QCon
SÃO PAULO
2017

Feature Engineering

Getting the most out of data for predictive models

Gabriel Moreira
@gspmoreira

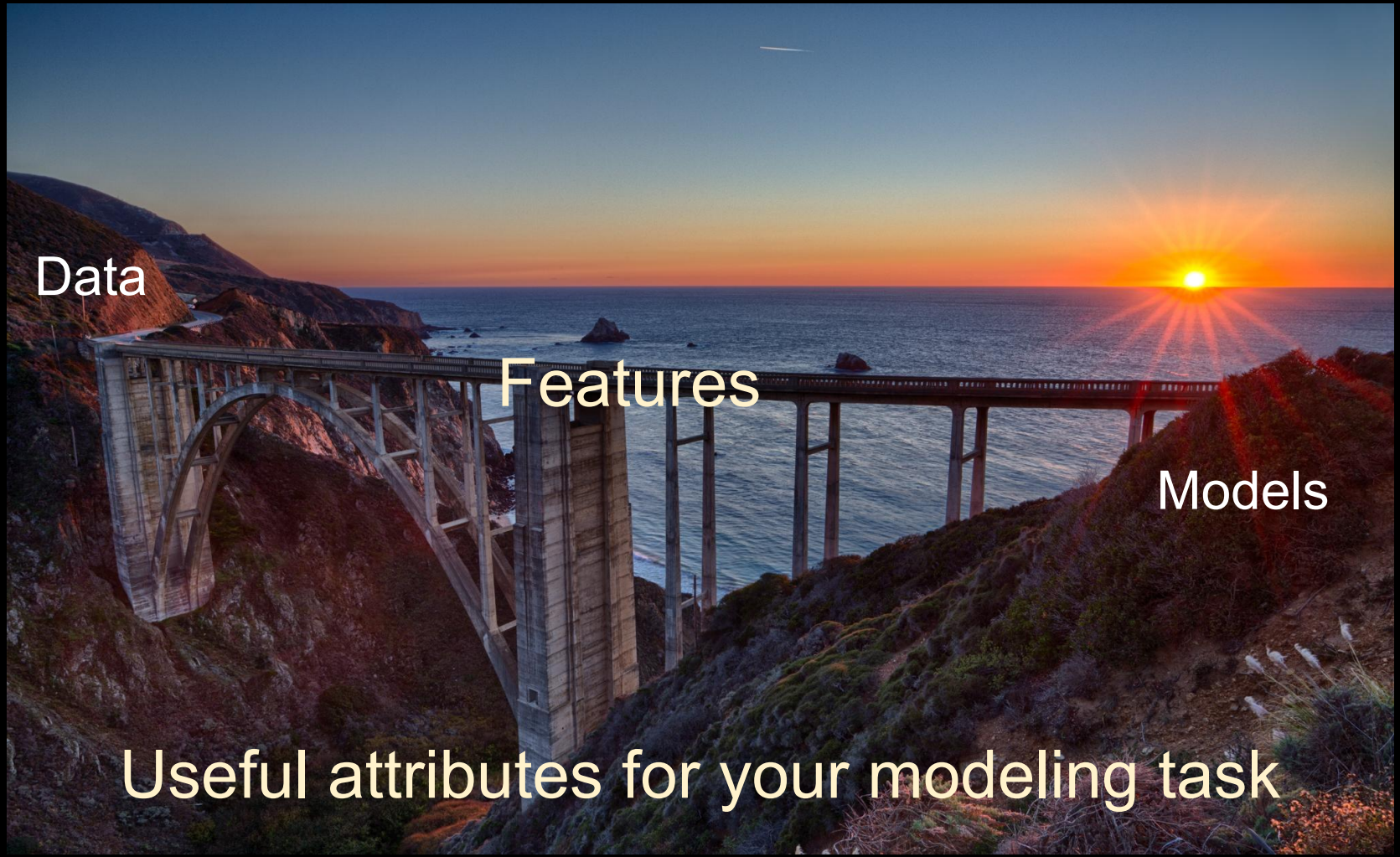
Lead Data Scientist
ciandt.com

DSc. student
ITA

Agenda

- Machine Learning Pipeline
- Data Munging
- Feature Engineering
  - Numerical features
  - Categorical features
  - Temporal and Spatial features
  - Textual features
- Feature Selection

Extra slides marker


Data
Features
Models
Useful attributes for your modeling task

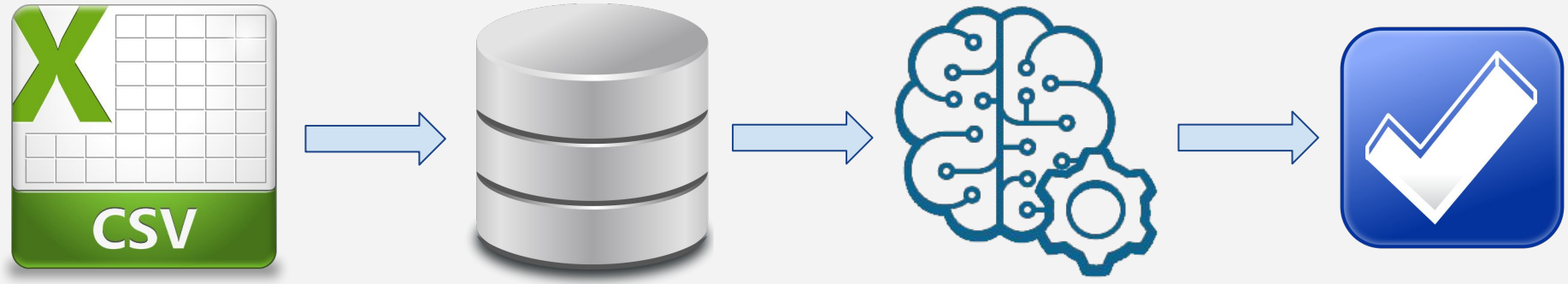
"Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data."

