Project 1

A. Burak Gulhan

February 22, 2023

Contents

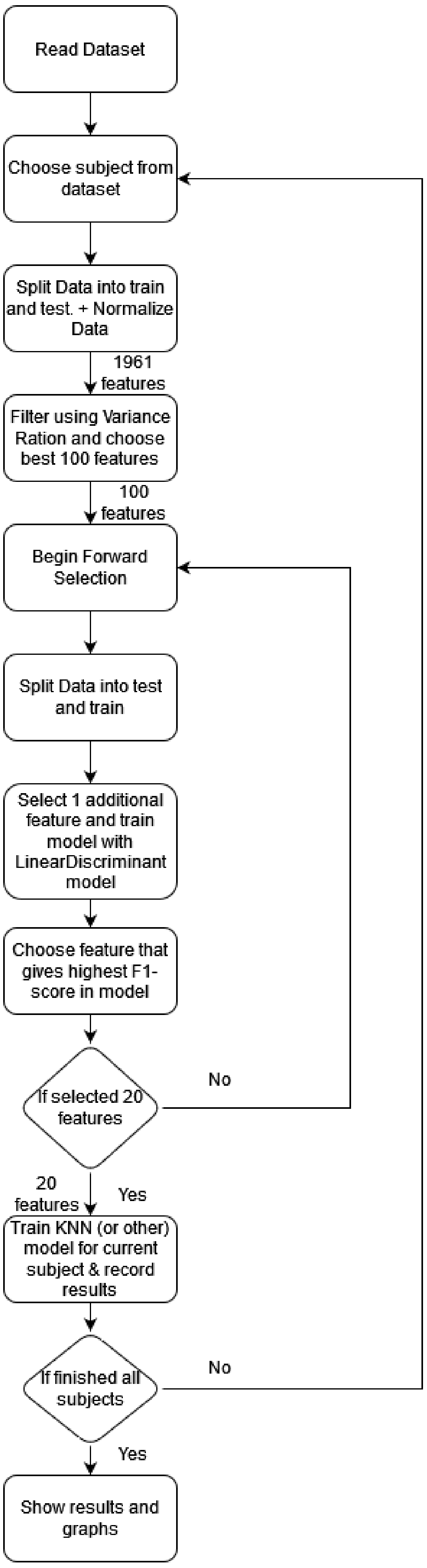
1 Introduction and Flowchart

In this project I implemented a filtering and wrapper method to perform feature selection. First the data is normalized using global normalization (I did several other normalization methods, and the default global normalization gave the best results). For the filtering step, I used **variance ratio** to score and select the best 100 features. The main reason for selecting variance ratio compared to other scoring methods was that it was the simplest to code and implement. For the wrapper, I implemented forward selection. I started with an empty set and split the input data into 2/3rd train and 1/3 test with the split done in a way so that classes are balanced (using the stratify parameter in numpy.test_train_split()). The data split was repeated for each feature. For the current feature set, I added 1 unselected feature, and trained the model with **LinearDiscriminantAnalysis** since this was the fastest method I found (I also used different models, which is explained in the extra credit section). The performance of the model was evaluated using **F1-score** as the criterion, which I chose because it can be used to maximize both precision and recall. Since there are multiple classes, the F1-score can be computed in different ways. I used the 'micro' averaging method to compute F1-score, since it gave the best results (more details on this in extra credit section).

After performing feature reduction, the model is trained using **KNN** (performance of other models shown extra credit section).

The flowchart of the program can be seen in figure 1.

NOTE: Summary and comparisons are done in both section 3 and in Extra credit section 5 (for extra credit related results).


Read Dataset
Choose subject from dataset
Split Data into train and test. + Normalize Data
1961 features
Filter using Variance Ration and choose best 100 features
100 features
Begin Forward Selection
Split Data into test and train
Select 1 additional feature and train model with LinearDiscriminant model
Choose feature that gives highest F1-score in model
If selected 20 features
No
20 features
Yes
Train KNN (or other) model for current subject & record results
If finished all subjects
No
Yes
Show results and graphs

2 Visualization

The results obtained for these figures were by using the models and criteria described in Section 1. These results and figures will be compared and summarized in Section 3.

2.1 Histogram

Figure 2 shows a histogram of the 20 most discriminative features from the filter and Figure 3 shows a histogram of the 20 most commonly chosen features from the wrapper.

