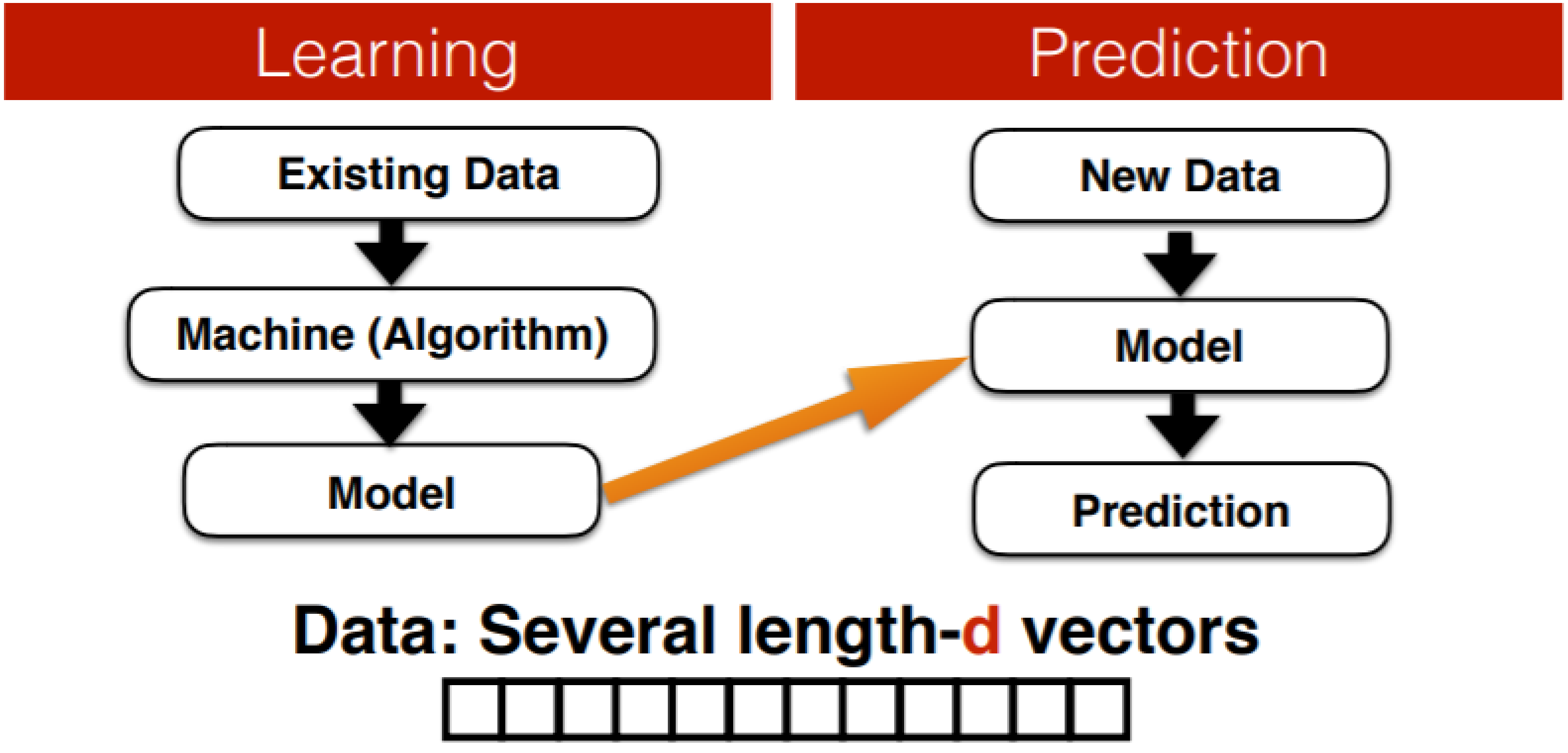
Feature Engineering and Preprocessing

shaileshsivan@gmail.com

Overview

- Dimensionality Reduction
- Components of Dimensionality Reduction
- Methods of Dimensionality Reduction
- Feature Reduction Iris Dataset
- Preprocessing

What machine learning does?