– Jason Brownlee

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

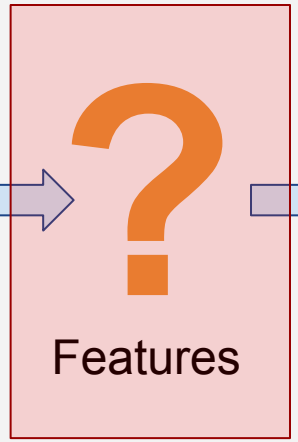
– Andrew Ng

The Dream...



… The Reality

Raw data



Features

ML Ready dataset

Model

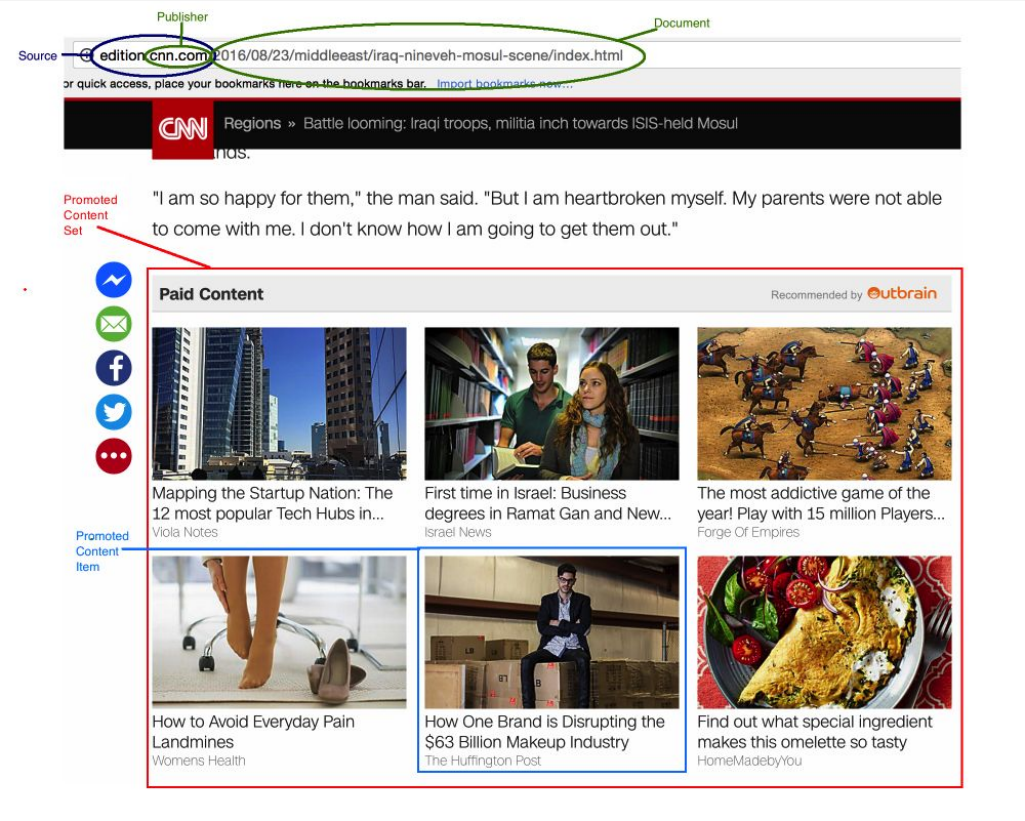
Task

Here are some Feature Engineering techniques for your Data Science toolbox...

Case Study

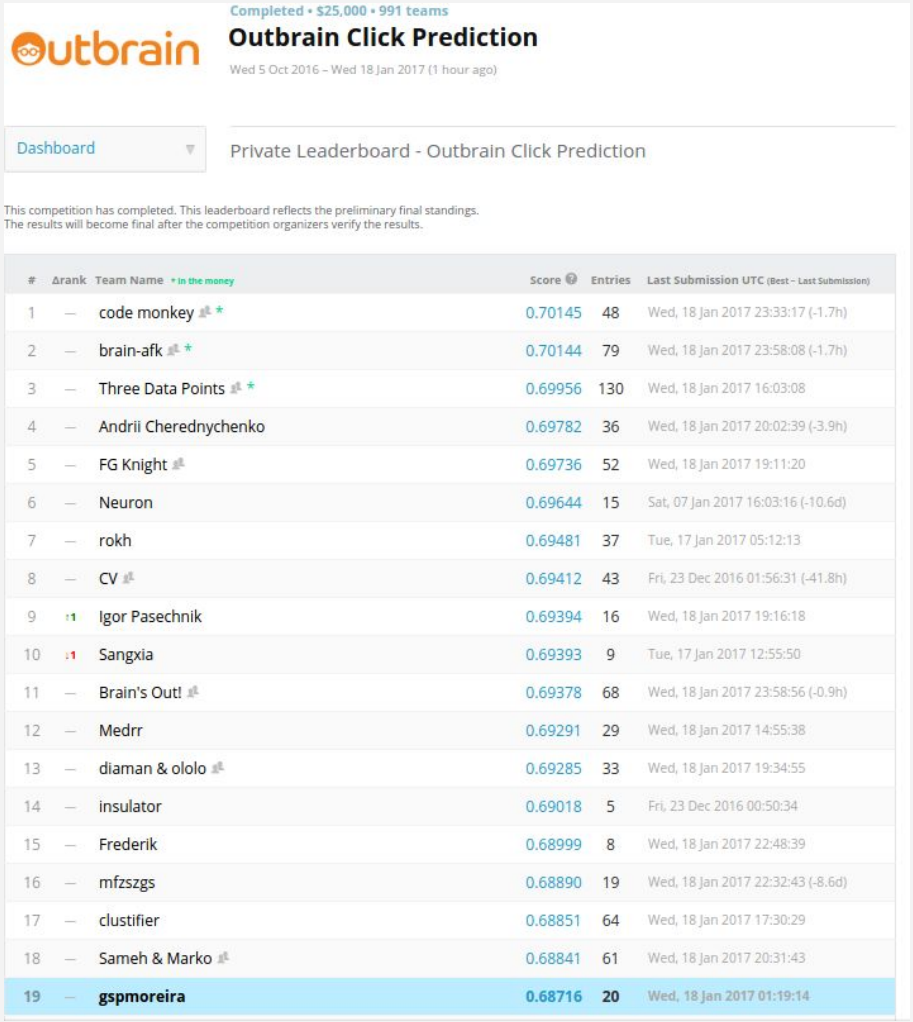
Outbrain Click Prediction - Kaggle competition

