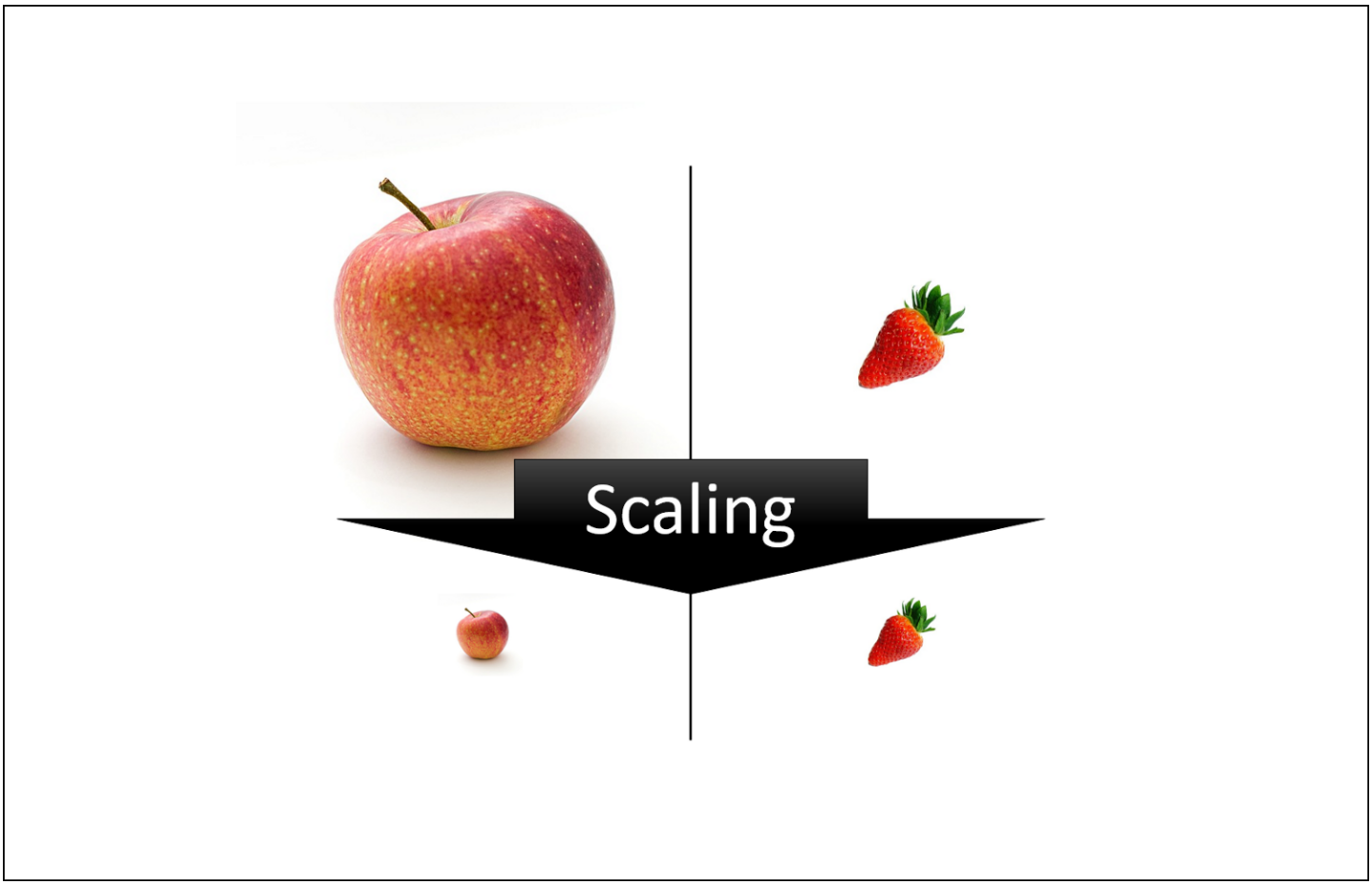


Apple & Strawberry Image Credit: Pixabay

IN DEPTH ANALYSIS

All about Feature Scaling

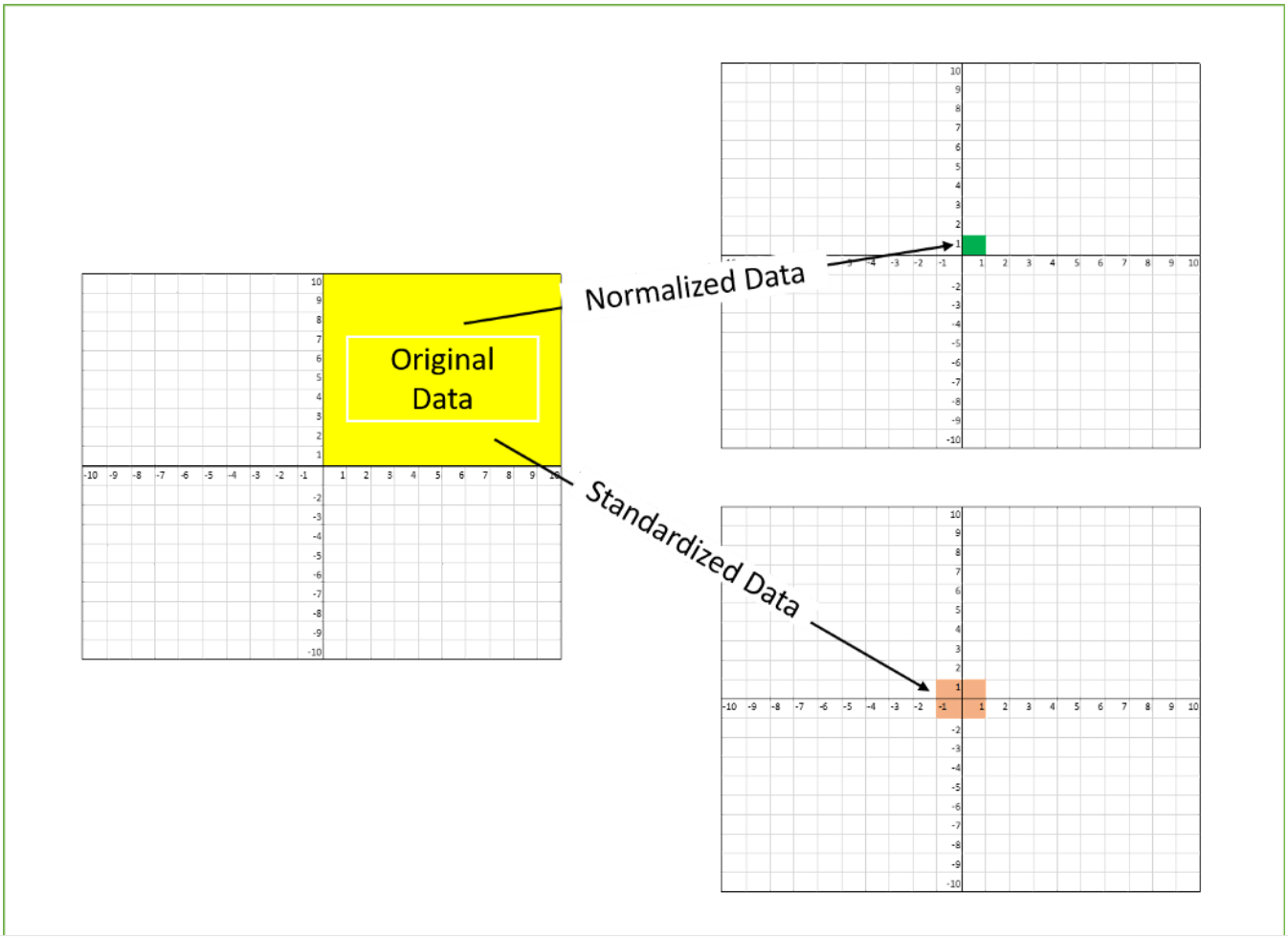
Scale data for better performance of Machine Learning Model

Machine learning is like making a mixed fruit juice. If we want to get the best-mixed juice, we need to mix all fruit not by their size but based on their right proportion. We just need to remember apple and strawberry are not the same unless we make them similar in some context to compare their attribute. Similarly, in many machine learning algorithms, to bring all features in the same standing, we need to do scaling so that one significant number doesn't impact the model just because of their large magnitude.

Feature scaling in machine learning is one of the most critical steps during the pre-processing of data before creating a machine learning model. Scaling can make a difference between a weak machine learning model and a better one.

The most common techniques of feature scaling are Normalization and Standardization.

Normalization is used when we want to bound our values between two numbers, typically, between [0,1] or [-1,1]. While Standardization transforms the data to have zero mean and a variance of 1, they make our data **unitless**. Refer to the below diagram, which shows how data looks after scaling in the X-Y plane.



Machine learning algorithm just sees number — if there is a vast difference in the range say few ranging in thousands and few ranging in the tens, and it makes the underlying assumption that higher ranging numbers have superiority of some sort. So these more significant number starts playing a more decisive role while training the model.

The machine learning algorithm works on numbers and does not know what that number represents. A weight of 10 grams and a price of 10 dollars represents completely two different things — which is a no brainer for humans, but for a model as a feature, it treats both as same.

Suppose we have two features of weight and price, as in the below table. The "Weight" cannot have a meaningful comparison with the "Price." So the assumption algorithm makes that since "Weight" > "Price," thus "Weight," is more important than "Price."

Name	Weight	Price
Orange	15	1
Apple	18	3
Banana	12	2
Grape	10	5

So these more significant number starts playing a more decisive role while training the model. Thus feature scaling is needed to bring every feature in the same footing without any upfront importance. Interestingly, if we convert the weight to "Kg," then "Price" becomes dominant.

Another reason why feature scaling is applied is that few algorithms like Neural network gradient descent **converge much faster** with feature scaling than without it.

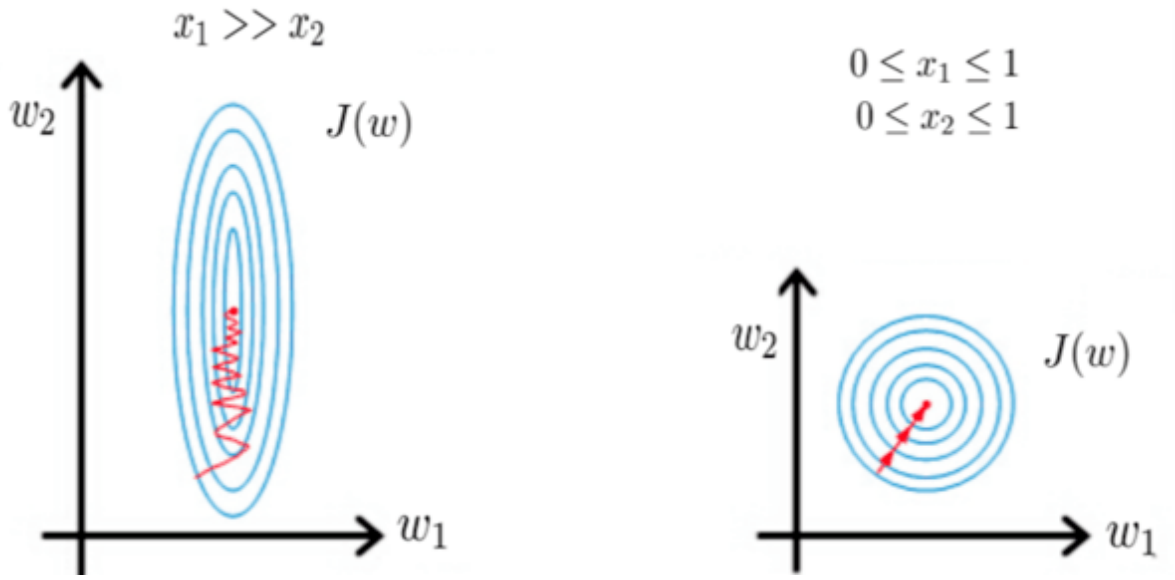


Photo Credit

One more reason is **saturation**, like in the case of sigmoid activation in Neural Network, scaling would help not to saturate too fast.

