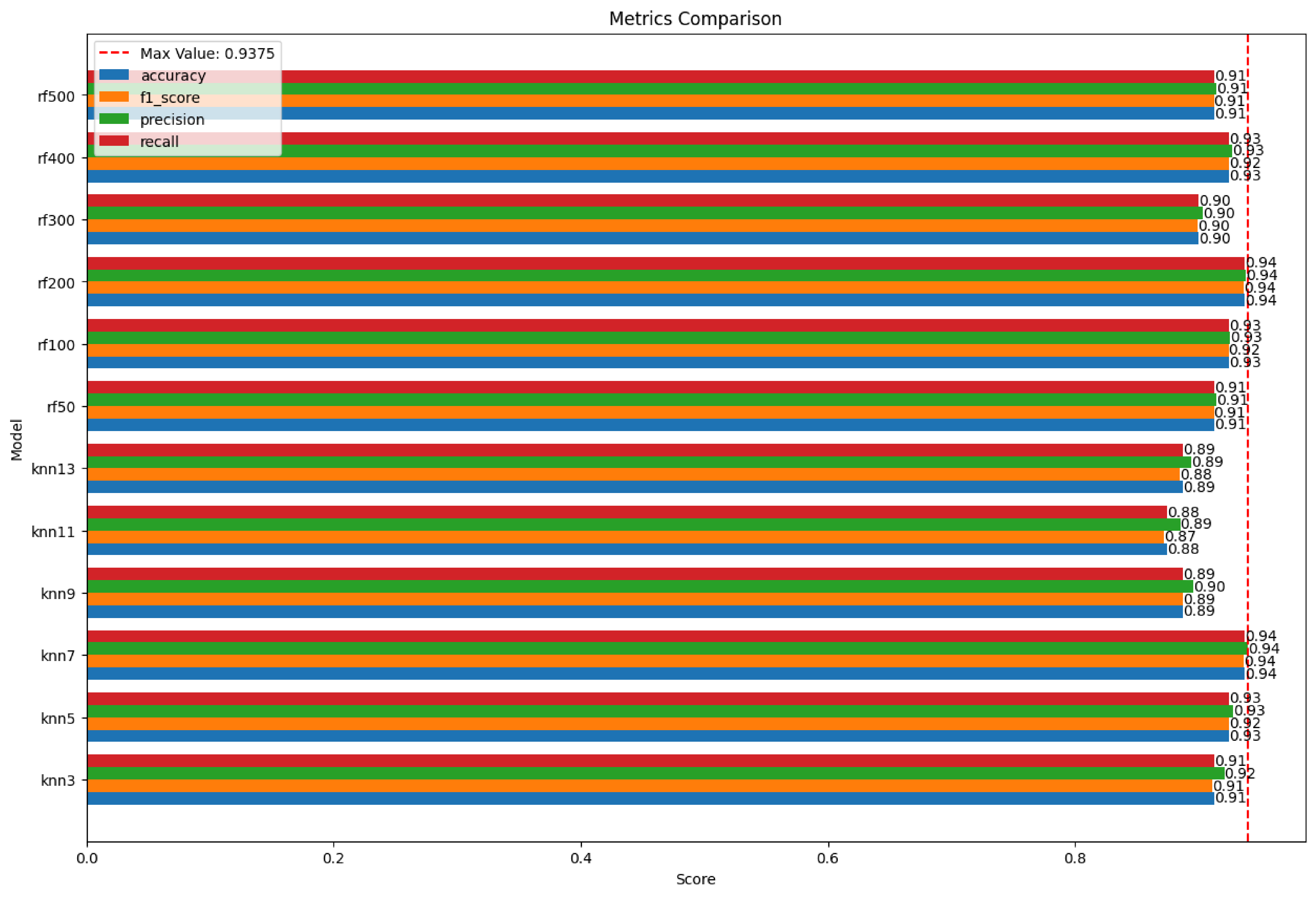
FEATURE ENGINEERING (SESONAL DECOMPOSE)

