Support Vector Machines – Testing & Parameter Tuning

Emanuel Zaymus

Dataset

Kyoto – ADL Activities

Features

1. SECONDS FROM MIDNIGHT - of the first record in the window
2. DAY OF THE WEEK - MON..SUN => 1..7 - of the last record in the window
3. SECONDS ELAPSED – between the last and the first record of the window

4.-28. SIMPLE COUNTS OF THE SENSORS

(29. CLASS of the feature vector - index of the activity 0..4 - of the last record of the window)

Used library

Scikit-learn => `sklearn.svm.SVC` (C-Support Vector Classification)

Testing

For testing purposes I chose 5-folds cross-validation with shuffling data and fixed random state to value 0:

```
sklearn.model_selection.KFold(n_splits=5, shuffle=True,
random_state=0)
```

Feature scaling

I used `sklearn.preprocessing.StandardScaler` to scale feature vectors and to improve the performance.

From Scikit-learn documentation:

`StandardScaler` - Standardize features by removing the mean and scaling to unit variance. The standard score of a sample ***x*** is calculated as: ***z = (x - u) / s*** where ***u*** is the mean of the training samples or zero if `with_mean=False`, and ***s*** is the standard deviation of the training samples or one if `with_std=False`.

Defaults I used are: `with_mean=True` and `with_std=True`.

Docs: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

Also a lot of tutorials say that scaling should be used!

I also tried to make a simple (maybe stupid) visualization of various scaling techniques – at the bottom of this document.

Window size

At the beginning it is important to try multiple window sizes of a feature vector – number of records from which a feature vector consists. Because it is the first step I am using default settings of `SVC()`. SVC defaults are: `SVC(kernel='rbf', C=1.0, gamma='scale')`

Table 1

Window size	Accuracy score	Improvement	
5	75.9229	-	
7	79.6540	3.7311	
10	83.7593	4.1053	
13	86.8080	3.0487	MAJOR IMPROVEMENT
15	88.4888	1.6808	
17	89.9049	1.4161	
20	91.1803	1.2754	
23	92.2692	1.0889	
25	92.8293	0.5601	
27	93.1864	0.3571	
30	**93.7462**	0.5598	
33	93.9465	0.2003	MINOR IMPROVEMENT
35	93.9603	0.0138	
37	94.1148	0.1545	
40	94.2060	0.0912	

Based on the results above I chose window size **30** as the ideal window size, because it is one of the first values with minor improvement and I wanted to keep it more or less "small".

Parameter tuning - Kernel

This parameter specifies the kernel type to be used in the algorithm. It must be one of: `'linear'` (linear kernel), `'poly'` (polynomial kernel), `'rbf'` (radial-basis function kernel or squared-exponential kernel), `'sigmoid'`, `'precomputed'` (needs square matrix) or a custom kernel.

More about kernel functions: https://scikit-learn.org/stable/modules/svm.html#svm-kernels

Since there are too many parameters for every kernel that can be adjusted, I decided to run a test for every kernel with its default parameters and choose the one with the best result.

Table 2

Kernel	Accuracy score
Linear	90.8536
Polynomial	92.2452
RBF	**93.7462**
Sigmoid	76.1413
Precomputed	needs square matrix

The best result yields **RBF**.

Parameter tuning – C and Gamma

The **C** is a regularization parameter or penalty parameter. It tells the SVM optimization how much error is bearable. A smaller value of C creates a small-margin hyperplane and a larger value of C creates a larger-margin hyperplane. It has to be strictly positive. Default is C=1. In general is better to select a smaller C.

The **Gamma** is a kernel coefficient. A lower value of Gamma will loosely fit the training dataset, whereas a higher value of gamma will exactly fit the training dataset, which causes over-fitting. That means that more convenient is to choose a Gamma of lower value. There are two predefined values: **'scale'** (default) and **'auto'**.

- 'scale' = 1 / (n_features * X.var()) ; where X.var() is variance of the data array
- 'auto' = 1 / n_features

As we can see in the *Table 3*, accuracy scores for 'scale ' and 'auto ' are exactly the same. Also the best result are yielded when C is super large. High value of C causes too large margin – 3 it is better to keep it the smallest possible.

