EL844 - Workshop 1 Report

Bastien Dhiver – bfrd2

Task 1a:

We try to extract features from static signatures.

I coded 4 functions to address the problem:

- f2DMoment: computes the 2D moment of order (p + q)
- centralMoment: computes the central moment of order (p + q)
- normalisedCentralMoments: computes the normalized central moment of order (p + q)
- HuInvariantMoments: computes the 7 Hu invariant moments of the given image

Using the script `task1a.m`, we end up with this table:

```
>> task1a
```

	user1_1	user1_2	user2_1	user2_2
ϕ1	1.401	1.4778	1.1827	0.98267
ϕ2	1.6899	1.8836	1.0427	0.70775
ϕ3	0.064567	0.085767	0.021778	0.044015
ϕ4	0.0076309	0.014977	0.0057587	0.01709
ϕ5	-1.3659e-06	0.00025652	6.22e-05	0.00046367
ϕ6	-0.0065211	0.0060271	0.0058615	0.014348
ϕ7	-0.00016938	0.00047149	-1.7033e-05	-6.8658e-05

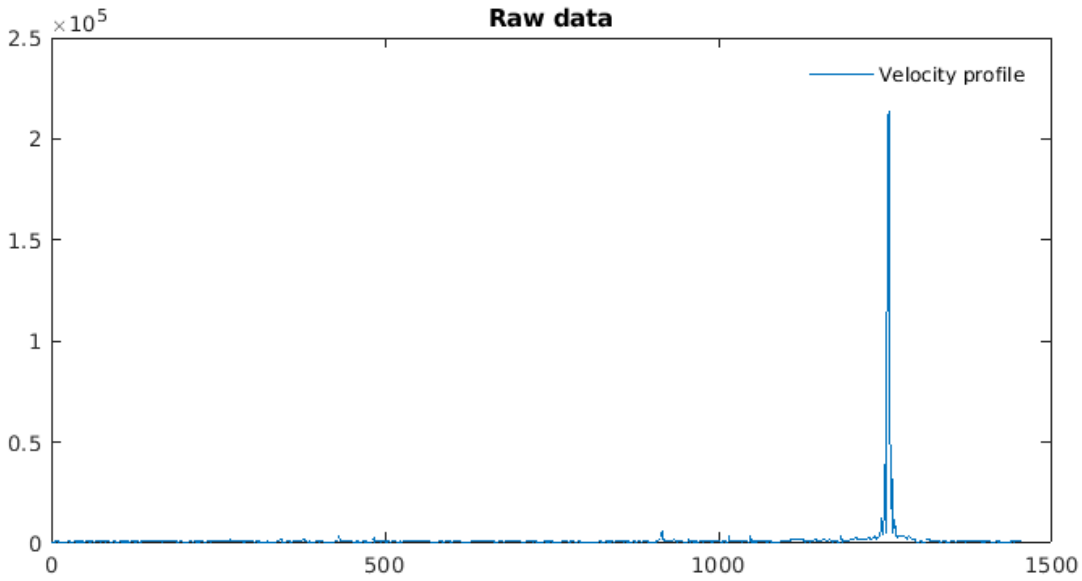
We can notice here that the extracted features (Hu invariant moments) for the signatures `user1_1` and `user1_2` are very similar. On the other hand, `user2_1` and `user2_2` features are more distinct. We can visually see that the writing is a little bit thinner on `user2_1`. We can distinguish the 2 two dots as well as the letters ‘A’ and ‘o’. This explains why these moments differ.

Task 1b:

We try to extract features from dynamic signatures.

For this task, I will focus on the ‘user1_1.pen’ file for the analysis and then display statistics for all the .pen files.