Can you predict which recommended content each user will click?



Dataset

- Sample of users page views and clicks during 14 days on June, 2016
- 2 Billion page views
- 17 million click records
- 700 Million unique users
- 560 sites



Completed • $25,000 • 991 teams

Outbrain Click Prediction

Wed 5 Oct 2016 – Wed 18 Jan 2017 (1 hour ago)

Dashboard

Private Leaderboard - Outbrain Click Prediction

This competition has completed. This leaderboard reflects the preliminary final standings.
The results will become final after the competition organizers verify the results.

#	Δrank	Team Name * in the money	Score	Entries	Last Submission UTC (Best – Last Submission)
1	—	code monkey *	0.70145	48	Wed, 18 Jan 2017 23:33:17 (-1.7h)
2	—	brain-afk *	0.70144	79	Wed, 18 Jan 2017 23:58:08 (-1.7h)
3	—	Three Data Points *	0.69956	130	Wed, 18 Jan 2017 16:03:08
4	—	Andrii Cherednychenko	0.69782	36	Wed, 18 Jan 2017 20:02:39 (-3.9h)
5	—	FG Knight	0.69736	52	Wed, 18 Jan 2017 19:11:20
6	—	Neuron	0.69644	15	Sat, 07 Jan 2017 16:03:16 (-10.6d)
7	—	rokh	0.69481	37	Tue, 17 Jan 2017 05:12:13
8	—	CV	0.69412	43	Fri, 23 Dec 2016 01:56:31 (-41.8h)
9	↑1	Igor Pasechnik	0.69394	16	Wed, 18 Jan 2017 19:16:18
10	↓1	Sangxia	0.69393	9	Tue, 17 Jan 2017 12:55:50
11	—	Brain's Out!	0.69378	68	Wed, 18 Jan 2017 23:58:56 (-0.9h)
12	—	Medrr	0.69291	29	Wed, 18 Jan 2017 14:55:38
13	—	diaman & ololo	0.69285	33	Wed, 18 Jan 2017 19:34:55
14	—	insulator	0.69018	5	Fri, 23 Dec 2016 00:50:34
15	—	Frederik	0.68999	8	Wed, 18 Jan 2017 22:48:39
16	—	mfzszgs	0.68890	19	Wed, 18 Jan 2017 22:32:43 (-8.6d)
17	—	clustifier	0.68851	64	Wed, 18 Jan 2017 17:30:29
18	—	Sameh & Marko	0.68841	61	Wed, 18 Jan 2017 20:31:43
19	—	**gspmoreira**	**0.68716**	**20**	**Wed, 18 Jan 2017 01:19:14**

I got **19th** position
from about
1000 competitors
(top 2%),
mostly due to
Feature Engineering
techniques.

