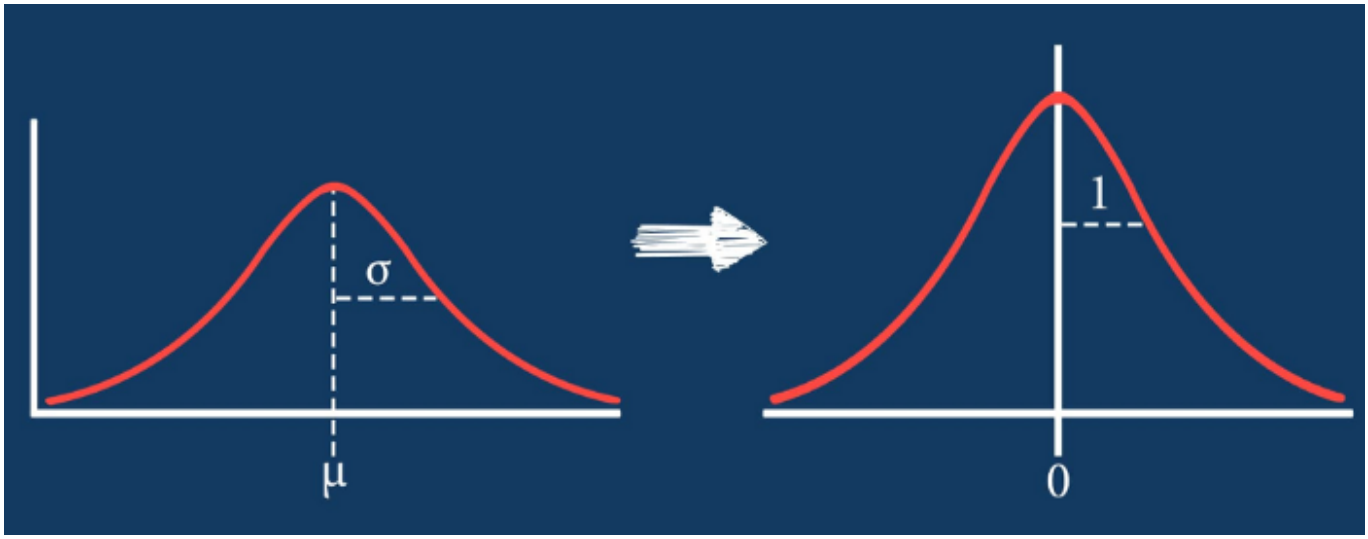
Normalization vs Standardization — Quantitative analysis

Stop using StandardScaler from Sklearn as a default feature scaling method can get you a boost of 7% in accuracy, even when you hyperparameters are tuned!

Shay Geller Follow
Apr 4 · 13 min read ★



https://365datascience.com/standardization/

Every ML practitioner knows that feature scaling is an important issue (read more here).

The two most discussed scaling methods are Normalization and Standardization. ***Normalization*** typically means rescales the values into a range of [0,1]. ***Standardization*** typically means rescales data to have a mean of 0 and a standard deviation of 1 (unit variance).

In this blog, I conducted a few experiments and hope to answer questions like:

1. Should we always scale our features?
2. Is there a single best scaling technique?
3. How different scaling techniques affect different classifiers?
4. Should we consider scaling technique as an important hyperparameter of our model?

I'll analyze the empirical results of applying different scaling methods on features in multiple experiments settings.

Table of Contests

0. Why are we here?

First, I was trying to understand what is the difference between Normalization and Standardization.
So, I encountered this excellent blog by Sebastian Raschka that supplies a mathematical background that satisfied my curiosity. **Please take 5 minutes to read this blog if you**

are not familiar with Normalization or Standardization concepts.
There is also a great explanation of the need for scaling features when dealing with classifiers that trained using gradient descendent methods(like neural networks) by famous Hinton here.

Ok, we grabbed some math, that's it? Not quite.
When I checked the popular ML library Sklearn, I saw that there are lots of different scaling methods. There is a great visualization of the effect of different scalers on data with outliers. But they didn't show how it affects classification tasks with different classifiers.

I saw a lot of ML pipelines tutorials that use StandardScaler (usually called Z-score Standardization) or MinMaxScaler (usually called min-max Normalization) to scale features. Why does no one use other scaling techniques for classification? Is it possible that StandardScaler or MinMaxScaler are the best scaling methods?
I didn't see any explanation in the tutorials about why or when to use each one of them, so I thought I'd investigate the performance of these techniques by running some experiments. **This is what this notebook is all about**

Project details

Like many Data Science projects, lets read some data and experiment with several out-of-the-box classifiers.

Dataset

Sonar dataset. It contains 208 rows and 60 feature columns. It's a classification task to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock.