Some Facts

Expectation :

- We have good enough data
- Do focus on designing better algorithms

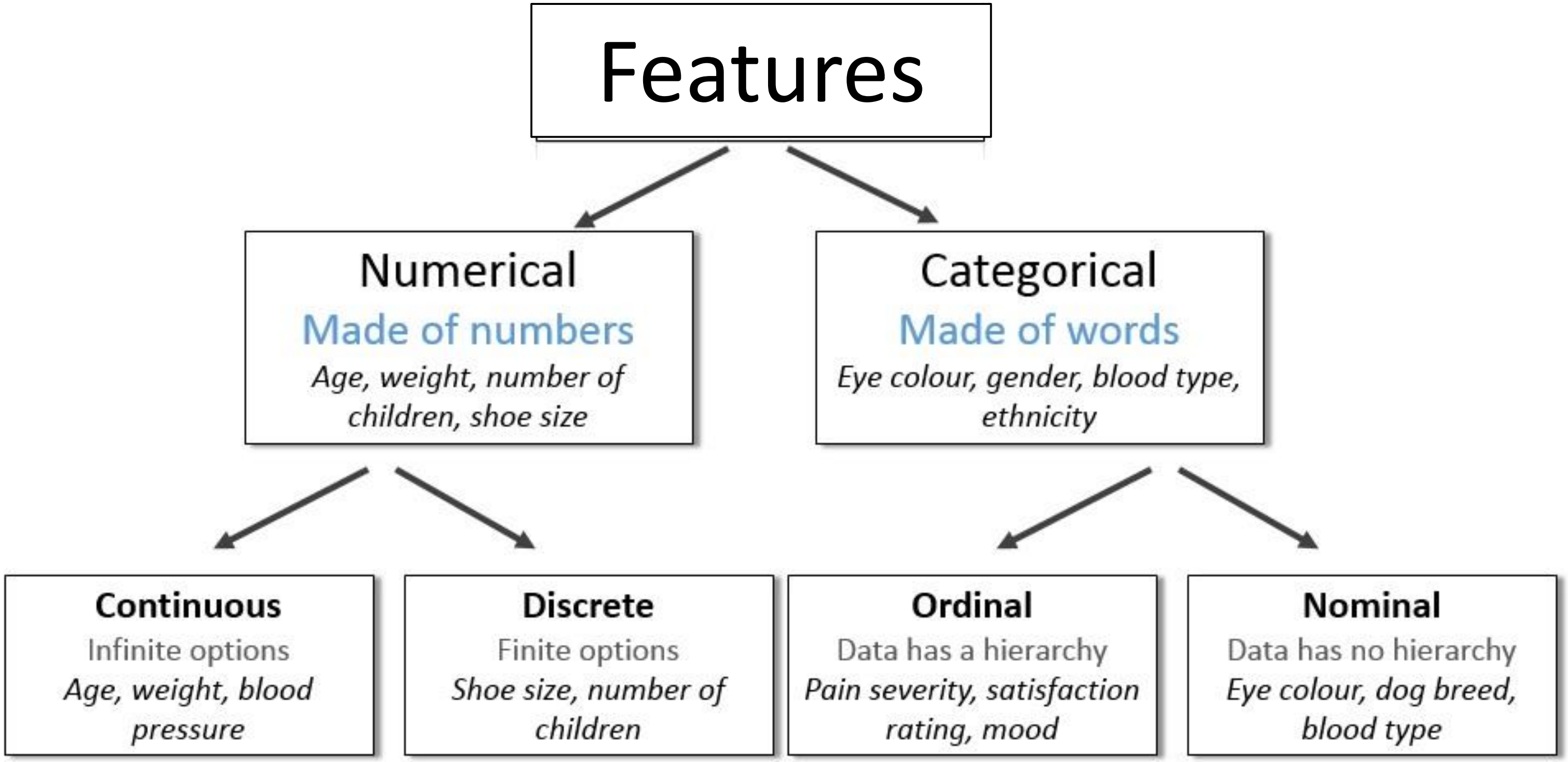
Reality :

- We have large amount of data but not *good enough*
- How to transform your data into learning compactable ?

