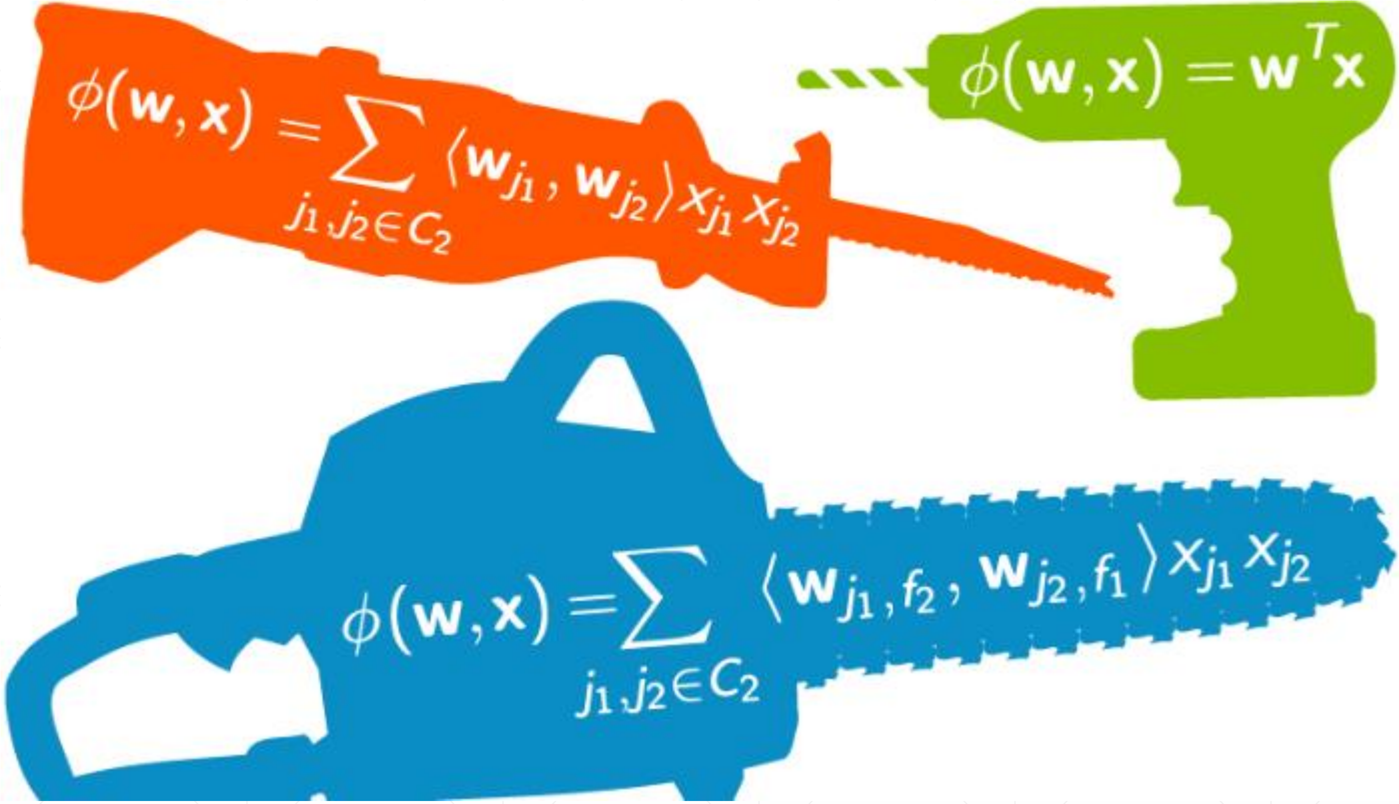


Feature Engineering

Mundher Al-Shabi

- Adapted from Gabriel Moreira’s talk

Big Data Borat
@BigDataBorat

Follow

In Data Science, 80% of time spent prepare data, 20% of time spent complain about need for prepare data.