	0	1	2	3	4	5	6	7	8	9	...	51	52	53	54	55	56	57	58	59	60
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0027	0.0065	0.0159	0.0072	0.0167	0.0180	0.0084	0.0090	0.0032	R
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.0084	0.0089	0.0048	0.0094	0.0191	0.0140	0.0049	0.0052	0.0044	R
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.0232	0.0166	0.0095	0.0180	0.0244	0.0316	0.0164	0.0095	0.0078	R
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.0121	0.0036	0.0150	0.0085	0.0073	0.0050	0.0044	0.0040	0.0117	R
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.0031	0.0054	0.0105	0.0110	0.0015	0.0072	0.0048	0.0107	0.0094	R
5	0.0286	0.0453	0.0277	0.0174	0.0384	0.0990	0.1201	0.1833	0.2105	0.3039	...	0.0045	0.0014	0.0038	0.0013	0.0089	0.0057	0.0027	0.0051	0.0062	R
6	0.0317	0.0956	0.1321	0.1408	0.1674	0.1710	0.0731	0.1401	0.2083	0.3513	...	0.0201	0.0248	0.0131	0.0070	0.0138	0.0092	0.0143	0.0036	0.0103	R
7	0.0519	0.0548	0.0842	0.0319	0.1158	0.0922	0.1027	0.0613	0.1465	0.2838	...	0.0081	0.0120	0.0045	0.0121	0.0097	0.0085	0.0047	0.0048	0.0053	R
8	0.0223	0.0375	0.0484	0.0475	0.0647	0.0591	0.0753	0.0098	0.0684	0.1487	...	0.0145	0.0128	0.0145	0.0058	0.0049	0.0065	0.0093	0.0059	0.0022	R
9	0.0164	0.0173	0.0347	0.0070	0.0187	0.0671	0.1056	0.0697	0.0962	0.0251	...	0.0090	0.0223	0.0179	0.0084	0.0068	0.0032	0.0035	0.0056	0.0040	R

It's a balanced dataset:

```
sonar[60].value_counts() # 60 is the label column name

M    111
R     97
```

All the features in this dataset are between 0 to 1, **but** it's not ensured that 1 is the max value or 0 is the min value in each feature.

I chose this dataset because, from one hand, it is small, so I can experiment pretty fast. On the other hand, it's a hard problem and none of the classifiers achieve anything close to 100% accuracy, so we can compare meaningful results.

We will experiment with more datasets in the last section.

Code

As a preprocessing step, I already calculated all the results (it takes some time). So we only load the results file and work with it.

The code that produces the results can be found in my GitHub:
https://github.com/shaygeller/Normalization_vs_Standardization.git

I pick some of the most popular classification models from Sklearn, denoted as:

Name	Sklearn_Class
LR	LogisticRegression
LDA	LinearDiscriminantAnalysis
KNN	KNeighborsClassifier
CART	DecisionTreeClassifier
NB	GaussianNB
SVM	SVC
RF	RandomForestClassifier
MLP	MLPClassifier

(MLP is Multi-Layer Perceptron, a neural network)

The scalers I used are denoted as:

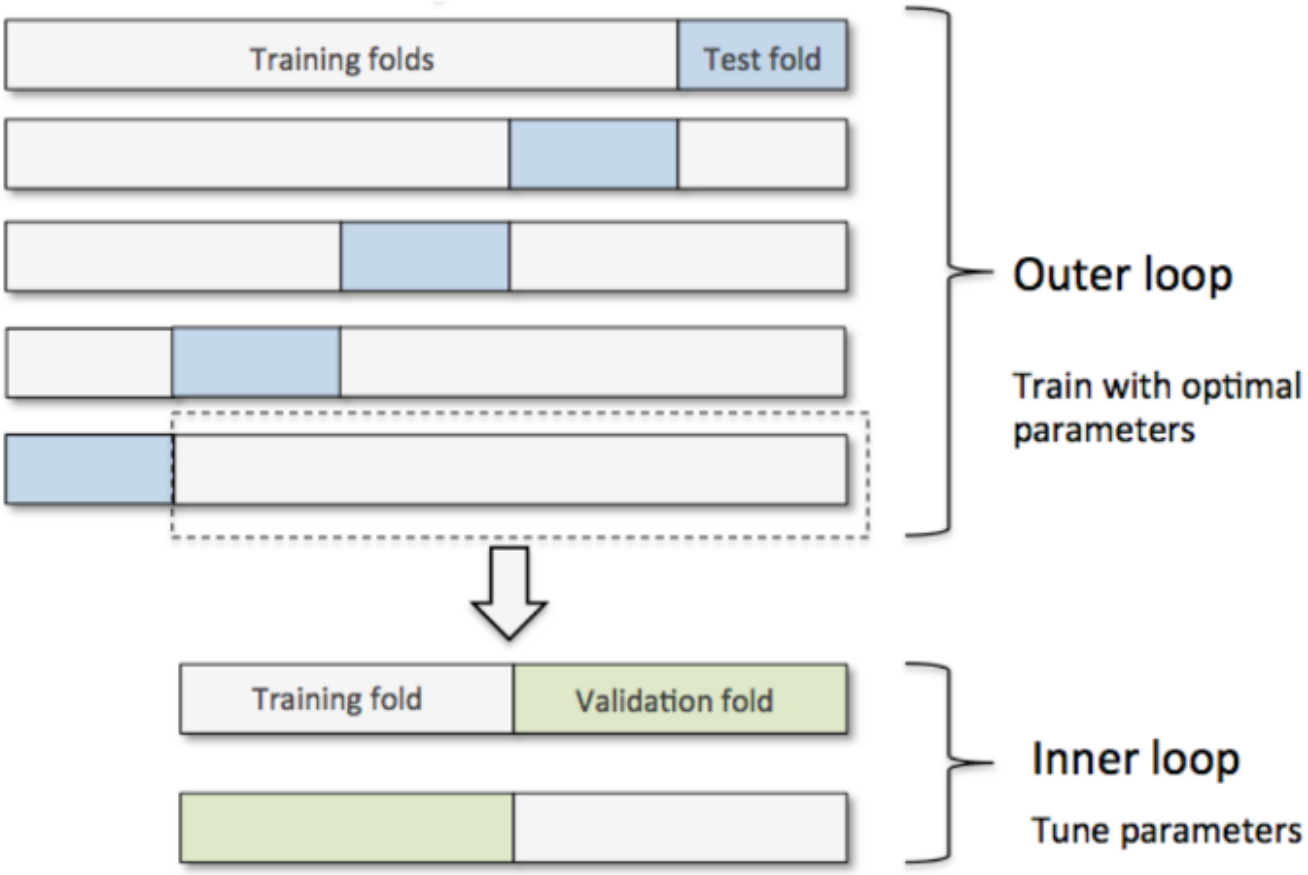
Name	Sklearn_Class
StandardScaler	StandardScaler
MinMaxScaler	MinMaxScaler
MaxAbsScaler	MaxAbsScaler
RobustScaler	RobustScaler
QuantileTransformer-Normal	QuantileTransformer(output_distribution='normal')
QuantileTransformer-Uniform	QuantileTransformer(output_distribution='uniform')
PowerTransformer-Yeo-Johnson	PowerTransformer(method='yeo-johnson')
Normalizer	Normalizer

