Final Project : SVM Application

Kernel-Based Learning

Martin Guy and Hannes Leskelä

December 29, 2016

Abstract

In this project, we work with EEG (electroencephalography) data of users performing a **Grasp-and-Lift** action. The goal is to apply an SVM classifier in order to predict which action is done by the user according to the collected electrical brain activity. After presenting the data set and some "prerequisite" of EEG and BCI (Brain Computer Interfaces), we present how we faced this problem and applied SVM to classify the EEG data. We trained two different models to classify our six classes: one multiclass SVM and six separate SVM. The results showed that the multiclass SVM was a better predictor and generalized better than the other method. It was also shown that a lot of pre-processing of the data set was needed to train good models, independent of method choice.

Contents

1 Introduction

A brain computer interface (BCI) is an interface of direct communication between a brain and an external device. In our case the principle is simple: collect "information" from the brain of a user in order to interpret what he wants to do. After pre-processing the collected data, we train a classifier using supervised learning and the pre-processed data to generate several models. Models that can predict which movement that is made based on which electrical brain activity. The goal is to detect a specific action, for instance which hand the user want to move, left or right.

We will work on a Kaggle dataset **Grasp-and-Lift EEG Detection** containing electric brain signals of some test persons.

2 The Data Set

This data set is composed of raw electric signals of the brain collected with an Electroencephalogram (EEG) headset. All the subjects perform the same "Grasp and Lift" action, which consist of moving his hand forward, grasp an object, lift it up, then put it down and move the hand back to its original position. This entire action is separated into 6 sub-actions:

1. HandStart: when the hand start moving;
2. FirstDigitTouch: when a finger touch the object;
3. BothStartLoadPhase: when we want to lift the object (short phase)
4. LiftOff: when we start lifting the object;
5. Replace: the object is back to its original place;
6. BothReleased: the object is released.

We have **12 different subjects**, each of them performing **10 series of test**. However we only have the labels for each actions from the first 8 series, as the objective is to predict actions on series 9 and 10. For convenience, we will ignore the last two series and focus on the ones of which we

have labels. Moreover, since the data is really big we will only use the first five series to facilitate training the SVM. Also, we will only consider three different test subjects. So, from more than 1 million of observations for one subject, we will consider 10,000 observations only.

2.1 A word about the labels

In the data set there are six labeled actions that we will consider as six separate classes. The files containing the labels are represented as a matrix with six columns, with values $\in 0, 1$, each columns representing one class. Here are some examples:

$$
\begin{matrix}
1\ 0\ 0\ 0\ 0\ 0 \\
0\ 1\ 1\ 1\ 0\ 0 \\
0\ 0\ 0\ 0\ 0\ 1 \\
.\ .\ .\ .\ .\ .
\end{matrix}
$$

We can observe that we can have more than one class at the same time in our data. This is because the labels are marked as one when the corresponding event has occurred within +/-150ms. This can be problematic as this does not clearly define the six different classes, which we shall later see how to deal with.

3 Background

Electroencephalography is complicated. The computations depend on a lot of parameters, as it varies between people and the task done. Moreover, the folds of the brain are different between any two people (including identical twins), and these folds also change with time. This is why we decided to keep more than one subject, to see if there is any practical difference between models of two different subjects.

The resolution of the sensors (that we will call channels from now) is pretty bad, especially since the scalp is highly conductive. Thus, the value collected by an electrode is quite similar to the value collected by a nearby electrode. Moreover, **there is a lot of noise**: the muscles of the neck, of the face, the eyes and also cardiac activity add noise the output signal. Fortunately, there

exists methods in order to increase the ratio of the desired signal over the noise, i.e. increase the quality of the acquired signal. We will explain the filtering of the data, but will not go into too much detail as it is out of the scope for this report.

3.1 Pre-processing the Data

The filtering is done in **two steps**:

1. Frequency filtering.

2. Spatial filtering, in order to emphasize some characteristics of the signal.

3.1.1 Band-Pass Filter

Most of the time, **depending on the action that we are performing** we only want to keep some frequencies of the signal. Applying a band-pass filter will then remove a part of the noise. One type of electrical activity collected by an EEG headset is called neural oscillation. There are many **neural oscillation**, each one in a particular frequency band, and depends on the psychological state of the person (whether the person is calm, focused etc.). We can denote the oscillations, alpha, beta, gamma, delta, theta and mu.

3.1.2 Spatial Filtering

However, **filtering on the frequencies is not enough**, there is still some noise present in the data. One can apply a spatial filter in order to improve signal quality. Note that this is not the only method but we used a spatial filter as it seems to be the most effective method in practice. [1] In the same way that a temporal filter suppresses some temporal frequencies, a spatial filter removes some spatial frequencies of a signal.

