Feature Scaling

What is feature scaling?

Feature scaling is a data manipulation technique used in many machine-learning models and algorithms. The general idea is to transform raw values so that they are within the same scale. The general objective of feature scaling is to avoid having one feature dominate other features during training due to a having a differing range. For example, say we have two features: score and effort. Further, suppose the values for score exist on a scale from [0, 100], and those for effort exist on a scale from [0, 1000000]. During training, the relatively large values of effort will dominate those of score. To account for this, we scale each of the raw data sets so that the values for both score and effort exist within the same scale, say [0,1]. During training then the machine learning models that expect these ranges to be on the same scale will not inadvertently introduce any bias.

Why is feature scaling necessary?

In general, the ranges of raw data vary widely and machine learning algorithms see nothing but numbers. While humans understand that 10 lbs and $100 000 are vastly different things, machine learning algorithms simply see 10 and 100 000. So these numbers play an integral role while training machine models. One example of this can be seen in gradient descent algorithms that seek a minimum of a cost function. Their goal is to find parameters such that the cost function is minimized. The only thing the computer may do is randomly choose values for the parameters. The value chosen for the next parameter is a function of the current value, and some information about the gradient vector. But if the ranges are dissimilar, the gradient vector will likely fail to point directly towards the minima, thus presenting a jagged path, as opposed to a rather straight one. Thus, the time spent finding the minima will not be optimized. This explains just one reason for the scaling of features. More generally, without scaling features, we risk introducing bias to our model due to the algorithm being biased towards the features with relatively larger values. So we must scale the features of our model so that the machine learning process treats them all equally.

What does normalization and standardization do to the data and noise?

Standardization rescales features such that they have the characteristics of a standard normally distributed graph, with mean 0 and standard deviation of 1. It is widely to determine the distribution mean [2] (Eq 1) and standard deviation [3] (Eq 2) for each feature. The values of each of the features is divided by its standard deviation (Eq 3).

$$\text{Eq (1)}\ \ \mu = \frac{1}{N}\sum_{i=1}^{N} x \qquad \text{Eq (2)}\ \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)} \qquad \text{Eq (3)}\ x' = \frac{x-\mu}{\sigma}$$

Normalization, on the other hand, transforms features to fit the scale of the range [0, 1] or [-1, 1] (Eq 4). The selection of target range depends on the aspects of the data.

$$\text{Eq (4)}\ x' = \frac{x-\min(x)}{\max(x)-\min(x)}$$

Standardization tends to be helpful when the data follows a Gaussian distribution; however, since standardization does not have a bounding range, outliers will continue to be outliers [7]. Note that standardization *can* become skewed or biased if the input contains heavy outliers, so one approach is to ignore the outliers from the calculation of the mean and standard deviation, then to scale the outliers themselves [8]. For standardization, generally, the mean and standard deviation calculations will be affected by a smaller margin. Standardization would be more intuitive to use when the data sample are in form of normal distribution and the data contains heavy outliers that would affect min/max calculations [6]. If the data does not follow a standard distribution, or if the algorithm being used does not assume any distribution at all, it may make more sense to use normalization [7]. In normalization the noise of the data is scaled into the appropriate range. Given min and max values, outliers will therefor weigh heavily in the calculations. We should use normalization when the data set is not normally distributed and is clean from particularly heavy outliers.

References

[1] Feature scaling : https://en.wikipedia.org/wiki/Feature_scaling

[2] Mean : https://en.wikipedia.org/wiki/Mean#Arithmetic_mean_(AM)

[3] Standard Deviation : https://en.wikipedia.org/wiki/Standard_deviation

[4] Gradient Descent : https://en.wikipedia.org/wiki/Gradient_descent

[5] Plot scaling importance https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html

[6] https://www.enjoyalgorithms.com/blog/need-of-feature-scaling-in-machine-learning

[7] https://www.atoti.io/articles/when-to-perform-a-feature-scaling/

[8] https://machinelearningmastery.com/robust-scaler-transforms-for-machine-learning/

Q2. Smoothened Weeks

To produce the chart to your right we first smoothened the Global Intensity data for each week using the moving average, with a time window of 10 consecutive observations.