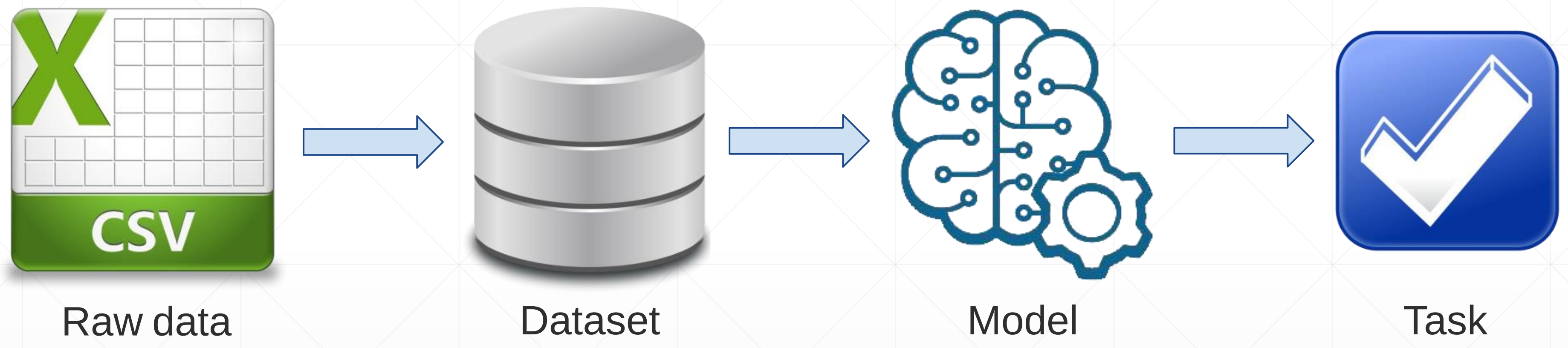
"Coming up with features is difficult, time-consuming, requires expert knowledge. 'Applied machine learning' is basically feature engineering."

– Andrew Ng

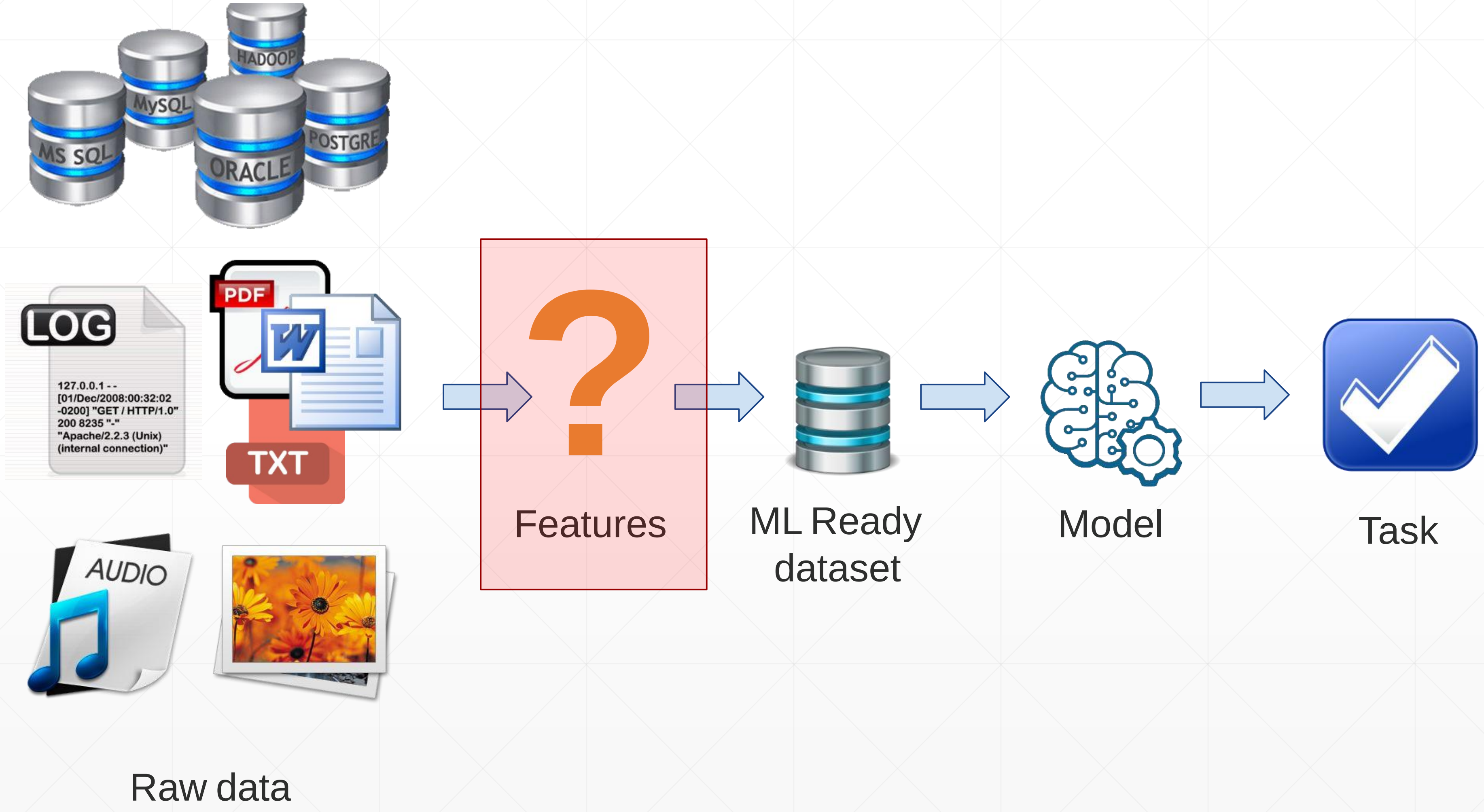
"More data beats clever algorithms, but better data beats more data."