*Do not confuse Normalizer, the last scaler in the list above with the min-max normalization technique I discussed before. The min-max normalization is the second in the list and named MinMaxScaler. The Normalizer class from Sklearn normalizes samples individually to unit norm. **It is not column based but a row based normalization technique.**

Experiment details:

- The same seed was used when needed for reproducibility.
- I randomly split the data to train-test sets of 80%-20% respectively.
- All results are accuracy scores on 10-fold random cross-validation splits from the **train set**.
- I do not discuss the results on the test set here. Usually, the test set should be kept hidden, and all of our conclusions about our classifiers should be taken only from the cross-validation scores.
- In part 4, I performed nested cross-validation. One inner cross-validation with 5 random splits for hyperparameter tuning, and another outer CV with 10 random

splits to get the model's score using the best parameters. Also in this part, all data taken only from the train set. A picture is worth a thousand words:



https://sebastianraschka.com/faq/docs/evaluate-a-model.html

Let's read the results file

```
import os
import pandas as pd

results_file = "sonar_results.csv"
results_df =
pd.read_csv(os.path.join("..","data","processed",results_file)).dropn
a().round(3)
results_df
```

1. Out-of-the-box classifiers

```
import operator

results_df.loc[operator.and_(results_df["Classifier_Name"].str.starts
with("_"),
~results_df["Classifier_Name"].str.endswith("PCA"))].dropna()
```

	Dataset	Classifier_Name	CV_mean	CV_std	Test_acc
0	sonar	_LR	0.754	0.114	0.714
20	sonar	_LDA	0.699	0.132	0.833
40	sonar	_KNN	0.753	0.081	0.762
60	sonar	_CART	0.735	0.067	0.810
80	sonar	_NB	0.657	0.131	0.690
100	sonar	_SVM	0.608	0.116	0.738
120	sonar	_RF	0.710	0.126	0.810
140	sonar	_MLP	0.778	0.125	0.833

Nice results. By looking at the CV_mean column, we can see that at the moment, MLP is leading. SVM has the worst performance.

Standard deviation is pretty much the same, so we can judge mainly by the mean score. All the results below will be the mean score of 10-fold cross-validation random splits.

Now, let's see how different scaling methods change the scores for each classifier

2. Classifiers+Scaling

```
import operator
temp =
results_df.loc[~results_df["Classifier_Name"].str.endswith("PCA")].dr
opna()
temp["model"] = results_df["Classifier_Name"].apply(lambda sen:
sen.split("_")[1])
temp["scaler"] = results_df["Classifier_Name"].apply(lambda sen:
sen.split("_")[0])

def df_style(val):
    return 'font-weight: 800'
```

```
pivot_t = pd.pivot_table(temp, values='CV_mean', index=["scaler"],
columns=['model'], aggfunc=np.sum)
pivot_t_bold = pivot_t.style.applymap(df_style,

subset=pd.IndexSlice[pivot_t["CART"].idxmax(),"CART"])
for col in list(pivot_t):
    pivot_t_bold = pivot_t_bold.applymap(df_style,

subset=pd.IndexSlice[pivot_t[col].idxmax(),col])
pivot_t_bold
```

model scaler	CART	KNN	LDA	LR	MLP	NB	RF	SVM
	0.735	0.753	0.699	0.754	0.778	0.657	0.71	0.608
MaxAbsScaler	0.735	0.808	0.699	0.778	0.767	0.657	0.71	0.705
MinMaxScaler	0.735	**0.813**	0.699	0.778	0.767	0.657	0.71	0.711
Normalizer	0.716	0.765	0.692	0.699	0.724	0.662	**0.723**	0.524
PowerTransformer-Yeo-Johnson	0.729	0.813	**0.747**	0.76	**0.839**	**0.752**	0.71	0.814
QuantileTransformer-Normal	0.735	0.783	0.694	0.718	0.808	0.752	0.717	0.832
QuantileTransformer-Uniform	**0.74**	0.789	0.742	**0.808**	0.808	0.752	0.717	0.772
RobustScaler	0.735	0.76	0.699	0.735	0.808	0.657	0.71	0.776
StandardScaler	0.735	0.796	0.699	0.742	0.82	0.657	0.71	**0.849**