Let's see if it can be done in a better way.

Table 3

C	Gamma	Accuracy score
0.25	scale	91.5260
0.25	auto	91.5260
0.5	scale	92.6831
0.5	auto	92.6831
1	scale	93.7462
1	auto	93.7462
2	scale	94.6217
2	auto	94.6217
5	scale	95.2940
5	auto	95.2940
7	scale	95.5755
7	auto	95.5755
10	scale	95.8413
10	auto	95.8413
13	scale	96.1070
13	auto	96.1070
15	scale	96.3415
15	auto	96.3415
17	scale	96.3884
17	auto	96.3884
20	scale	96.4197
20	auto	96.4197
30	scale	96.6073
30	auto	96.6073
50	scale	96.8106
50	auto	96.8106
75	scale	97.0450
75	auto	97.0450
100	scale	97.0920
100	auto	97.0920
150	scale	97.2952
150	auto	97.2952
200	**scale**	**97.3578**
200	**auto**	**97.3578**
500	scale	97.3577
500	auto	97.3577
1000	scale	97.2640
1000	auto	97.2640

Table 4

C	Gamma	Accuracy
1	0.01	90.7912
1	0.02	92.3547
1	0.05	94.5435
1	0.1	95.5286
1	0.15	96.0133
1	0.2	96.3884
1	0.25	96.6855
1	0.3	96.7636
1	0.35	96.7323
1	0.4	96.5447
1	0.5	96.1383
1	0.75	95.4972
1	1	93.4492
2	0.01	91.6668
2	0.02	93.7150
2	0.05	95.1221
2	0.1	95.9819
2	0.15	96.7011
2	0.2	96.8887
2	0.25	97.0919
2	0.3	97.0450
2	0.35	97.2014
2	0.4	97.1389
2	0.5	96.5760
2	0.75	95.8099
2	1	94.1682
5	0.01	93.2928
5	0.02	94.4966
5	0.05	95.7474
5	0.1	96.7949
5	0.15	97.2170
5	0.2	97.3265
5	0.25	97.4516
5	0.3	97.6704
5	0.35	97.6235
5	0.4	97.4203
5	0.5	96.8574
5	0.75	95.8255
5	1	93.9650
7	0.01	93.8400
7	0.02	94.8249
7	0.05	96.1539
7	0.1	96.9200
7	0.15	97.2014
7	0.2	97.4203
7	0.25	97.6235
7	0.3	97.6392
7	0.35	97.6392
7	0.4	97.4672
7	0.5	96.8575
7	0.75	95.7474
7	1	93.9963
10	0.01	94.2621
10	0.02	95.1533
10	0.05	96.3572
10	0.1	97.2014
10	0.15	97.4516
10	0.2	97.6079
10	0.25	97.6235
10	**0.3**	**97.7486**
10	**0.35**	**97.7330**
10	0.4	97.5297
10	0.5	96.7793
10	0.75	95.7161
10	1	93.9494

C	Gamma	Accuracy
13	0.01	94.4185
13	0.02	95.4347
13	0.05	96.5604
13	0.1	97.2171
13	0.15	97.5454
13	0.2	97.6235
13	**0.25**	**97.7173**
13	**0.3**	**97.7330**
13	**0.35**	**97.7017**
13	0.4	97.4515
13	0.5	96.7480
13	0.75	95.6692
13	1	93.9650
15	0.01	94.4184
15	0.02	95.5598
15	0.05	96.6542
15	0.1	97.2640
15	0.15	97.5766
15	0.2	97.6548
15	**0.25**	**97.7486**
15	**0.3**	**97.7486**
15	0.35	97.6548
15	0.4	97.4203
15	0.5	96.7011
15	0.75	95.6535
15	1	93.9181
17	0.01	94.5435
17	0.02	95.6536
17	0.05	96.7324
17	0.1	97.3421
17	0.15	97.6079
17	0.2	97.6548
17	**0.25**	**97.7330**
17	0.3	97.6861
17	0.35	97.6079
17	0.4	97.3734
17	0.5	96.7011
17	0.75	95.6535
17	1	93.9181
20	0.01	94.7311
20	0.02	95.6693
20	0.05	96.8731
20	0.1	97.4359
20	0.15	97.5766
20	**0.2**	**97.7643**
20	0.25	97.6861
20	0.3	97.6704
20	0.35	97.5766
20	0.4	97.3265
20	0.5	96.6073
20	0.75	95.6535
20	1	93.9337
30	0.01	94.9813
30	0.02	95.9351
30	0.05	96.9200
30	0.1	97.5298
30	**0.15**	**97.7017**
30	0.2	97.6392
30	0.25	97.5923
30	0.3	97.5766
30	0.35	97.5141
30	0.4	97.2952
30	0.5	96.6385
30	0.75	95.6379
30	1	93.9337