– Peter Norvig

The Dream...



The Reality



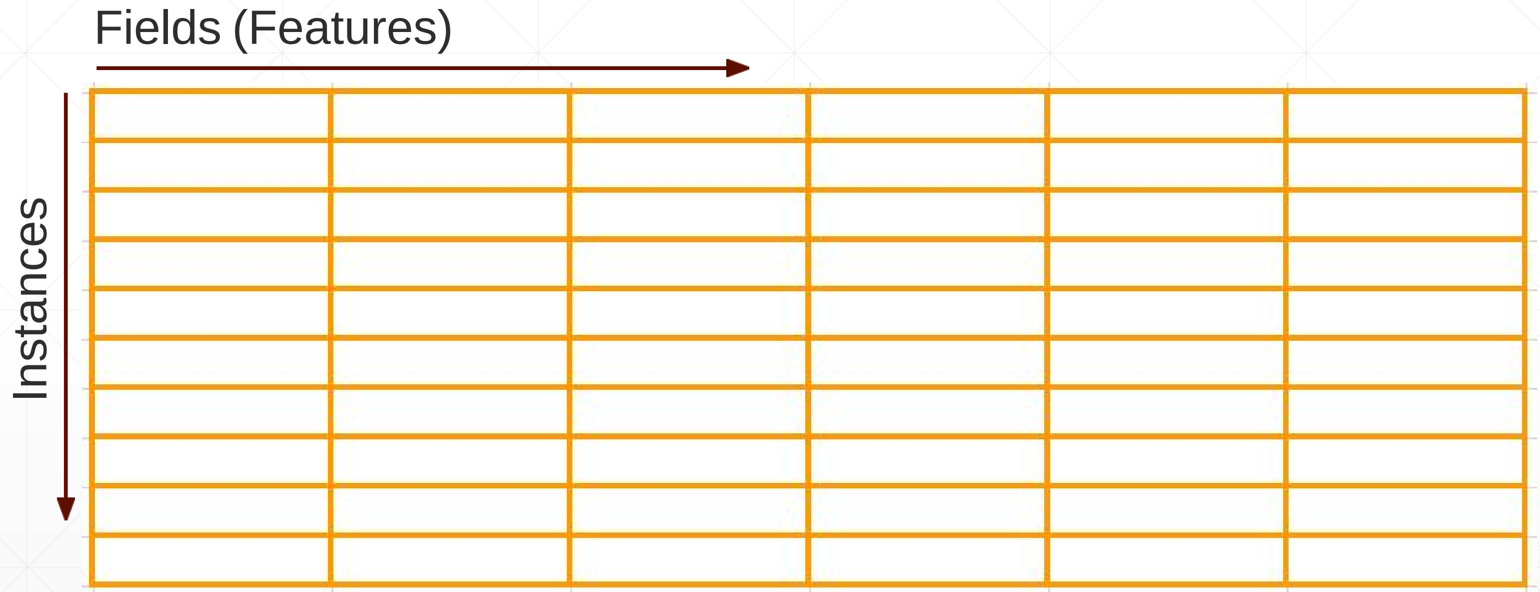
HADOOP
MySQL
MS SQL
ORACLE
POSTGRE
LOG
127.0.0.1 - -
[01/Dec/2008:00:32:02
-0200] "GET / HTTP/1.0"
200 8235 "-"
"Apache/2.2.3 (Unix)
(internal connection)"
PDF
W
TXT
AUDIO
?
Features
ML Ready
dataset
Model
Task
Raw data