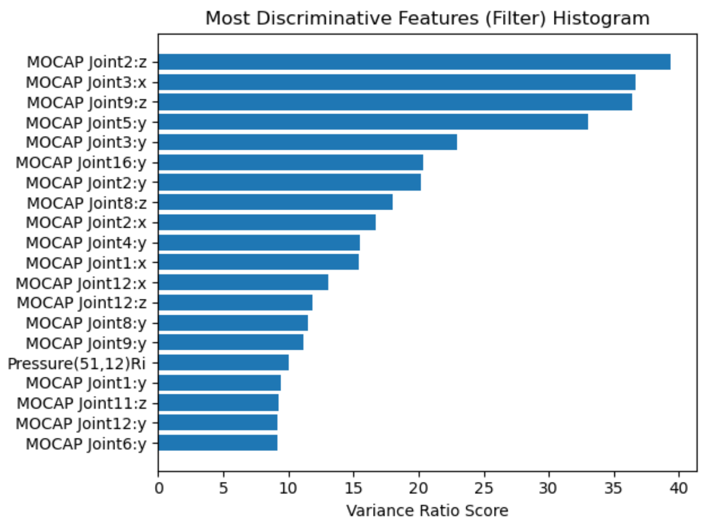


Figure 2: Histogram for filter (using filter and wrapper)

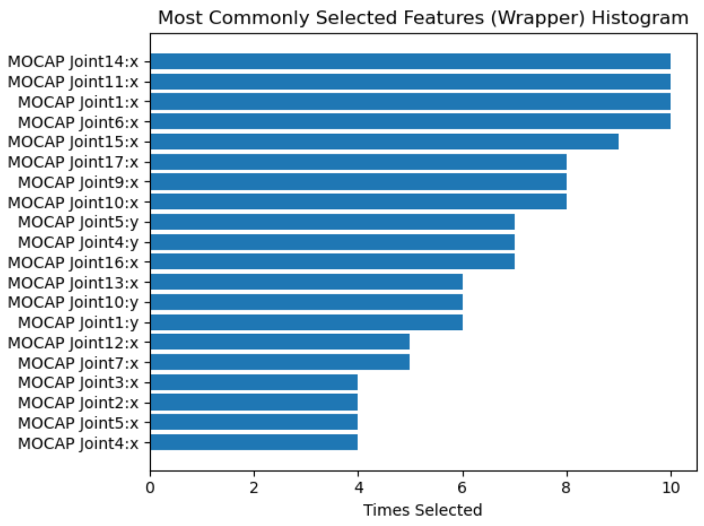


Figure 3: Histogram for wrapper (using filter and wrapper)

2.2 Results

Training and testing accuracy and standard deviation (across subject accuracies) is shown in table 1 and 2 for the methods below.

2.2.1 Using both filter and wrapper

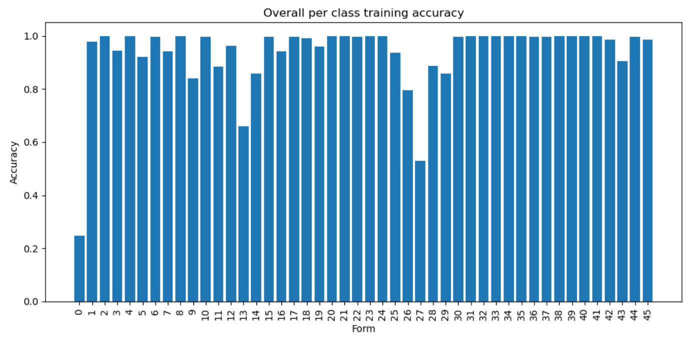
Figure 4 shows the the train test accuracy per subject. Figure 7 shows the confusion matrices for train and test. Figure 10 shows the subject-wise train and test accuracies.

2.2.2 Using filter but no wrapper

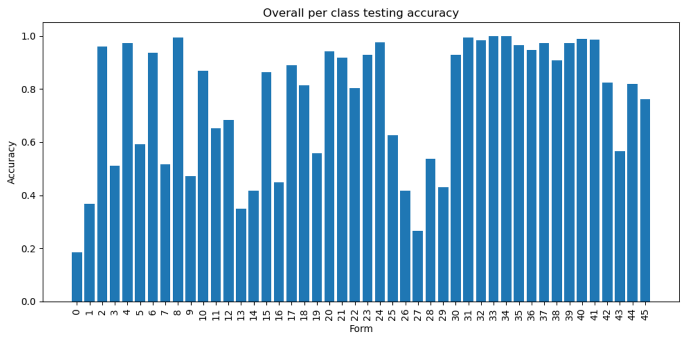
Figure 5 shows the the train and test accuracy per subject. Figure 8 shows the confusion matrices for train and test. Figure 11 shows the subject-wise train and test accuracies.

2.2.3 Using no filter and no wrapper

Figure 6 shows the the train and test accuracy per subject. Figure 9 shows the confusion matrices for train and test. Figure 12 shows the subject-wise train and test accuracies.

