Feature Scaling and Normalization

# The Why?

Features consists of two parts: Magnitude and Unit.

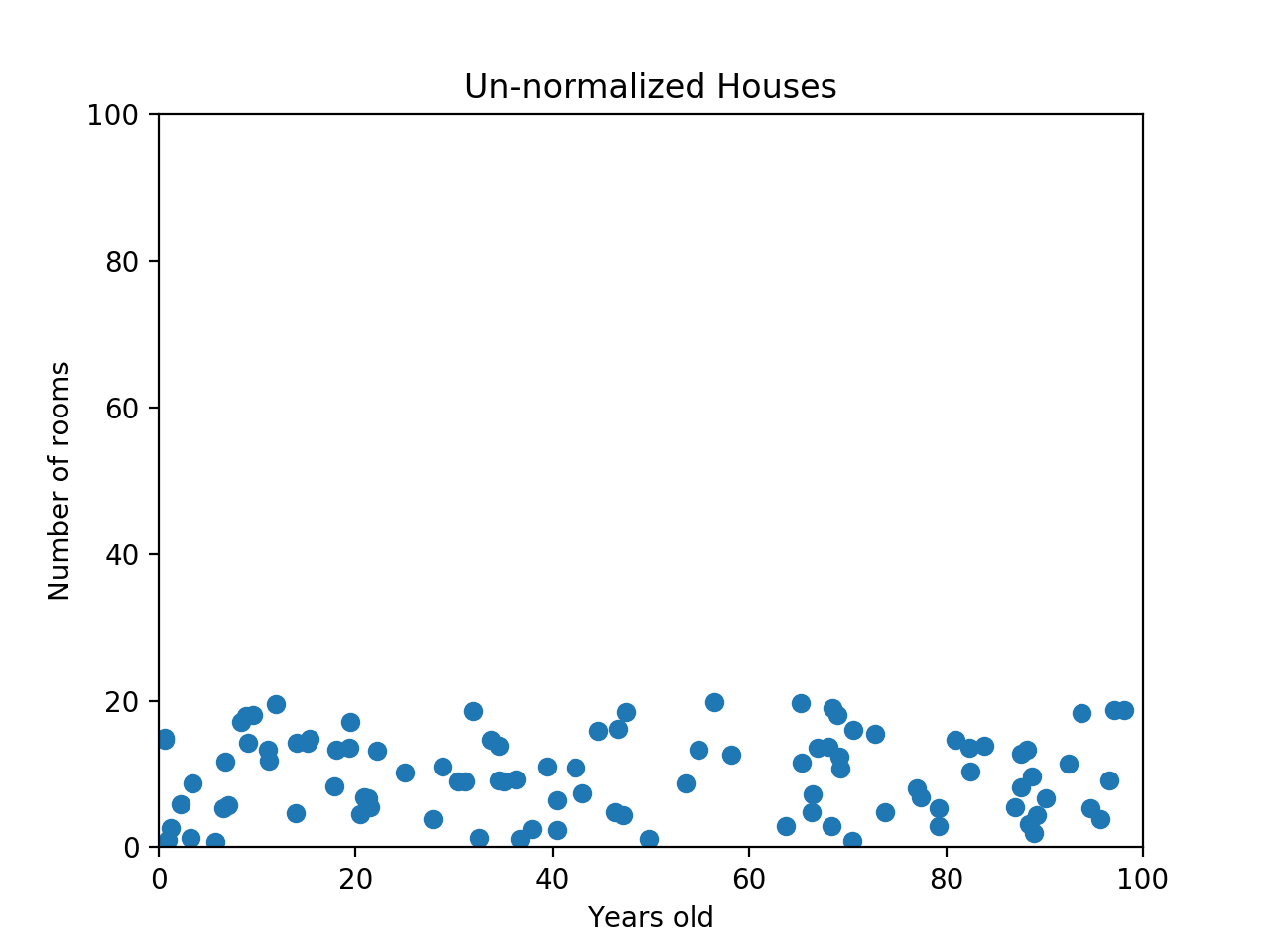
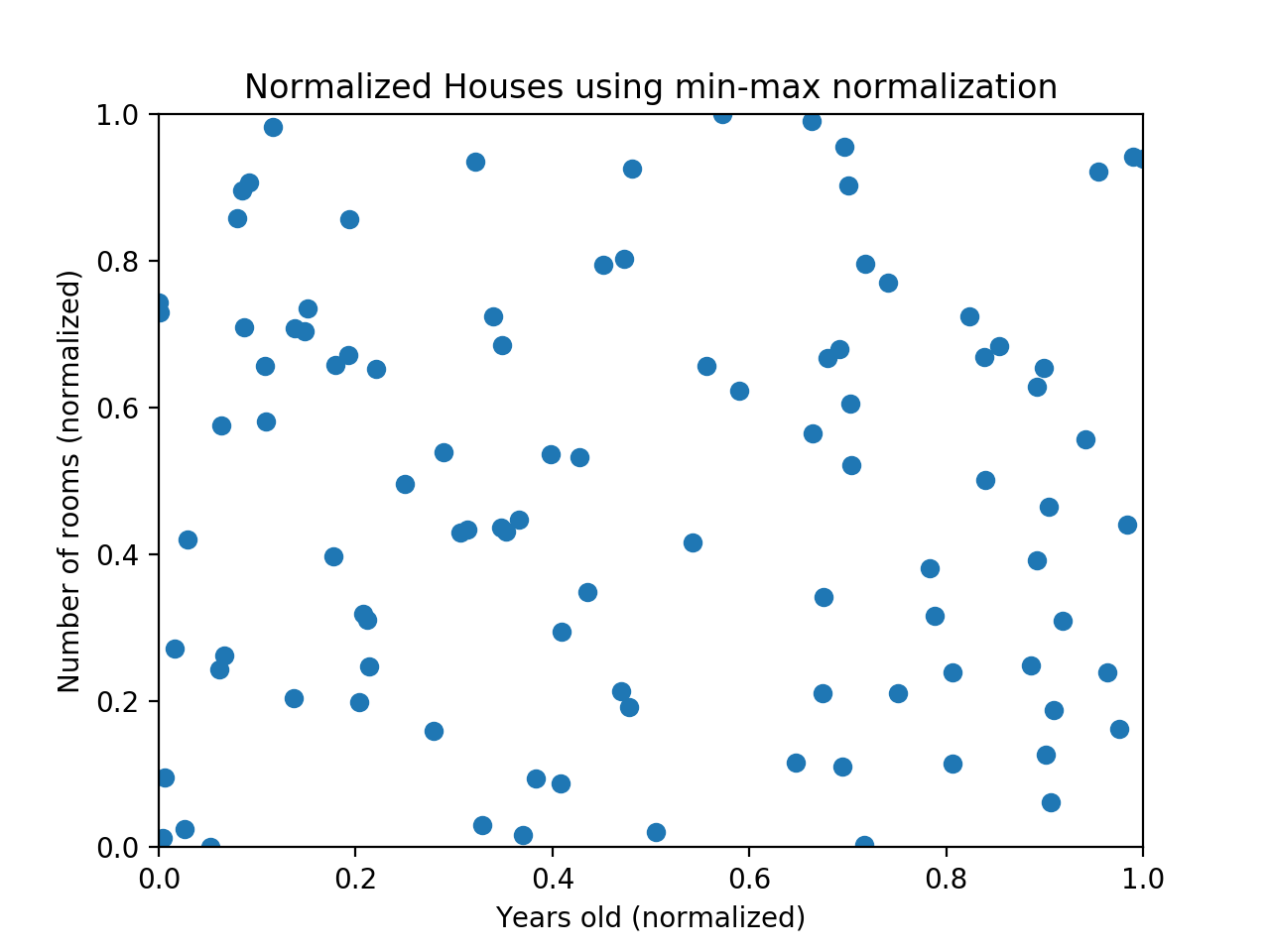
Most of the times, the dataset will contain features highly varying in magnitudes, units and range. And since, most of the machine learning algorithms use Euclidian distance between two data points in their computations, this is a problem