- The technique we used is seasonal decomposition of time-series data, where each part of the data (x, y, z) is decomposed into trend and seasonal components using an additive model, excluding the residual component. The period for decomposition is set to 39 (1/8 of the length of the data), and the decomposed seasonal and trend components are concatenated for each time series, providing a feature-rich representation that captures both regular patterns and underlying trends in the data

```
seasonal train data, label shape: (240, 1890) , (240,)
seasonal validation data, label shape: (80, 1890) , (80,)
seasonal test data, label shape: (120, 1890) , (120,)

seasonal normalized train data, label shape: (240, 1890) , (240,)
seasonal normalized validation data, label shape: (80, 1890) , (80,)
seasonal normalized test data, label shape: (120, 1890) , (120,)
```

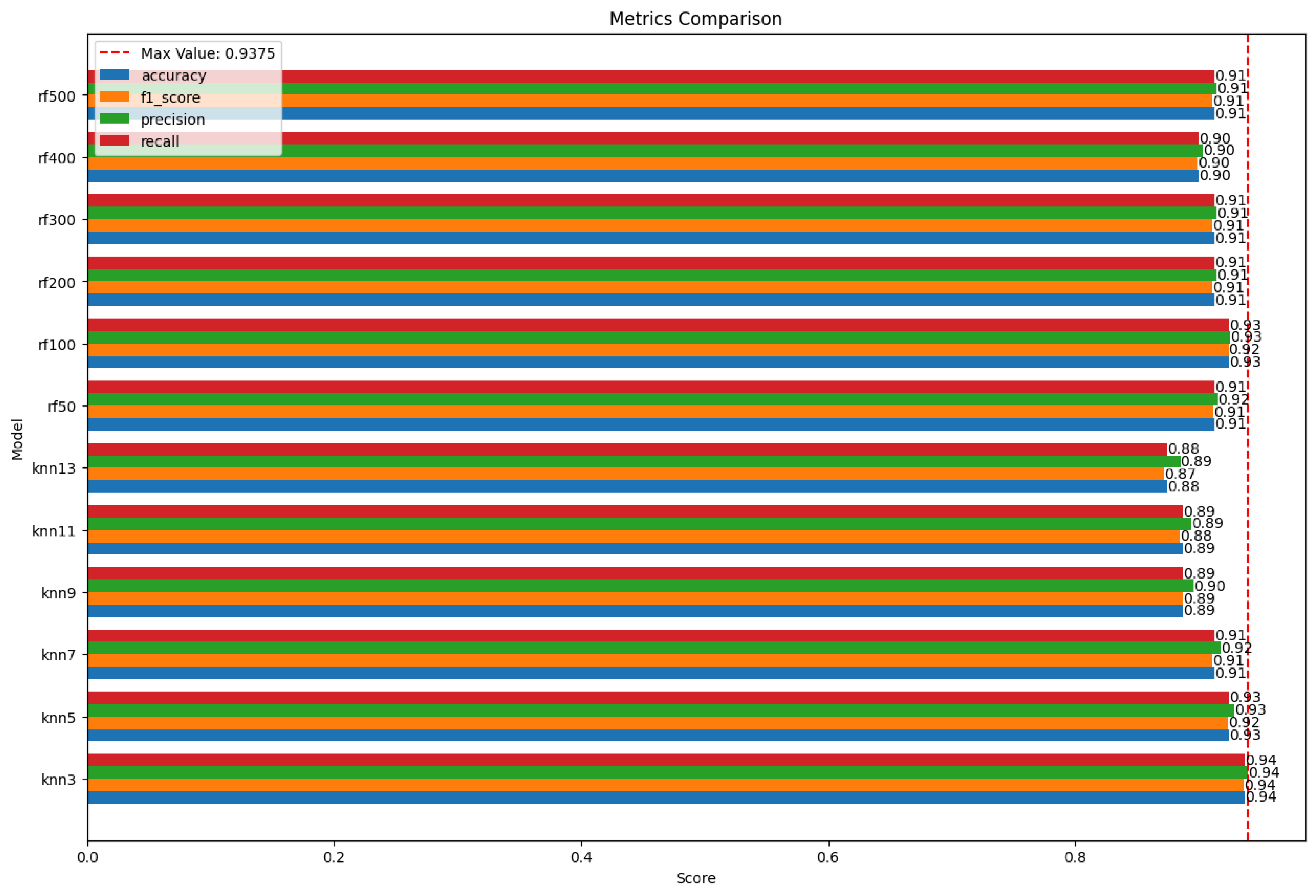
FEATURE ENGINEERING (SESONAL DECOMPOSE)




```
best seasonal decomposed model on test data
Model: rf200
accuracy: 0.8583333333333333
f1_score: 0.8575374755018804
precision: 0.8585794413919413
recall: 0.8583333333333333
confusion_matrix:
[[14  0  0  0  0  1  0  0]
 [ 0 14  0  0  0  0  0  1]
 [ 0  0 12  0  0  2  1  0]
 [ 0  0  1 13  0  0  0  1]
 [ 0  0  2  0 13  0  0  0]
 [ 1  0  1  1  0 10  0  2]
 [ 0  0  0  0  0  0 15  0]
 [ 0  1  0  1  1  0  0 12]]
```