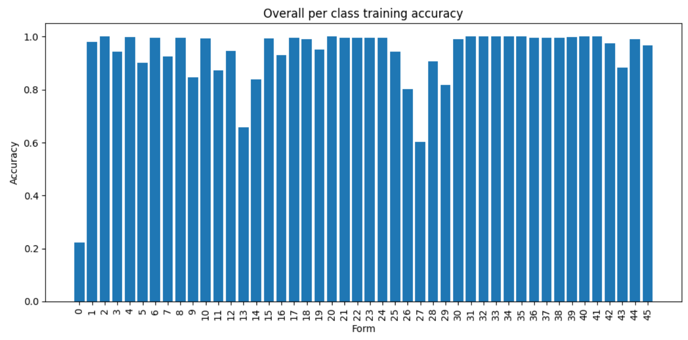


(a) train

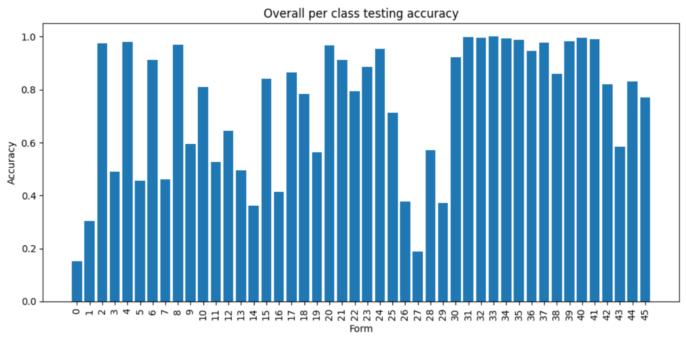


(b) test

Figure 4: Overall per class accuracy (using filter and wrapper)

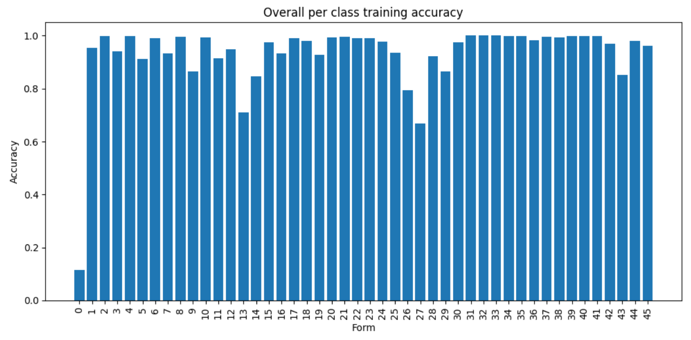


(a) train

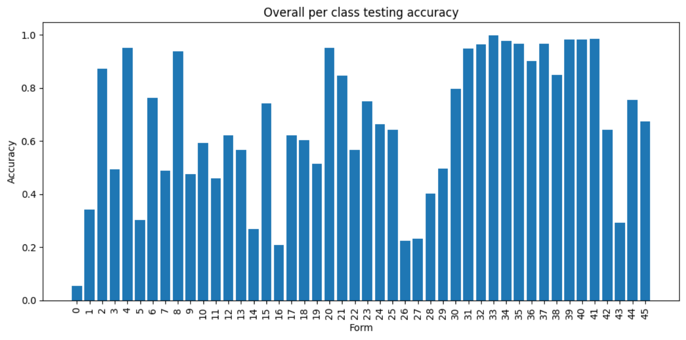


(b) test

Figure 5: Overall per class accuracy (using filter and no wrapper)

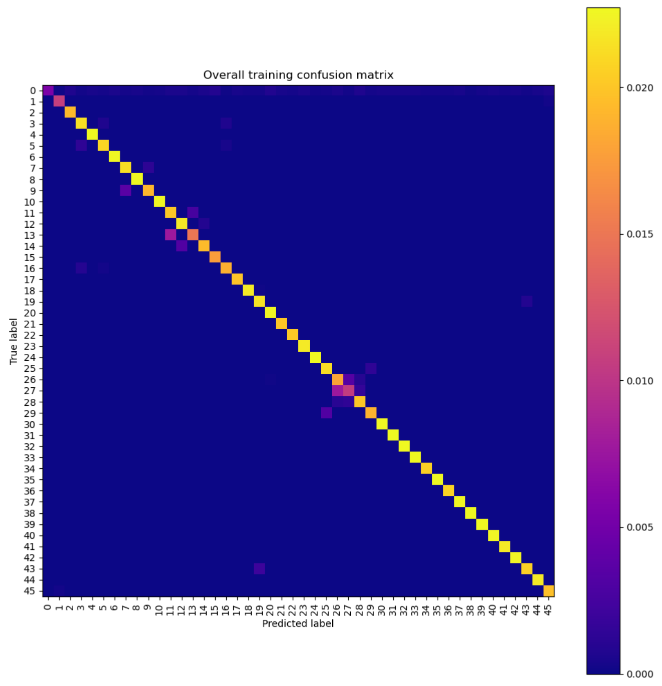


(a) train

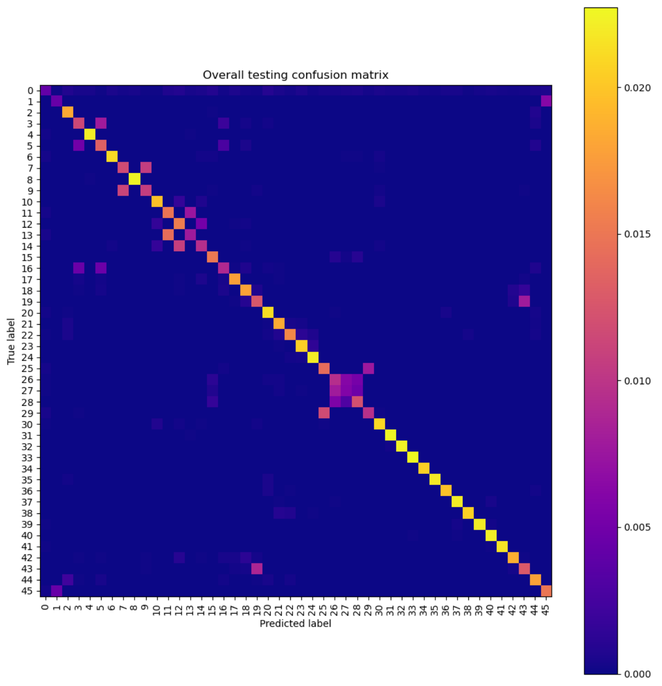


(b) test

Figure 6: Overall per class accuracy (no filter and no wrapper)