First at all … a closer look at your data

- What does the data model look like?
- What is the features distribution?
- What are the features with missing or inconsistent values?
- What are the most predictive features?
- Conduct a Exploratory Data Analysis (EDA)

ML-Ready Dataset

- Tabular data (rows and columns)
  - Usually denormalized in a single file/dataset
  - Each row contains information about one instance
  - Each column is a feature that describes a property of the instance



Data Cleansing

- Homogenize missing values and different types of in the same feature, fix input errors, types, etc.

Name	Date	Duration (s)	Genre	Plays
Highway star	1984-05-24	-	Rock	139
Blues alive	1990/03/01	281	Blues	239
Lonely planet	2002-11-19	5:32s	Techno	42
Dance, dance	02/23/1983	312	Disco	N/A
The wall	1943-01-20	218	Reagge	83
Offside down	1965-02-19	4 minutes	Techno	895
The alchemist	2001-11-21	418	Bluesss	178
Bring me down	18-10-98	328	Classic	21
The scarecrow	1994-10-12	269	Rock	734

Original data

Cleaned data

Name	**Date**	**Duration (s)**	**Genre**	**Plays**
Highway star	1984-05-24		Rock	139
Blues alive	1990-03-01	281	Blues	239
Lonely planet	2002-11-19	332	Techno	42
Dance, dance	1983-02-23	312	Disco	
The wall	1943-01-20	218	Reagge	83
Offside down	1965-02-19	240	Techno	895
The alchemist	2001-11-21	418	Blues	178
Bring me down	1998-10-18	328	Classic	21
The scarecrow	1994-10-12	269	Rock	734

Numerical features

- Usually easy to ingest by mathematical models.
- Can be prices, measurements, counts, ...
- Easier to impute missing data
- Distribution and scale matters to many models

Imputation for missing values

- Datasets contain missing values, often encoded as blanks, NaNs or other placeholders
- Ignoring rows and/or columns with missing values is possible, but at the price of loosing data which might be valuable
- Better strategy is to infer them from the known part of data
- Strategies
  - **Mean**: Basic approach
  - **Median**: More robust to outliers
  - **Mode**: Most frequent value
  - **Using a model**: Can expose algorithmic bias

Imputation for missing values

```
>>> import numpy as np
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
>>> imp.fit([[1, 2], [np.nan, 3], [7, 6]])
Imputer(axis=0, copy=True, missing_values='NaN', strategy='mean',
verbose=0)
>>> X = [[np.nan, 2], [6, np.nan], [7, 6]]
>>> print(imp.transform(X))
[[ 4.          2.       ]
 [ 6.          3.666...]
 [ 7.          6.       ]]
```

Missing values imputation with scikit-learn

Binarization

- Transform discrete or continuous numeric features in binary features Example: Number of user views of the same document

document_id	uuid	views_count
25792	6d82e412aa0f0d	8
25792	571016386ffee7	6
25792	6a91157d820e37	6
25792	ad45fc764587b0	6
25792	a743b03f2b8ddc	3

document_id	uuid	viewed
25792	6d82e412aa0f0d	1
25792	571016386ffee7	1
25792	6a91157d820e37	1
25792	ad45fc764587b0	1
25792	8d87becfb35857	1
25792	abcdefg1234567	0

```
>>> from sklearn import preprocessing
>>> X = [[ 1., -1.,  2.],
...           [ 2.,  0.,  0.],
...           [ 0.,  1., -1.]]

>>> binarizer =
preprocessing.Binarizer(threshold=1.0)
>>> binarizer.transform(X)
array([[ 1., 0.,  1.],
          [ 1., 0.,  0.],
          [ 0., 1.,  0.]])
```