The first row, the one without index name, is the algorithm without applying any scaling method.

```
import operator

cols_max_vals = {}
cols_max_row_names = {}
for col in list(pivot_t):
    row_name = pivot_t[col].idxmax()
    cell_val = pivot_t[col].max()
    cols_max_vals[col] = cell_val
    cols_max_row_names[col] = row_name

sorted_cols_max_vals = sorted(cols_max_vals.items(), key=lambda kv:
kv[1], reverse=True)

print("Best classifiers sorted:\n")
counter = 1
```

```
for model, score in sorted_cols_max_vals:
    print(str(counter) + ". " + model + " + "
+cols_max_row_names[model] + " : " +str(score))
    counter +=1
```

Best classifier from each model:

1. SVM + StandardScaler : 0.849
2. MLP + PowerTransformer-Yeo-Johnson : 0.839
3. KNN + MinMaxScaler : 0.813
4. LR + QuantileTransformer-Uniform : 0.808
5. NB + PowerTransformer-Yeo-Johnson : 0.752
6. LDA + PowerTransformer-Yeo-Johnson : 0.747
7. CART + QuantileTransformer-Uniform : 0.74
8. RF + Normalizer : 0.723

Let's analyze the results

1. **There is no single scaling method to rule them all.**

2. We can see that scaling improved the results. SVM, MLP, KNN, and NB got a significant boost from different scaling methods.

3. Notice that NB, RF, LDA, CART are unaffected **by some** of the scaling methods. This is, of course, related to how each of the classifiers works. Trees are not affected by scaling because the splitting criterion first orders the values of each feature and then calculate the gini\entropy of the split. Some scaling methods keep this order, so no change to the accuracy score.
   NB is not affected because the model's priors determined by the count in each class and not by the actual value. Linear Discriminant Analysis (LDA) finds it's coefficients using the variation between the classes (check this), so the scaling doesn't matter either.

4. Some of the scaling methods, like QuantileTransformer-Uniform, doesn't preserve the exact order of the values in each feature, hence the change in score even in the above classifiers that were agnostic to other scaling methods.

3. Classifier+Scaling+PCA

We know that some well-known ML methods like PCA can benefit from scaling (blog). Let's try adding PCA(n_components=4) to the pipeline and analyze the results.

```
import operator
temp = results_df.copy()
temp["model"] = results_df["Classifier_Name"].apply(lambda sen:
sen.split("_")[1])
temp["scaler"] = results_df["Classifier_Name"].apply(lambda sen:
sen.split("_")[0])

def df_style(val):
    return 'font-weight: 800'

pivot_t = pd.pivot_table(temp, values='CV_mean', index=["scaler"],
columns=['model'], aggfunc=np.sum)
pivot_t_bold = pivot_t.style.applymap(df_style,

subset=pd.IndexSlice[pivot_t["CART"].idxmax(),"CART"])
for col in list(pivot_t):
    pivot_t_bold = pivot_t_bold.applymap(df_style,

subset=pd.IndexSlice[pivot_t[col].idxmax(),col])
pivot_t_bold
```

model	CART	CART-PCA	KNN	KNN-PCA	LDA	LDA-PCA	LR	LR-PCA	MLP	MLP-PCA	NB	NB-PCA	RF	RF-PCA	SVM	SVM-PCA
scaler																
	0.735	0.657	0.753	0.742	0.699	0.704	0.754	0.699	0.778	0.705	0.657	0.699	0.71	0.711	0.608	0.699
MaxAbsScaler	0.735	0.717	0.808	**0.777**	0.699	0.759	0.778	0.759	0.767	0.741	0.657	0.753	0.71	0.729	0.705	0.759
MinMaxScaler	0.735	**0.742**	**0.813**	0.759	0.699	0.753	0.778	0.753	0.767	0.747	0.657	0.753	0.71	0.728	0.711	0.759
Normalizer	0.716	0.571	0.765	0.742	0.692	0.642	0.699	0.631	0.724	0.619	0.662	0.638	**0.723**	0.643	0.524	0.589
PowerTransformer-Yeo-Johnson	0.729	0.71	0.813	0.777	**0.747**	0.782	0.76	0.782	**0.839**	0.77	**0.752**	**0.758**	0.71	0.71	0.814	0.74
QuantileTransformer-Normal	0.735	0.706	0.783	0.741	0.694	0.753	0.718	0.741	0.808	0.729	0.752	0.723	0.717	**0.766**	0.832	0.729
QuantileTransformer-Uniform	**0.74**	0.71	0.789	0.74	0.742	**0.783**	**0.808**	**0.789**	0.808	0.776	0.752	0.735	0.717	0.668	0.772	0.747
RobustScaler	0.735	0.715	0.76	0.758	0.699	0.734	0.735	0.734	0.808	0.771	0.657	0.661	0.71	0.703	0.776	**0.788**
StandardScaler	0.735	0.692	0.796	0.753	0.699	0.741	0.742	0.753	0.82	**0.777**	0.657	0.752	0.71	0.675	**0.849**	0.776

Let's analyze the results

1. Most of the time scaling methods improve models with PCA, **but** no specific scaling method is in charge.
   Let's look at "QuantileTransformer-Uniform", the method with most of the high

scores.
In LDA-PCA it improved the results from 0.704 to 0.783 (8% jump in accuracy!), but in RF-PCA it makes things worse, from 0.711 to 0.668 (4.35% drop in accuracy!)
On the other hand, using RF-PCA with "QuantileTransformer-Normal", improved the accuracy to 0.766 (5% jump in accuracy!)