(a) Overall train confusion matrix



(b) Overall test confusion matrix

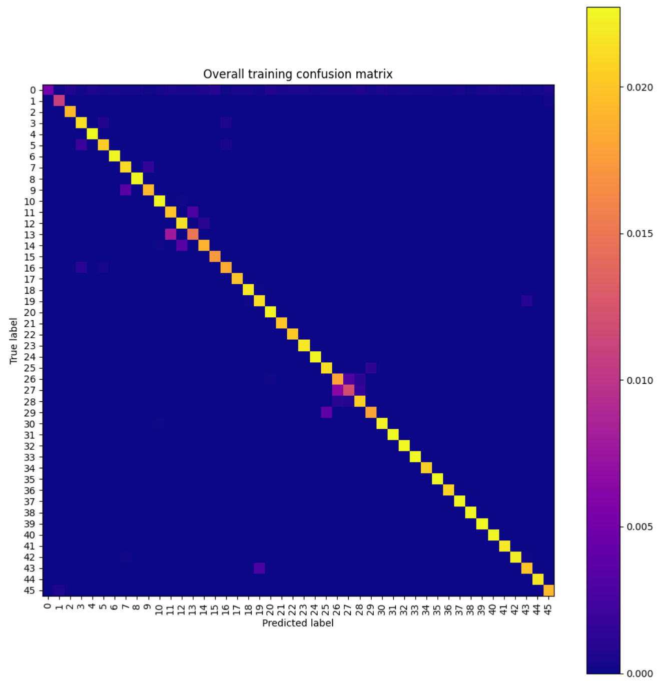
Figure 7: Confusion matrices (using filter and wrapper)

Table 1: Training accuracy and std

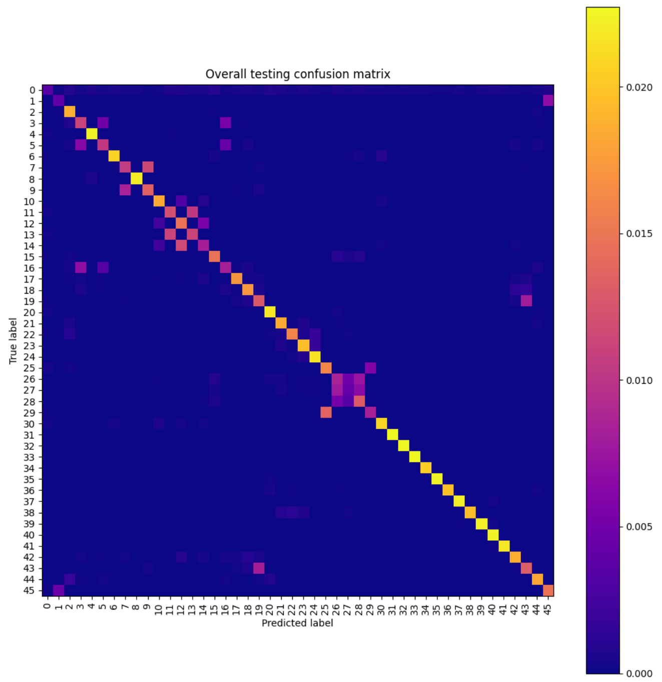
Method	Accuracy	Std
Filter+Wrapper	0.9329	0.0028
Filter+No Wrapper	0.9298	0.0009
No Filter+No Wrapper	0.9286	0.0012

Table 2: Testing accuracy and std

Method	Accuracy	Std
Filter+Wrapper	0.7541	0.0267
Filter+No Wrapper	0.7437	0.0252
No Filter+No Wrapper	0.6618	0.0418