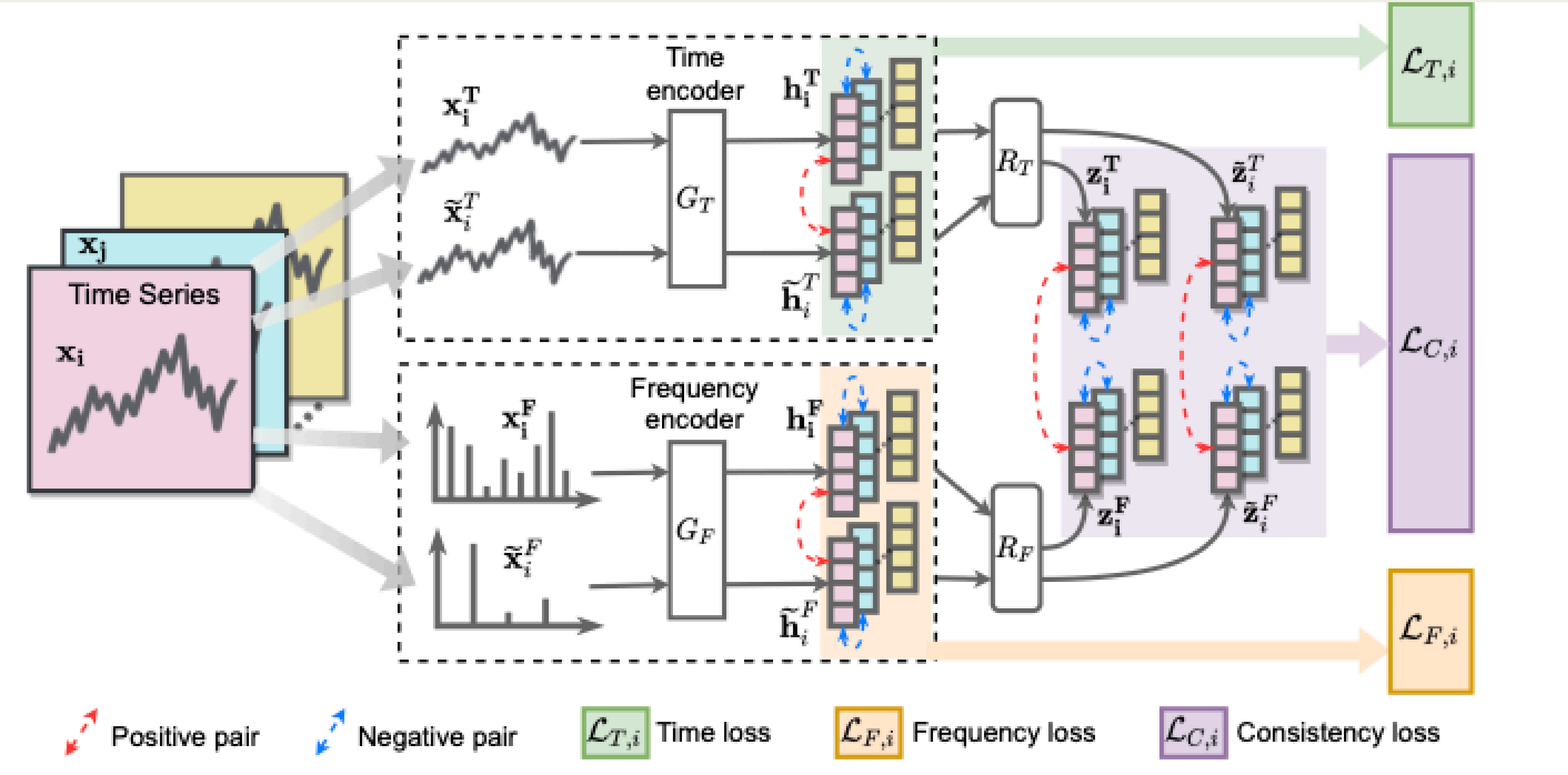
FEATURE ENGINEERING (SESONAL DECOMPOSE NM)