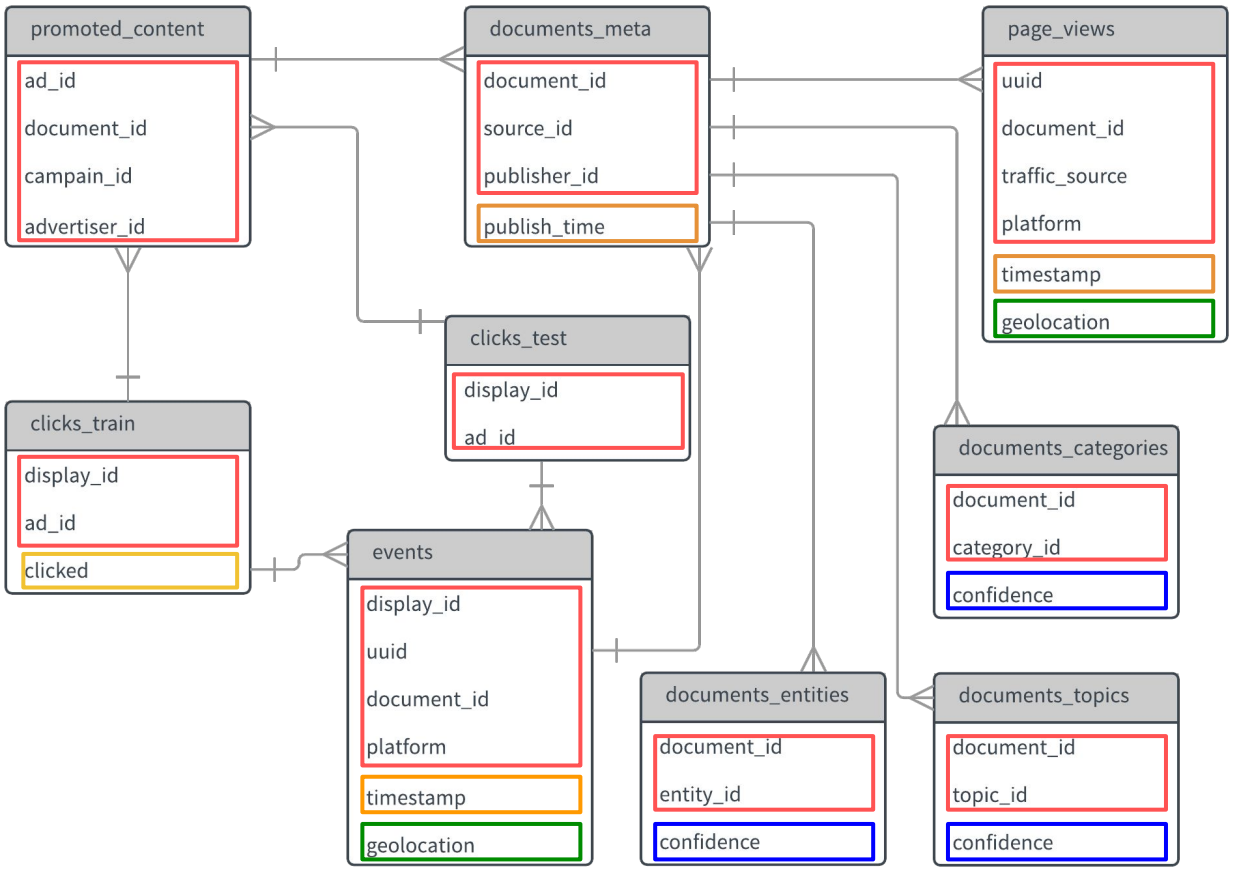
Data Munging

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)

Outbrain Click Prediction - Data Model



Numerical

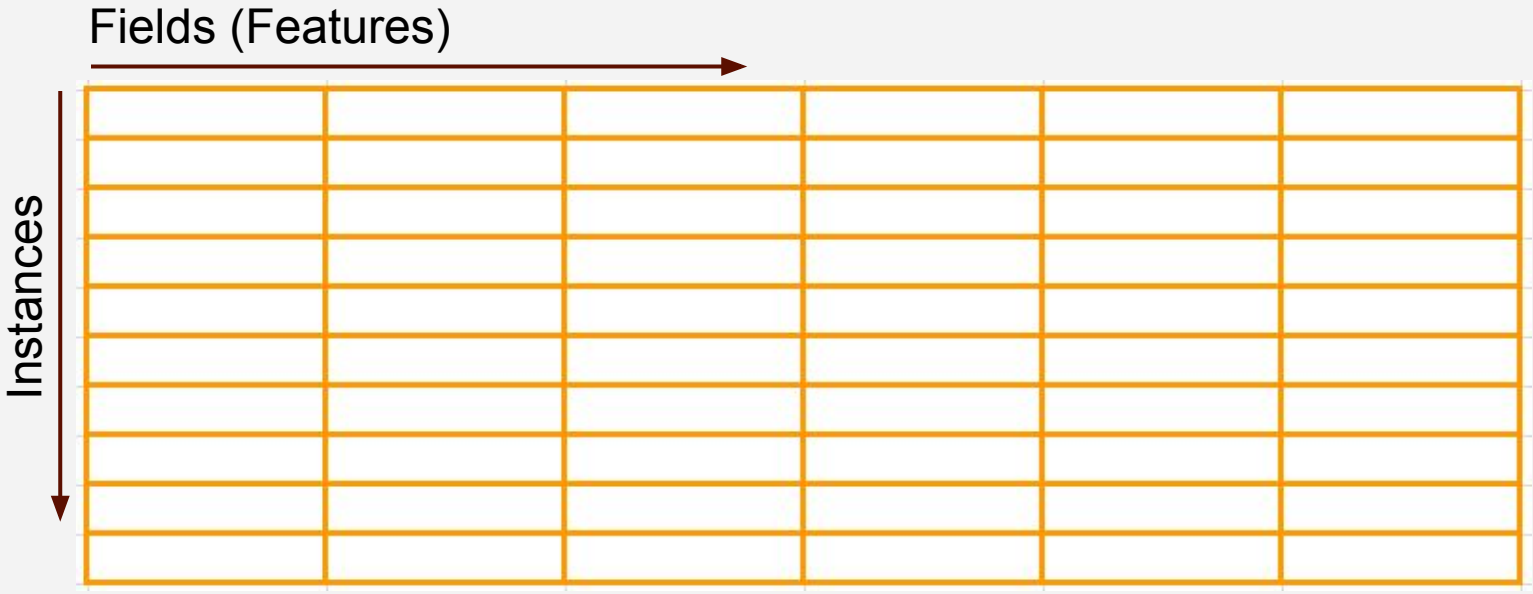
Temporal

Spatial

Categorical

Target

ML-Ready Dataset



Tabular data (rows and columns)

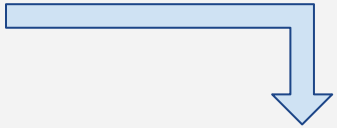
- Usually denormalized in a single file/dataset
- Each row contains information about one instance
- Each column is a feature that describes a property of the instance

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

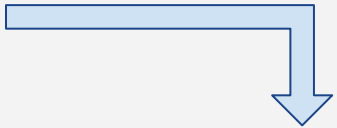
Name	Date	Duration (s)	Genre	Plays
Highway star	1984-05-24		Rock	139
Blues alive	1990-03-01	281	Blues	239
Lonely planet	2002-11-19	332	Techno	42
Dance, dance	1983-02-23	312	Disco	
The wall	1943-01-20	218	Reagge	83
Offside down	1965-02-19	240	Techno	895
The alchemist	2001-11-21	418	Blues	178
Bring me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

Aggregating

Necessary when the entity to model is an aggregation from the provided data.