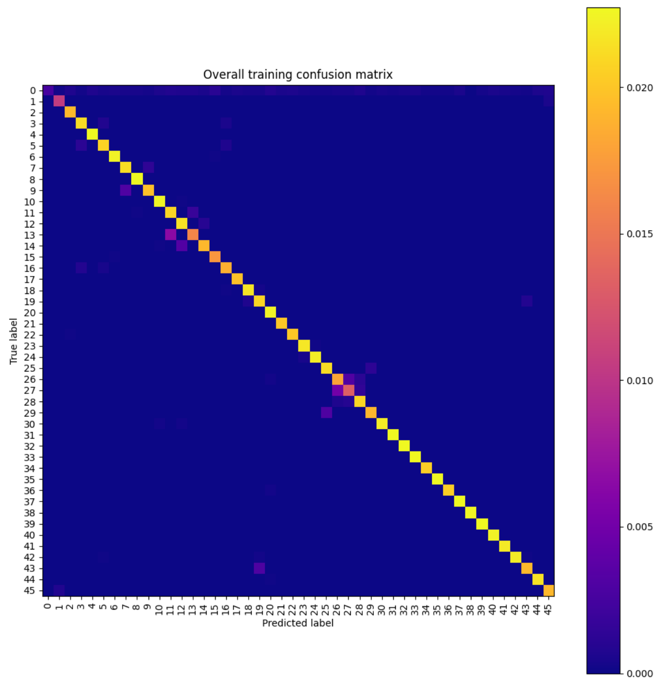


(a) Overall train confusion matrix

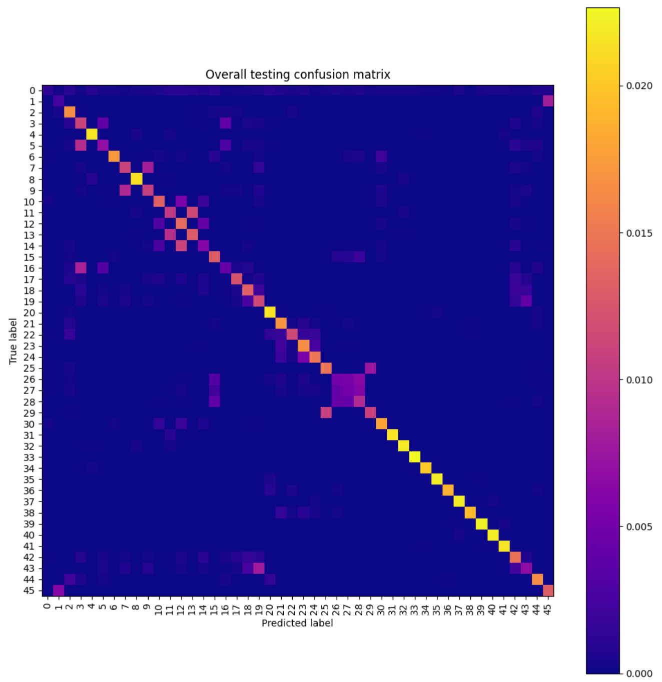


(b) Overall test confusion matrix

Figure 8: Confusion matrices (using filter and no wrapper)



(a) Overall train confusion matrix



(b) Overall test confusion matrix

Figure 9: Confusion matrices (no filter and no wrapper)

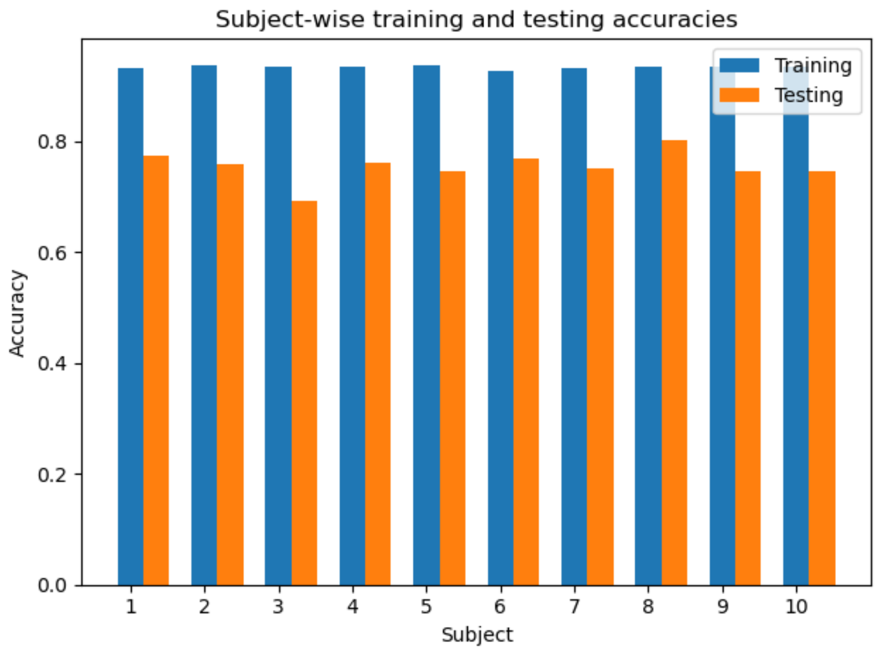


Figure 10: Subjectwise Accuracy (using filter and wrapper)