2. We can see that PCA only improve LDA and RF, so PCA is not a magic solution.
   It's fine. We didn't hypertune the n_components parameter, and even if we did, PCA doesn't guarantee to improve predictions.

3. We can see that StandardScaler and MinMaxScaler achieve best scores only in 4 out of 16 cases. So we should think carefully what scaling method to choose, even as a default one.

We can conclude that even though PCA is a known component that benefits from scaling, no single scaling method always improved our results, and some of them even cause harm(RF-PCA with StandardScaler).

The dataset is also a great factor here. To better understand the consequences of scaling methods on PCA, we should experiment with more diverse datasets (class imbalanced, different scales of features and datasets with numerical and categorical features). I'm doing this analysis in section 5.

4. Classifiers+Scaling+PCA+Hyperparameter tuning

There are big differences in the accuracy score between different scaling methods for a given classifier. One can assume that when the hyperparameters are tuned, the difference between the scaling techniques will be minor and we can use StandardScaler or MinMaxScaler as used in many classification pipelines tutorials in the web.
Let's check that!

model / scaler	CART	CART-PCA	KNN	KNN-PCA	LDA	LDA-PCA	LR	LR-PCA	MLP	MLP-PCA	RF	RF-PCA	SVM	SVM-PCA
	0.734	0.704	0.85	0.771	0.77	0.676	0.76	0.689	0.734	0.651	0.771	0.699	0.789	0.67
MaxAbsScaler	0.734	0.723	0.843	0.741	0.765	0.76	0.736	0.766	0.728	**0.754**	0.776	0.758	0.759	0.743
MinMaxScaler	0.734	0.657	0.837	0.753	0.782	0.754	0.711	0.766	0.746	0.742	0.776	0.722	0.735	0.737
PowerTransformer-Yeo-Johnson	0.65	**0.74**	0.874	0.777	0.778	0.771	0.789	0.777	**0.807**	0.681	0.776	0.772	**0.873**	0.728
QuantileTransformer-Normal	0.64	0.687	0.806	0.731	0.771	**0.778**	0.795	0.766	0.694	0.735	**0.795**	**0.773**	0.837	0.735
QuantileTransformer-Uniform	0.651	0.717	**0.891**	0.771	**0.783**	0.765	**0.814**	**0.778**	0.758	0.741	0.746	0.741	0.814	0.753

RobustScaler	0.734	0.686	0.838	**0.789**	0.776	0.73	0.758	0.724	0.783	0.71	0.776	0.711	0.771	**0.759**
StandardScaler	0.734	0.736	0.825	0.776	0.783	0.753	0.741	0.748	0.777	0.742	0.776	0.722	0.861	0.718

First, NB is not here, that's because NB has no parameters to tune.

We can see that almost all the algorithms benefit from hyperparameter tuning compare to results from o previous step. An interesting exception is MLP that got worse results. It's probably because neural networks can easily overfit the data (especially when the number of parameters is much bigger than the number of training samples), and we didn't perform a careful early stopping to avoid it, nor applied any regularizations.

Yet, even when the hyperparameters are tuned, there are still big differences between the results using different scaling methods. If we would compare different scaling techniques to the broadly used StandardScaler technique, we can **gain up to 7% improvement in accuracy** (KNN column) when experiencing with other techniques.

The main conclusion from this step is that even though the hyperparameters are tuned, changing the scaling method can dramatically affect the results. So, we should consider the scaling method as a crucial hyperparameter of our model.

Part 5 contains a more in-depth analysis of more diverse datasets. If you don't want to deep dive into it, feel free to jump to the conclusion section.

5. All again on more datasets

To get a better understanding and to derive more generalized conclusions, we should experiment with more datasets.

We will apply Classifier+Scaling+PCA like section 3 on several datasets with different characteristics and analyze the results. All datasets were taken from Kaggel.

- For the sake of convenience, I selected only the numerical columns out of each dataset. In multivariate datasets (numeric and categorical features), there is an ongoing debate about how to scale the features.
- I didn't hypertune any parameters of the classifiers.

5.1 Rain in Australia dataset

Link

Classification task: Predict is it's going to rain?

Metric: Accuracy

Dataset shape: (56420, 18)

Counts for each class:

No 43993

Yes 12427

Here is a sample of 5 rows, we can't show all the columns in one picture.