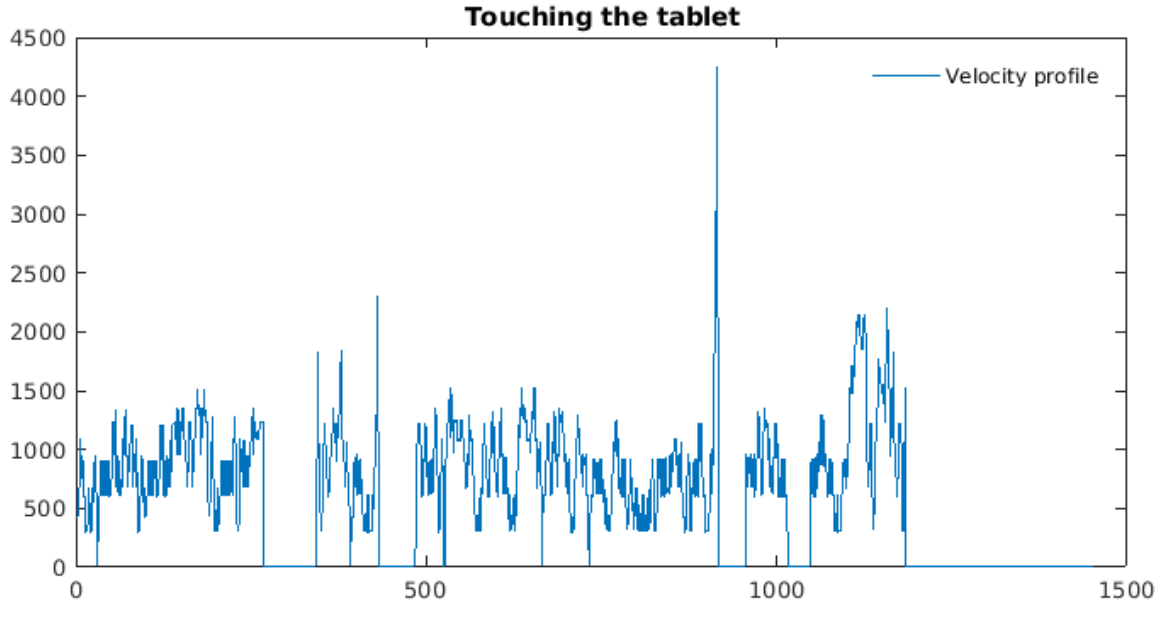
Thanks to the script ‘task1b.m’ we can plot the velocity profile for ‘user1_1.pen’



We can see a significant spike on the curve near the measure n°1250. After inspecting the .pen file, we notice that the recorded measures are going very high when the pen is not touching the tablet.

1251	4.1667	1617	187	0
1252	4.1700	1631	185	0
1253	4.1733	1855	270	0
1254	4.1767	2346	466	0
1255	4.1800	2992	730	0
1256	4.1833	3643	1002	0
1257	4.1867	4145	1220	0
1258	4.1900	4387	1341	0

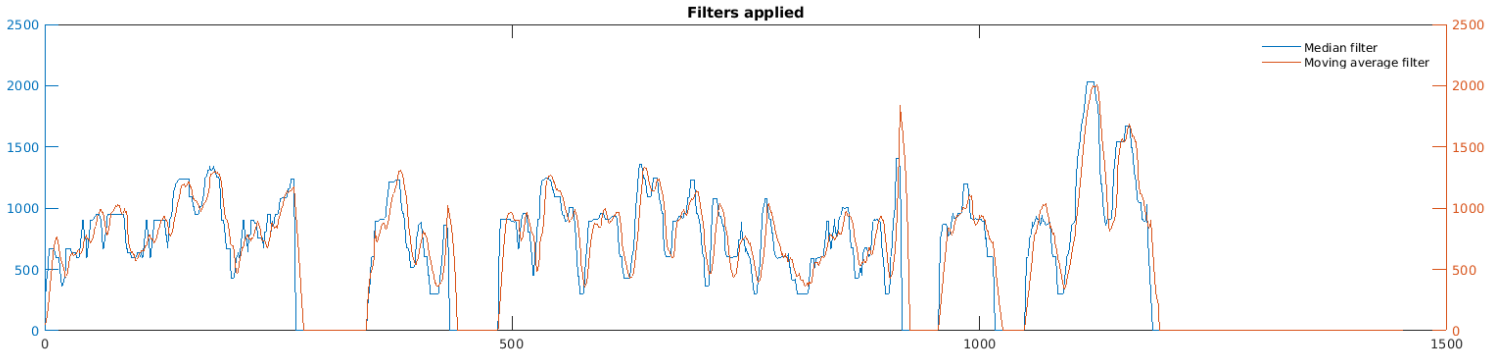
We decide to set to 0 the velocity when the pen does not touch the screen.



Once this special case is resolved, we can see the shape of our velocity profile.