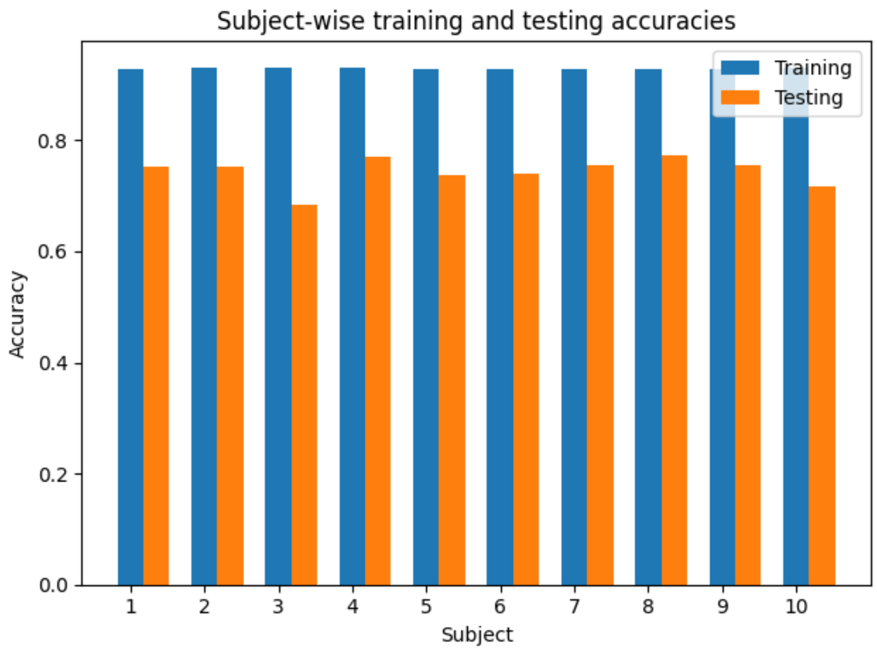


Figure 11: Subjectwise Accuracy (using filter and no wrapper)

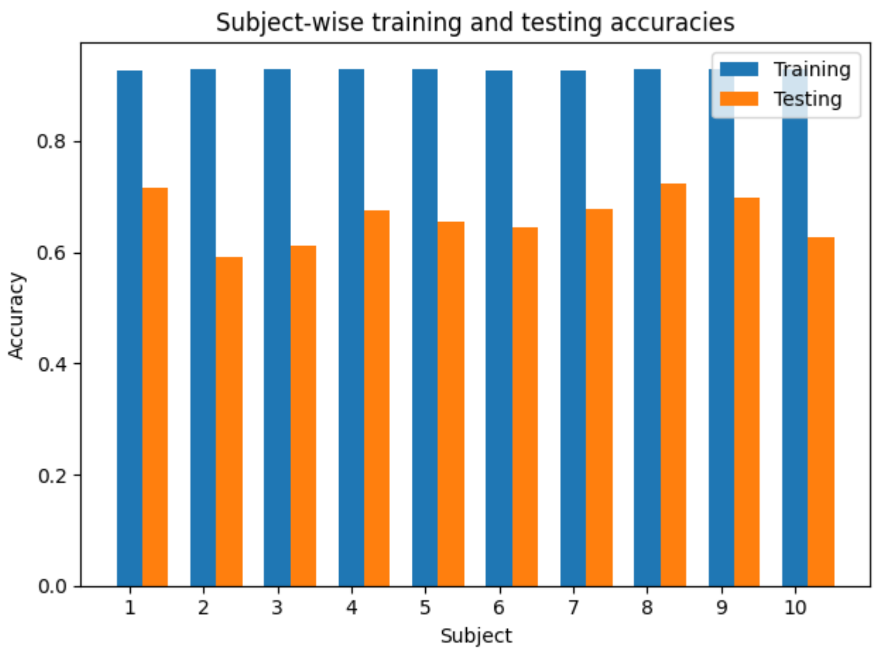


Figure 12: Subjectwise Accuracy (no filter and no wrapper)

3 Summary and Comparison of Results

NOTE: In the extra credit section 5, there are additional summaries and comparison of results obtained in the extra credit experiments.

From the results we see that using a filter and wrapper gives the best results, while using no filter and no wrapper gives the worse results. It is interesting that filtering gives very good improvement from using no filter and no wrapper (around 0.08 improvement in test accuracy) compared to using a filter and wrapper which gives a very small improvement (less than 0.01 increase in testing accuracy), for a very large increase in runtime.

From the histograms, we can see in Figure 2 that joint2:z, joint3:x, joint9:z and joint5:y are the most discriminative features by a large margin, yet in the most commonly selected features from the wrapper, in Figure 3 we see that half of these are not even selected by the model (only joint3:x and joint5:y are selected). This may be due to using a simple selection criterion (Variance Ratio) for filtering. Maybe using a more sophisticated criterion may give better results.

Another interesting observation from the histograms is that in Figure 3 we can see that only x and y coordinates are selected for the final 20 features, which makes sense, since the z coordinate shouldn't change much (I assume that in Taiji there is not much vertical movement). Furthermore, we can observe that foot pressure data is not used at all in the final 20 features. The reason for this is that I think that foot pressure data is redundant compared to the coordinate features (we can use coordinates to calculate center of gravity, which can be used to guess foot pressure).