When to do scaling?

Feature scaling is essential for machine learning algorithms that calculate **distances between data**. If not scale, the feature with a higher value range starts dominating when calculating distances, as explained intuitively in the "why?" section.

The ML algorithm is sensitive to the "**relative scales of features,**" which usually happens when it uses the numeric values of the features rather than say their rank.

In many algorithms, when we desire **faster convergence**, scaling is a MUST like in Neural Network.

Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions do not work correctly without normalization. For example, the majority of classifiers calculate the distance between two points by the distance. If one of the features has a broad range of values, the distance governs this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

Even when the conditions, as mentioned above, are not satisfied, you may still need to rescale your features if the ML algorithm expects some scale or a saturation

Rule of thumb we may follow here is an algorithm that computes distance or assumes normality, **scales your features.**

Some examples of algorithms where feature scaling matters are:

- **K-nearest neighbors** (KNN) with a Euclidean distance measure is sensitive to magnitudes and hence should be scaled for all features to weigh in equally.
- **K-Means** uses the Euclidean distance measure here feature scaling matters.
- Scaling is critical while performing **Principal Component Analysis(PCA)**. PCA tries to get the features with maximum variance, and the variance is high for high magnitude features and skews the PCA towards high magnitude features.
- We can speed up **gradient descent** by scaling because θ descends quickly on small ranges and slowly on large ranges, and oscillates inefficiently down to the optimum when the variables are very uneven.

Algorithms that do not require normalization/scaling are the ones that **rely on rules**. They would not be affected by any monotonic transformations of the variables. Scaling is a monotonic transformation. Examples of algorithms in this category are all the tree-based algorithms — **CART, Random Forests, Gradient Boosted Decision Trees**. These algorithms utilize rules (series of inequalities) and **do not require normalization**.

Algorithms like **Linear Discriminant Analysis(LDA), Naive Bayes is** by design equipped to handle this and give weights to the features accordingly. Performing features scaling in these algorithms may not have much effect.

Few key points to note :

- Mean centering does not affect the covariance matrix
- Scaling of variables does affect the covariance matrix
- Standardizing affects the covariance

How to perform feature scaling?

1) Min Max Scaler
2) Standard Scaler
3) Max Abs Scaler
4) Robust Scaler
5) Quantile Transformer Scaler
6) Power Transformer Scaler
7) Unit Vector Scaler

For the explanation, we will use the table shown in the top and form the data frame to show different scaling methods.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
df = pd.DataFrame({'WEIGHT': [15, 18, 12,10],
                   'PRICE': [1,3,2,5]},
                   index = ['Orange','Apple','Banana','Grape'])
print(df)

WEIGHT  PRICE
Orange      15      1
Apple       18      3
Banana      12      2
Grape       10      5
```

1)Min-Max scaler

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

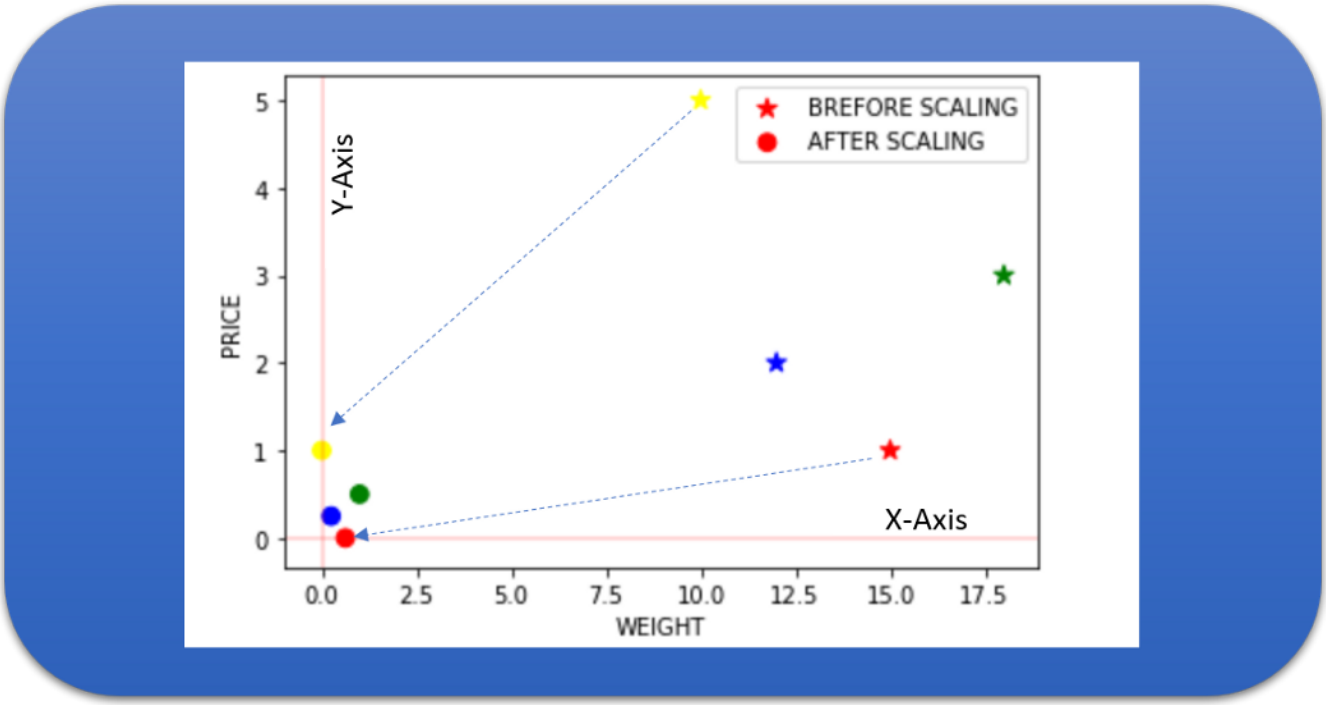
Transform features by scaling each feature to a given range. This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g., between zero and one. This Scaler shrinks the data within the range of -1 to 1 if there are negative values. We can set the range like [0,1] or [0,5] or [-1,1].

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

df1 = pd.DataFrame(scaler.fit_transform(df),
                   columns=['WEIGHT','PRICE'],
                   index = ['Orange','Apple','Banana','Grape'])ax =
df.plot.scatter(x='WEIGHT', y='PRICE',color=
['red','green','blue','yellow'],
                    marker = '*',s=80, label='BREFORE SCALING');

df1.plot.scatter(x='WEIGHT', y='PRICE', color=
['red','green','blue','yellow'],
                  marker = 'o',s=60,label='AFTER SCALING', ax = ax);

plt.axhline(0, color='red',alpha=0.2)
plt.axvline(0, color='red',alpha=0.2);
```



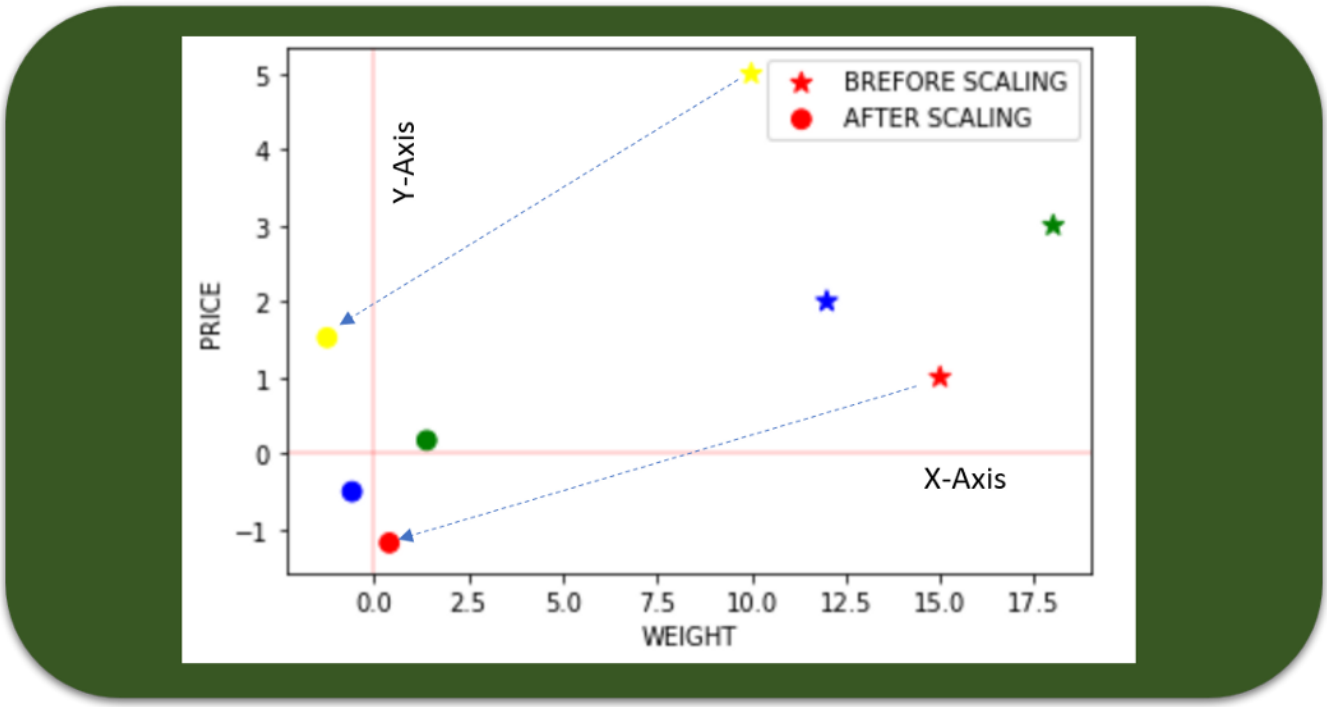
2) Standard Scaler

$$x_{new} = \frac{x - \mu}{\sigma}$$

1.