Binarization with scikit-learn

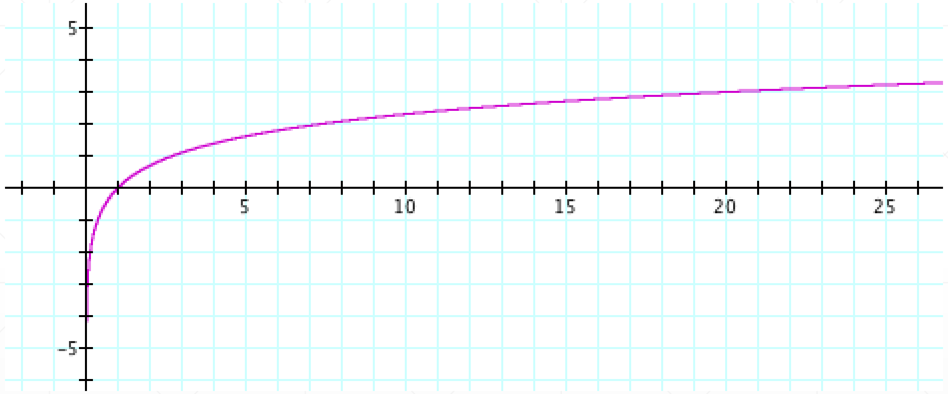
Rounding

- Form of lossy compression: retain most significant features of the data.
- Sometimes too much precision is just noise
- Rounded variables can be treated as categorical variables
- Example:
  - Some models like Association Rules work only with categorical features. It is possible to convert a percentage into categorical feature this way

document_id	topic_id	confidence	ROUND(confidence*10)
25792	1205	0.9594	10
15454	1545	0.1254	1
78764	958	0.1854	2
21548	1510	0.5454	5
48877	25	0.3655	4

Log transformation

- Compresses the range of large numbers and expand the range of small numbers. Eg. The larger x is, the slower log(x) increments