If unscaled, these algorithms only take in the magnitude of features neglecting the units. **The features with high magnitudes will weigh in a lot more in the distance calculations than features with low magnitudes**. For example, A variable that ranges between 0 and 1000 will outweigh a variable that ranges between 0 and 1. For an algorithm, 1 dollar is same as 1 day. To supress this effect, we need to bring all features to the same level of magnitudes. This can be achieved by scaling.

In scaling, we’re changing the range of our data while in normalization we’re mostly changing the shape of the distribution of our data. We need to normalize the data if we’re going to use a machine learning algorithm that assumes that the data is normally distributed e.g., linear regression, linear discriminant analysis (LDA) and Gaussian Naive Bayes.

With an example: Un normalized data

Consider a dataset of houses. Two potential features might be the number of rooms in the house, and the total age of the house in years. A machine learning algorithm could try to predict which house would be best for you. However, when the algorithm compares data points, the feature with the larger scale will completely dominate the other. Take a look at the image below:

The machine learning algorithm should realize that there is a huge difference between a house with 2 rooms and a house with 20 rooms. But right now, because two houses can be 100 years apart, the difference in the number of rooms contributes less to the overall difference. As a more extreme example, imagine what the graph would look like if the x-axis was the cost of the house. The data would look even more squished; the difference in the number of rooms would be even less relevant because the cost of two houses could have a difference of thousands of dollars.

The goal of normalization is to make every data point have the same scale so each feature is equally important.

There are 4 ways to do so:

**Normalization and Standardization**

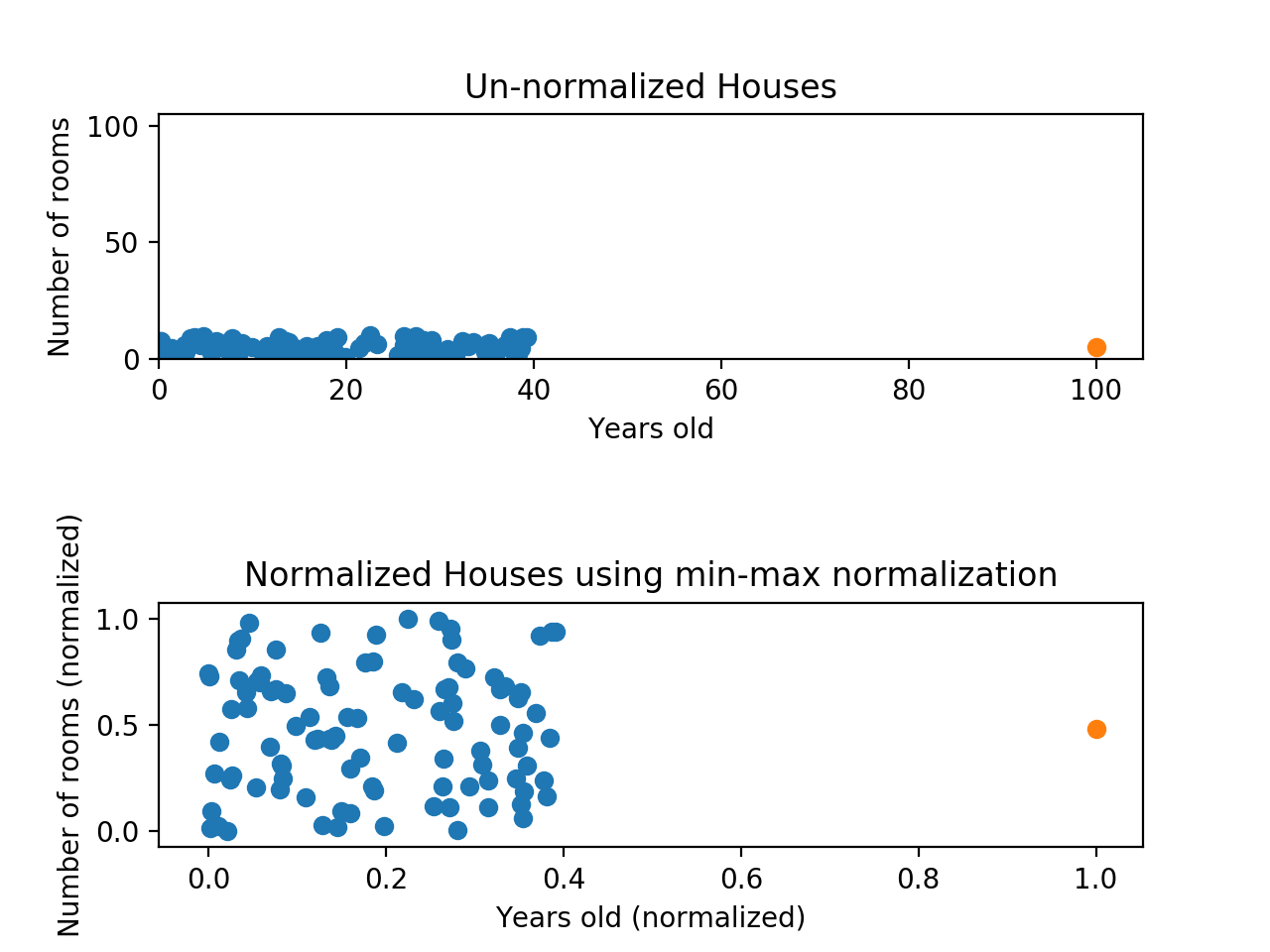
The point of normalization is to change your observations so that they can be described as a normal distribution.