What is a Feature in ML

- A **feature** is a measurable property of the object you're trying to analyze. In datasets, features appear as columns
- **Feature engineering** is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data.
- **Feature engineering** turn your inputs into things the algorithm can understand

Types of Features

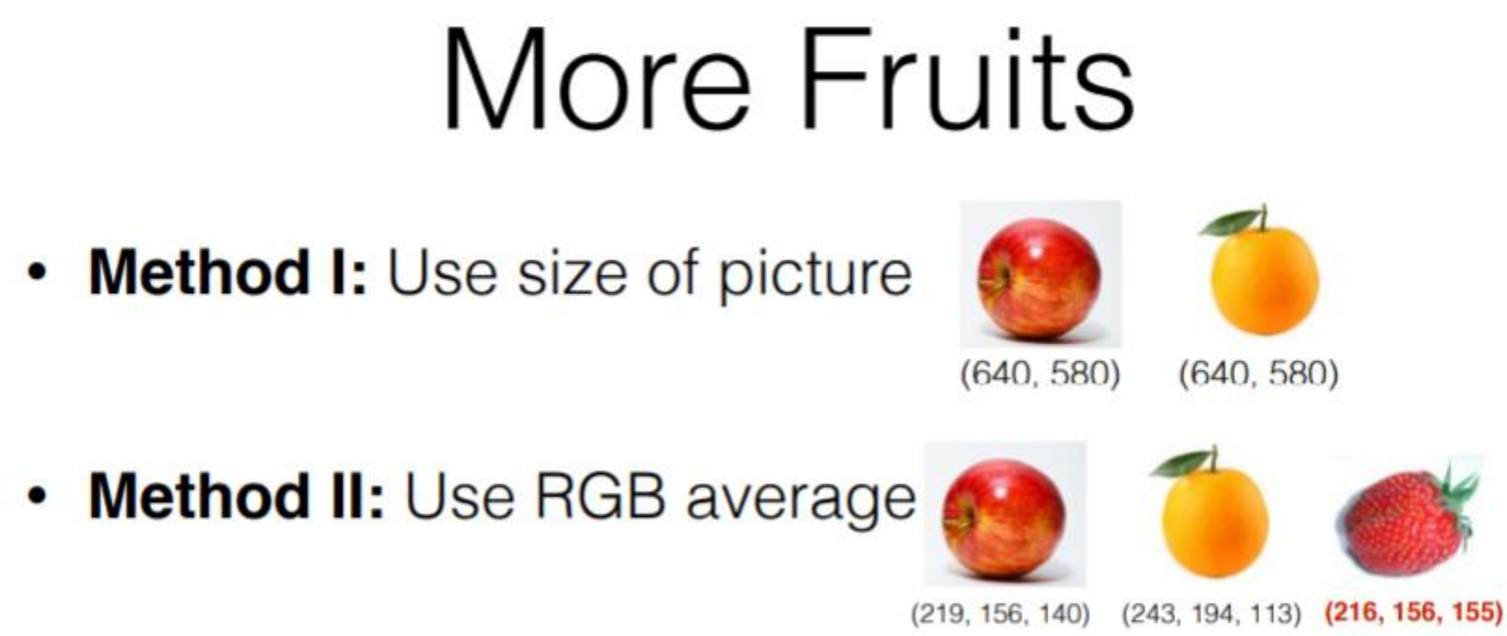


Features from observation

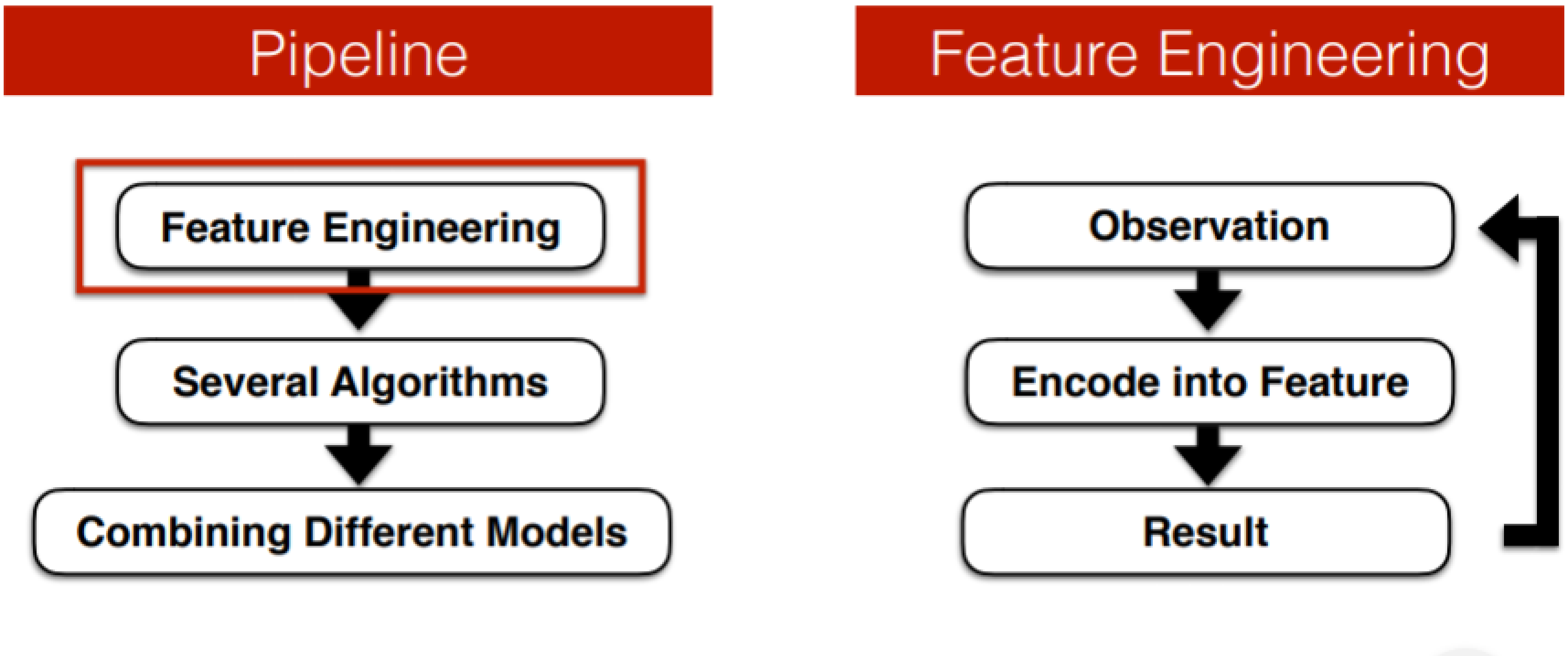
An Apple