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. If data is not normally distributed, this is not the best Scaler to use.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df2 = pd.DataFrame(scaler.fit_transform(df),
                   columns=['WEIGHT','PRICE'],
                   index = ['Orange','Apple','Banana','Grape'])
ax = df.plot.scatter(x='WEIGHT', y='PRICE',color=
['red','green','blue','yellow'],
                     marker = '*',s=80, label='BREFORE SCALING');
df2.plot.scatter(x='WEIGHT', y='PRICE', color=
['red','green','blue','yellow'],
                  marker = 'o',s=60,label='AFTER SCALING', ax = ax)
plt.axhline(0, color='red',alpha=0.2)
plt.axvline(0, color='red',alpha=0.2);
```

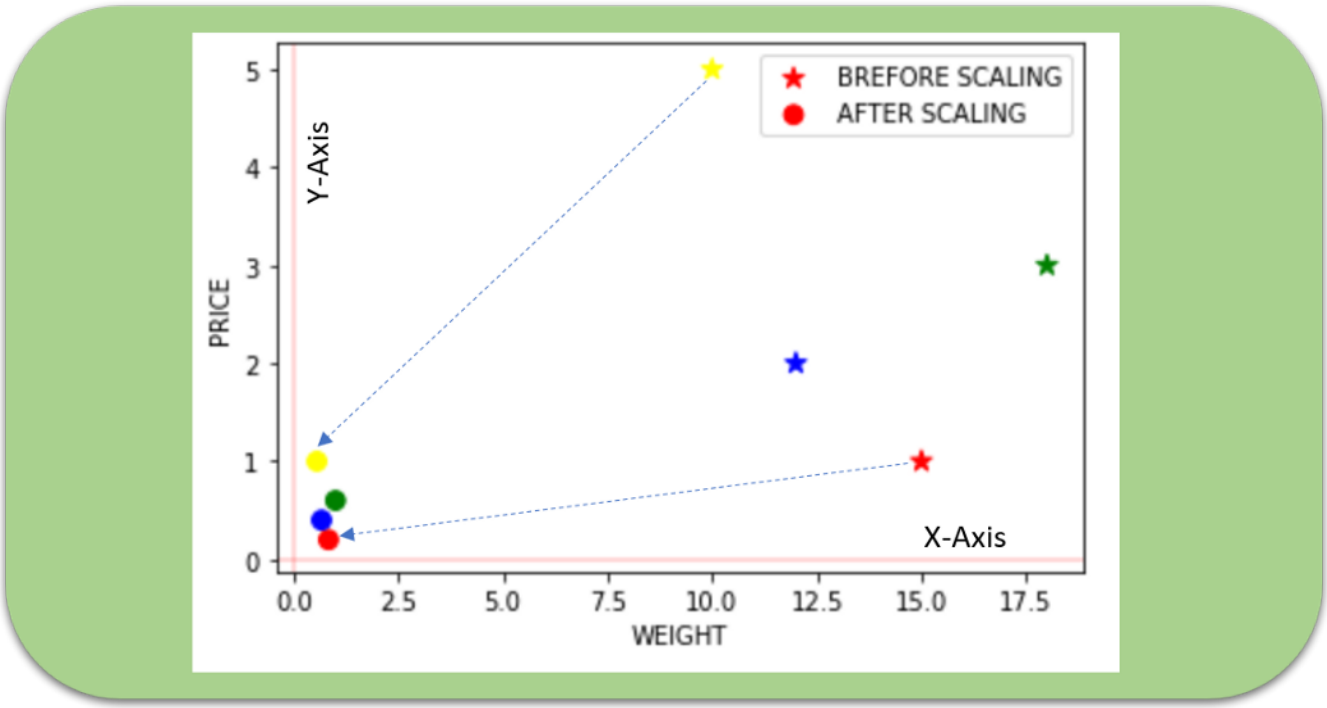


3) Max Abs Scaler

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

On positive-only data, this Scaler behaves similarly to Min Max Scaler and, therefore, also suffers from the presence of significant **outliers**.

```
from sklearn.preprocessing import MaxAbsScaler
scaler = MaxAbsScaler()
df4 = pd.DataFrame(scaler.fit_transform(df),
                   columns=['WEIGHT','PRICE'],
                   index = ['Orange','Apple','Banana','Grape'])
ax = df.plot.scatter(x='WEIGHT', y='PRICE',color=
['red','green','blue','yellow'],
                     marker = '*',s=80, label='BREFORE SCALING');
df4.plot.scatter(x='WEIGHT', y='PRICE', color=
['red','green','blue','yellow'],
                  marker = 'o',s=60,label='AFTER SCALING', ax = ax)
plt.axhline(0, color='red',alpha=0.2)
plt.axvline(0, color='red',alpha=0.2);
```



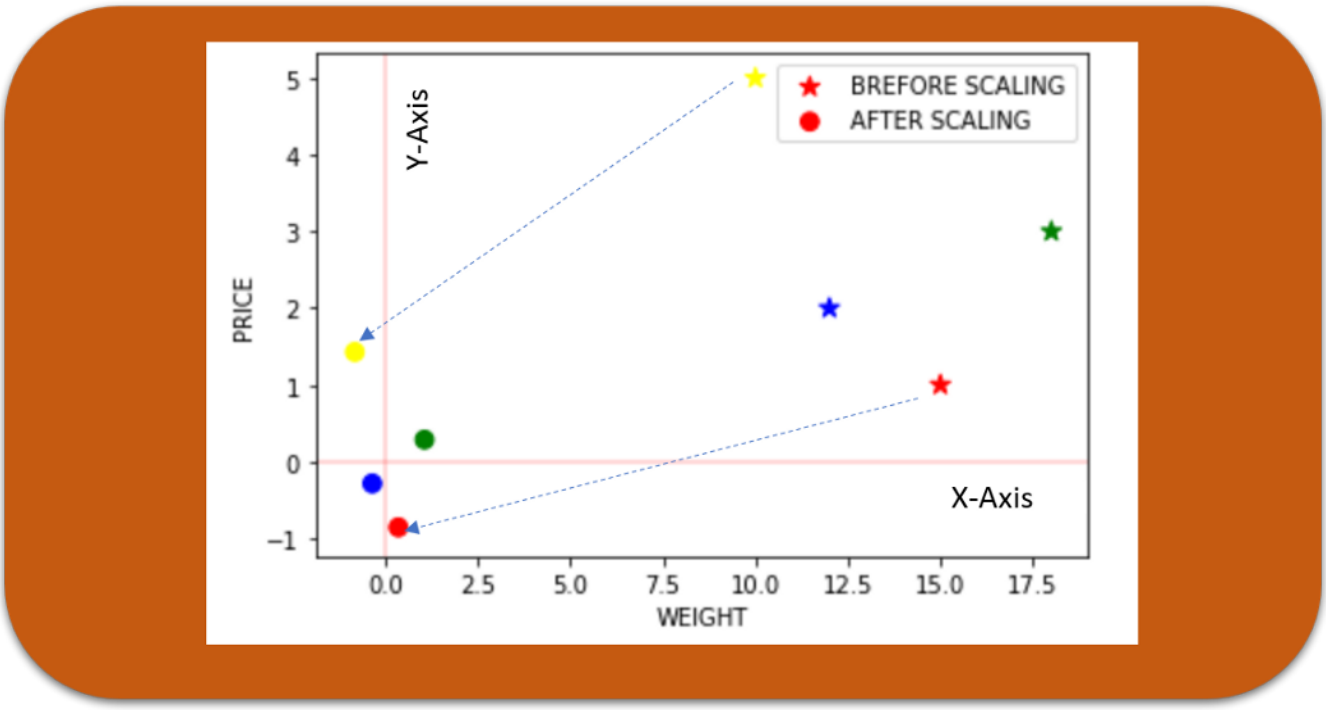
4) Robust Scaler

As the name suggests, this Scaler is **robust** to outliers. If our data contains many **outliers**, scaling using the mean and standard deviation of the data won't work well.

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

few numbers of huge marginal outliers. Note that the outliers themselves are still present in the transformed data. If a separate outlier clipping is desirable, a non-linear transformation is required.

```
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()
df3 = pd.DataFrame(scaler.fit_transform(df),
                   columns=['WEIGHT','PRICE'],
                   index = ['Orange','Apple','Banana','Grape'])
ax = df.plot.scatter(x='WEIGHT', y='PRICE',color=
['red','green','blue','yellow'],
                     marker = '*',s=80, label='BREFORE SCALING');
df3.plot.scatter(x='WEIGHT', y='PRICE', color=
['red','green','blue','yellow'],
                  marker = 'o',s=60,label='AFTER SCALING', ax = ax)
plt.axhline(0, color='red',alpha=0.2)
plt.axvline(0, color='red',alpha=0.2);
```



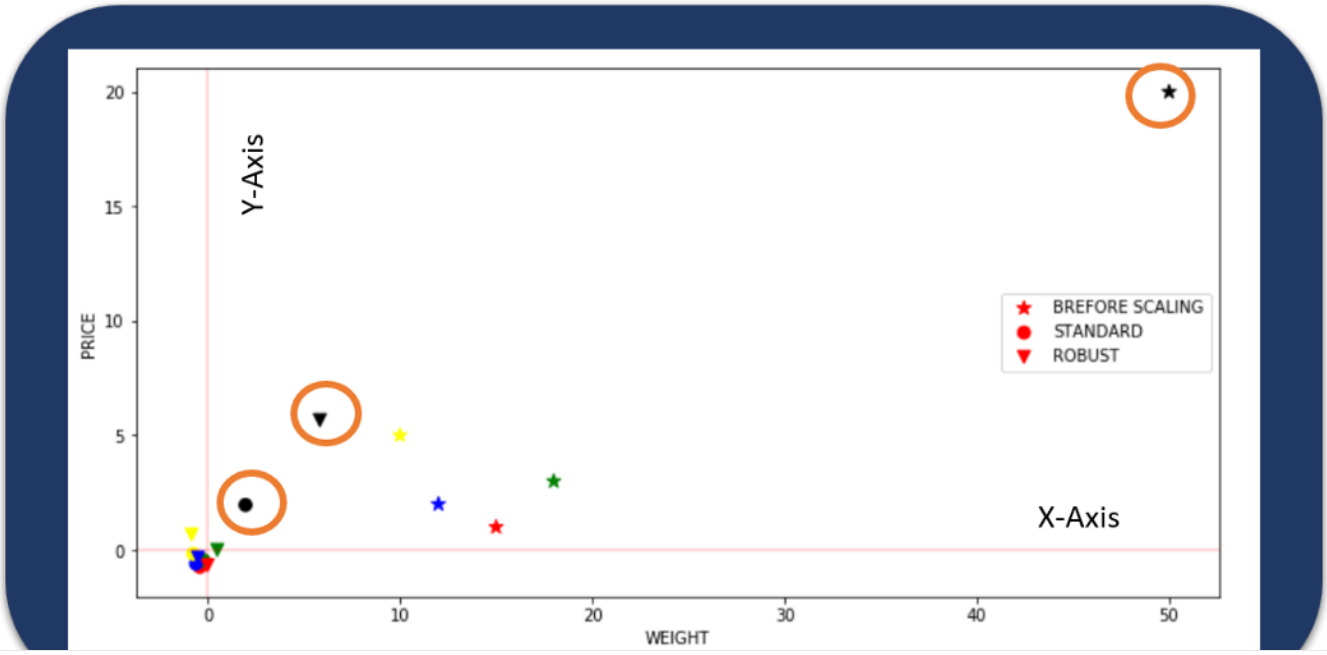
Let's now see what happens if we introduce an outlier and see the effect of scaling using Standard Scaler and Robust Scaler (a circle shows outlier).