**1. Min-Max Scaling (Normalization):** This scaling brings the values within a specific range e.g. 0 to 1

Algorithms: SVM, KNN, K-Means

Min-max normalization: Guarantees all features will have the exact same scale but does not handle outliers well.

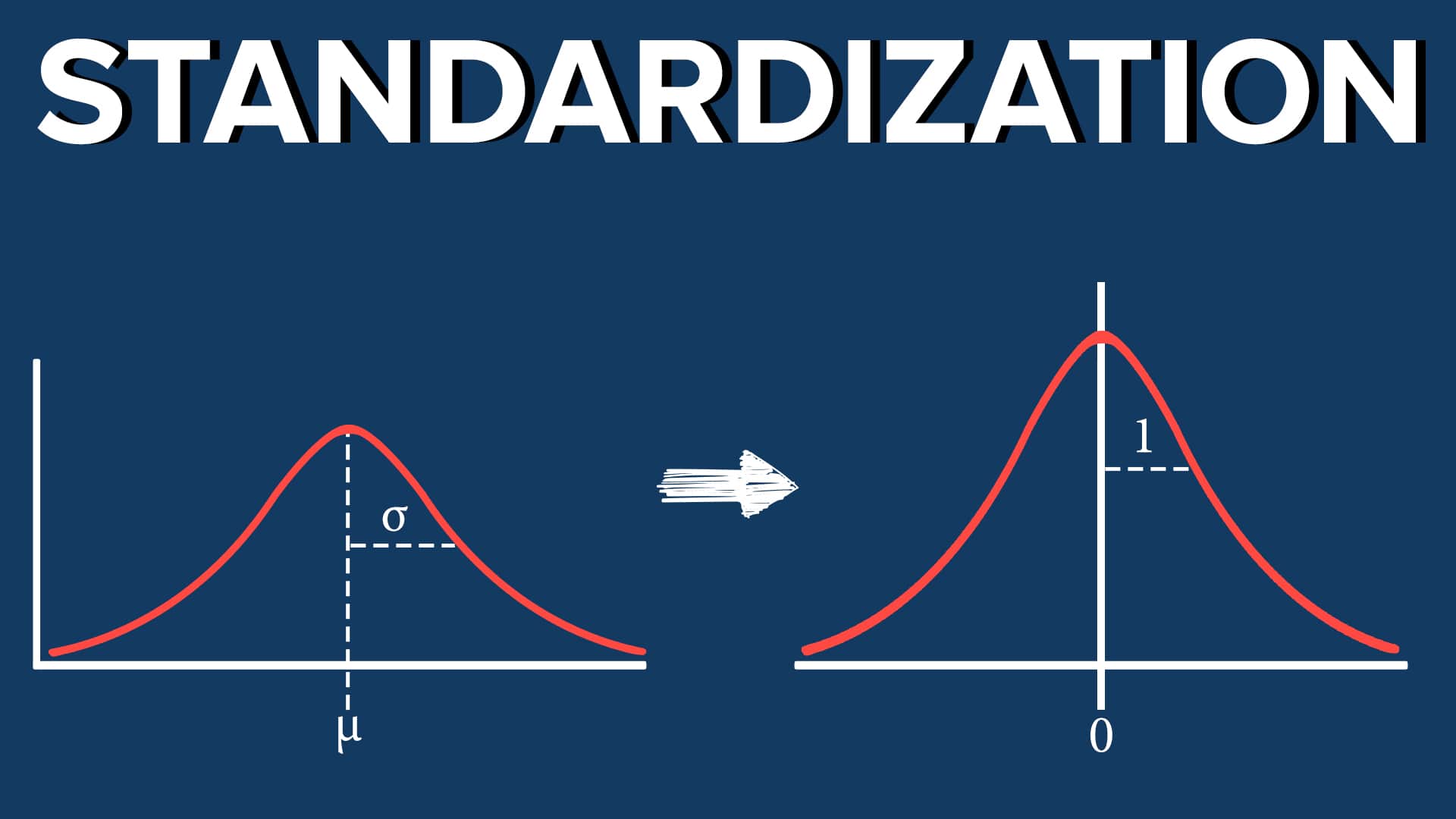
Looking back at the housing example, if you have 99 values between 0 and 40, and one value is 100, then the 99 values will all be transformed to a value between 0 and 0.4. That data is just as squished as before! Take a look at the image below:

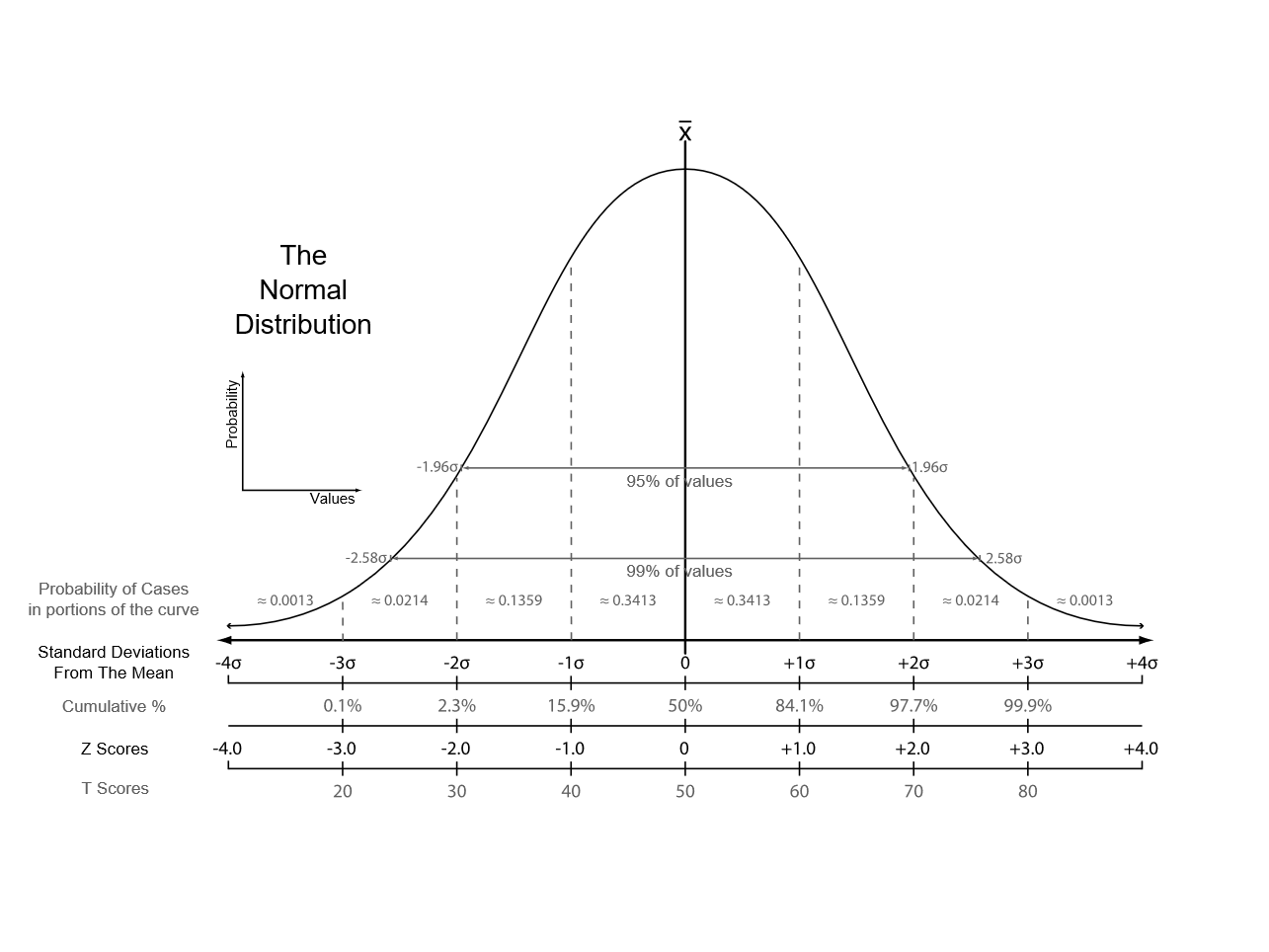


Normalizing fixed the squishing problem on the y-axis, but the x-axis is still problematic. Now if we were to compare these points, the y-axis would dominate; the y-axis can differ by 1, but the x-axis can only differ by 0.4.

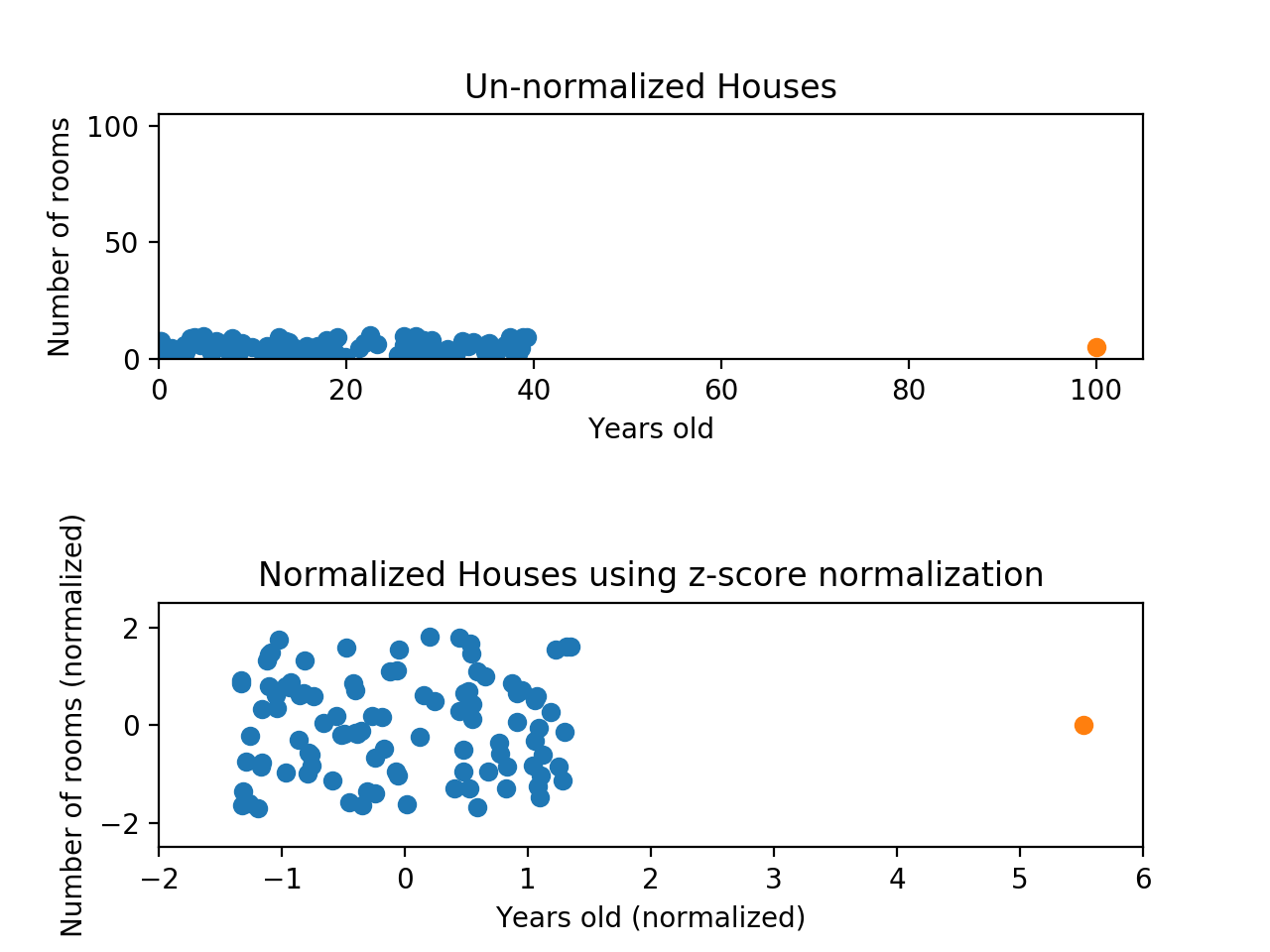
**2. Standardisation (z-score Normalization):** Standardisation replaces the values by their Z scores. This redistributes the features with their mean μ = 0 and standard deviation σ =1. The range of the new negative and positive numbers is determined by the standard deviation of the original feature. If the un-normalized data had a large standard deviation, the normalized values will be closer to 0.