We then computed the average smoothened week. This was produced by finding the averaged value for time t for all times for one whole week.

Letting:
X_i = The smoothened week for week i
Y = average smoothened week

We then computed:

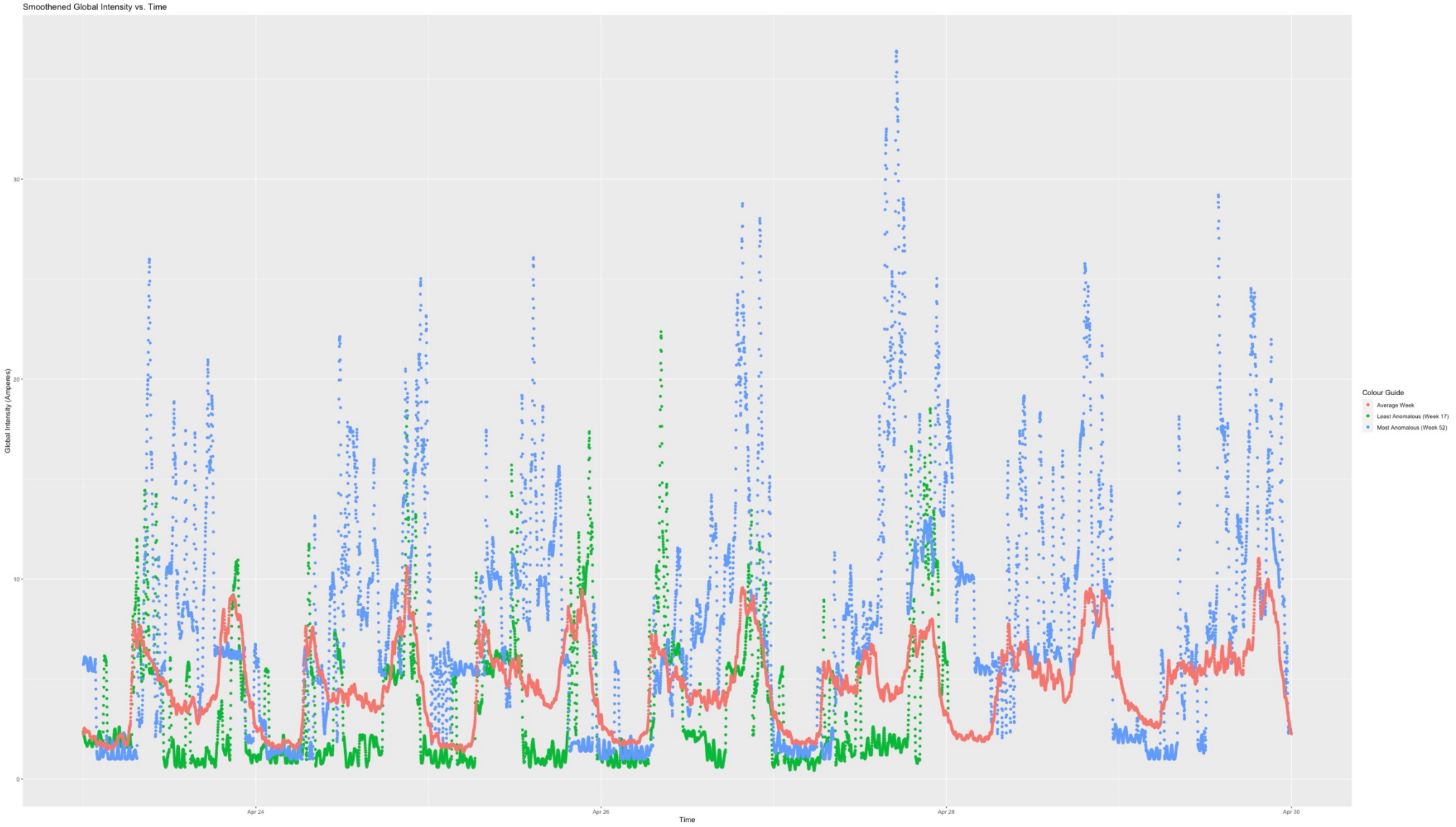
MSE = $\frac{1}{n} \sum (Y - f(X_i))^2$ for each i