From the confusion matrices (Figures 7, 8 and 9) and overall per class accuracy (Figures 4, 5 and 6) we can see that features 3 to 5, 7 to 9, 10 to 14 and 25 to 29 have less accuracy compared to others (for all 3 methods). We see that minor errors in the training are much more exaggerated in the testing, this is particularly easy to see in the confusion matrices - faint dots that are not in the diagonal of the train confusion matrix are much more brighter in the test confusion matrix.

In the Subject-wise accuracy figures (4, 5 and 6) we see that there is not too much of a difference across subject accuracies for a given method. The most interesting thing I could observe was that subjects have varying amounts of improvement from having filter and wrapper applied. For example, subject 2 had the worst accuracy in no filter and no wrapper (figure 6) but it is has one of the highest accuracies from having filter and wrapper applied (figure 4). However, we can also see that subject 3, which had the second lowest accuracy with no filter and no wrapper, has the lowest accuracy with using both filter and wrapper – subject 3 had much less improvement compared to subject 2, despite having similar accuracies in the no filter and no wrapper method.

4 Data Size Sufficiency Question

Question: Justify that this Taiji data set is or is not sufficiently large for this classification task.

Answer:

One way to justify that a dataset is sufficient is to use that data to train a model and check if the results are good. If we do get good results, then we can say that the dataset if sufficient for classification. If we get a bad result, then we cannot say for certain by just looking at the result whether the data is sufficient or not (since our chosen model may be the problem).

Since the classifier gives good results for the Taiji data set I would say that this data set is sufficiently large.

Another way to check if a data set is large enough is to use the rule of thumb where it states we need around 5 data points per feature `https://en.wikipedia.org/wiki/Curse_of_dimensionality`. We have 1961 features in the Taiji dataset and 49774 data points, which seems to satisfy the rule of thumb, but I think that that rule wouldn't be suitible to apply for the Taiji dataset for two reasons.

First, the dimensions (features) in the data set are not independent of each other. We can easily see this in pressure data, where pressure in one location means that adjacent points also have pressure. Also in project 1 we did fisher projection, which reduces the data dimension down to 64 features, which also supports that the features are dependent on each other.

Additionally, the number of samples may not be indicative of whether we have enough data points, since this depends on how fast we sample data from the sensors. The faster we sample, the more points we obtain, but the difference between subsequent points would be very small.

5 Extra Credit: Obtaining Better Classification Results

5.1 Adjusting F1-Score in Wrapper

For scoring models in the wrapper, I chose F1-Score since it uses both precision and recall, thus maximizing F1-Score would maximize both those metrics. However, since this is a multi class classification, there are different ways of computing F1-score. In sklearn's f1_score() function there is a parameter "average" that can be set to micro, macro or weighted (`https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html`). Micro calculates F1-score globally, by taking into account the total number of true/false positive/negatives; macro calculates F1-score per label and gets their mean; weighted is similar to macro, but the weighted mean is taken to account for label imbalance.

I tested these 3 averaging methods for my wrapper. I used the method described in 1 for my model, wrapper and filter. The accuracy results I obtained, shown in table 3 show that micro averaging gives the best result for testing accuracy. This makes sense, since when calculating accuracy, we check the total number of true/false positive/negatives; we don't calculate them for each label nor do we do any weighting.

Table 3: F1-score accuracy and std

Averaging Method	Train Acc	Test Acc
micro	0.9274	**0.7395**
macro	0.9288	0.7315
weighted	0.9272	0.7358

5.2 Normalization

I also tried several normalization methods to see if those gave better results. I thought that the global normalization used in the starter code was not the best method, since we are getting the max and min values from the entire dataset for normalization. However, there are two types of data in the dataset, coordinate data and pressure data. Coordinate data ranges from around -1000 to 1000 and are floats, but pressure data contains only non negative integers. The max and min chooses for the data are all from coordinate features (min=-885.398 and max=999.789). Meaning that we are normalizing the pressure data with values from coordinate data.

The first thing I tried, was to use column wise normalization, so that the min and max values used in normalization are selected for each feature and not across different feature types. However this gave worse results even compared to using no normalization. This is probably due to losing scaling information across features (for example a higher pressure may be significant in classification, but with column-wise normalization we cannot tell the pressure different between different pressure locations).

I tried several other methods such as global normalization for coordinate and pressure data separately; column-wise normalization for coordinates and global for pressure; and column-wise normalization for coordinates and no normalization for pressure. These train/testing accuracy results are shown in table 4.

All of the results I obtained show that the original global normalization performs the best (I used the method described in 1 and used F1-score with micro averaging for the wrapper). I was expecting that global normalization for coordinate and pressure features separately would give the best result, but maybe doing this gives some bias to one of the feature types since the scaling factor is different. Another normalization method I was planning on doing but didn't have enough time was to do global normalization for x, y, z coordinates and pressure separately, since these would all have different ranges (eg. z axis can only increase so much do to a person's height vs x and y being constrained by the size of the room).