C	Gamma	Accuracy
50	0.01	95.4191
50	0.02	96.3103
50	0.05	97.2171
50	0.1	97.6392
50	0.15	97.6079
50	0.2	97.5454
50	0.25	97.5610
50	0.3	97.5297
50	0.35	97.4203
50	0.4	97.2170
50	0.5	96.5760
50	0.75	95.6692
50	1	93.9494
75	0.01	95.6379
75	0.02	96.5604
75	0.05	97.3578
75	0.1	97.6079
75	0.15	97.5297
75	0.2	97.6235
75	0.25	97.5297
75	0.3	97.4984
75	0.35	97.4359
75	0.4	97.2014
75	0.5	96.5916
75	0.75	95.6692
75	1	93.9337
100	0.01	95.8569
100	0.02	96.6699
100	0.05	97.4203
100	0.1	97.5454
100	0.15	97.4984
100	0.2	97.6548
100	0.25	97.5141
100	0.3	97.4828
100	0.35	97.4046
100	0.4	97.1857
100	0.5	96.6073
100	0.75	95.6692
100	1	93.9494
150	0.01	96.0289
150	0.02	96.7324
150	0.05	97.3890
150	0.1	97.4047
150	0.15	97.4984
150	0.2	97.5453
150	0.25	97.4984
150	0.3	97.4515
150	0.35	97.4203
150	0.4	97.2170
150	0.5	96.6229
150	0.75	95.6692
150	1	93.9181
200	0.01	96.2165
200	0.02	96.8262
200	0.05	97.4046
200	0.1	97.4047
200	0.15	97.4828
200	0.2	97.4984
200	0.25	97.4828
200	0.3	97.4359
200	0.35	97.4359
200	0.4	97.2170
200	0.5	96.6229
200	0.75	95.6692
200	1	93.9337

Now we take a look at the *Table 4*. I tested the C parameter for values: 1, 2, 5, 7, 10, 13, 15, 17, 20, 30, 50, 75, 100, 150 and 200 while Gamma parameter was having values: 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4 and 0.45.

Little explanation for the table:

- Lines in red color have accuracy more than 97.5 %
- **Bold lines in darkred color have accuracy more than 97.7 %**

Best accuracy I could get was 97.7643 % with C = 20 and Gamma = 0.2. But my humble opinion is that C = 13 and Gamma = 0.3 is a better parameter choice. I am having this impression because this part of the table has the most accuracy scores over 97.7 % => C = 13 and Gamma = 0.25, 0.3, 0.35 (see the table).

Conclusion

There are a lot more into SVM, but I think this is maximum I could do for now. There are even more parameters in `sklearn.svm.SVC` I was leaving by default. It could be investigated deeper.

Best result I reached are:

- **97.7643 % with C = 20 and Gamma = 0.2**
- **97.7330 % with C = 13 and Gamma = 0.3**

Reports and confusion matrices or every fold are in the files:

- report_conf_matrix_c20_g02.txt
- report_conf_matrix_c13_g03.txt

Github

https://github.com/emanuelzaymus/ActivityRecognition

Possible visualization of the data

I tried to visualize the dataset based on splitting the features into two groups:

- Count of all ITEM sensors
- Count of all MOTION sensors

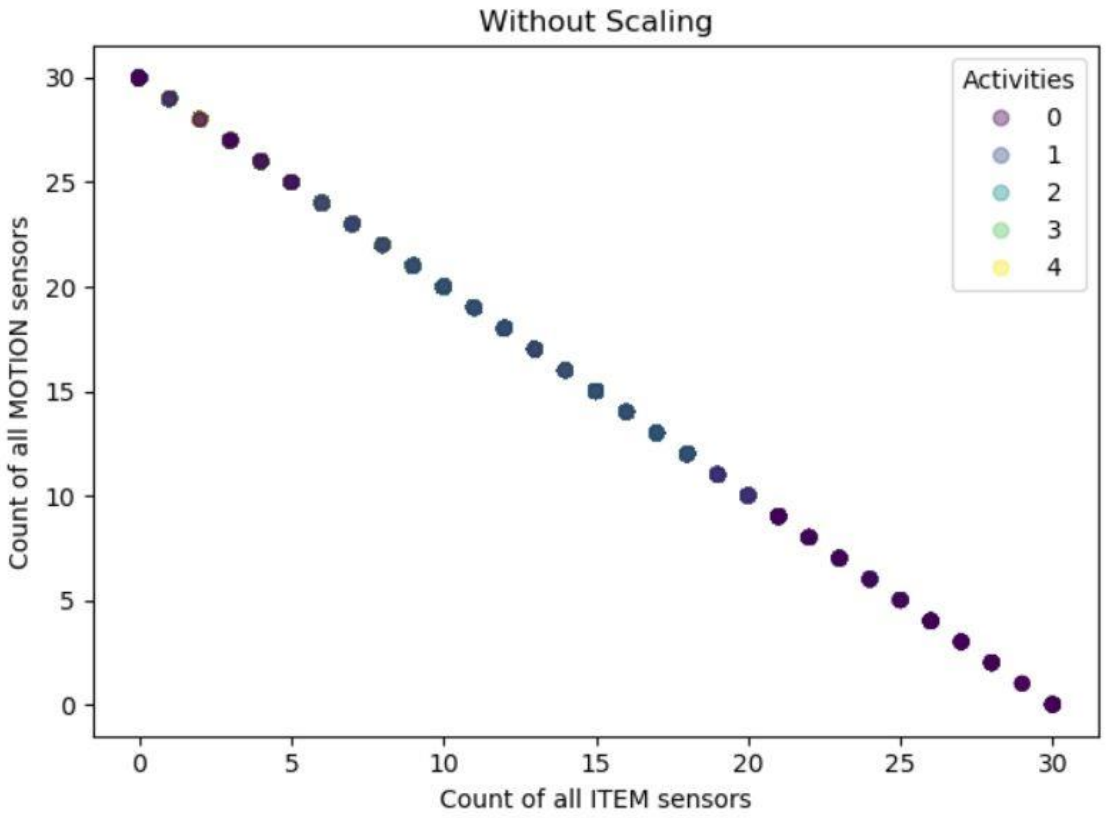
Used features are created with window size of 30.

Tried also various scaling options by `sklearn.preprocessing`: `StandardScaler`, `RobustScaler`, `MinMaxScaler` and `MaxAbsScaler`