```
dfr = pd.DataFrame({'WEIGHT': [15, 18, 12,10,50],
                    'PRICE': [1,3,2,5,20]},
                    index =
```

```
scaler = StandardScaler()
df21 = pd.DataFrame(scaler.fit_transform(dfr),
                    columns=['WEIGHT','PRICE'],
                    index =
['Orange','Apple','Banana','Grape','Jackfruit'])
ax = dfr.plot.scatter(x='WEIGHT', y='PRICE',color=
['red','green','blue','yellow','black'],
                      marker = '*',s=80, label='BREFORE SCALING');
df21.plot.scatter(x='WEIGHT', y='PRICE', color=
['red','green','blue','yellow','black'],
                  marker = 'o',s=60,label='STANDARD', ax =
ax,figsize=(12,6))
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()
df31 = pd.DataFrame(scaler.fit_transform(dfr),
                    columns=['WEIGHT','PRICE'],
                    index =
['Orange','Apple','Banana','Grape','Jackfruit'])

df31.plot.scatter(x='WEIGHT', y='PRICE', color=
['red','green','blue','yellow','black'],
                  marker = 'v',s=60,label='ROBUST', ax = ax,figsize=
(12,6))
plt.axhline(0, color='red',alpha=0.2)
plt.axvline(0, color='red',alpha=0.2);
```

```
WEIGHT  PRICE
Orange         15      1
Apple          18      3
Banana         12      2
Grape          10      5
Jackfruit      50     20
```



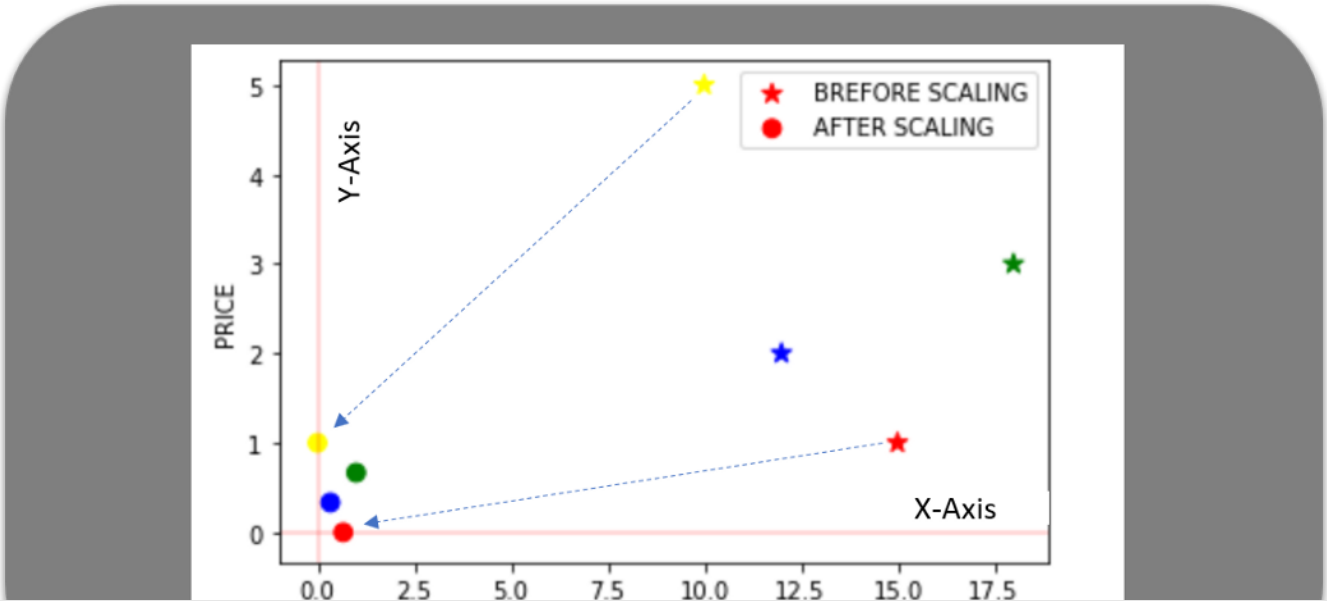
5) Quantile Transformer Scaler

Transform features using quantiles information.

This method transforms the features to follow a **uniform or a normal** distribution. Therefore, for a given feature, this transformation tends to spread out the most frequent values. It also reduces the impact of (marginal) outliers: this is, therefore, a **robust pre-processing** scheme.

The cumulative distribution function of a feature is used to project the original values. Note that this transform is non-linear and may distort linear correlations between variables measured at the same scale but renders variables measured at different scales more directly comparable. This is also sometimes called as **Rank scaler.**

```
from sklearn.preprocessing import QuantileTransformer
scaler = QuantileTransformer()
df6 = pd.DataFrame(scaler.fit_transform(df),
                   columns=['WEIGHT','PRICE'],
                   index = ['Orange','Apple','Banana','Grape'])
ax = df.plot.scatter(x='WEIGHT', y='PRICE',color=
['red','green','blue','yellow'],
                     marker = '*',s=80, label='BREFORE SCALING');
df6.plot.scatter(x='WEIGHT', y='PRICE', color=
['red','green','blue','yellow'],
                  marker = 'o',s=60,label='AFTER SCALING', ax =
ax,figsize=(6,4))
plt.axhline(0, color='red',alpha=0.2)
plt.axvline(0, color='red',alpha=0.2);
```



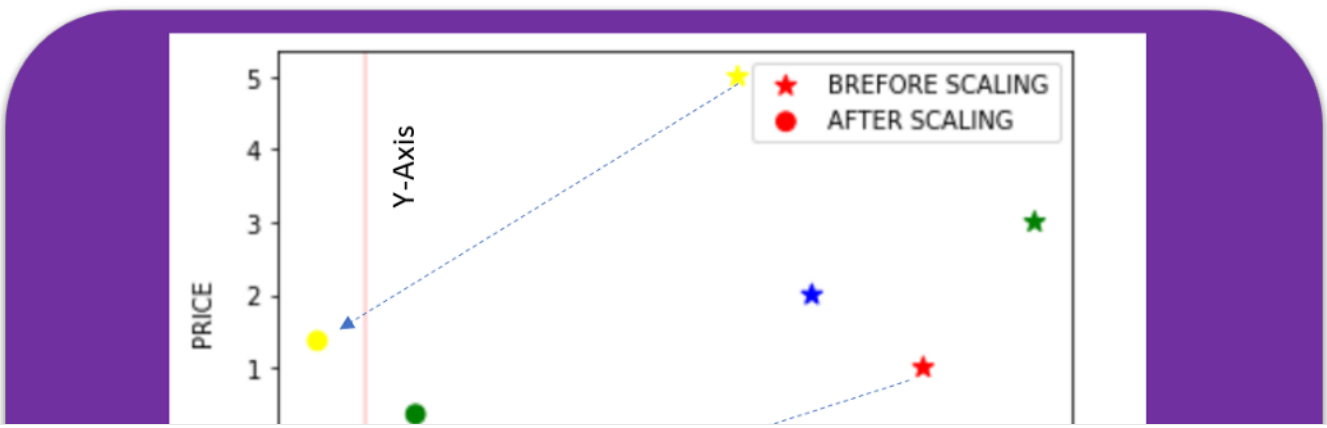
The above example is just for illustration as Quantile transformer is useful when we have a large dataset with many data points usually more than 1000.

6) Power Transformer Scaler

The power transformer is a family of parametric, monotonic transformations that are applied to **make data more Gaussian-like**. This is useful for modeling issues related to the variability of a variable that is unequal across the range (heteroscedasticity) or situations where normality is desired.

The power transform finds the optimal scaling factor in stabilizing variance and minimizing skewness through maximum likelihood estimation. Currently, Sklearn implementation of PowerTransformer supports the Box-Cox transform and the Yeo-Johnson transform. The optimal parameter for stabilizing variance and minimizing skewness is estimated through maximum likelihood. Box-Cox requires input data to be strictly positive, while Yeo-Johnson supports both positive or negative data.

```
from sklearn.preprocessing import PowerTransformer
scaler = PowerTransformer(method='yeo-johnson')
df5 = pd.DataFrame(scaler.fit_transform(df),
                   columns=['WEIGHT','PRICE'],
                   index = ['Orange','Apple','Banana','Grape'])
ax = df.plot.scatter(x='WEIGHT', y='PRICE',color=
['red','green','blue','yellow'],
                     marker = '*',s=80, label='BREFORE SCALING');
df5.plot.scatter(x='WEIGHT', y='PRICE', color=
['red','green','blue','yellow'],
                  marker = 'o',s=60,label='AFTER SCALING', ax = ax)
plt.axhline(0, color='red',alpha=0.2)
plt.axvline(0, color='red',alpha=0.2);
```



7) Unit Vector Scaler

$$x' = \frac{x}{||x||}$$

Scaling is done considering the whole feature vector to be of unit length. This usually means dividing each component by the Euclidean length of the vector (L2 Norm). In some applications (e.g., histogram features), it can be more practical to use the L1 norm of the feature vector.

Like Min-Max Scaling, the Unit Vector technique produces values of range [0,1]. When dealing with features with hard boundaries, this is quite useful. For example, when dealing with image data, the colors can range from only 0 to 255.