A spatial filter is a matrix $W \in \mathbb{R}^{n*T}$ where $n \in [0; ch]$ is the number of selected filters and ch denotes the number of channels, and T the number of samples. By applying a spatial filter, we then reduce the dimension of our input signal (denoted by $X \in \mathbb{R}^{ch*T}$) and obtain $X \in \mathbb{R}^{n*T}$ such that:

$$X_f = W * X$$

The spatial filter that we decided to use is known as Common Spatial Pattern (CSP)[2], as it is commonly used in state of the art models for brain computer interfaces.

Common Spatial Pattern (CSP) Common spatial pattern separates a multivariate signal into additive subcomponents that have a maximum differences in variance between two windows. The idea is to find a spatial filter such that it **maximizes the variance of one class** and **minimizes the variance of the other class**. The CSP algorithm only works with two classes [1]. On the following, we consider two classes denoted respectively as `1` and `2`.

Let S_1,$S_2 \in \mathbb{R}^{ch \times ch}$ be estimates of the covariance matrices of the **filtered signal** X_f and t_1 (resp. t_2) the set on indexes such that the signal is from the class `1` (resp. class `2`). We have:

$$S_1 = \frac{1}{|t_1|} * \sum_{i \in t_1} X_i X_i^T$$

$$S_2 = \frac{1}{|t_2|} * \sum_{i \in t_2} X_i X_i^T$$

The filter W is then calculated by finding that W such that:

$$\begin{cases} W^T S_1 W = V_1 \\ W^T S_1 W = V_2 \\ V_1 + V_2 = I \end{cases}$$

[1] Even though there exists some variants with more than two classes.

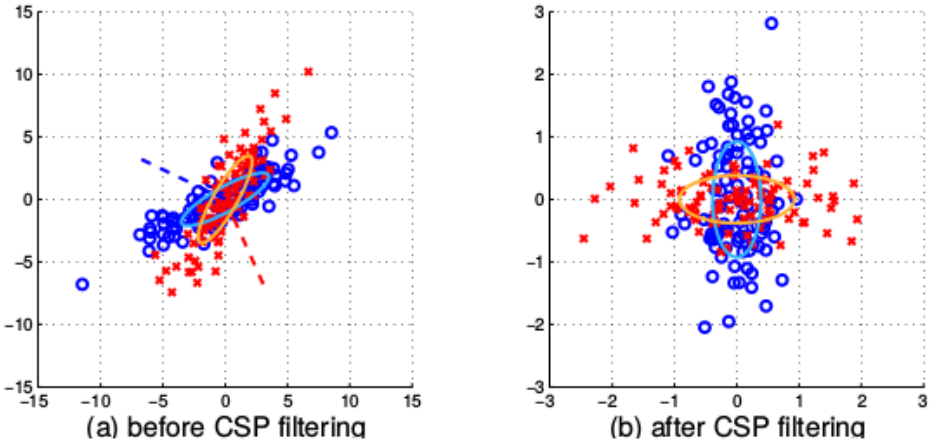

Figure 1: Effect of applying CSP [3]

4 Methods

4.1 About the pre-processing

The pre-processing has been done in **Python** since there are several Python libraries for Brain Computer Interfaces, like for instance `MNE`[4]. Moreover we mainly used a code of Alexandre Barachant[5]. This way, we could focus on the main part of the project which is to model and train the SVM. Even so, here are some technical notes regarding the pre-processing of the data:

- We selected a pass-band between 7-30 Hz, as it is known that the sensorimotor cortex is expressed by change in signal power in the beta (∼20 Hz) and mu (∼10 Hz) frequencies. [6]
- As stated previously, CSP only works on two classes. However, here we have 6 classes. When applying it, **we decided to consider `HandStart` as the first class**, and all the other actions as the second class. We chose this split because this was the choice of Alexandre Barachant, who in turn chose HandStart because the movement is significantly different from the other actions. But this is a personal choice and doing the split in another way would most surely change the results. We also selected **4 components** for CSP as it is a commonly used value that generally produces good results. [3]

4.2 Dealing with the labels

As we have seen it during the lectures SVM works with two classes, but here we have 6 classes (i.e. 6 potential labels for each line of data).

Our first idea was to search for a multi-class SVM and apply it. However, in the label files, we sometime have more than one 1 in a row, so this would lead to a model with more than 6 unique classes. Adding this complexity to the project seemed cumbersome, so we looked for alternative approaches as well.

Our second idea then was to do six independent SVM to detect if it is one of the classes. This would solve the problem of having more than one 1 in a row, since each SVM predicted 1 or 0 for a class independently of the other SVM.

In the end, we decided to tackle the problem with both of these solutions and compare the results of the two approaches.