- Same approach but on normalized data




```
best seasonal decomposed model
on normalized test data
Model: knn3
accuracy: 0.9166666666666666
f1_score: 0.9162372290406708
precision: 0.9216409412955465
recall: 0.9166666666666666
confusion_matrix:
[[14  0  0  0  0  0  1  0]
 [ 0 15  0  0  0  0  0  0]
 [ 0  0 13  0  0  2  0  0]
 [ 0  0  0 13  2  0  0  0]
 [ 0  0  0  0 15  0  0  0]
 [ 0  0  0  2  2 11  0  0]
 [ 0  0  0  0  0  0 15  0]
 [ 1  0  0  0  0  0  0 14]]
```

TF-C (TIME-FREQUENCY CONSITENCY)



TF-C MODEL

- We downloaded the model from author's github and pretrained it on the HAR dataset. After pretraining we used gesture dataset for fine-tuning and calculating overall performance of the model. During fine-tuning we had to change some parts of the code causing errors and also switch some of the hyperparameters as the achieved results were not satisfying.
- Our results:

```
################## Best testing performance! #########################
Best Testing Performance: Acc=61.6667| Precision = 58.6642 | Recall = 61.6667 | F1 = 56.3479 | AUROC= 87.0239 | AUPRC=66.8749
```

- TF-C Paper results:

Table 1: One-to-one pre-training evaluation (Scenario 3). Pre-training is performed on HAR, followed by fine-tuning on GESTURE. Results for other three scenarios are shown in Tables 4-6.

Models	Accuracy	Precision	Recall	F1 score	AUROC	AUPRC
TF-C	**0.7824**$\pm$**0.0237**	**0.7982**$\pm$**0.0496**	**0.8011**$\pm$**0.0322**	**0.7991**$\pm$**0.0296**	0.9052$\pm$0.0136	**0.7861**$\pm$**0.0149**

CONCLUSIONS

1) Model Performance:

- Random Forests performed worse on the test set, potentially due to overfitting. High tree depth or insufficient regularization may have caused the model to memorize the training data, reducing its ability to generalize.
- The TF-C neural network underperformed, potentially due to implementation issues, suboptimal hyperparameters, or the limited size of the gesture dataset.

2) Feature Engineering Approaches:

- Sliding Window: Extracting statistical features (mean, median, standard deviation, etc.) provided compact representations that improved model interpretability.
- Seasonal Decomposition: Capturing trends and seasonal patterns enriched the dataset, enabling better recognition of gestures.
- Normalization: Improved results for certain models, emphasizing the importance of scaling features.

3) Evaluation Metrics:

- The use of accuracy, precision, recall, and F1-score offered a comprehensive evaluation, particularly valuable for imbalanced class distributions.

4) General Conclusion:

- Simple machine learning models, when paired with appropriate feature engineering, regularization techniques, and methodological tuning, can achieve highly satisfactory results for certain problems, demonstrating their value as robust and efficient solutions.

SOURCES

- *TF-C Paper: https://openreview.net/pdf?id=OJ4mMfGKLN*
- *Github Repo: https://github.com/mims-harvard/TFC-pretraining/tree/main*

Thank you!