```
## Unit vector with L1 norm
df8 = df.apply(lambda x : x/np.linalg.norm(x,1))
df8
```

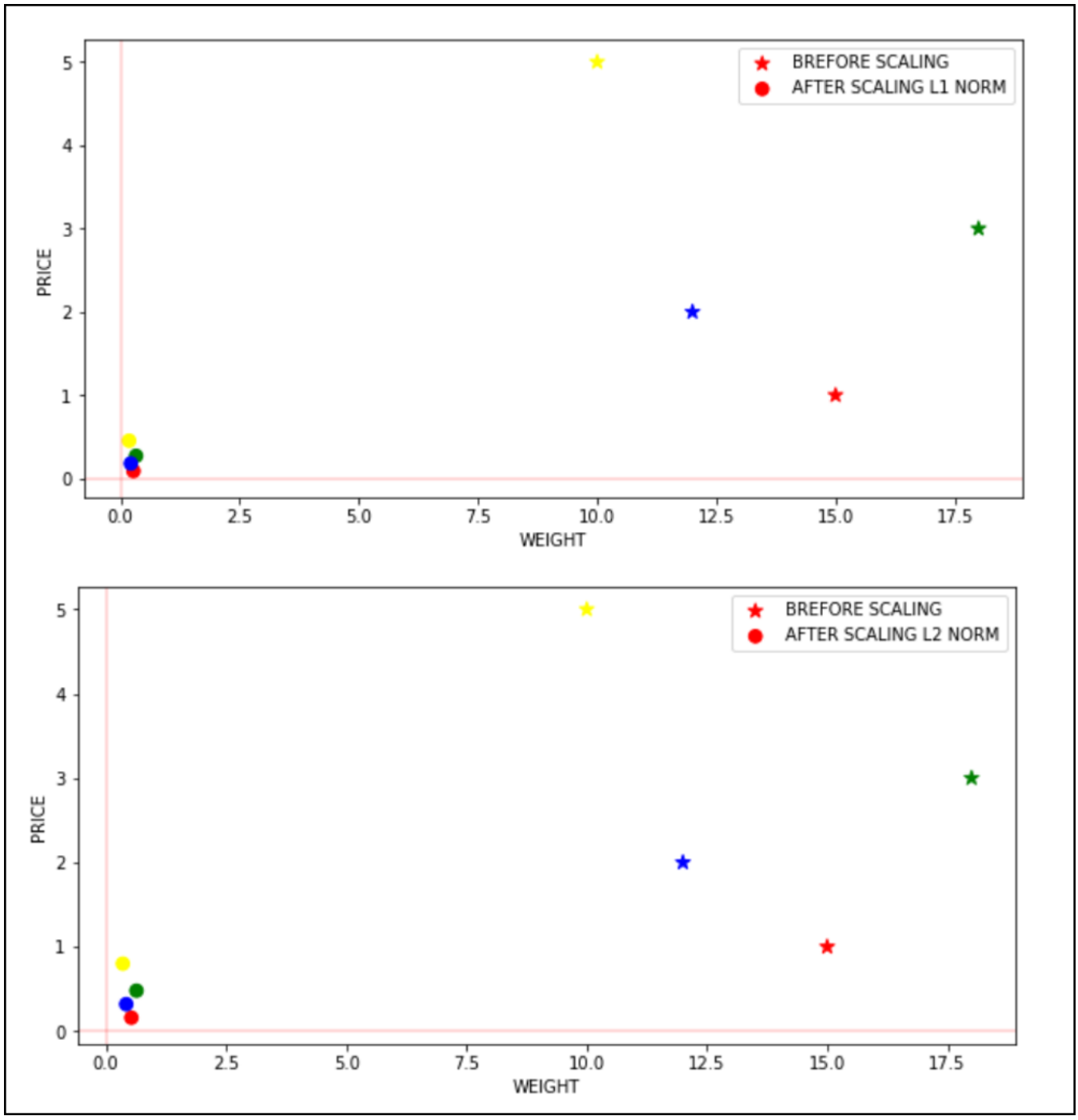
	WEIGHT	PRICE
Orange	0.272727	0.090909
Apple	0.327273	0.272727
Banana	0.218182	0.181818
Grape	0.181818	0.454545

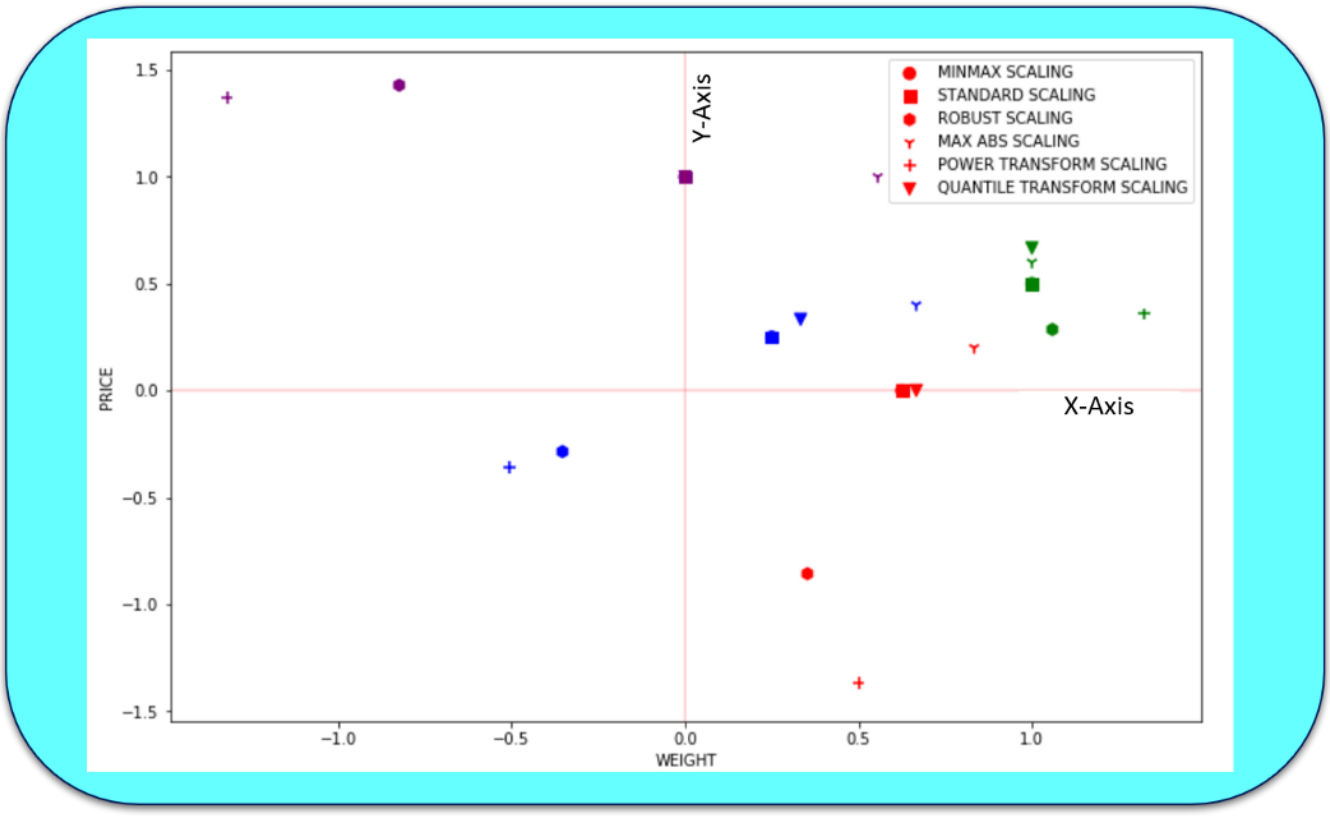
```
## Unit vector with L1 norm
df9 = df.apply(lambda x : x/np.linalg.norm(x,2))
df9
```

| | WEIGHT | PRICE |

Apple	0.639199	0.480384
Banana	0.426132	0.320256
Grape	0.355110	0.800641

If we plot, then it would look as below for L1 and L2 norm, respectively.





Final Note:

Feature scaling is an essential step in Machine Learning pre-processing. Deep learning requires feature scaling for faster convergence, and thus it is vital to decide which feature scaling to use. There are many comparison surveys of scaling methods for various algorithms. Still, like most other machine learning steps, feature scaling too is a trial and error process, not a single silver bullet.

I look forward to your comment and share if you have any unique experience related to feature scaling. Thanks for reading. You can connect me @LinkedIn.

Reference:

http://sebastianraschka.com/Articles/2014_about_feature_scaling.html

https://www.kdnuggets.com/2019/04/normalization-vs-standardization-quantitative-analysis.html

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

Effect of feature scaling on accuracy

Asked 7 years, 1 month ago Active 10 months ago Viewed 9k times

10

9

I am working on image classification using Gaussian Mixture Models. I have around 34,000 features, belonging to three classes, all lying in a 23 dimensional space. I performed feature scaling on both the training and testing data using different methods, and I observed that accuracy actually **reduces after performing scaling**. I performed feature scaling because there was a difference of many orders between many features. I am curious to know why this is happening, I thought that feature scaling would increase the accuracy, especially given the large differences in features.

normalization mixture-model

Share Improve this question Follow

asked Oct 31 '14 at 5:52

Raghuram
402 1 7 20

1) How many training samples do you have? 2) Are you doing testing by testing on data not seen during training? (In this case since you are doing GMM, you are just doing clustering.) 3) What is the accuracy B4 and AFTER your change? – lightalchemist Oct 31 '14 at 6:59

I should have added this in the question itself, sorry about it. I am using the GMM to build a Bayes classifier. Well before the scaling, I was getting an accuracy of 70%, and after the scaling, the accuracy reduces to 45%. – Raghuram Nov 1 '14 at 13:10

@Raghuram Do you have native pictures? What happens if you enlarge the native picture? Accuracy should not be lost here. – Léo Léopold Hertz 준영 Mar 25 '16 at 10:19

1 Answer

Active | Oldest | Votes

24

> I thought that feature scaling would increase the accuracy, especially given the large differences in features.

Welcome to the real world buddy.

In general, it is quite true that you want features to be in the same "scale" so that you don't have some features "dominating" other features. This is especially so if your machine learning algorithm is inherently "geometrical" in nature. By "geometrical", I mean it treats the samples are *points* in a space, and relies on distances (usually Euclidean/L2 as is your case) between points in making its predictions, i.e., the spatial relationships of the points matter. GMM and SVM are algorithms of this nature.

However, feature scaling can screw things up, especially if some features are categorical/ordinal in nature, and you didn't properly preprocess them when you appended them to the rest of your features. Furthermore, depending on your feature scaling method, presence of outliers for a particular feature can also screw up the feature scaling for that feature. For e.g., a "min/max" or "unit variance" scaling is going to be sensitive to outliers (e.g., if one of your feature encodes yearly income or cash balance and there are a few mi/billionaires in your dataset).

Also, when you experience a problem such as this, the cause may not be obvious. It does not mean you perform feature scaling, result goes bad, then feature scaling is at fault. It could be that your method was screwed up to begin with, and the result after feature scaling just happens to be more screwed up.

So what could be other cause(s) of your problem?

1. My guess for the most likely cause is that you have high-dimensional data and not enough training samples. This is because your GMM is going to estimating covariance matrices using data that is 34000 in dimension. Unless you have a lot of data, chances are one or more of your covariance matrices (one for each gaussian) are going to be near singular or singular. This means the predictions from your GMM are nonsense to begin with because your gaussians "blew" up, and/or the EM algorithm just gave up after a predefined number of iterations.
2. Poor testing methodology. You did not have data divided into proper training/validation/test sets, and you did not perform the testing properly. What "good" performance you have in the beginning was not credible. This is actually very common, as the natural tendency is to test using the training data the model was fitted on and not on a validation or test set.

So what can you do?

1. Don't use a GMM for image categorization. Use a proper *supervised* learning algorithm, especially if you have known image categories as labels. In particular, to avoid the feature scaling altogether, use random forest or its variants (e.g., extremely randomized trees).
2. Get more training data. Unless you are classifying "simple" (i.e., "toy"/synthetic images) or you are classifying them into a few image classes (e.g., <= 5. Note this is just a random small number I pulled out of the air.), you really to have a good deal of images per class. A good starting point is to get at least a couple of hundreds per class, or use a more sophisticated algorithm to exploit the structure within your data to arrive at better performance.

Basically, my point is don't (just) treat machine learning field/algorithms as black boxes and a bunch of tricks which you memorize and try at random. Try to understanding the algorithm/math under the hood. That way, you'll be better able to diagnose the problem(s) you encounter.

For papers, the only one I can recall off the top of my head is A Practical Guide to Support Vector Classification by the authors of LibSVM. Examples therein show the importance of feature scaling for SVM on various datasets. E.g., consider the RBF/Gaussian kernel. This kernel uses the square L2 norm. If your features are of different scale, this will affect the value.

Also, how you *represent* your features matter. E.g., changing a variable that represents height from meters to cm or inches will affect algorithms such as PCA (because variance along direction for that feature has changed.) Note this is different from the "typical" scaling (e.g., min/max, Z-score etc.) in that this is a matter of *representation*. The person is still the same height regardless of the *unit*. Whereas typical feature scaling "transform" the data, which changes the "height" of the person. Prof. David Mackay, on the Amazon page of his book, Information Theory for Machine Learning, has a comment in this vein when asked why he did not include PCA in his book.

For ordinal and categorical variables, they are mentioned briefly in Bayesian Reasoning for Machine Learning, The Elements of Statistical Learning. They mention ways to encode them as features, for e.g., replacing a variable that can represent 3 categories with 3 binary variables, with one set to "1" to indicate the sample has that category. This is important for methods such as Linear Regression (or Linear Classifiers). Note this is about encoding categorical variables/features, *not* scaling per se, but they are part of the feature preprocessing set up, and hence useful to know. More can be found in Hal Duame III's book below.

The book A Course in Machine Learning by Hal Duame III. Search for "scaling". One of the earliest example in the book is how it affects KNN (which just uses L2 distance, which GMM, SVM etc. uses if you use the RBF/gaussian kernel). More details are given in the chapter 4, "Machine Learning in Practice". Unfortunately the images/plots are not shown in the PDF. This book has one of the nicest treatments on feature encoding and scaling, especially if you work on Natural Language Processing (NLP). E.g., see his explanation of applying the logarithm to features (i.e., log transform). That way, sums of logs become log of product of features, and "effects"/"contributions" of these features are tapered by the logarithm.

Note that all the aforementioned textbooks are freely downloadable from the above links.

Share Improve this answer Follow

edited Feb 8 '18 at 19:30

Archie
1,859 16 34

answered Oct 31 '14 at 7:37

lightalchemist
9,372 4 41 52

Well, I did test my data on a training set. I am using GMMs to do the image categorization because I was curious to see how it would work out. Thanks for the advice!! – Raghuram Nov 1 '14 at 13:14

This is a pretty good answer, however can you reference some papers? I am trying to find some scientific evidence of the effect of feature scaling. – Zee Feb 17 '15 at 23:34