4.3 C parameter tuning

The C parameter decides on the width of the margin for the separating hyperplane, and thus dictate how much misclassifications we can tolerate. A small C value means that we have a large margin for our separating hyperplane, even if it means that we incorrectly classify some points. Our approach is to use a C from 0.1 ranging up to 1000, to see if any improvements could be made. The C was selected arbitrarily, but we ended up using the following values:

C	0.1	1	300	600	1000

Table 1: Different values of C tried

4.4 Class weighting

When training a statistical model, it is important to try and have a balanced data set. If one class is more dominant than the others, the model might make it easy for itself and only predict that class in order to get a high accuracy. To combat a biased data set, one can introduce class weighting to tell the model that a class represented by fewer values should be considered more important than

a common class. By looking at table 2, we can see a clear need to weight our classes to even out the bias for some of the classes.

Class number	**1**	**2**	**3**	**4**	**5**	**6**
Population	2113	2059	1868	175	691	2812

Table 2: Distribution of the classes for test set

In order to correct this, we selected weights w_i for the classes such that:

$$w_i = \frac{\sum_{i=1}^{6} |c_i|}{|c_i|}$$

where $|c_i|$ is the number of samples of the class i. The weight was then used in training of the model with the **class.weights** parameter set for **ksvm** function call.

5 Results

In this section, due to space limitations, we will present the **plots for the first subject only**. Full results can be found in appendix.[2] For each model we compute the accuracy, and also the **AUC score** to have another test metric, since accuracy is not always an optimal measurement of performance.

5.1 One SVM for each class

This is the obtained accuracy separately for each class. We find an average accuracy of **72%**, **72%** and **70%** for the subject 1, 2 and 3 respectively.

[2] As stated previously, there exists some differences between test subjects, leading to different results.

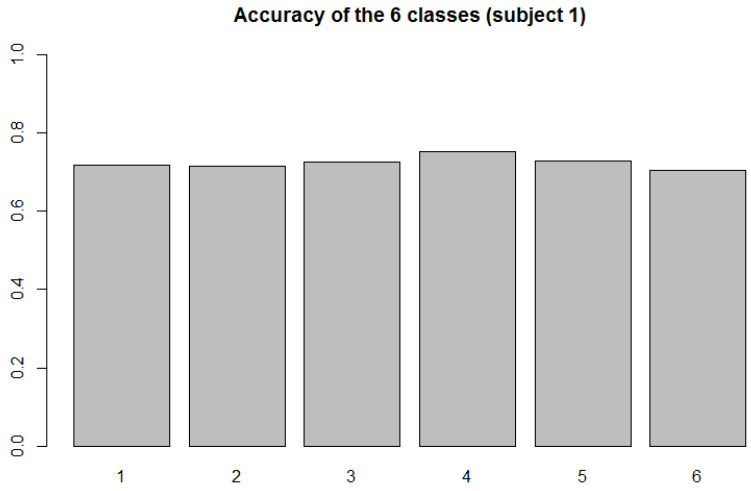

Figure 2: Accuracy on the 6 classes (6-SVM)

The following ROC curves were obtained for the six different SVM for subject 1:

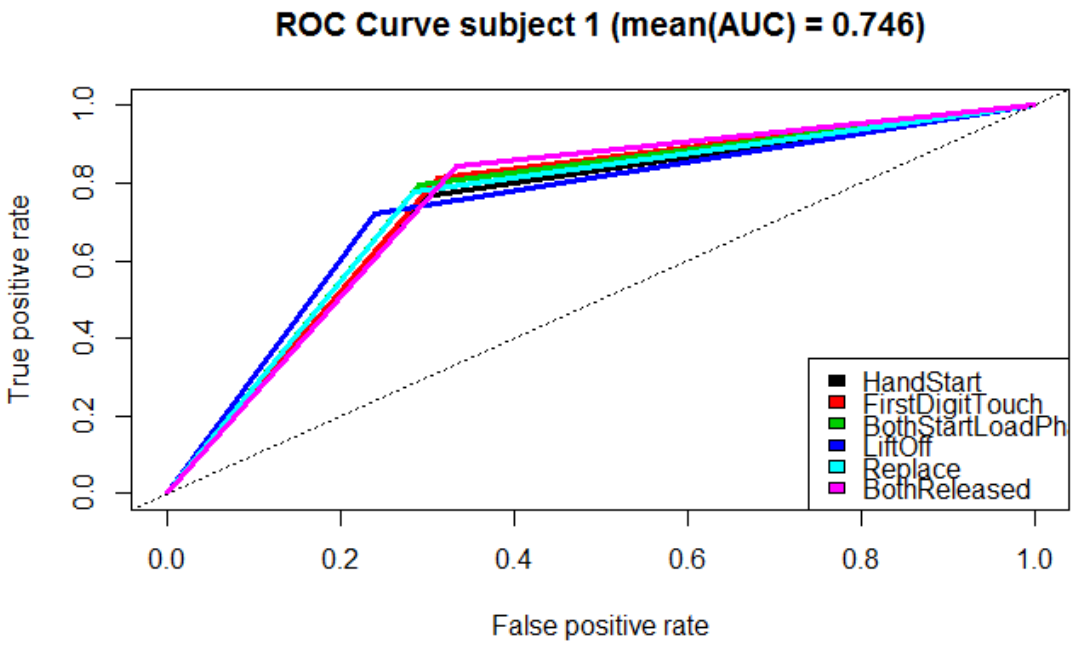

Figure 3: ROC curve of the first subject (6-SVM)

The average AUC is **0.746**, **0.751** and **0.719** respectively.