Noises are amplified since we are computing a delta between 2 positions. This is why I applied 2 different filters to smooth the results. I will apply 2 common methods used for smoothing noisy data:

- Noise Suppression by Median Filtering
- Moving-Average Filter



The last plot is way better. As a parameter to the median filtering, I've used a 10th-order median filter to smooth the signal. As of the moving-average filter, I've used a window size of 10.
Both filters are working great with this parameters. We now have a correct velocity profile with 2 different filters.

All the plot exports with the different steps are in the archive folder next to this report. (They didn't fit well in this document)

After generating the 4 signature velocity profiles, we can see a huge difference between user1 and user2 dynamic signatures. 'user1_1' and 'user1_2' profiles looks the same as well as 'user2_1' and 'user2_2'. This is interesting to see that the smoothing done by the 2 different methods is quite similar.

Complementary, some statistics have been computed for every .pen file.

Input file: 'user1_1.pen'

	Raw	Touching
max velocity	2.138e+05	4253.2
max velocity time	4.18	3.0467
velocity average	1754.1	874.03

Input file: 'user1_2.pen'

	Raw	Touching
max velocity	9122.4	2589.1
max velocity time	1.5833	3.35
velocity average	893.19	885.33

Input file: 'user2_1.pen'

	Raw	Touching
max velocity	6135.9	3163.7
max velocity time	4.15	1.1367
velocity average	1300.8	1149.4

Input file: 'user2_2.pen'

	Raw	Touching
max velocity	9136.6	3163.7
max velocity time	3.9033	1.1067
velocity average	1270.3	1108.2

We separate the raw statistics from the one when the pen was touching the tablet.

We've seen that the Hu invariant moments feature can help us to have a first idea on how different 'user1' and 'user2' static signatures can be. The smoothed velocity profiles instantly showed how different the signatures were made. We also notice that the computed velocity averages are similar per user.

Task 2:

We apply classification algorithms on features.

The classifier function 'fitcnb' performs very well when N=7 for example. Just one wrong guess in this example.

test_label

20x1 double

	1
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20

predicted_label

20x1 double

	1	2
1	1	
2	2	
3	3	
4	4	
5	5	
6	6	
7	7	
8	8	
9	9	
10	10	
11	11	
12	12	
13	13	
14	10	
15	15	
16	16	
17	17	
18	18	
19	19	
20	20	

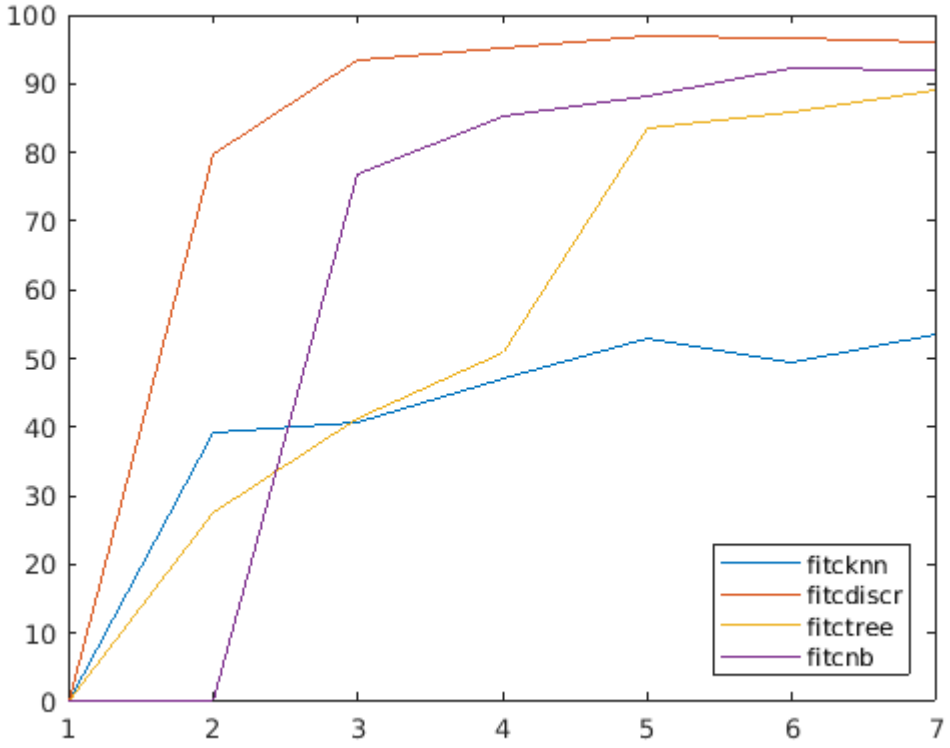
From time to time the naïve Bayes classifier (fitcnb) function **raises an error**:

```
Error using ClassificationNaiveBayes/fitNonMNDists (line 222)
A normal distribution cannot be fit for the combination of class 12 and predictor x4. The data has zero variance.
```