How to describe this picture?

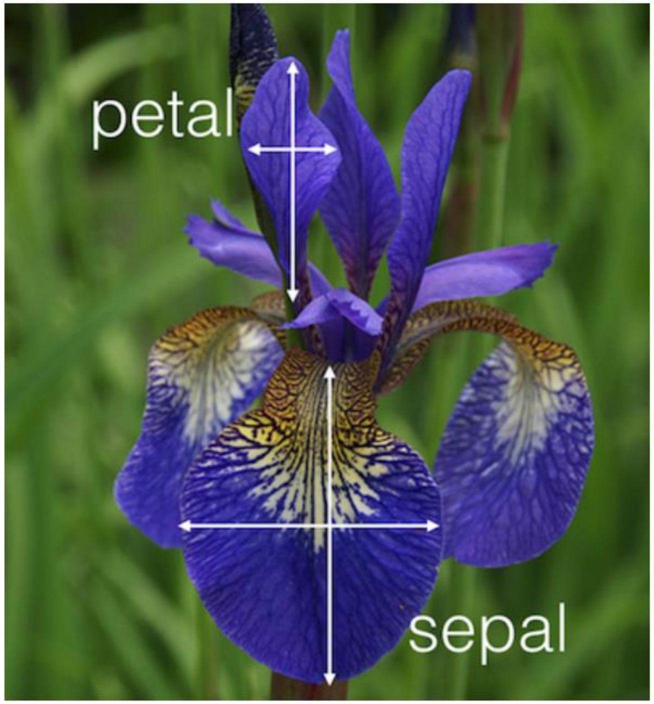
Features from observation



Feature Engineering



Features in iris data set



Source : Kaggle
https://www.kaggle.com/uciml/iris

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

first 5 rows of the Iris Dataset

Iris dataset in scikit-learn

- **scikit-learn** – machine learning in python
- Simple and efficient tools for data mining and data analysis
- several machine learning algorithms