Table 4: Normalization methods

Normalization Method	Train Acc	Test Acc
Global norm (default)	0.9330	**0.7522**
No norm	0.9331	0.7517
column-wise norm	0.9274	0.7395
column-wise only for coord	0.9228	0.6728
column-wise for coord+global for pressure	0.9271	0.7373
global for coord and pressure separately	0.9323	0.7485

5.3 Ensemble Learning (Random Forest) vs KNN

All of the results previously were obtained by using KNN as the model for each subject. I tried using Random Forest as the model and obtained better results. An interesting observation is that Random Forest has very high overfitting (perfect training results), yet it still obtains better testing accuracy compared to KNN. The accuracy and standard deviation results for Random Forest are shown in table 5 and the confusion matrices are shown in Figures 13a and 13b.

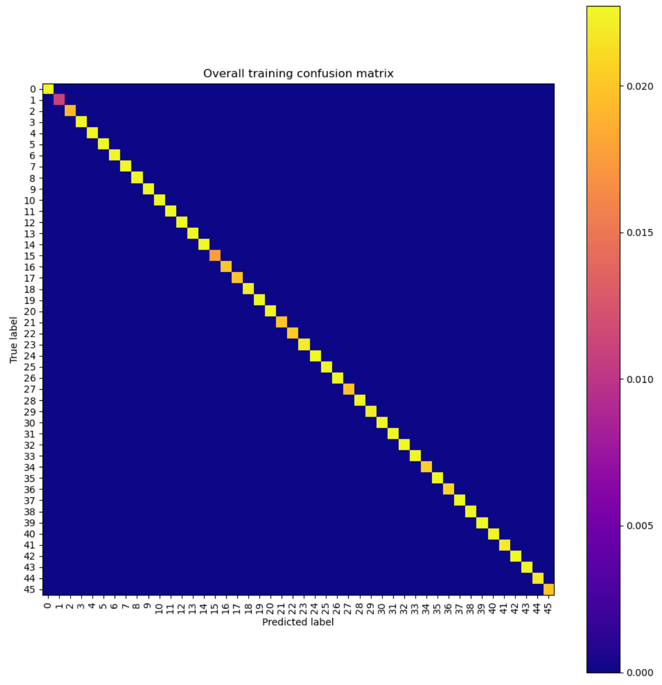
From the confusion matrices we can see that Random Forest performs bad on the same labels as with KNN.

Table 5: KNN vs Random Forest

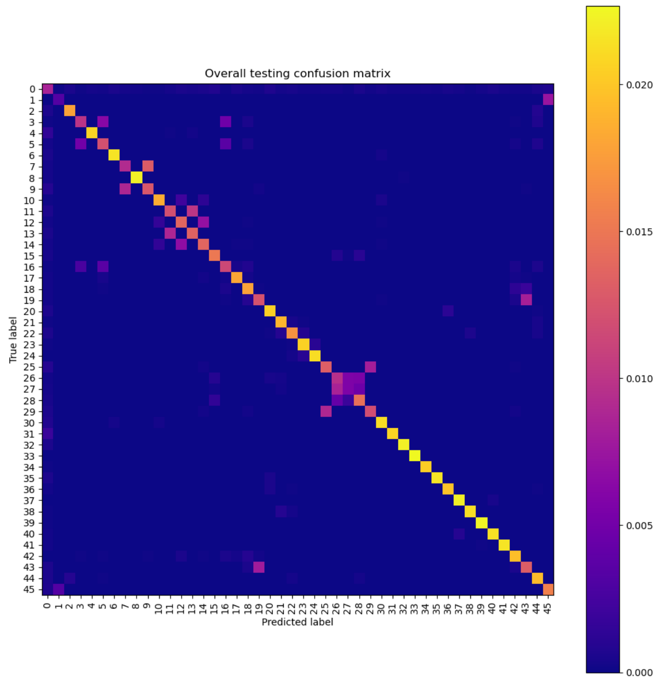
Model	Train Acc+Std	Test Acc+Std
Random Forest	1.0 ±0	0.7627 ±0.0323
KNN	0.9329 ±0.0028	0.7541 ±0.0267

5.4 Different Wrapper Classifiers

I tested several classifiers to use in the wrapper for forward selection. I tried LinearDiscriminantAnalysis, DecisionTree, RandomForest and KNN. However, due to RandomForest and KNN taking a very long time to finish (80 min per subject) I did not have



(a) Overall train confusion matrix



(b) Overall test confusion matrix

Figure 13: Random Forest confusion matrices

enough time to get results for them. I obtained results for LinearDiscriminantAnalysis and DecisionTree shown in table 6. We can see that LinearDiscriminatAnalysis gives better results compared to DecisionTree as the model in forward selection. Furthermore, decision tree takes around 4 times as long (2 min vs 8 min per subject) to train compared to LinearDiscriminantAnalysis.

Table 6: Wrapper Classifiers

Model	Train Acc+Std	Test Acc+Std
DecisionTree	1.0 ±0	0.7515 ±0.0395
LinearDiscriminantAnalysis	1.0 ±0	0.7634 ±0.0323

6 Difficulties

There was an issue with getting feature names from data["feat_names"] for drawing the histogram. Each pressure feature spans 2 elements in data["feat_names"] for some reason, so data["feat_names"] has shape (3871,). I had to fix this in the load_dataset() function in utils.py.