After asking an assistant, sometimes this classifier does not have enough data to train on. I tried to mitigate this issue by running the function only when N > 2 but it isn't perfect. If the zero variance error arises, I re-run the program.

We will now look at the accuracy rate of the classifiers thanks to plots.

With the original features:

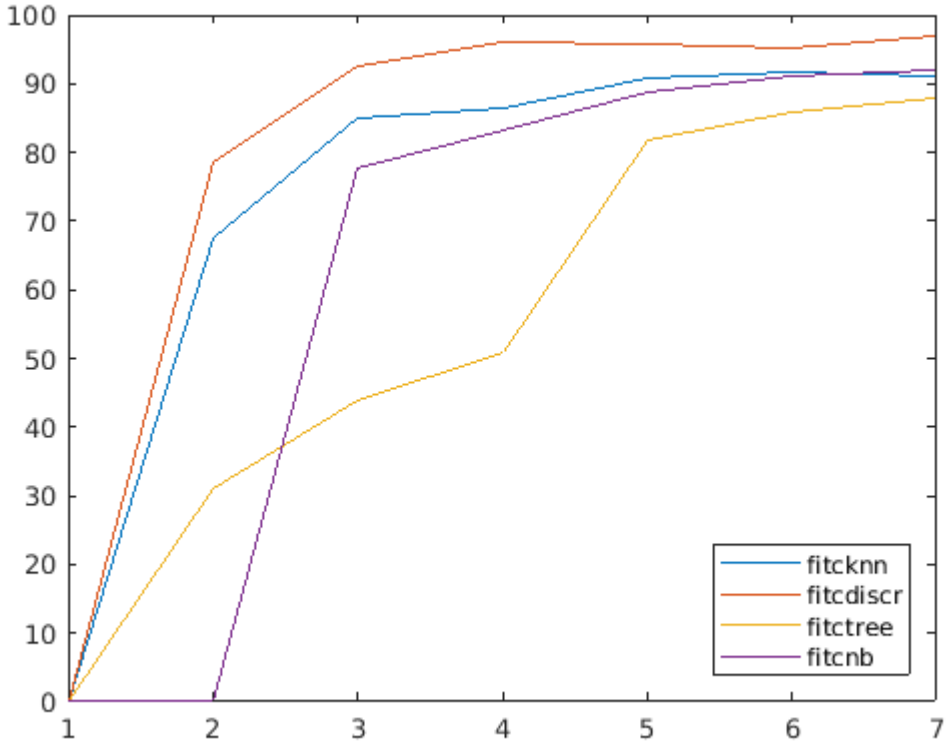



```
>> task2
```

	fitcknn_avg	fitcdiscr_avg	fitctree_avg	fitcnb_avg
N2	39.333	79.833	27.417	0
N3	40.8	93.4	41.3	76.8
N4	47.125	95.25	51	85.375
N5	53	96.833	83.667	88.333
N6	49.5	96.75	85.75	92.25
N7	53.5	96	89	92

The ‘fitcdiscr’ classifier function performs the best and the ‘fitcknn’ classifier function performs the worst.

With the normalized features:




```
>> task2
```

	fitcknn_avg	fitcdiscr_avg	fitctree_avg	fitcnb_avg
N2	67.417	78.583	31.083	0
N3	85	92.7	44	77.8
N4	86.375	96.125	51	83.375
N5	90.833	95.833	81.833	88.667
N6	91.75	95.25	86	91
N7	91	97	88	92

We notice that with the normalized data, the ‘fitcknn’ classifier performs way better.

If I had to pick a classifier in this context of feature analysis, I’ll go with the ‘fitcknn’ because it performed the best on ‘original’ and ‘normalized’ features.