Original data (list of playbacks)

Content	Genre	Duration	Play Time	User	Device
Highway star	Rock	190	2015-05-12 16:29:33	User001	TV
Blues alive	Blues	281	2015-05-13 12:31:21	User005	Tablet
Lonely planet	Techno	332	2015-05-13 14:26:04	User003	TV
Dance, dance	Disco	312	2015-05-13 18:12:45	User001	Tablet
The wall	Reagge	218	2015-05-14 09:02:55	User002	Smartphone
Offside down	Techno	240	2015-05-14 11:26:32	User005	Tablet
The alchemist	Blues	418	2015-05-14 21:44:15	User003	TV
Bring me down	Classic	328	2015-05-15 06:59:56	User001	Tablet
The scarecrow	Rock	269	2015-05-15 12:37:05	User003	Smartphone



Aggregated data (list of users)

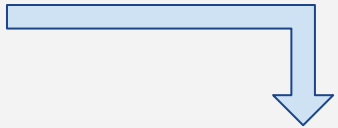
User	Num.Playbacks	Total Time	Pref.Device
User001	3	830	Tablet
User002	1	218	Smartphone
User003	3	1019	TV
User005	2	521	Tablet

Pivoting

Necessary when the entity to model is an aggregation from the provided data.

Original data

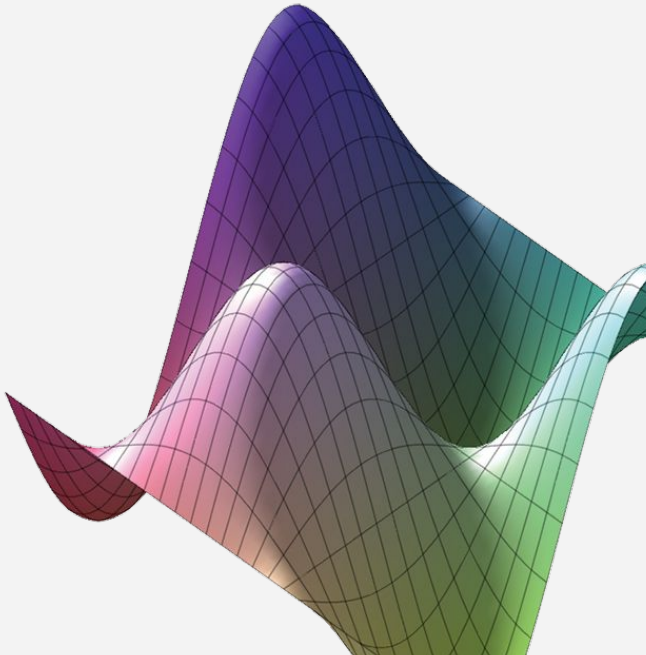
Content	Genre	Duration	Play Time	User	Device
Highway star	Rock	190	2015-05-12 16:29:33	User001	TV
Blues alive	Blues	281	2015-05-13 12:31:21	User005	Tablet
Lonely planet	Techno	332	2015-05-13 14:26:04	User003	TV
Dance, dance	Disco	312	2015-05-13 18:12:45	User001	Tablet
The wall	Reagge	218	2015-05-14 09:02:55	User002	Smartphone
Offside down	Techno	240	2015-05-14 11:26:32	User005	Tablet
The alchemist	Blues	418	2015-05-14 21:44:15	User003	TV
Bring me down	Classic	328	2015-05-15 06:59:56	User001	Tablet
The scarecrow	Rock	269	2015-05-15 12:37:05	User003	Smartphone



Aggregated data with pivoted columns

User	Num.Playback	Total Time	Pref.Device	# playbacks by device			Play duration by device		
				NP_TV	NP_Tablet	NP_Smartphone	TT_TV	TT_Tablet	TT_Smartphone
User001	3	830	Tablet	1	2	0	190	640	0
User002	1	218	Smartphone	0	0	1	0	0	218
User003	3	1019	TV	2	0	1	750	0	269
User005	2	521	Tablet	0	2	0	0	521	0

Numerical Features



Numerical features

- Usually easy to ingest by mathematical models, but feature engineering is indeed necessary.
- Can be floats, counts, ...
- Easier to impute missing data
- Distribution and scale matters to some 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

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 categorial 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