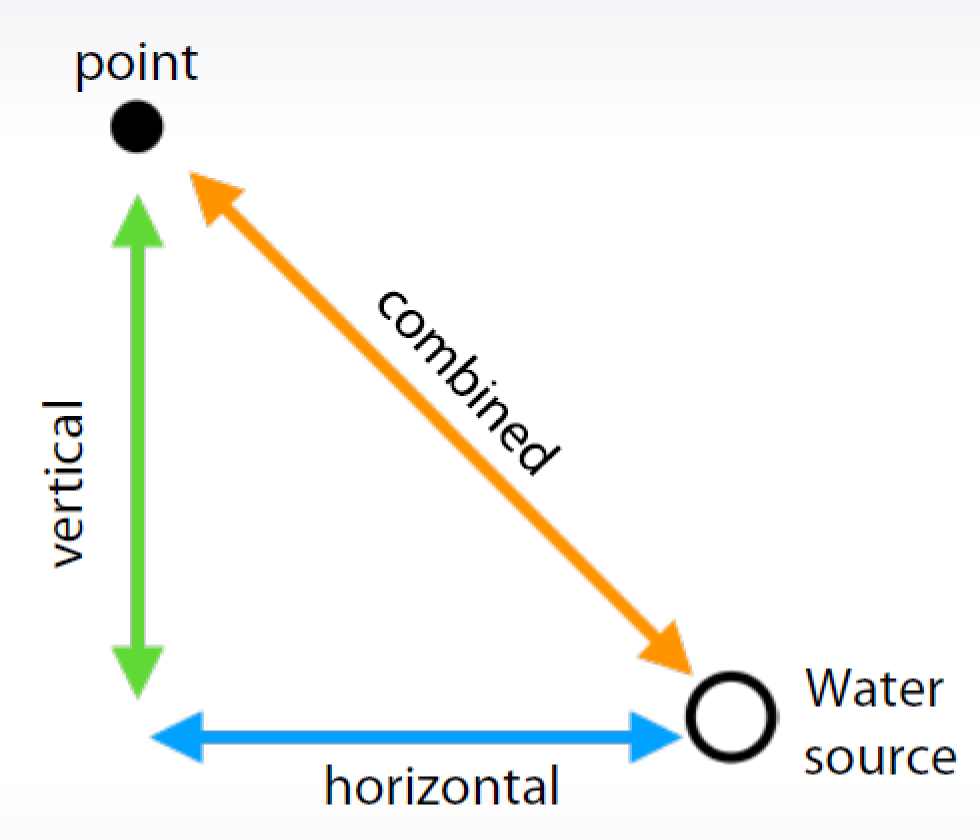


user_id	views_count
a	1000
b	500
c	300
d	200
e	150
f	100
g	70
h	50
i	30
j	20
k	10
l	5
m	1

log(1+views_count)
6.91
6.22
5.71
5.30
5.02
4.62
4.26
3.93
3.43
3.04
2.40
1.79
0.69

Feature generation

- Combined = (horizontal ** 2 + vectical ** 2) ** 0.5



Feature generation

price	fractional_part
0.99	0.99
2.49	0.49
1.0	0.0
9.99	0.99

Scaling

- Models that are smooth functions of input features are sensitive to the scale of the input (eg. Linear Regression)
- Scale numerical variables into a certain range, dividing values by a normalization constant (no changes in single-feature distribution)
- Popular techniques
  - MinMax Scaling
  - Standard (Z) Scaling

Min-max scaling

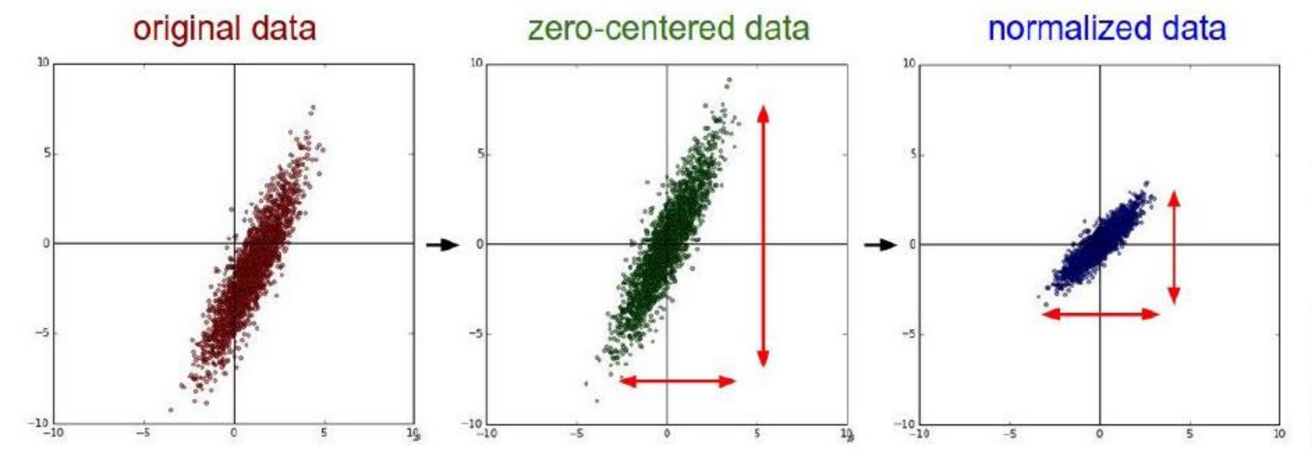
- Squeezes (or stretches) all values within the range of [0, 1] to add robustness to very small standard deviations and preserving zeros for sparse data.

```
>>> from sklearn import preprocessing
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0., 1., -1.]])
...
>>> min_max_scaler =
preprocessing.MinMaxScaler()
>>> X_train_minmax =
min_max_scaler.fit_transform(X_train)
array([[ 0.5       ,  0.        ,  1.        ],
       [ 1.        ,  0.5       ,  0.33333333],
       [ 0.        ,  1.        ,  0.        ]])
```