1 @Zee I've included more details. Feature scaling is more of an "art" or "preprocessing step" in research and is mentioned sporadically in different papers, often investigated in the context for a particular applications. I don't recall offhand any theoretical research on it, although if you understand the math for the algorithms, it is not hard to see how the scale changes things. Results in literature are mostly

@lightalchemist thanks for the refs. I am familiar with basic normalisation and standardisation but I wanted some more info. Especially examples for success uses. Thank you for your answer it is exactly what I was looking for. – Zee Feb 18 '15 at 15:30

1 The part on representation of features could be better explained if we simply say that the large variance of a feature is dominating other features of smaller variance. e.g. Suppose we want to classify people based on height (in metres) and weight (in kilograms). The height attribute has a low variability, ranging from 1.5m to 1.85m, whereas the weight attribute may vary from 50kg to 250kg. If the scale of the attributes are not taken into consideration, the distance measure may be dominated by differences in the weights of a person. Source: Introduction to Data Mining, Ch.5, Tan Pan-Ning – ruhong Apr 22 '18 at 2:11

What algorithms need feature scaling, beside from SVM?

Asked 5 years, 1 month ago Active 1 year, 3 months ago Viewed 48k times

33

I am working with many algorithms: RandomForest, DecisionTrees, NaiveBayes, SVM (kernel=linear and rbf), KNN, LDA and XGBoost. All of them were pretty fast except for SVM. That is when I got to know that it needs feature scaling to work faster. Then I started wondering if I should do the same for the other algorithms.

42

machine-learning svm random-forest naive-bayes boosting

Share Cite Improve this question

Follow

edited Dec 27 '16 at 22:14

Franck Dernoncourt

41.3k 29 152 269

asked Nov 6 '16 at 15:09

Aizzaac

909 2 11 21

Related: How and why do normalization and feature scaling work? – Franck Dernoncourt Jan 6 '17 at 1:23

Also: Variables are often adjusted (e.g. standardised) before making a model - when is this a good idea, and when is it a bad one? – Franck Dernoncourt Jan 6 '17 at 1:37

3 Answers

Active | Oldest | Votes

In general, algorithms that exploit *distances* or *similarities* (e.g. in the form of scalar product) between data samples, such as k-NN and SVM, are sensitive to feature transformations.

36

Graphical-model based classifiers, such as Fisher LDA or Naive Bayes, as well as Decision trees and Tree-based ensemble methods (RF, XGB) are invariant to feature scaling, but still, it might be a good idea to rescale/standardize your data.

Share Cite Improve this answer Follow

edited Sep 2 '20 at 9:31

veeresh d

3 2

answered Dec 20 '16 at 20:41

yell

526 4 5

3 +1. Just note that XGBoost actually implements a second algorithm too, based on linear boosting. Scaling will make a difference there. – usεr11852 Dec 20 '16 at 22:11

3 Could you elaborate more about rescaling/standarizing data for RF and XGB? I don't see how it can influence the quality of model. – Tomek Tarczynski May 11 '17 at 9:27

33

Here is a list I found on http://www.dataschool.io/comparing-supervised-learning-algorithms/ indicating which classifier needs feature scaling:

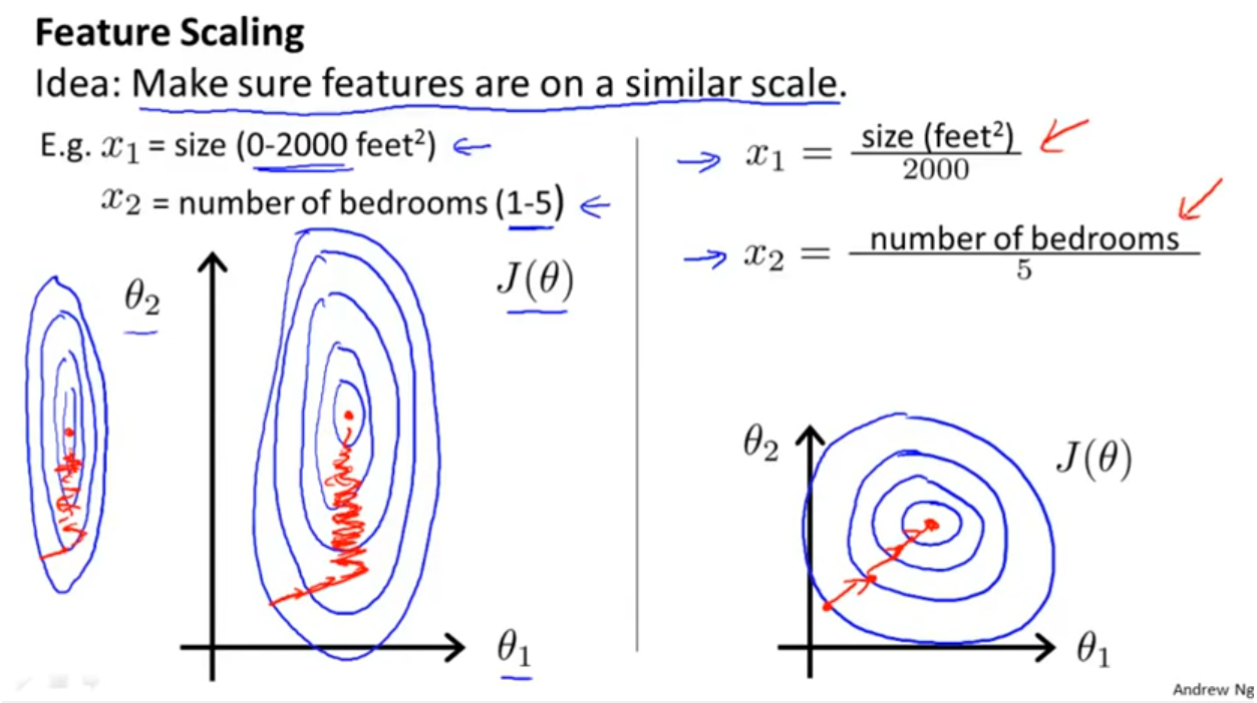
	A	B	N
1	**Algorithm**	**Problem Type**	**Features might need scaling?**
2	KNN	Either	Yes
3	Linear regression	Regression	No (unless regularized)
4	Logistic regression	Classification	No (unless regularized)
5	Naive Bayes	Classification	No
6	Decision trees	Either	No
7	Random Forests	Either	No
8	AdaBoost	Either	No
9	Neural networks	Either	Yes

Full table:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	**Algorithm**	**Problem Type**	**Results interpretable by you?**	**Easy to explain algorithm to others?**	**Average predictive accuracy**	**Training speed**	**Prediction speed**	**Amount of parameter tuning needed (excluding feature selection)**	**Performs well with small number of observations?**	**Handles lots of irrelevant features well (separates signal from noise)?**	**Automatically learns feature interactions?**	**Gives calibrated probabilities of class membership?**	**Parametric?**	**Features might need scaling?**	**Algorithm**
2	KNN	Either	Yes	Yes	Lower	Fast	Depends on n	Minimal	No	No	No	Yes	No	Yes	KNN
3	Linear regression	Regression	Yes	Yes	Lower	Fast	Fast	None (excluding regularization)	Yes	No	No	N/A	Yes	No (unless regularized)	Linear regression
4	Logistic regression	Classification	Somewhat	Somewhat	Lower	Fast	Fast	None (excluding regularization)	Yes	No	No	Yes	Yes	No (unless regularized)	Logistic regression
5	Naive Bayes	Classification	Somewhat	Somewhat	Lower	Fast (excluding feature extraction)	Fast	Some for feature extraction	Yes	Yes	No	No	Yes	No	Naive Bayes
6	Decision trees	Either	Somewhat	Somewhat	Lower	Fast	Fast	Some	No	No	Yes	Possibly	No	No	Decision trees
7	Random Forests	Either	A little	No	Higher	Slow	Moderate	Some	No	Yes (unless noise ratio is very high)	Yes	Possibly	No	No	Random Forests
8	AdaBoost	Either	A little	No	Higher	Slow	Fast	Some	No	Yes	Yes	Possibly	No	No	AdaBoost
9	Neural networks	Either	No	No	Higher	Slow	Fast	Lots	No	Yes	Yes	Possibly	No	Yes	Neural networks
10															

In k-means clustering you also need to normalize your input.

In addition to considering whether the classifier exploits distances or similarities as Yell Bond mentioned, Stochastic Gradient Descent is also sensitive to feature scaling (since the learning rate in the update equation of Stochastic Gradient Descent is the same for every parameter {1}):



References:

- {1} Elkan, Charles. "Log-linear models and conditional random fields." Tutorial notes at CIKM 8 (2008). https://scholar.google.com/scholar?cluster=5802800304608191219&hl=en&as_sdt=0,22 ; https://pdfs.semanticscholar.org/b971/0868004ec688c4ca87aa1fec7ffb7a2d01d8.pdf

Share Cite Improve this answer Follow

edited May 23 '17 at 12:39
Community Bot
1

answered Dec 27 '16 at 22:09
Franck Dernoncourt
41.3k 29 152 269

What lacks from this answer is some explication of why!! See my answer for that. – kjetil b halvorsen ♦ Jan 1 '17 at 21:33

2 @kjetilbhalvorsen well I explained for k-means and SGD, but there are many other algorithms and models. There is a 30k-char limit on Stack Exchange :) – Franck Dernoncourt Jan 1 '17 at 21:36

Somewhat related: stats.stackexchange.com/questions/231285/... – kjetil b halvorsen ♦ Jul 8 '18 at 17:33

@FranckDernoncourt Can I ask you a question building on this? I have a dataset of both categorical and continuous data, for which I'm building an SVM. The continuous data is highly skewed (long tail). For transformation on the continuous should I do a `log transformation / Box-Cox` and *then* also `normalise the resultant data to get limits between 0 and 1` ? So i'll be normalising the log values. Then calculate the SVM on the continuous and categorical (0-1) data together? Cheers for any help you can provide. – Chuck Oct 25 '18 at 10:28

8

Adding to the excellent (but too short) answer by Yell Bond. Look at what happens with a linear regression model, we write it with only two predictors but the issue do not depend on that.

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 z_i + \epsilon_i$$

$i = 1, \ldots, n$. Now if you, say, center and scale the predictors to get

$$x_i^* = (x_i - \bar{x})/\mathrm{sd}(x)$$
$$z_i^* = (z_i - \bar{z})/\mathrm{sd}(z)$$

and instead fit the model (using ordinary least squares)

$$Y_i = \beta_0^* + \beta_1^* x_i^* + \beta_2^* z_i^* + \epsilon_i$$

Then the fitted parameters (betas) will change, but they change in a way that you can calculate by simple algebra from the transformation applied. So if we call the estimated betas from the model using transformed predictors for $\beta_{1,2}^*$ and the denote the betas from the untransformed model with $\hat{\beta}_{1,2}$, we can calculate one set of betas from the other one, knowing the means and standard deviations of the predictors. The realtionship between the transformed and untransformed parameters is the same as between their estimates, when based on OLS. Some algebra will give the relationship as

$$\beta_0 = \beta_0^* - \frac{\beta_1^* \bar{x}}{\mathrm{sd(x)}} - \frac{\beta_2^* \bar{z}}{\mathrm{sd(z)}}, \quad \beta_1 = \frac{\beta_1^*}{\mathrm{sd(x)}}, \quad \beta_2 = \frac{\beta_2^*}{\mathrm{sd(z)}}$$

So standardization is not a necessary part of modelling. (It might still be done for other reasons, which we do not cover here). This answer depends also upon us using ordinary least squares. For some other fitting methods, such as ridge or lasso, standardization is important, because we loose this invariance we have with least squares. This is easy to see: both lasso and ridge do regularization based on the size of the betas, so any transformation which change the relative sizes of the betas will change the result!

And this discussion for the case of linear regression tells you what you should look after in other cases: Is there invariance, or is it not? Generally, **methods which depends on distance measures among the predictors will not show invariance**, so standardization is important. Another example will be clustering.

Share Cite Improve this answer Follow

edited Jun 15 '19 at 20:33

answered Dec 20 '16 at 21:09

kjetil b halvorsen ♦
61.8k 26 140 453