5.2 Multiclass SVM

`kernlab` automatically implements a multi-class SVM with an all-versus-all strategy to combine several binary SVM. In order to deal with the problem of more than one 1 in a row, we filtered the data set, including only rows that have exactly one 1 for our train and test data.

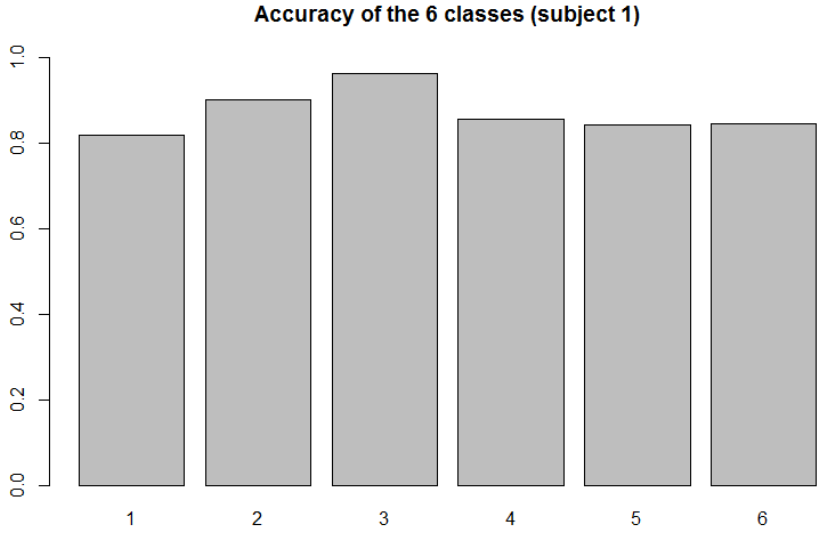

Figure 4: Accuracy on the 6 classes (mutli-class SVM)

We find an accuracy of **62%**, **67%** and **57%** for the subject 1, 2 and 3 respectively, when checking for exact labels.

When we consider each classes separately, we find an average accuracy of **87%**, **89%** and **86%** for the subject 1, 2 and 3 respectively, which is a **higher** score than what we previously obtained by training 6 independent SVM.

The average AUC is **0.793**, **0.805** and **0.775** which is higher than for the six independent SVM.

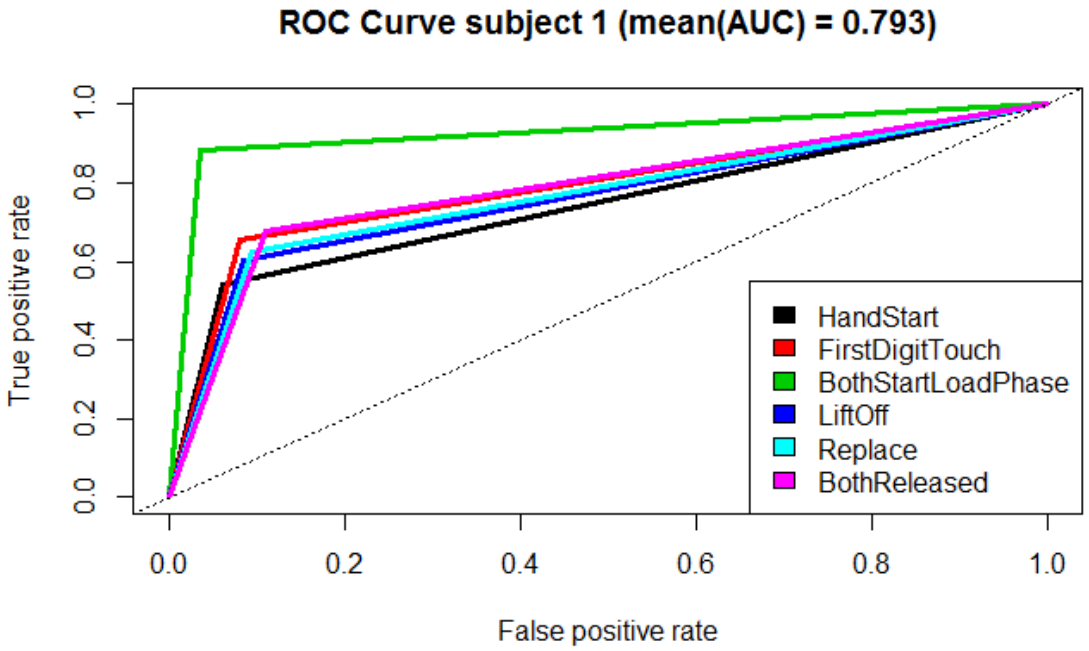

Figure 5: ROC curve for the first subject (mutli-class SVM)

6 Discussion

At first glance it was a bit daunting being confronted with this data set. Indeed, the data are very large and we have 6 labels, some of them being present at the same time, which is not a classic problem for SVM. Initially, it was not really clear how to tackle this problem, but the results from our two models of choice seem promising. They are by no means perfect models, but it is far better than guessing which is basically the worst case performance of an SVM.