Extra slides marker

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

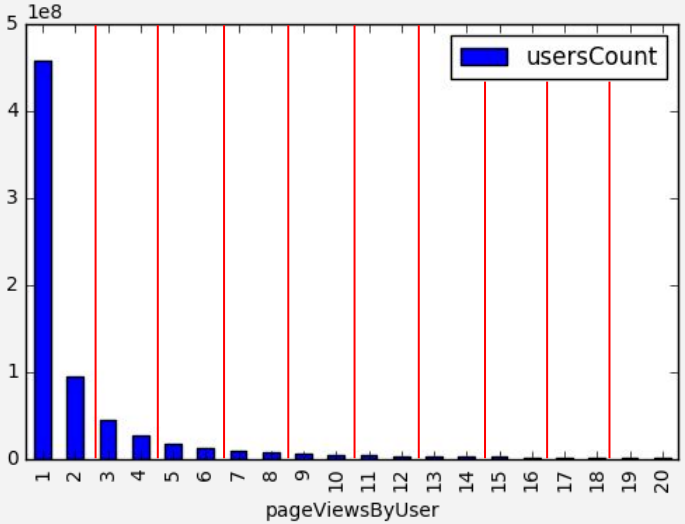
```
>>> from sklearn import preprocessing
>>> X = [[ 1., -1.,  2.],
...      [ 2.,  0.,  0.],
...      [ 0.,  1., -1.]]

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

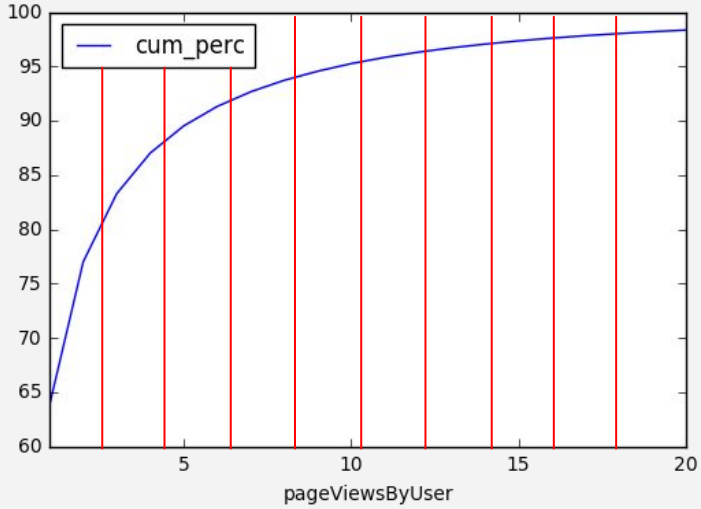
Binarization with scikit-learn

Binning

- Split numerical values into bins and encode with a bin ID
- Can be set arbitrarily or based on distribution
- **Fixed-width binning**

Does fixed-width binning make sense for this long-tailed distribution?



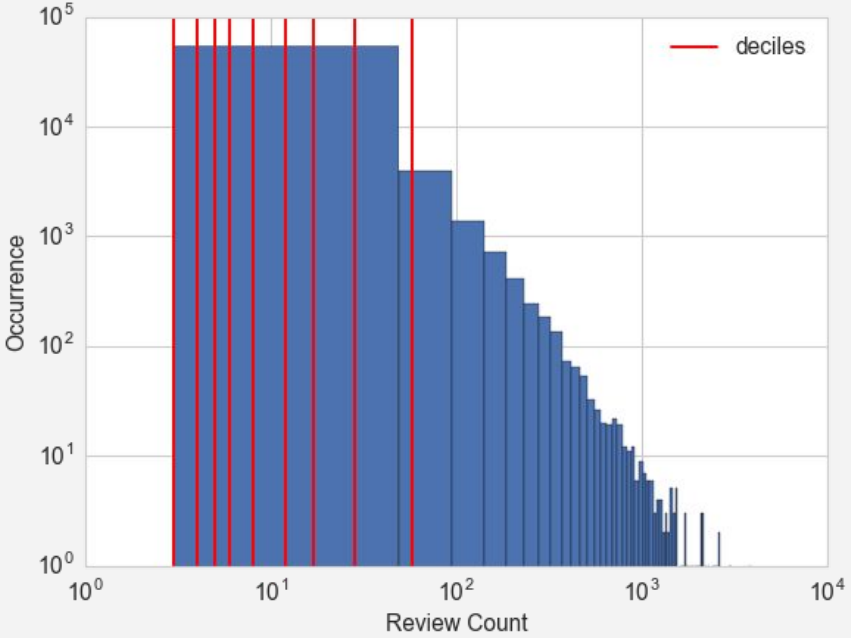


Most users (458,234,809 ~ 5*10^8) had only 1 pageview during the period.

Binning

- **Adaptative or Quantile binning**

  Divides data into equal portions (eg. by median, quartiles, deciles)