1 Can you explicitly show how one calculates one set of betas from the other in this particular example of scalings you have applied? – Mathews24 Aug 7 '18 at 2:35

@kjetil Can I ask you a question building on this? I have a dataset of both categorical and continuous data, for which I'm building an SVM. The continuous data is highly skewed (long tail). For transformation on the continuous should I do a log transformation / Box-Cox and then also normalise the resultant data to get limits between 0 and 1? So i'll be normalising the log values. Then calculate the SVM on the continuous and categorical (0-1) data together? Cheers for any help you can provide – Chuck Oct 25 '18 at 10:28

1 Can you please add this as a new question? with reference back here! – kjetil b halvorsen ♦ Oct 25 '18 at 10:31

Getting Started

+ New Topic

Bhushan Kapkar

Topic Author

Which Machine Learning requires Feature Scaling(Standardization and Normalization)? And Which not?

Posted in Getting Started a year ago

32

Beginner sklearn Exploratory Data Analysis Tabular Data

The feature scaling is the most important step in data preparation. Whether to use feature scaling or not depend upon the algorithm you are using.

Many of us, still wondering why feature scaling requires? Why we need to scale the variables?

1. Having features on same scale that can contribute equally to the result. Can enhance the performance of machine learning algorithms.
2. If you don't scale features then large scale variables will dominate the small scale features. Example: Suppose, the dataset contains X variable(might be of 2 digit number) and Y variable(might be 5-6 digit number) variables. There is huge gap in scale. As we don't want our algorithm to be biased towards one feature.This will effect the accuracy of model or towards the performance of algorithm or might get wrong predictions.

Certain machine learning algorithms such as distance based algorithms , curve based algorithms or matrix factorization, decomposition or dimensionality reduction or gradient descent based algorithms are sensitive towards feature scaling (standardization and normalization for numerical variables).

And there are certain tree based algorithms which are insensitive towards feature scaling as they are rule based algorithms such as Classification and Regression trees, Random Forests or Gradient Boosted decision Trees.

Sr No.	Algorithms	Feature Scaling
1.	Linear/Non-Linear Regressions	Yes
2.	Logistic Regression	Yes
3.	KNN	Yes
4.	SVM	Yes
5.	Neural Networks	Yes
6.	K-means clustering	Yes
7.	CART	No
8.	Random Forests	No
9.	Gradient Boosted Decision Trees	No
10.	Naïve Bayes	N0
11.	PCA	Yes
12.	SVD	Yes
13.	Factorization Machines	Yes

Fig: Feature Scaling requires based on Machine Learning

Cons of Feature Scaling: You will lose the original value will transforming to other values. So, there is loss of interpretation of the values.

Standardization v/s Normalization
Standardization:
The idea behind using standardization before applying machine learning algorithm is to transform you data such that its distribution will have mean value 0 and standard deviation as 1.

Mu=0
Sd=1

Normalization:

This method will scale/shift/rescale the data between the range of 0 and 1. So, this is also called as Min-Max scaling.

Cons: This method will make us to lose some of the information of the data such as outliers.

For most of the applications, Standardization method performances better than Normalization.
**Note: For the best possible results, you need to start fitting the actual whole model(default), normalized and standardized and compare the results.

Hope! This is useful. Please do give your additional Inputs...!!!!

Quote | Follow | Bookmark | Report | 32 Upvoters

Comments (31) Sort by Hotness

Before you can post on Kaggle, you'll need to verify your account with a phone number.

This comment will be made public once posted. Post Comment

Neil • 8 months ago • Options • Report • Reply 1

Wow, Very precise and accurate information.

Thanks for sharing.

Praful Mohanan • 9 months ago • Options • Report • Reply 1

Thank you for sharing this. I was searching around to clear this up 👍

FoolishSpector • a year ago • Options • Report • Reply 1

Good one @bpkapkar
All we know is how to use it, but little we know when to use it.
Thanks

Bhushan Kapkar Topic Author • a year ago • Options • Report • Reply 0

Yeah, hopefully now its much more clear when to use it and make it more sense to model as well😊

Gaurav Dudeja • a year ago • Options • Report • Reply 1

A big thanks for providing such important information with so much clarity 🙌

Bhushan Kapkar Topic Author • a year ago • Options • Report • Reply 0

Yes, during early days we do face such problems and confusion on the same. Thought to share this for more clarity and hopefully its meeting the purpose...!!!!

Edwin Tanaga • a year ago • Options • Report • Reply 1

Thanks for sharing Bhushan Kapkar.
This article is useful for me

Bhushan Kapkar Topic Author • a year ago • Options • Report • Reply 0

Its pleasure, to share knowledge....!!!

Tip : Kindly complete the basic required action given by kaggle to upgrade your progression from Nvoice to Contributor, that will help you to get better support ,as contributor upvote make the difference to the person

Explore further with atoti

ARTICLES < HTTPS://WWW.ATOTI.IO/ARTICLES/>

When to perform a Feature Scaling?

By Raghav Vashisht < https://www.atoti.io/author/raghav/>

6 January 2021 < https://www.atoti.io/when-to-perform-a-feature-scaling/>

In this article, we shall discuss one of the ubiquitous steps in the machine learning pipeline — Feature Scaling. This article's origin lies in one of the coffee discussions in my office on what all models actually are affected by feature scaling and then what is the best way to do it — to normalize or to standardize or something else?

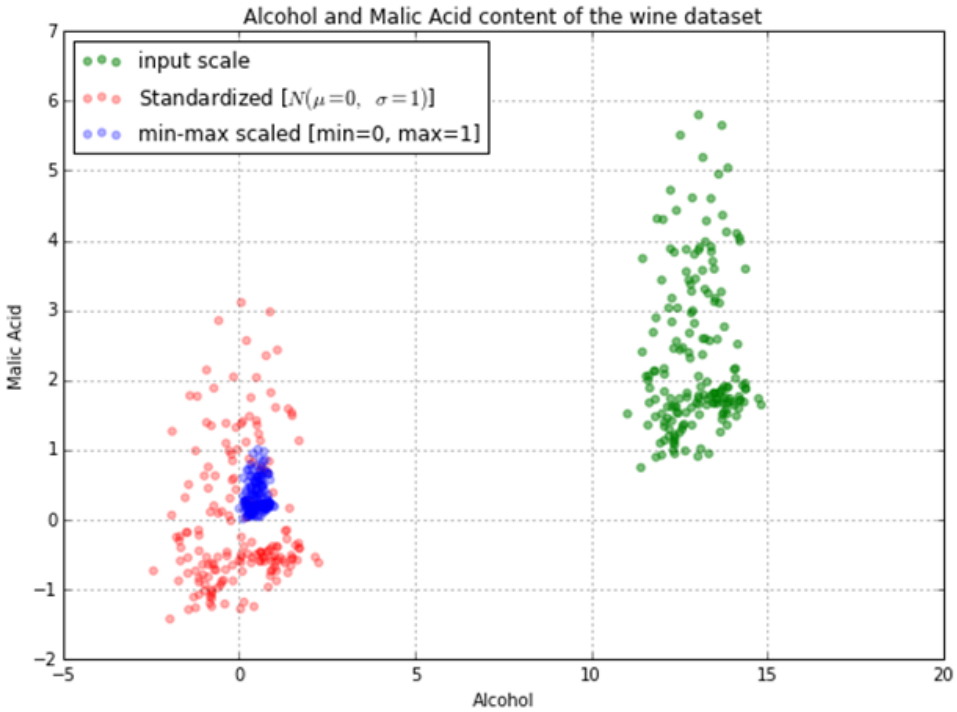
In this article, in addition to the above, we would also cover a gentle introduction to feature scaling, the various feature scaling techniques, how it might lead to data leakage, when to perform feature scaling, and when NOT to perform feature scaling. So tighten your reading glasses and read on...

An increasing number of features on different scales make machine learning problems hard to handle, is feature scaling the solution?

What is Feature Scaling?

Feature scaling is a method used to normalize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing step. Just to give you an example — if you have multiple independent variables like age, salary, and height; With their range as (18–100 Years), (25,000–75,000 Euros), and (1–2 Meters) respectively, feature scaling would help them all to be in the same range, for example- centered around 0 or in the range (0,1) depending on the scaling technique.

In order to visualize the above, let us take an example < https://sebastianraschka.com/Articles/2014_about_feature_scaling.html> of the independent variables of alcohol and Malic Acid content in the wine dataset from the “Wine Dataset” that is deposited on the UCI machine learning repository. Below you can see the impact of the two most common scaling techniques (Normalization and Standardization) on the dataset.



The impact of Standardization and Normalisation on the Wine dataset

Methods for Scaling

Now, since you have an idea of what is feature scaling. Let us explore what methods are available for doing feature scaling. Of all the methods available, the most common ones are:

Normalization

Also known as min-max scaling or min-max normalization, it is the simplest method and consists of rescaling the range of features to scale the range in [0, 1]. The general formula for normalization is given as:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Here, max(x) and min(x) are the maximum and the minimum values of the feature respectively.

We can also do a normalization over different intervals, e.g. choosing to have the variable laying in any [a, b] interval, a and b being real numbers. To rescale a range between an arbitrary set of values [a, b], the formula becomes:

$$x' = a + \frac{(x - \min(x))(b - a)}{\max(x) - \min(x)}$$

Standardization

Feature standardization makes the values of each feature in the data have zero mean and unit variance. The general method of calculation is to determine the distribution mean and standard deviation for each feature and calculate the new data point by the following formula:

$$x' = \frac{x - \bar{x}}{\sigma}$$

Here, σ is the standard deviation of the feature vector, and $\bar{x}$ is the average of the feature vector.

Scaling to unit length

The aim of this method is to scale the components of a feature vector such that the complete vector has length one. This usually means dividing each component by the Euclidean length of the vector:

$$x' = \frac{x}{\|x\|}$$

$\|x\|$ Is the Euclidean length of the Feature Vector.

In addition to the above 3 widely-used methods, there are some other methods to scale the features viz. Power Transformer, Quantile Transformer, Robust Scaler, etc. For the scope of this discussion, we are deliberately not diving into the details of these techniques.

The million-dollar question: Normalization or Standardization