6.1 The impact of the C parameter and class weighting

In the beginning, we just wanted to try which one of the two approaches would fit a model to the data the best, the multiclass SVM, or six class specific SVM. So we took C=1 by default, and created our models. But this approach proved to generate poor results. With C=1 we got 81% on the accuracy for the first model (6-SVM) on testing (which is better than our final result of 72%). For the second model we had an accuracy around only 35%. However, we only had an **AUC score**

around 0.5 for both of these models, which is really bad.

Without considering the C parameter, there were also another problem: we did not have an equal amount of samples in each class, and this is important as we are considering only around 10,000 observations per subject. As we can see in table 6 in the appendix, the proportion of each classes on our multi-class test data set is highly skewed.

We can also observe that the model cannot predict the classes that are not as heavily presented in the data set, and overpredicts the most common class. This is why we decided to use class weighting in addition to tuning the C parameter. Note that we finally did the same for the 6-SVM approach. To select the C parameter, we focused mainly on the lowest training error, but also, to some extent, the computational time of the model.

From class weighting, we got a **huge** gain in accuracy for our models. **From 35%** for multi-class SVM we went **to 62%**. The accuracy even improved for predicting each classes separately.

Regarding the AUC score, this time **the difference was even bigger**. From having AUC scores around 0.5 for both models, we managed to have almost 0.8 for the multi-class SVM, and around 0.74 for the 6-SVM approach. This means that we can now trust the models more when it comes to generalization. Even the 6-SVM model, where the accuracy dropped, should be more accurate on other data since the new AUC score compensates for the loss of accuracy.

7 Conclusion

Working on EEG data is not an easy task as there are a lot of factors related with signal processing to consider. It is necessary to have a lot of data in order to train your models properly, and it can be problematic when one uses just a small amount of data, like in this project. Moreover, it is important to take care about bias in our data, and thus consider using class weighting in order to remove some classes that are too heavily represented.

When calculating if a model is good there are plenty of methods to use, but we often only look at accuracy. In this project we could observe that only using accuracy is not the best choice, as the AUC score could help indicate when the models were bad. Moreover, the definition of accuracy is ambiguous and depending on which one we used we got different results. The two measurements

of accuracy that we used was: accuracy for each classes, or accuracy for exact labels (multi-class SVM only).

As the pre-processing plays an important part in the results, it could be interesting to change it and see how it affects the models. Examples of changes include: more CSP components, filtering other frequency bands, other class choices for CSP, or even multi-class CSP.

Finally, we did not consider the length of the computation time. However it is highly important to account for this in EEG research, as we have to calibrate the EEG headset for each subject. This means that a faster calibration leads to a better overall model for the user in the end. Maybe other classifiers could do this fit faster, and thus be more appropriate for a project like this, but that is a question left for further research.

References

[1] Common Spatial Pattern - Wikipedia

https://en.wikipedia.org/wiki/Common_spatial_pattern

[2] Introduction to Modern Brain-Computer Interface Design - Christian A. Kothe

https://www.youtube.com/watch?v=EvVEC7O6zlc&index=18&list=PLbbCsk7MUIGcO_lZMbyymWU2UezVHNaMq

Retrieved 2016-12-14

[3] Optimizing Spatial Filters for Robust EEG Single-Trial Analysis

http://doc.ml.tu-berlin.de/bbci/publications/BlaTomLemKawMue08.pdf

[4] MEG + EEG analysis and visualization tools http://martinos.org/mne/stable/index.html

Retrieved 2016-12-28

[5] Alexandre Barachant's code for the Grasp and Lift EEG Detection competition on kaggle.com https://www.kaggle.com/alexandrebarachant/grasp-and-lift-eeg-detection/beat-the-benchmark-0-67

Retrieved 2016-11-02

[6] Rythme crbral - Wikipedia french page of Neural oscillation https://fr.wikipedia.org/wiki/Rythme_c%C3%A9r%C3%A9bral

[7] Introduction to Common Spatial Pattern Filters for EEG Motor Imagery Classification - Tatsuya Yokota

goo.gl/CJggTE

Retrieved 2016-12-20

[8] A review of classification algorithms for EEG-based braincomputer interfaces - Fabien Lotte et. al.

https://hal.archives-ouvertes.fr/inria-00134950/document

Retrieved 2016-12-19

[9] Common Spatial Pattern Method for channel Selection in Motor Imagery Based Brain-Computer-Interface

https://sccn.ucsd.edu/~yijun/pdfs/EMBC05.pdf

Appendices

We got these results, for the first subject:

	6-SVM	**Multiclass SVM**	
		Factors	Each class
Accuracy	72%	62%	87%
mean(AUC)	0,746	-	0,793

Table 3: Results of the different models for the subject 1

second subject:

	6-SVM	**Multiclass SVM**	
		Factors	Each class
Accuracy	72%	67%	89%
mean(AUC)	0,751	-	0,805

Table 4: Results of the different models for the subject 2

third subject:

	6-SVM	**Multiclass SVM**	
		Factors	Each class
Accuracy	70%	57%	86%
mean(AUC)	0,719	-	0,775

Table 5: Results of the different models for the subject 3

Predictions for first subject:

6-SVM:

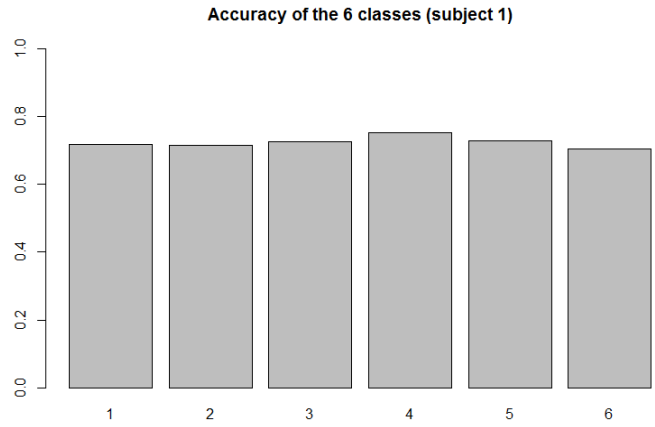

Figure 6: Box-plot of the first subject

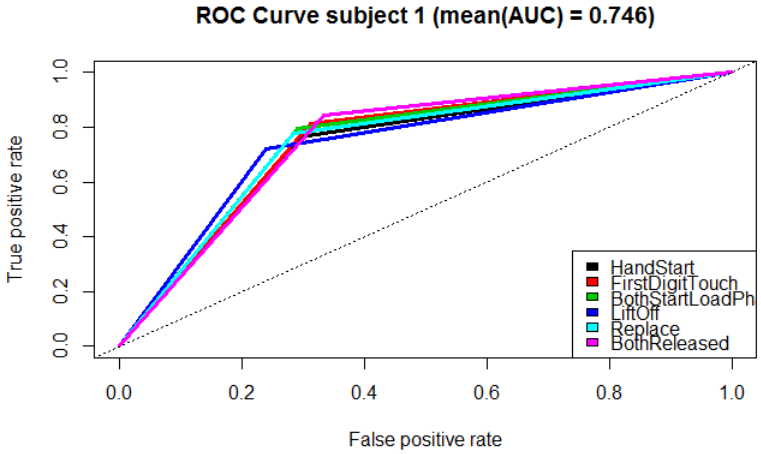

Figure 7: ROC curve of the first subject

Multi-class SVM:

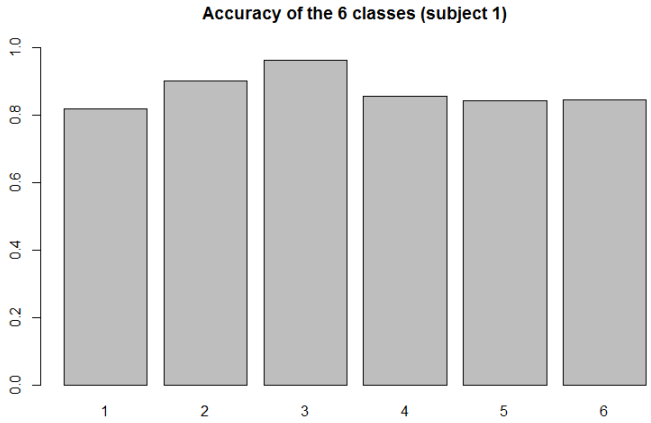

Figure 8: Box-plot of the first subject

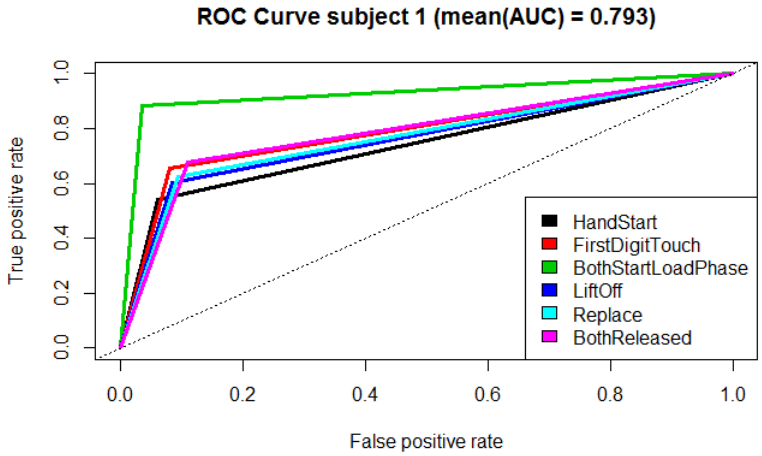

Figure 9: ROC curve of the first subject

Predictions for second subject:

6-SVM:

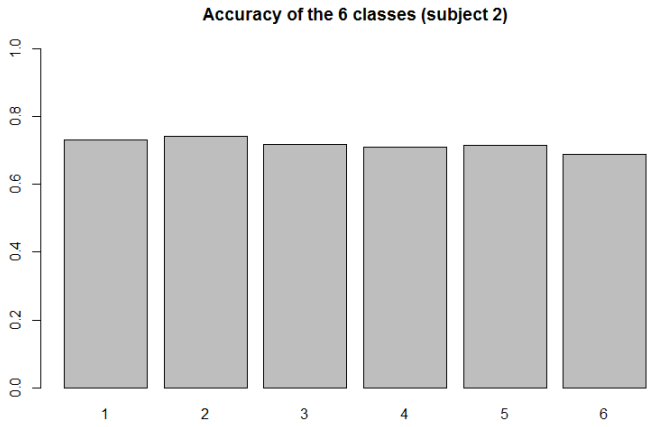

Figure 10: Box-plot of the second subject

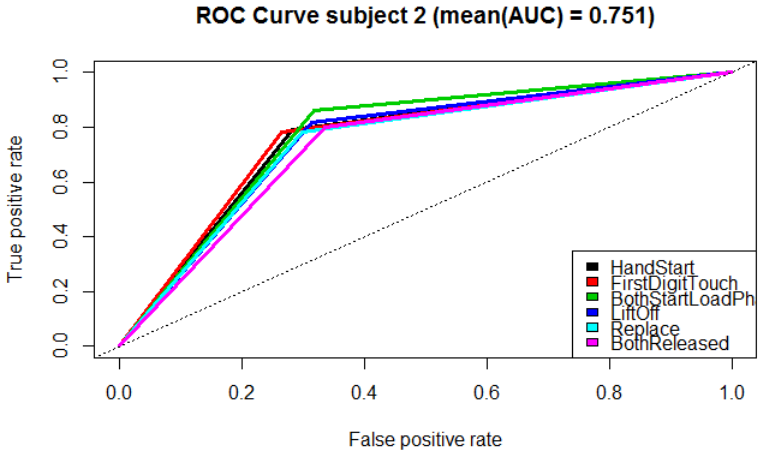

Figure 11: ROC curve of the second subject

Multi-class SVM:

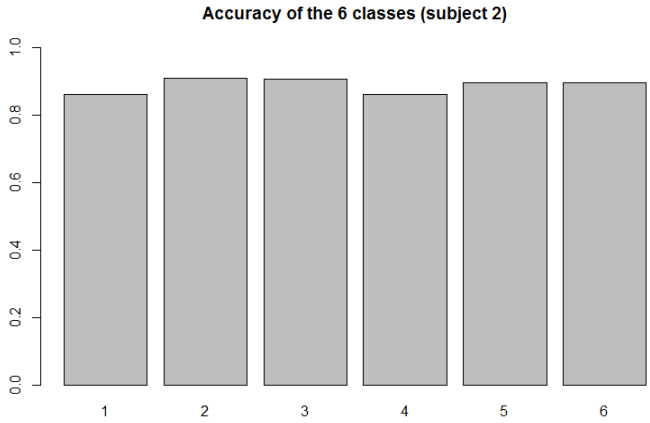

Figure 12: Box-plot of the second subject

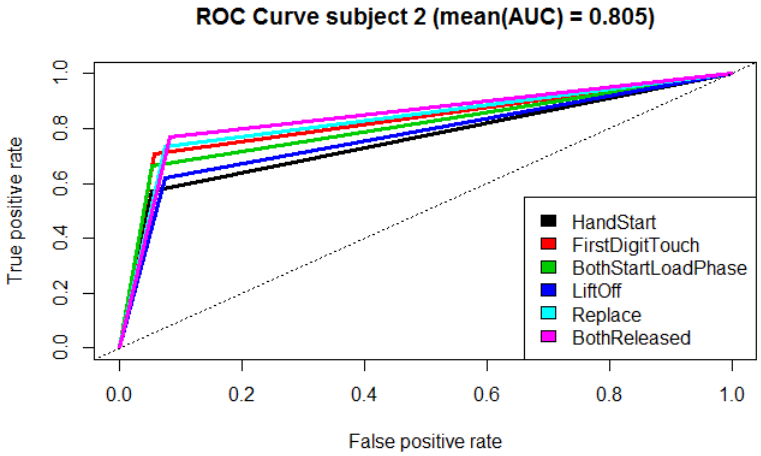

Figure 13: ROC curve of the second subject

Predictions for third subject:

6-SVM:

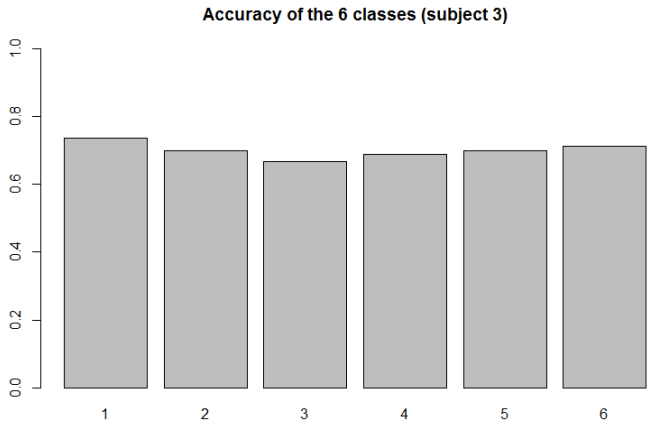

Figure 14: Box-plot of the third subject

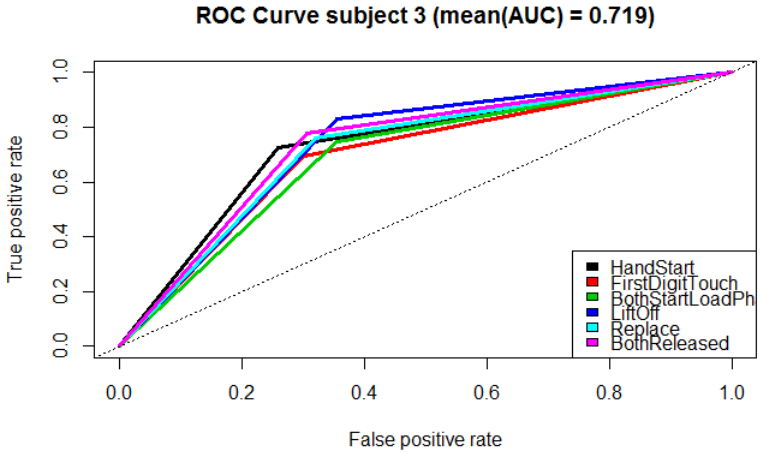

Figure 15: ROC curve of the third subject

Multi-class SVM:

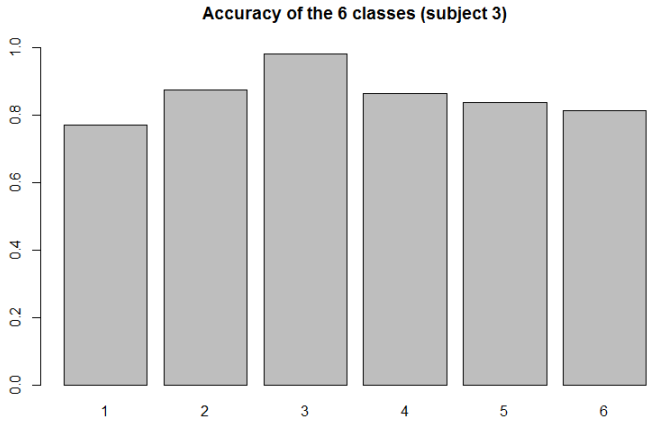

Figure 16: Box-plot of the third subject

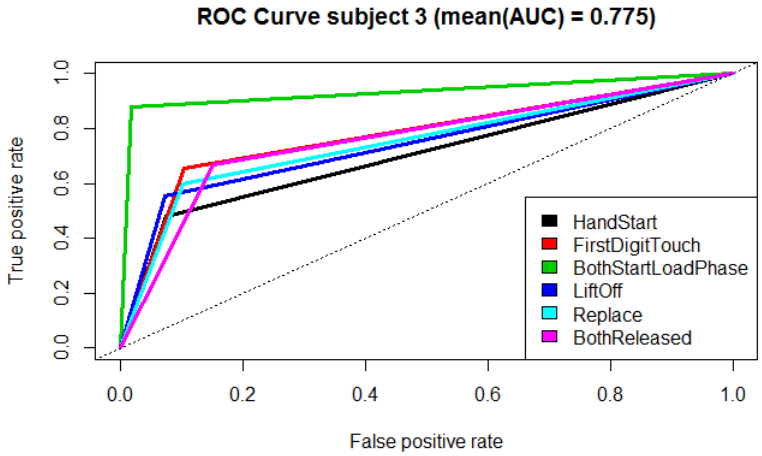

Figure 17: ROC curve of the third subject

Class number	**1**	**2**	**3**	**4**	**5**	**6**
Population	1357	1415	1193	139	447	1928
Predicted	317	591	269	2	6	5294

Table 6: Distribution of classes for test set, and predictions from the multiclass SVM