```
>>> deciles = dataframe['review_count'].quantile([.1, .2, .3, .4, .5, .6, .7, .8, .9])
>>> deciles
0.1     3.0
0.2     4.0
0.3     5.0
0.4     6.0
0.5     8.0
0.6    12.0
0.7    17.0
0.8    28.0
0.9    58.0
```

Quantile binning with Pandas

Log transformation

Compresses the range of large numbers and expand the range of small numbers.

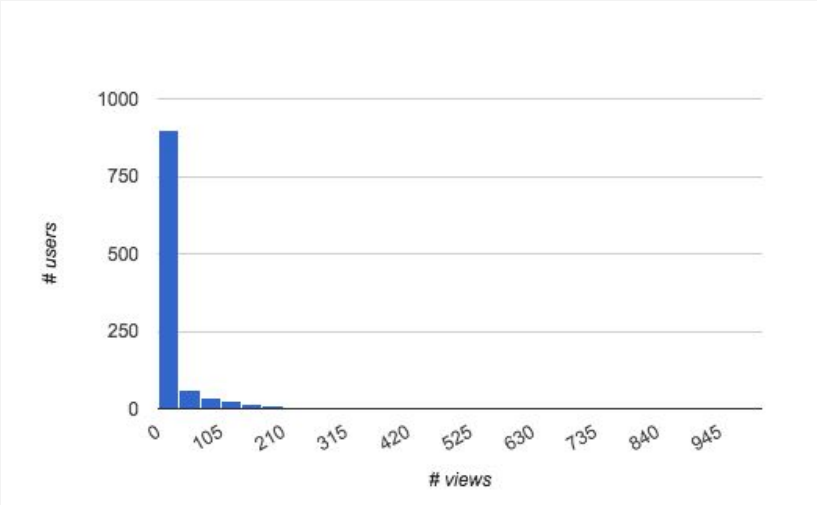
Eg. The larger x is, the slower log(x) increments.

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

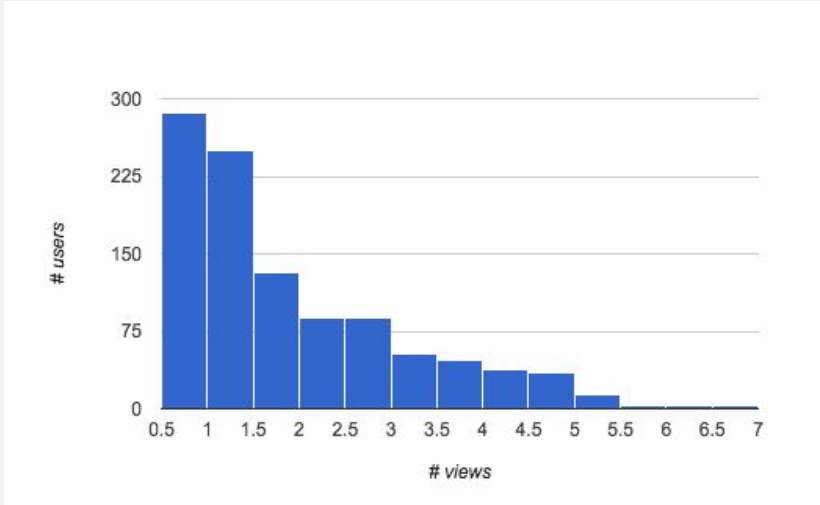
log(1+views_count)
6.91
6.22
5.71
5.30
5.02
4.62
4.26
3.93
3.43
3.04
2.40
1.79
0.69

Log transformation

Smoothing long-tailed data with log



Histogram of # views by user



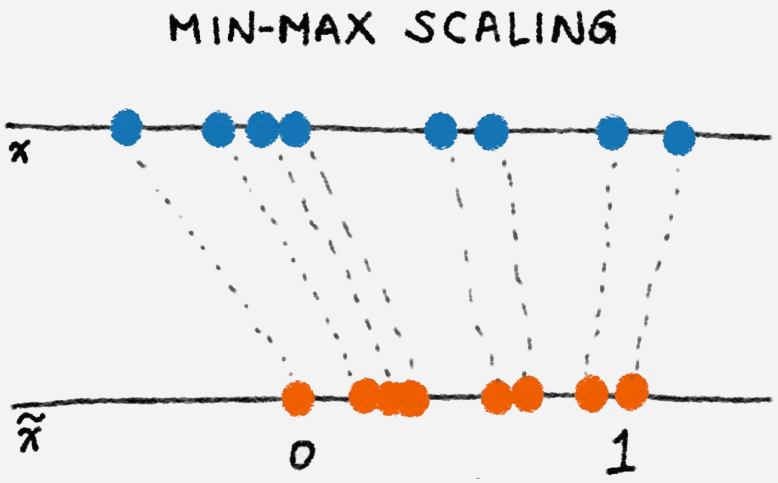
Histogram of # views by user smoothed by log(1+x)

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.



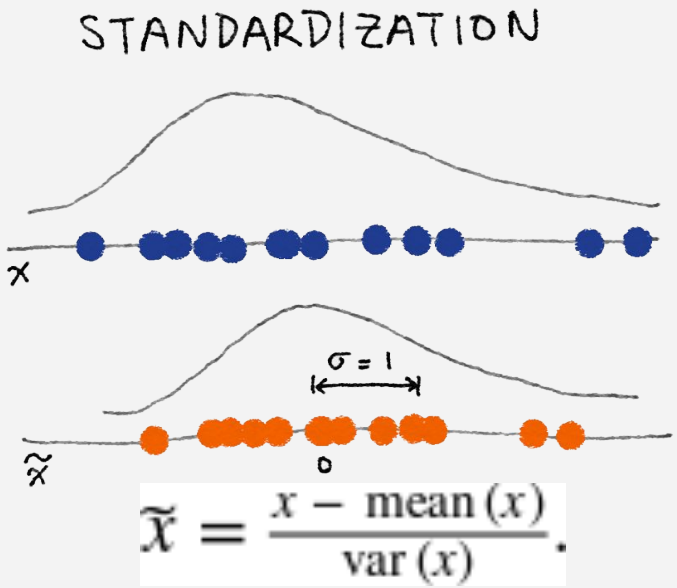
$$\tilde{x} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

```
>>> 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)
>>> X_train_minmax
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)