If you have ever built a machine learning pipeline, you must have always faced this question of whether to Normalize or to Standardize. While there is no obvious answer to this question, it really depends on the application, there are still a few generalizations that can be drawn < https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/> .

Normalization is good to use when the distribution of data does not follow a Gaussian distribution. It can be useful in algorithms that do not assume any

distribution of the data like K-Nearest Neighbors.

In Neural Networks algorithm that require data on a 0–1 scale, normalization is an essential pre-processing step. Another popular example of data normalization is image processing, where pixel intensities have to be normalized to fit within a certain range (i.e., 0 to 255 for the RGB color range).

Standardization can be helpful in cases where the data follows a Gaussian distribution. Though this does not have to be necessarily true. Since standardization does not have a bounding range, so, even if there are outliers in the data, they will not be affected by standardization.

In clustering analyses, standardization comes in handy to compare similarities between features based on certain distance measures. Another prominent example is the Principal Component Analysis, where we usually prefer standardization over Min-Max scaling since we are interested in the components that maximize the variance.

There are some points which can be considered while deciding whether we need Standardization or Normalization

- Standardization may be used when data represent Gaussian Distribution, while Normalization is great with Non-Gaussian Distribution
- Impact of Outliers is very high in Normalization

To conclude, you can always start by fitting your model to raw, normalized, and standardized data and compare the performance for the best results.

The link between Data Scaling and Data Leakage

In order to apply Normalization or Standardization, we can use the prebuilt functions in scikit-learn or can create our own custom function.

Data leakage mainly occurs when some information from the training data is revealed to the validation data. In order to prevent the same, the point to pay attention to is to fit the scaler on the train data and then use it to transform the test data. For further details on data leakage, you can check out my article < https://medium.com/atoti/what-is-data-leakage-and-how-to-mitigate-it-5be11f6d2f94> on data leakage and how to mitigate it.

When to scale your data?

After building an understanding of how to do data scaling and which data scaling techniques to use, we can now talk about where to use these data scaling techniques.

Gradient Descent Based Algorithms

If an algorithm uses gradient descent, then the difference in ranges of features will cause different step sizes for each feature. To ensure that the gradient descent moves smoothly towards the minima and that the steps for gradient descent are updated at the same rate for all the features, we scale the data before feeding it to the model. Having features on a similar scale will help the gradient descent converge more quickly towards the minima.

Specifically, in the case of Neural Networks Algorithms, feature scaling benefits optimization by:

- It makes the training faster
- It prevents the optimization from getting stuck in local optima
- It gives a better error surface shape

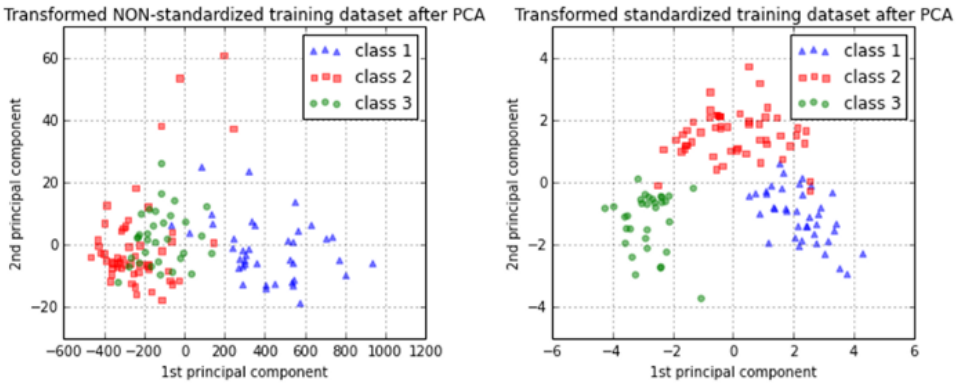
- Weight decay and Bayes optimization can be done more conveniently

Distance-Based Algorithms

Distance-based algorithms like KNN, K-means, and SVM are most affected by the range of features. This is because behind the scenes they are using distances between data points to determine their similarity and hence perform the task at hand. Therefore, we scale our data before employing a distance-based algorithm so that all the features contribute equally to the result.

For feature engineering using PCA

In PCA < https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html> we are interested in the components that maximize the variance. If one component (e.g. age) varies less than another (e.g. salary) because of their respective scales, PCA might determine that the direction of maximal variance more closely corresponds with the 'salary' axis, if those features are not scaled. As a change in the age of one year can be considered much more important than the change in salary of one euro, this is clearly incorrect.



Impact of data standardization on PCA < https://sebastianraschka.com/Articles/2014_about_feature_scaling.html>

What about regression?

In regression, it is often recommended to scale the features so that the predictors have a mean of 0. This makes it easier to interpret the intercept term as the expected value of Y when the predictor values are set to their means. There are few other aspects that vouch for feature centering in case of regression:

- When one variable has a very large scale: e.g. if you are using the population size of a country as a predictor. In that case, the regression coefficients may be on a ***very*** small order of magnitude (e.g. e^-9) which can be a little annoying when you're reading computer output, so you may convert the variable to, for example, population size in millions or just perform a Normalization.
- While creating power terms: Let's say you have a variable, X, that ranges from 1 to 2, but you suspect a curvilinear relationship with the response variable, and so you want to create an X^2 term. If you don't center X first, your squared term will be highly correlated with X, which could muddy the estimation of the beta. Centering ***first*** addresses this issue.
- Creating interaction terms: If an interaction/product term is created from two variables that are not centered on 0, some amount of collinearity will be induced (with the exact amount depending on various factors).

Centering/scaling does not affect your statistical inference in regression models < https://stats.stackexchange.com/questions/29781/when-conducting-multiple-regression-when-should-you-center-your-predictor-varia> — the estimates are adjusted appropriately and the p-values will be the same. The scale and location of the explanatory variables do not affect the ***validity*** of the regression model in any way.

The betas are estimated such that they convert the units of each explanatory variable into the units of the response variable appropriately.

Consider the model:

$y=\beta0+\beta1\times1+\beta2\times2+...+\epsilon y=\beta0+\beta1\times1+\beta2\times2+...+\epsilon.$

The [least-squares estimators <](http://en.wikipedia.org/wiki/Linear_regression#Least-squares_estimation_and_related_techniques>) of $\beta1,\beta2,...\beta1,\beta2,...$ are not affected by shifting. The reason is that these are the slopes of the fitting surface — how much the surface changes if you change $x1,x2,...x1,x2,...$ one unit. This does not depend on location. The scaling doesn't affect the estimators of the other slopes. Thus, scaling simply corresponds to scaling the corresponding slopes.

To conclude, technically, feature scaling does not make a difference in the regression, but it might give us some practical benefits and in further feature engineering steps.

When scaling your data is NOT necessary?

Tree-based algorithms

Tree-based algorithms are fairly insensitive to the scale of the features. A decision tree is only splitting a node based on a single feature. The decision tree splits a node on a feature that increases the homogeneity of the node. This split on a feature is not influenced by other features. Hence, there is virtually no effect of the remaining features on the split. This is what makes them invariant to the scale of the features.

Should you ALWAYS do feature engineering?

If there are some algorithms that are not really affected by feature scaling and can work with or without feature scaling, but then there are some algorithms that just cannot work without the features being scaled, does it not make sense to ALWAYS perform feature engineering?

Well, not ALWAYS < https://stats.stackexchange.com/questions/121886/when-should-i-apply-feature-scaling-for-my-data> — imagine classifying something that has equal units of measurement recorded with noise. Like a photograph or microarray or some spectrum. In this case, you already know a-priori that your features have equal units. If you were to scale them all you would amplify the effect of features that are constant across all samples, but were measured with noise. (Like the background of the photo). This again will have an influence on KNN and might drastically reduce performance if your data had more noisy constant values compared to the ones that vary.

Some questions that you should ask yourself to decide whether scaling is a good idea:

- What would normalization do to your data wrt solving the task at hand? Should that become easier or do you risk deleting important information?
- Is the algorithm sensitive to the scale of the data?
- Does the algorithm or its actual implementation perform its own normalization?

Now you know-all about feature scaling, the next step is to apply it correctly in your ML pipelines.

Thank you for your time and attention.

Please check out our GitHub gallery < https://github.com/atoti/notebooks> for some interesting use cases and to see a python library with an in-built BI tool in action!

You can also see similar QnA and other articles we have published on our medium page < https://medium.com/atoti> .