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am
5939	17.9	35.2	0.0	12.0	12.3	48.0	6.0	20.0	20.0	13.0	1006.3
5940	18.4	28.9	0.0	14.8	13.0	37.0	19.0	19.0	30.0	8.0	1012.9
5942	19.4	37.6	0.0	10.8	10.6	46.0	30.0	15.0	42.0	22.0	1012.3
5943	21.9	38.4	0.0	11.4	12.2	31.0	6.0	6.0	37.0	22.0	1012.7
5944	24.2	41.0	0.0	11.2	8.4	35.0	17.0	13.0	19.0	15.0	1010.7

```
dataset.describe()
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm
count	56420.000000	56420.000000	56420.000000	56420.000000	56420.000000	56420.000000	56420.000000	56420.000000	56420.000000	56420.000000
mean	13.464770	24.219206	2.130397	5.503135	7.735626	40.877366	15.667228	19.786778	65.874123	49.601985
std	6.416689	6.970676	7.014822	3.696282	3.758153	13.335232	8.317005	8.510180	18.513289	20.197040
min	-6.700000	4.100000	0.000000	0.000000	0.000000	9.000000	2.000000	2.000000	0.000000	0.000000
25%	8.600000	18.700000	0.000000	2.800000	5.000000	31.000000	9.000000	13.000000	55.000000	35.000000
50%	13.200000	23.900000	0.000000	5.000000	8.600000	39.000000	15.000000	19.000000	67.000000	50.000000
75%	18.400000	29.700000	0.600000	7.400000	10.700000	48.000000	20.000000	26.000000	79.000000	63.000000
max	31.400000	48.100000	206.200000	81.200000	14.500000	124.000000	67.000000	76.000000	100.000000	100.000000

We will suspect that scaling will improve classification results due to the different scales of the features (check min max values in the above table, it even get worse on some of the rest of the features).

Results

model	CART	CART-PCA	KNN	KNN-PCA	LDA	LDA-PCA	LR	LR-PCA	MLP	MLP-PCA	NB	NB-PCA	RF	RF-PCA	SVM	SVM-PCA
scaler																
	1	0.78	0.89	0.836	0.88	0.85	**1**	0.85	0.992	0.851	0.95	0.844	**0.97**	0.831	0.78	0.78
MaxAbsScaler	1	0.753	0.84	0.808	0.88	0.819	0.887	0.821	0.998	0.827	0.95	0.814	0.97	0.809	0.868	0.82
MinMaxScaler	1	0.759	0.845	0.814	0.88	0.824	0.887	0.825	0.998	0.83	0.95	0.823	0.97	0.801	0.869	0.824
Normalizer	1	0.78	0.893	0.834	0.884	0.848	0.827	0.821	0.999	0.849	0.942	0.844	0.948	0.839	0.78	0.781

PowerTransformer-Yeo-Johnson	1	0.818	0.974	0.858	**0.961**	0.869	1	0.872	**1**	0.873	**0.978**	0.864	0.97	0.861	0.997	0.873
QuantileTransformer-Normal	1	0.943	0.917	0.954	0.884	0.872	1	0.954	0.999	0.965	0.891	**0.893**	0.97	0.907	0.985	0.965
QuantileTransformer-Uniform	1	0.844	0.919	0.878	0.884	**0.885**	0.994	0.885	0.999	0.888	0.916	0.88	0.97	0.859	0.994	0.886
RobustScaler	1	**0.997**	**0.991**	**0.995**	0.88	0.86	1	**1**	1	**1**	0.95	0.871	0.97	**0.947**	**0.998**	**0.998**
StandardScaler	1	0.792	0.89	0.839	0.88	0.853	0.994	0.853	1	0.855	0.95	0.834	0.97	0.846	0.978	0.854

Results analysis

- We can see the StandardScaler never got the highest score, nor MinMaxScaler.
- We can see **differences of up to 20%** between StandardScaler and other methods. (CART-PCA column)
- We can see that scaling usually improved the results. Take for example SVM that **jumped from 78% to 99%.**

5.2 Bank Marketing dataset

Link
Classification task: Predict has the client subscribed a term deposit?
Metric: AUC **(The data is imbalanced)**
Dataset shape: (41188, 11)
Counts for each class:
no 36548
yes 4640

Here is a sample of 5 rows, we can't show all the columns in one picture.

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
0	56	261	1	999	0	1.1	93.994	-36.4	4.857	5191.0
1	57	149	1	999	0	1.1	93.994	-36.4	4.857	5191.0
2	37	226	1	999	0	1.1	93.994	-36.4	4.857	5191.0
3	40	151	1	999	0	1.1	93.994	-36.4	4.857	5191.0
4	56	307	1	999	0	1.1	93.994	-36.4	4.857	5191.0