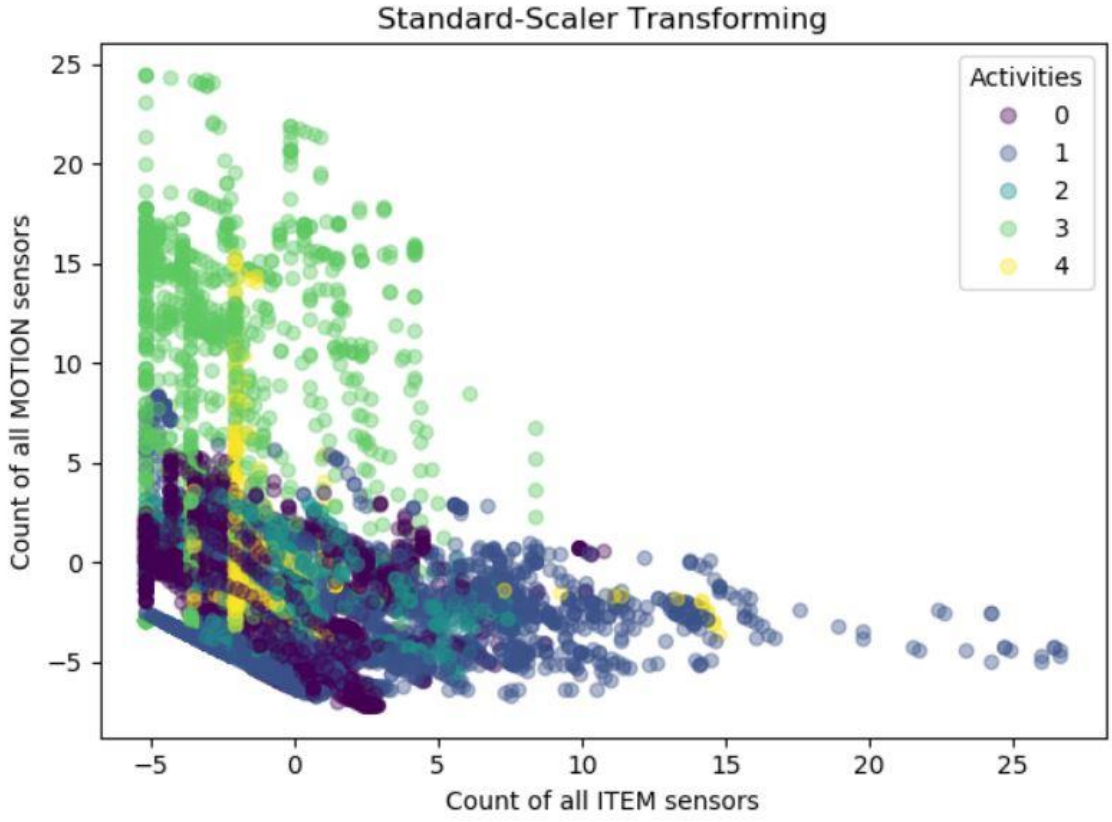
Explanation for legend – Activities:

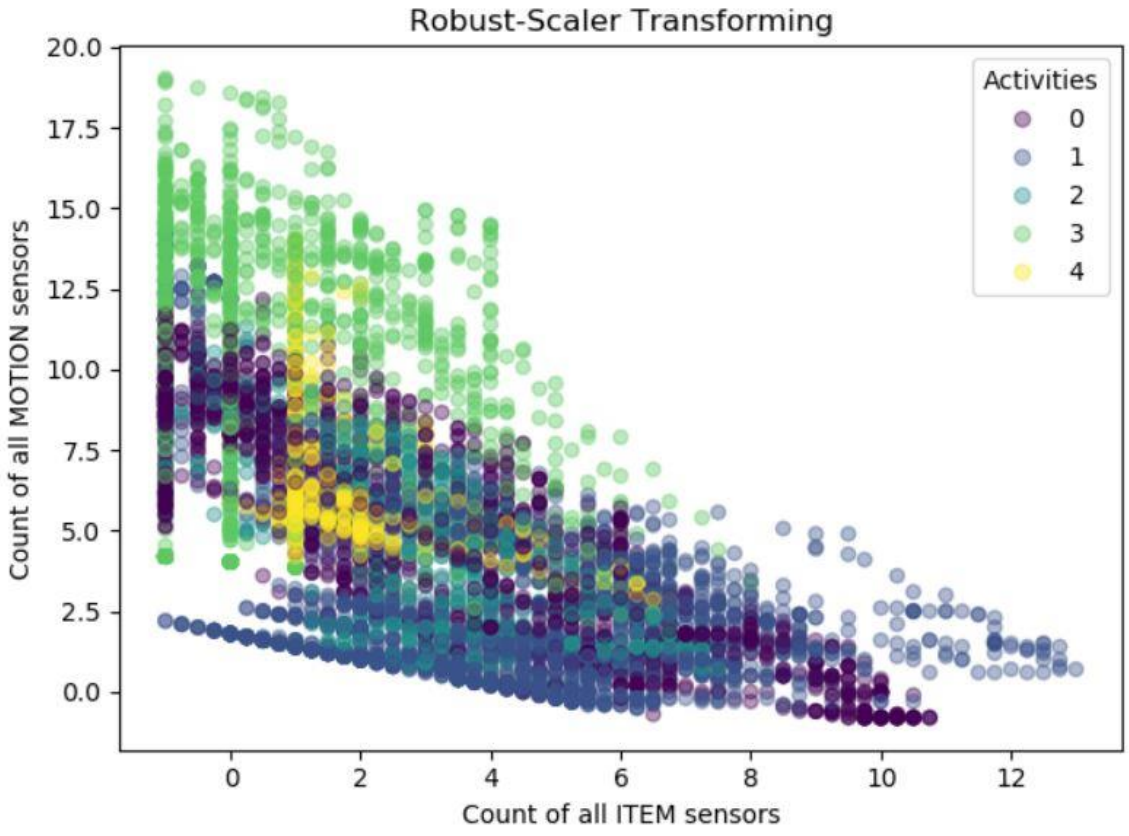
- 0 – Clean
- 1 – Cook
- 2 – Eat
- 3 – Phone_Call
- 4 – Wash_Hand