Algorithms:SVM, PCA, logistics regression and neural networks.





Looking back at the housing example, While the data still looks squished, notice that the points are now on roughly the same scale for both features — almost all points are between -2 and 2 on both the x-axis and y-axis. The only potential downside is that the features aren’t on the exact same scale.



With min-max normalization, we were guaranteed to reshape both of our features to be between 0 and 1. Using z-score normalization, the x-axis now has a range from about -1.5 to 1.5 while the y-axis has a range from about -2 to 2. This is certainly better than before; the x-axis, which previously had a range of 0 to 40, is no longer dominating the y-axis.

Z-score normalization: Handles outliers, but does not produce normalized data with the exact same scale.

**3. Mean Normalisation:** This distribution will have values between -1 and 1 with μ=0.

Algorithms:PCA

**4. Unit Vector:** This scaling brings the value between 0 and 1.

# The when?

When Should you Use Normalization And Standardization:

***Normalization*** is a good technique to use when you do not know the distribution of your data or when you know the distribution is not Gaussian (a bell curve). In this approach, the data is scaled to a fixed range — **usually 0 to 1.**

In contrast to standardization, the cost of having this bounded range is that we will end up with smaller standard deviations, which can suppress the effect of outliers. Thus Min/Max Scalar is sensitive to outliers.

Normalization is useful when your data has varying scales and the algorithm you are using does not make assumptions about the distribution of your data, such as k-nearest neighbors and artificial neural networks.

***Standardization*** assumes that your data has a Gaussian (bell curve) distribution. This does not strictly have to be true, but the technique is more effective if your attribute distribution is Gaussian.

However, the minimum and maximum values after rescaling, vary according to how spread out the variable was, to begin with, and is highly influenced by the presence of outliers.

Standardization is useful when your data has varying scales and the algorithm you are using does make assumptions about your data having a Gaussian distribution, such as linear regression, logistic regression, and linear discriminant analysis.

***Min-Max scaling and Unit Vector scaling:*** 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.

Some examples of algorithms where feature scaling is required are:

|  |  |
| --- | --- |
| **Required** | **Not required** |
| k-nearest neighbors  Euclidean distance measure is sensitive to magnitudes and hence should be scaled for all features to weigh in equally. | Decision Trees |
| Principal Component Analysis(PCA)  PCA tries to get the features with maximum variance and the variance is high for high magnitude features. This skews the PCA towards high magnitude features. | Random Forests |
| Linear Regression  We can speed up gradient descent by scaling | XG boost |
|  | Linear Discriminant Analysis(LDA) |
| Naive Bayes |