Min-max scaling with scikit-learn

Standard (Z) Scaling

- After Standardization, a feature has mean of 0 and variance of 1 (assumption of many learning algorithms)




```
>>> from sklearn import preprocessing
>>> import numpy as np
>>> X = np.array([[ 1., -1.,  2.],
...                [ 2.,  0.,  0.],
...                [ 0.,  1., -1.]])
>>> X_scaled = preprocessing.scale(X)
>>> X_scaled
array([[ 0.  ..., -1.22...,  1.33...],
       [ 1.22...,  0.  ..., -0.26...],
       [-1.22..., 1.22..., -1.06...]])
>> X_scaled.mean(axis=0)
array([ 0., 0.,  0.])
>>> X_scaled.std(axis=0)
array([ 1., 1.,  1.])
```

Standardization with scikit-learn

Interaction Features

- Simple linear models use a linear combination of the individual input features, $x1$, $x2$, ... xn to predict the outcome y.
- $y = w1x1 + w2x2 + ... + wnxn$
- An easy way to increase the complexity of the linear model is to create feature combinations (nonlinear features).

Interaction Features

$$(X_1, X_2) \Longrightarrow (1, X_1, X_2, X_1^2, X_1X_2, X_2^2)$$

```
>>> import numpy as np
>>> from sklearn.preprocessing import PolynomialFeatures
>>> X = np.arange(6).reshape(3, 2)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5]])
>>> poly = poly = PolynomialFeatures(degree=2, interaction_only=False,
include_bias=True)
>>> poly.fit_transform(X)
array([[  1.,   0.,   1.,   0.,   0.,   1.],
       [  1.,   2.,   3.,   4.,   6.,   9.],
       [  1.,   4.,   5.,  16.,  20.,  25.]])
```

Polynomial features with scikit-learn

Categorical Features

- Nearly always need some treatment to be suitable for models
- Examples:
- Platform: [“desktop”, “tablet”, “mobile”] Document_ID or User_ID: [121545, 64845, 121545]
- High cardinality can create very sparse data
- Difficult to impute missing

Label encoding

- Give every categorical variable a unique numerical ID
- Useful for non-linear tree-based algorithms
- Does not increase dimensionality
- Randomize the cat_var -> num_id

```
>>> le = preprocessing.LabelEncoder()
>>> le.fit(["paris", "paris", "tokyo", "amsterdam"])
LabelEncoder()
>>> list(le.classes_)
['amsterdam', 'paris', 'tokyo']
>>> le.transform(["tokyo", "tokyo", "paris"])
array([2, 2, 1]...)
```

One-Hot Encoding

- Transform a categorical feature with m possible values into m binary features.
- If the variable cannot be multiple categories at once, then only one bit in the group can be on.

platform
desktop
mobile
tablet

platform=desktop	platform=mobile	platform=tablet
1	0	0
0	1	0
0	0	1

One-Hot Encoding

```
>>> import pandas as pd
>>> s = pd.Series(list('abca'))
>>> pd.get_dummies(s)
   a  b  c
0  1  0  0
1  0  1  0
2  0  0  1
3  1  0  0
```

Date and time

- Periodicity
  - Day number in week, month, season, year, second, minute, hour.
- Time since
  - Row-independent moment
    For example: since 00:00:00 UTC, 1 January 1970;
  - Row-dependent important moment
    Number of days left until next holidays/ time passed after last holiday.
- Difference between dates
  - datetime_feature_1 - datetime_feature_2

Periodicity. «Time since»

Date	weekday	daynumber_since_year_2014	is_holiday	days_till_holidays
01.01.2014	5	0	True	0
02.01.2014	6	1	False	3
03.01.2014	0	2	False	2
04.01.2014	1	3	False	1
05.01.2014	2	4	True	0
06.01.2014	3	5	False	9

Difference between dates

user_id	registration_date	*last_purchase_date*	*last_call_date*	date_diff	churn
14	10.02.2016	21.04.2016	26.04.2016	5	0
15	10.02.2016	03.06.2016	01.06.2016	-2	1
16	11.02.2016	11.01.2017	11.01.2017	1	1
20	12.02.2016	06.11.2016	08.02.2017	94	0

Coordinates

"...some machine learning projects succeed and some fail.

Where is the difference?

Easily the most important factor is the features used."

– Pedro Domingos