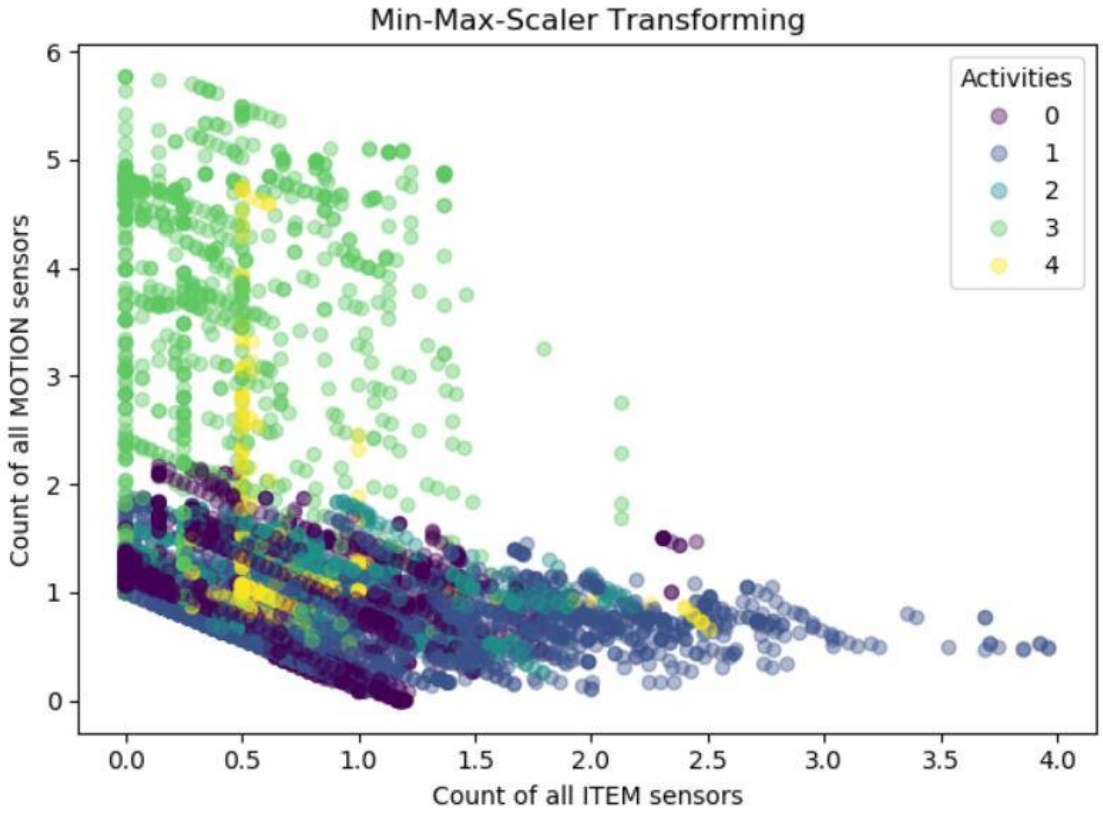
I used these numbers from 0 to 4 to represent the real activities in the feature vectors as well.



Data without scaling, of course, creates a line *x + y = 30* because window size is equal to 30.







Min-Max Scaler was giving exactly the same graph like Max-Abs Scaler.