```
dataset.describe()
```

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	93.575664	-40.502600	3.621291	5167.035911
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	0.578840	4.628198	1.734447	72.251528
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.634000	4963.600000
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.344000	5099.100000
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000	5191.000000
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000	5228.100000
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.045000	5228.100000

Again, features in different scales.

Results

model scaler	CART	CART-PCA	KNN	KNN-PCA	LDA	LDA-PCA	LR	LR-PCA	MLP	MLP-PCA	NB	NB-PCA	RF	RF-PCA	SVM	SVM-PCA
	0.734	0.71	0.873	0.87	0.922	**0.921**	0.921	**0.92**	0.912	0.91	0.868	**0.919**	0.91	**0.915**	0.868	**0.855**
MaxAbsScaler	0.734	0.686	**0.884**	0.825	0.922	0.748	0.917	0.747	0.937	0.782	0.868	0.758	0.91	0.767	0.924	0.627
MinMaxScaler	0.734	0.68	0.882	0.843	0.922	0.747	0.921	0.747	0.943	0.786	0.868	0.753	0.91	0.776	**0.93**	0.618
Normalizer	0.728	0.706	0.869	0.864	**0.923**	0.851	0.854	0.853	0.924	0.892	**0.889**	0.857	**0.921**	0.902	0.605	0.578
PowerTransformer-Yeo-Johnson	0.734	0.707	0.883	0.868	0.913	0.854	0.922	0.86	**0.944**	0.919	0.848	0.832	0.908	0.861	0.896	0.832
QuantileTransformer-Normal	**0.735**	0.7	0.883	0.821	0.902	0.701	0.908	0.726	0.942	0.79	0.839	0.741	0.909	0.783	0.919	0.806
QuantileTransformer-Uniform	0.735	0.691	0.882	0.851	0.894	0.746	0.9	0.749	0.942	0.9	0.833	0.751	0.909	0.791	0.916	0.669
RobustScaler	0.734	**0.72**	0.884	**0.874**	0.922	0.909	**0.923**	0.908	0.935	0.908	0.868	0.889	0.91	0.907	0.904	0.851
StandardScaler	0.735	0.713	0.884	0.866	0.922	0.892	0.923	0.9	0.944	**0.92**	0.868	0.873	0.91	0.881	0.898	0.83

Results analysis

- We can see that in this dataset, even though the features are on different scales, scaling when using PCA doesn't always improve the results. **However,** the second-best score in each PCA column is pretty close to the best score. It might indicate that hypertune the number of components of the PCA and using scaling will improve the results over not scaling at all.
- Again, there is no one single scaling method that stood out.
- Another interesting result is that in most models, all the scaling methods didn't affect that much (usually 1%–3% improvement). Let's remember that this is an unbalanced dataset and we didn't hypertune the parameters. Another reason is that the AUC score is already high (~90%), so it's harder to see major improvements.

5.3 Sloan Digital Sky Survey DR14 dataset

Link

Classification task: Predict if an object to be either a galaxy, star or quasar.

Metric: Accuracy (multiclass)

Dataset shape: (10000, 18)

Counts for each class:

GALAXY 4998

STAR 4152

QSO 850

Here is a sample of 5 rows, we can't show all the columns in one picture.

	objid	ra	dec	u	g	r	i	z	run	rerun	camcol	field	specobjid	redshift	plate	mjd
0	1.237650e+18	183.531326	0.089693	19.47406	17.04240	15.94699	15.50342	15.22531	752	301	4	267	3.722360e+18	-0.000009	3306	54922
1	1.237650e+18	183.598371	0.135285	18.66280	17.21449	16.67637	16.48922	16.39150	752	301	4	267	3.638140e+17	-0.000055	323	51615
2	1.237650e+18	183.680207	0.126185	19.38298	18.19169	17.47428	17.08732	16.80125	752	301	4	268	3.232740e+17	0.123111	287	52023
3	1.237650e+18	183.870529	0.049911	17.76536	16.60272	16.16116	15.98233	15.90438	752	301	4	269	3.722370e+18	-0.000111	3306	54922
4	1.237650e+18	183.883288	0.102557	17.55025	16.26342	16.43869	16.55492	16.61326	752	301	4	269	3.722370e+18	0.000590	3306	54922

```
dataset.describe()
```

	objid	ra	dec	u	g	r	i	z	run	rerun	camcol
count	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.0	10000.000000
mean	1.237650e+18	175.529987	14.836148	18.619355	17.371931	16.840963	16.583579	16.422833	981.034800	301.0	3.648700
std	0.000000e+00	47.783439	25.212207	0.828656	0.945457	1.067764	1.141805	1.203188	273.305024	0.0	1.666183
min	1.237650e+18	8.235100	-5.382632	12.988970	12.799550	12.431600	11.947210	11.610410	308.000000	301.0	1.000000
25%	1.237650e+18	157.370946	-0.539035	18.178035	16.815100	16.173333	15.853705	15.618285	752.000000	301.0	2.000000
50%	1.237650e+18	180.394514	0.404166	18.853095	17.495135	16.858770	16.554985	16.389945	756.000000	301.0	4.000000
75%	1.237650e+18	201.547279	35.649397	19.259232	18.010145	17.512675	17.258550	17.141447	1331.000000	301.0	5.000000
max	1.237650e+18	260.884382	68.542265	19.599900	19.918970	24.802040	28.179630	22.833060	1412.000000	301.0	6.000000

Again, features in different scales.

Results

model	CART	CART-PCA	KNN	KNN-PCA	LDA	LDA-PCA	LR	LR-PCA	MLP	MLP-PCA	NB	NB-PCA	RF	RF-PCA	SVM	SVM-PCA
scaler																
	0.985	0.707	0.783	0.782	0.919	0.786	0.796	0.796	0.616	0.737	0.796	0.796	**0.933**	0.798	0.534	0.501
MaxAbsScaler	0.985	0.718	0.869	0.778	0.912	0.778	0.875	0.797	0.974	0.797	0.942	0.768	0.933	0.748	0.857	0.797
MinMaxScaler	0.985	0.746	0.877	0.805	0.912	0.791	0.895	0.797	0.978	0.799	0.942	0.782	0.933	0.789	0.862	0.797
Normalizer	0.719	0.718	0.782	0.782	0.905	0.798	0.798	0.798	0.798	0.798	0.798	0.794	0.798	0.798	0.796	0.796

PowerTransformer-Yeo-Johnson	0.984	0.803	**0.956**	0.822	0.963	0.624	**0.978**	0.625	0.988	0.874	**0.985**	0.629	0.867	0.676	**0.985**	0.829
QuantileTransformer-Normal	0.985	0.818	0.944	0.86	**0.97**	0.825	0.971	0.822	0.985	0.858	0.925	0.835	0.933	0.813	0.977	0.861
QuantileTransformer-Uniform	0.985	0.818	0.955	0.866	0.968	0.813	0.955	0.821	0.984	0.864	0.924	0.826	0.933	0.83	0.975	0.846
RobustScaler	0.985	**0.971**	0.953	**0.97**	0.912	**0.853**	0.975	**0.957**	**0.989**	**0.985**	0.942	**0.88**	0.933	**0.854**	0.982	**0.98**
StandardScaler	0.985	0.804	0.904	0.846	0.912	0.838	0.955	0.847	0.985	0.861	0.942	0.84	0.933	0.851	0.954	0.856

Results analysis

- We can see that scaling highly improved the results. We could expect it because it contains features on different scales.
- We can see that RobustScaler almost always wins when we use PCA. It might be due to the many outliers in this dataset that shift the PCA eigenvectors. On the other hand, those outliers don't make such an effect when we do not use PCA. We should do some data exploration to check that.
- There is up to 5% difference in accuracy if we will compare StandardScaler to the other scaling method. So it's another indicator to the need for experiment with multiple scaling techniques.
- PCA almost always benefit from scaling.

5.4 Income classification dataset

Link
Classification task: Predict if income is >50K, <=50K.
Metric: AUC **(imbalanced dataset)**
Dataset shape: (32561, 7)
Counts for each class:
<=50K 24720
>50K 7841

Here is a sample of 5 rows, we can't show all the columns in one picture.

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
0	39	77516	13	2174	0	40
1	50	83311	13	0	0	13
2	38	215646	9	0	0	40

3	53	234721	7	0	0	40
4	28	338409	13	0	0	40