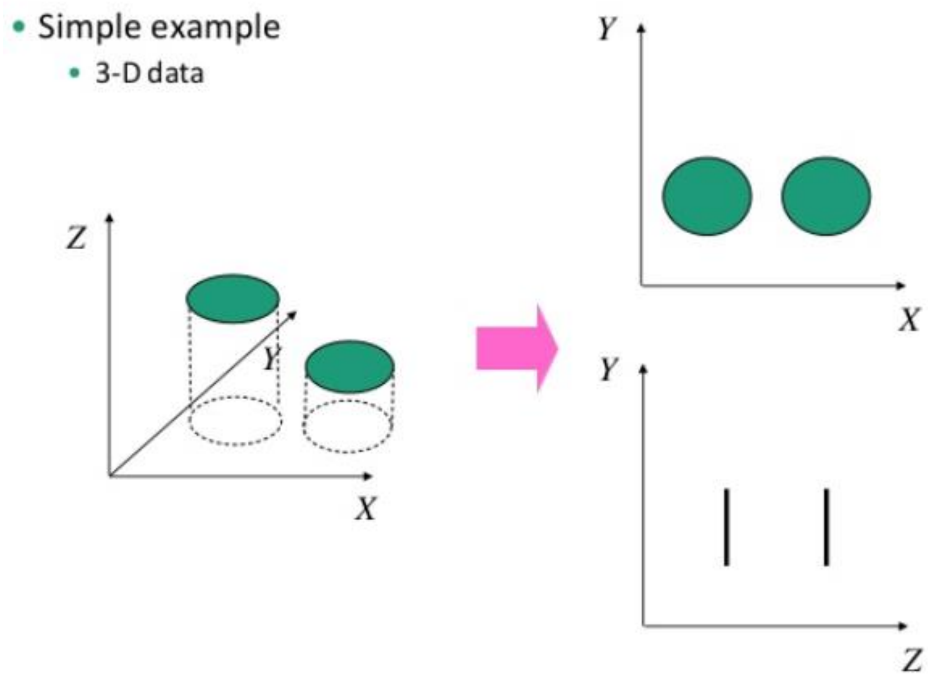
```python
from sklearn import datasets

# import some data to play with
iris = datasets.load_iris()
print(iris.data)
print(iris.target)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]]
[0 0 0 0 0 0 0 0 0 0]
```

Dimensionality Reduction

Dimensionality reduction - Reducing the number of random variables to consider.



Dimensionality Reduction

- ‘**Dimensionality**’ - simply refers to the number of features (i.e. input variables) in your dataset.
- When the number of features is very large relative to the number of observations in your dataset, certain algorithms struggle to train effective models.
- This is called the “Curse of Dimensionality,”
- It’s especially relevant for clustering algorithms that rely on distance calculations.

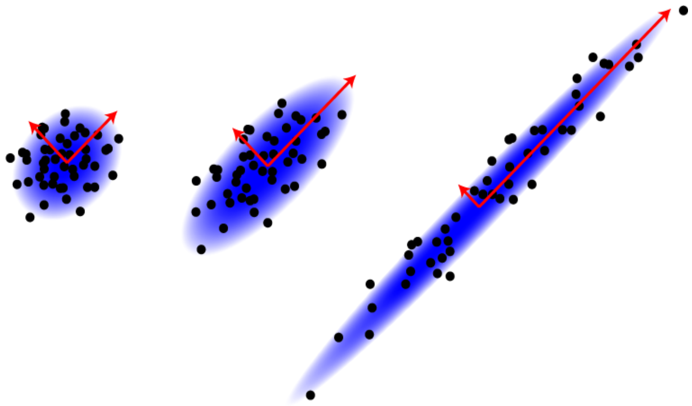
Components of Dimensionality Reduction

Feature selection: you select a subset of the original feature set.