This is what we see in the Mean Squared Error column, which represents how closely the observed values match the expected values. Smaller MSE values mean that the observed values were close to the predicted values, while larger MSE values mean the opposite.

From the rank column of the table, we can see that week 17 had the lowest MSE, while week 52 had the highest MSE. Thus, week 17 was the least anomalous, while week 52 is the most anomalous.

Please see the following page for a visual depiction of the findings.

	Week	Mean Squared Error	rank
1	17	7.855640	1
2	30	9.199078	2
3	37	9.941774	3
4	22	10.601794	4
5	39	10.652057	5
6	26	10.835727	6
7	29	11.050475	7
8	21	11.096155	8
9	28	11.188536	9
10	36	11.287955	10
11	16	11.421882	11
12	38	11.680019	12
13	31	11.746983	13
14	27	11.865673	14
15	23	12.111498	15
16	24	12.220430	16
17	25	12.326450	17
18	19	12.797205	18
19	40	13.049467	19
20	42	13.322988	20
21	15	13.325616	21
22	20	13.397533	22
23	18	14.851236	23
24	11	15.067981	24
25	41	15.264631	25
26	45	15.530791	26
27	35	15.716157	27
28	47	15.841395	28
29	44	17.223420	29
30	14	17.255287	30
31	6	17.319861	31
32	4	17.505312	32
33	9	17.522911	33
34	43	17.687334	34
35	50	17.816556	35
36	32	18.011579	36
37	33	19.105649	37
38	13	19.189228	38
39	46	19.362322	39
40	34	19.451300	40
41	51	19.698678	41
42	49	20.690451	42
43	12	21.135304	43
44	7	22.554368	44
45	10	22.586364	45
46	48	23.332091	46
47	2	25.759179	47
48	3	25.794341	48
49	5	27.934082	49
50	1	28.133050	50
51	8	28.490432	51
52	52	39.543071	52


Smoothened Global Intensity vs. Time
Global Intensity (Amperes)
30
20
10
0
Apr 24
Apr 26
Apr 28
Apr 30
Time
Colour Guide
Average Week
Least Anomalous (Week 17)
Most Anomalous (Week 52)

Question 3.

Problem 1: Given the observation sequence O = O1 , O2 , …, OT and a model λ = (A, B, π), how do we efficiently compute P(O|λ), the probability of the observation sequence given the model?

The meaning of the above problem regarding the detection of anomalous patterns in stream data received from a continuously operating system can be conceptualized by the following:

Suppose we are monitoring a supervisory control system that outputs a continual stream of data. Given a subset of this stream of data, and using machine learning techniques, we may produce a model λ = (A, B, π). That is, we may produce a model that, hopefully with some accuracy, is able to predict future observations or unearth the probability of seeing a sequence of observations.

Now, given the model we have created, suppose a set of data is produced by the supervisory control system that we are monitoring. We might conceptualize this new set as a sequence of observations O = O1 , O2 , …, OT. What we would like to know is, given the model that we produced based on the subset of stream data from the SCS, what is the probability of seeing this new set of observations? Is it anomalous? Is it "normal-ish?". Our challenge is to efficiently compute the probability of seeing this sequence of observations, given the model we have produced, so that we can make decisions about whether a cyber intrusion is occurring, or not, and if it is, how best to proceed.