```
dataset.describe()
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

Again, features in different scales.

Results

model / scaler	CART	CART-PCA	KNN	KNN-PCA	LDA	LDA-PCA	LR	LR-PCA	MLP	MLP-PCA	NB	NB-PCA	RF	RF-PCA	SVM	SVM-PCA
	0.695	0.659	0.662	0.662	0.822	0.751	0.583	0.764	0.602	0.644	0.824	0.768	**0.836**	0.772	0.582	0.586
MaxAbsScaler	**0.696**	0.673	0.786	0.775	0.822	0.808	0.831	0.808	0.852	**0.853**	**0.831**	0.812	0.836	0.834	0.836	0.818
MinMaxScaler	0.696	0.673	0.783	0.772	0.822	0.807	0.831	0.808	0.852	0.853	0.831	0.813	0.836	**0.837**	**0.84**	**0.819**
Normalizer	0.621	0.609	0.696	0.641	0.676	0.683	0.684	0.683	0.685	0.685	0.68	0.631	0.701	0.684	0.684	0.684
PowerTransformer-Yeo-Johnson	0.679	0.639	0.776	0.77	0.826	**0.827**	0.829	0.827	0.843	0.838	0.826	0.816	0.836	0.813	0.809	0.8
QuantileTransformer-Normal	0.696	0.679	0.786	**0.79**	0.823	0.819	0.829	0.825	**0.856**	0.849	0.825	0.815	0.836	0.831	0.822	0.813
QuantileTransformer-Uniform	0.696	0.646	0.778	0.773	**0.83**	0.827	**0.833**	**0.829**	0.845	0.839	0.828	**0.822**	0.836	0.814	0.819	0.811
RobustScaler	0.696	**0.703**	**0.821**	0.79	0.822	0.746	0.833	0.756	0.797	0.72	0.83	0.753	0.836	0.802	0.823	0.688
StandardScaler	0.695	0.662	0.792	0.779	0.822	0.823	0.833	0.828	0.856	0.85	0.831	0.769	0.836	0.8	0.817	0.809

Results analysis

- Here again, we have an imbalanced dataset, but we can see that scaling do a good job in improving the results (up to 20%!). This is probably because the AUC score is

lower (~80%) compared to the Bank Marketing dataset, so it's easier to see major improvements.

- Even though StandardScaler is not highlighted (I highlighted only the first best score in each column), in many columns, it achieves the same results as the best, but not always. From the running time results(no appeared here), I can tell you that running StandatdScaler is much faster than many of the other scalers. So if you are in a rush to get some results, it can be a good starting point. But if you want to squeeze every percent from your model, you might want to experience with multiple scaling methods.
- Again, no single best scale method.
- PCA almost always benefited from scaling

Conclusions

- Experiment with multiple scaling methods can dramatically increase your score on classification tasks, even when you hyperparameters are tuned. **So, you should consider the scaling method as an important hyperparameter of your model.**
- Scaling methods affect differently on different classifiers. Distance-based classifiers like SVM, KNN, and MLP(neural network) dramatically benefit from scaling. But even trees (CART, RF), that are agnostic to some of the scaling methods, can benefit from other methods.
- Knowing the underlying math behind models\preprocessing methods is the best way to understand the results. (For example, how trees work and why some of the scaling methods didn't affect them). It can also save you a lot of time if you know no to apply StandardScaler when your model is Random Forest.
- Preprocessing methods like PCA that known to be benefited from scaling, do benefit from scaling. **When it doesn't**, it might be due to a bad setup of the number of components parameter of PCA, outliers in the data or a bad choice of a scaling method.

If you find some mistakes or have proposals to improve the coverage or the validity of the experiments, please notify me.

Thanks to Dvir Cohen and Nicholas Hoernle.

Machine Learning Towards Data Science AI Data Science Data Preprocessing

About Help Legal