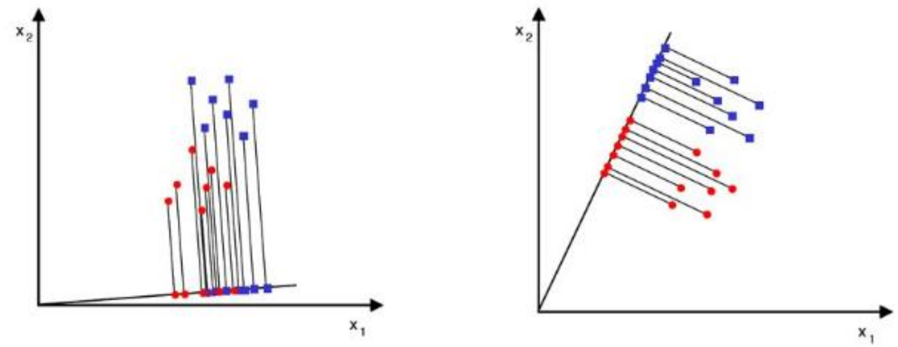
Feature extraction: you build a new set of features from the original feature set.

Methods of Dimensionality Reduction

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)



- Reduce dimensionality, preserve as much class discriminatory information as possible



Feature Reduction Iris Dataset

```
import matplotlib.pyplot as plt

from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

iris = datasets.load_iris()

X = iris.data
y = iris.target
target_names = iris.target_names

pca = PCA(n_components=2)
X_r = pca.fit(X).transform(X)
print(X_r)

lda = LDA(n_components=2)
X_r2 = lda.fit(X, y).transform(X)
print(X_r2)
```

Feature Reduction Iris Dataset

PCA OUTPUT

[[-2.68412563 0.31939725] [-2.71414169 -0.17700123] [-2.88899057 -0.14494943] [-2.74534286 -0.31829898] [-2.72871654 0.32675451] [-2.28085963 0.74133045] [-2.82053775 -0.08946138] [-2.62614497 0.16338496] [-2.88638273 -0.57831175] --[2.15943764 -0.21727758] [1.44416124 -0.14341341] [1.78129481 -0.49990168] [3.07649993 0.68808568] [2.14424331 0.1400642] [1.90509815 0.04930053] [1.16932634 -0.16499026] [2.10761114 0.37228787] [2.31415471 0.18365128] [1.9222678 0.40920347] [1.41523588 -0.57491635] [2.56301338 0.2778626] [2.41874618 0.3047982] [1.94410979 0.1875323] [1.52716661 -0.37531698] [1.76434572 0.07885885] [1.90094161 0.11662796] [1.39018886 -0.28266094]]

LDA OUTPUT

[[-8.06179978e+00 3.00420621e-01] [-7.12868772e+00 -7.86660426e-01] [-7.48982797e+00 -2.65384488e-01] [-6.81320057e+00 -6.70631068e-01] [-8.13230933e+00 5.14462530e-01] [-7.70194674e+00 1.46172097e+00] [-7.21261762e+00 3.55836209e-01] --[9.97610366e-01 -4.90530602e-01] [3.83525931e+00 -1.40595806e+00] [2.25741249e+00 -1.42679423e+00] [1.25571326e+00 -5.46424197e-01] [1.43755762e+00 -1.34424979e-01] [2.45906137e+00 -9.35277280e-01] [3.51848495e+00 1.60588866e-01] [2.58979871e+00 -1.74611728e-01] [-3.07487884e-01 -1.31887146e+00] [4.96774090e+00 8.21140550e-01] [5.88614539e+00 2.34509051e+00] [4.68315426e+00 3.32033811e-01]]

Preprocessing

Scaling

- The **MinMaxScaler** is the probably the most famous scaling algorithm, and follows the following formula for each feature:

$$\frac{x_i - \min(x)}{\max(x) - \min(x)}$$

- Shrinks the range such that the range is now between 0 and 1 (or -1 to 1 if there are negative values).

```
from sklearn.preprocessing import MinMaxScaler

data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
scaler = MinMaxScaler()
print(scaler.fit(data))
#MinMaxScaler(copy=True, feature_range=(0, 1))
print(scaler.transform(data))
#[[0.   0.  ]
# [0.25 0.25]
# [0.5  0.5 ]
# [1.   1.  ]]
print(scaler.transform([[2, 2]]))
#[[1.5 0. ]]
```

Preprocessing

Label Encoding

- Used to transform non-numerical labels that is categorical values to numerical labels.

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(["paris", "paris", "tokyo", "amsterdam"])
enlabels=le.transform(["paris","tokyo", "tokyo", "paris"])
print(enlabels)
#[1 2 2 1]
print(le.inverse_transform(enlabels))
#['paris' 'tokyo' 'tokyo' 'paris']
```

Thank You