Support Vector Machines – Preprocessing

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)

The last time I created feature vectors without distinguishing different praticipants what was causing huge values of SECONDS ELAPSED feature or also negative value of SECONDS ELAPSED feature – when samples were not ordered by time properly.

This time is feature extraction done in the right way. Feature vectors are made separately – a participant by participant.

Used library

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

I was focusing on preprocessing of data (feature vectors), so I left the SVC classifier with default settings/parameters: `SVC(kernel='rbf', C=1.0, gamma='scale')`

Testing

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

Window Size and Preprocessing

Normalization – `sklearn.preprocessing.Normalizer(norm='l2')`

From scikit-learn documentation:

Normalization is the process of **scaling individual samples to have unit norm**. This process can be useful if you plan to use a quadratic form such as the dot-product or any other kernel to quantify the similarity of any pair of samples.

Source: https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-normalization

Norms for the **`Normalizer`** are 'l1', 'l2' or 'max'. Explication of the different norms:

The options lead to different normalizations. if x is the vector of covariates of length n, and say that the normalized vector is $y = x/z$ then the three options denote what to use for z:

- L1: $z = \|x\|_1 = \sum_{i=1}^{n} |x_i|$
- L2: $z = \|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$
- Max: $z = \max x_i$

Source from: https://stats.stackexchange.com/questions/225564/scikit-learn-normalization-mode-l1-vs-l2-max

Table 1 – Comparison of accuracies of normalized features

Window size	Accuracy score (%)			
	No preprocessing	Normalized (L1)	Normalized (L2)	Normalized (MAX)
5	35.5507	35.5507	35.5507	35.5507
7	35.8224	35.8224	35.8224	35.8224
10	36.2378	36.2378	36.2378	36.2378
12	36.5201	36.5201	36.5201	36.5201
15	36.9516	36.9516	36.9516	36.9516
17	37.2455	37.2455	37.2455	37.2455
19	37.5438	37.5438	37.5438	37.5438
22	38.0003	38.0003	38.0003	38.0003
25	38.4681	38.4681	38.4681	38.4681
27	38.7863	38.7863	38.7863	38.7863
30	39.2734	39.2734	39.2734	39.2734
32	39.6057	39.6057	39.6057	39.6057
35	40.1139	40.1139	40.1139	40.1139
37	40.4606	40.4606	40.4606	40.4606
40	**40.9909**	**40.9909**	**40.9909**	**40.9909**

Every normalization changes the data in a different way but the accuracy score does not change.

Standardization

From scikit-learn documentation:

Standardization of datasets is **a common requirement for many machine learning estimators** implemented in scikit-learn; they might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with **zero mean and unit variance**.

In practice we often ignore the shape of the distribution and just transform the data to center it by removing the mean value of each feature, then scale it by dividing non-constant features by their standard deviation.

For instance, many elements used in the objective function of a learning algorithm (such as the RBF kernel of Support Vector Machines or the l1 and l2 regularizers of linear models) assume that all features are centered around zero and have variance in the same order. If a feature has a variance that

is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

Source: https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-scaler

Standard Scaler – `sklearn.preprocessing.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`.

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

Defaults: `with_mean=True` and `with_std=True`.

Robust Scaler – `sklearn.preprocessing.RobustScaler`

Scale features using statistics that are robust to outliers.

This Scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).

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

Defaults: `with_centering=True, with_scaling=True, quantile_range=(25.0, 75.0)`

Min-Max Scaler – `sklearn.preprocessing.MinMaxScaler`

Transform features by scaling each feature to a given range.

This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

The transformation is given by:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

where min, max = feature_range.

The transformation is calculated as:

```
X_scaled = scale * X + min - X.min(axis=0) * scale
where scale = (max - min) / (X.max(axis=0) - X.min(axis=0))
```

This transformation is often used as an alternative to zero mean, unit variance scaling.

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

Maximum-Absolute Value Scaler – `sklearn.preprocessing.MaxAbsScaler`

Scale each feature by its maximum absolute value.

This estimator scales and translates each feature individually such that the maximal absolute value of each feature in the training set will be 1.0. It does not shift/center the data, and thus does not destroy any sparsity.

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

Table 2 – Comparation of accuracies of scaled features

Window size	Accuracy score (%)				
	No preprocessing	Standard Scaler	Robust Scaler	Min-Max Scaler	Max-Abs Scaler
5	35.5507	75.9992	76.4258	74.9564	74.3086
7	35.8224	80.2579	79.8122	78.9365	77.9812
10	36.2378	84.2808	83.5237	82.2837	81.2368
12	36.5201	87.2264	85.9928	85.1650	84.1912
15	36.9516	89.7520	88.5859	87.7317	86.2372
17	37.2455	91.0777	90.0183	88.9258	87.4029
19	37.5438	91.7236	91.5733	89.6379	88.4530
22	38.0003	93.0081	92.9236	91.0996	90.0019
25	38.4681	93.6060	93.6744	92.0160	91.3663
27	38.7863	**94.0354**	**94.3113**	92.5356	91.8289
30	39.2734	**94.4842**	**94.3097**	93.2449	92.3722
32	39.6057	**94.6137**	**95.0008**	93.4695	92.5541
35	40.1139	**95.0970**	**95.1328**	93.6529	92.9398
37	40.4606	**95.2350**	**95.4687**	93.8142	93.1489
40	40.9909	**95.2086**	**95.6094**	93.8787	92.7674

Best results get features after Standard Scaling and Robust Scaling.

Combination of Normalization and Standardization

As scikit-learn documentation says, dataset should be normally distributed before standardization. Source: https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-scaler

I decided firstly normalized data and after then scale/standardize them.

I used `sklearn.preprocessing.Normalizer(norm='l2')` with default parameters. Then I scaled the data:

Table 3 – Accuracies – Scaling features after normalization

Window size	Accuracy score (%)			
	Standard Scaler	Robust Scaler	Min-Max Scaler	Max-Abs Scaler
5	75.6043	45.6155	73.5660	73.4396
7	79.5893	60.1497	76.6439	76.4529
10	83.6526	67.1442	80.1415	79.9644
12	86.5286	73.2677	83.0551	82.9902
15	89.0623	76.9582	85.8267	85.5803
17	90.5314	79.4074	87.4195	87.0554
19	91.1061	83.5305	88.3864	88.3364
22	92.5013	84.8167	89.7654	89.4108
25	93.2469	85.4165	91.1953	90.9731
27	93.8630	86.2781	91.5358	91.4496
30	**94.2746**	86.8039	92.2674	91.9008
32	**94.5609**	87.2027	92.5013	91.9204
35	**95.2040**	88.1440	92.6901	92.5476
37	**95.1990**	92.6815	92.8073	92.5915
40	**95.1722**	92.9680	92.6217	92.1662

In general, the accuracy dropped a little bit.

Conclusion

Normalization has impact only on internal representation of the data, but not for the final accuracy of the SVM.

Standardization has a great effect on the accuracy.

It is important to have in mind that **I did not tune any parameter of the SVM**. Most likely after parameter tuning we would reach success rate more than 97-98 % or even higher.

Github

https://github.com/emanuelzaymus/ActivityRecognition