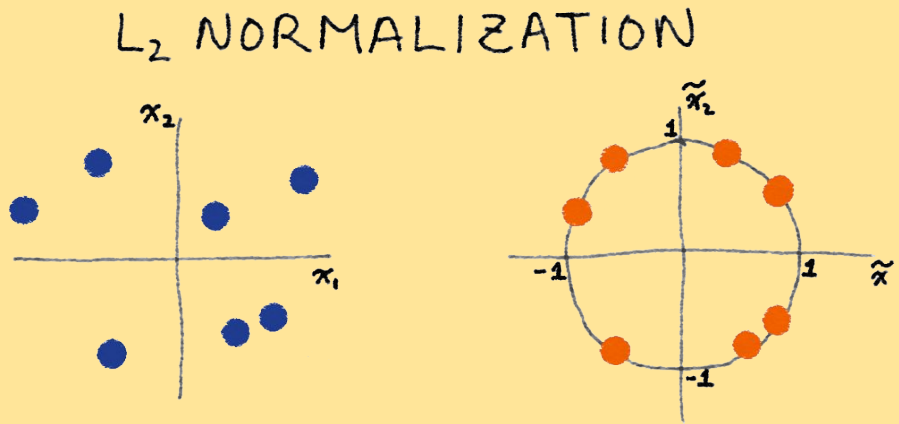
$$\tilde{x} = \frac{x - \text{mean}(x)}{\text{var}(x)}.$$

```
>>> 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

Normalization

- Scales individual samples (rows) to have unit vector, dividing values by vector's L^2 norm, a.k.a. the Euclidean norm
- Useful for quadratic form (like dot-product) or any other kernel to quantify similarity of pairs of samples. This assumption is the base of the **Vector Space Model** often used in **text classification and clustering** contexts



$$\tilde{x} = \frac{x}{\| x \|_2}.$$

Normalized vector

$$\| x \|_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_m^2}$$

Euclidean (L^2) norm

Normalization

```
>>> from sklearn import preprocessing
>>> X = [[ 1., -1.,  2.],
...      [ 2.,  0.,  0.],
...      [ 0.,  1., -1.]]
>>> X_normalized = preprocessing.normalize(X, norm='l2')
>>> X_normalized
array([[ 0.40..., -0.40...,  0.81...],
       [ 1.  ...,  0.  ...,  0.  ...],
       [ 0.  ...,  0.70..., -0.70...]])
```

Normalization with scikit-learn

Interaction Features

- Simple linear models use a linear combination of the individual input features, x_1, x_2, ... x_n to predict the outcome y.

$$y = w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

- An easy way to increase the complexity of the linear model is to create feature combinations (nonlinear features).
- Example:

Degree 2 interaction features for vector $x = (x_1, x_2)$

$$y = w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_4x_2^2$$

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

Interaction Features - Vowpal Wabbit

```
vw_line = '{} |i {} |m {} |z {} |c {}\n'.format(
          label,
          ' '.join(integer_features),
          ' '.join(ctr_features),
          ' '.join(similarity_features),
          ' '.join(categorical_features))
```

Separating features in namespaces in Vowpal Wabbit **(VW)** sparse format

```
1 |i 12:5 18:126 |m 2:0.015 45:0.123 |z 32:0.576 17:0.121 |c 16:1 295:1 3554:1
```

Sample data point (line in VW format file)

```
vw --loss_function logistic --link=logistic --ftrl --ftrl_alpha 0.005 --ftrl_beta 0.1
-q cc -q zc -q zm
-l 0.01 --l1 1.0 --l2 1.0 -b 28 --hash all
--compressed -d data/train_fv.vw -f output.model
```

Interacting (quadratic) features of some namespaces

Feature interactions with VW

Categorical Features

Categorical Features

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

One-Hot Encoding (OHE)

- 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

- Sparse format is memory-friendly
- Example: “platform=tablet” can be sparsely encoded as “2:1”

One-Hot Encoding (OHE)

```
from pyspark.ml.feature import OneHotEncoder, StringIndexer

df = spark.createDataFrame([
    (0, "a"),
    (1, "b"),
    (2, "c"),
    (3, "a"),
    (4, "a"),
    (5, "c")
], ["id", "category"])

stringIndexer = StringIndexer(inputCol="category", outputCol="categoryIndex")
model = stringIndexer.fit(df)
indexed = model.transform(df)

encoder = OneHotEncoder(inputCol="categoryIndex", outputCol="categoryVec")
encoded = encoder.transform(indexed)
encoded.show()
```

One-hot encoding with Spark ML

Large Categorical Variables

- Common in applications like targeted advertising and fraud detection
- Example:

categorical_feature	unique_values
landing_page_document_id	636482
ad_id	418295
ad_document_id	143856
content_entities	52439
advertiser	2052
publisher	830
country_state	1892

Some large categorical features from Outbrain Click Prediction competition

Feature hashing

- Hashes categorical values into vectors with fixed-length.
- Lower sparsity and higher compression compared to OHE
- Deals with new and rare categorical values (eg: new user-agents)
- May introduce collisions

100 hashed columns

country
brazil
chile
venezuela
colombia
... 222 countries

country_hashed_1	country_hashed_2	country_hashed_3	country_hashed_4	...
1	0	0	0	...
0	0	0	1	...
0	0	1	0	...
0	0	1	0	...
...	...	...	...	...

Feature hashing

```
import hashlib
def hashstr(s, nr_bins):
    return int(hashlib.md5(s.encode('utf8')).hexdigest(), 16)%(nr_bins-1)+1

CATEGORICAL_VALUE='ad_id=354424'  <-- Original category
MAX_BINS=100000
>>> hashstr(CATEGORICAL_VALUE, MAX_BINS)
49389  <-- Hashed category
```

Feature hashing with pure Python

```
import tensorflow as tf
ad_id_hashed = tf.contrib.layers.sparse_column_with_hash_bucket('ad_id',
                    hash_bucket_size=250000, dtype=tf.int64, combiner="sum")
```

Feature hashing with TensorFlow

Feature hashing

```
vw --loss_function logistic --link=logistic --ftrl --ftrl_alpha 0.005 --ftrl_beta 0.1
-q cc -q zc -q zm -l 0.01 --l1 1.0 --l2 1.0
-b 18 --hash all
--compressed -d data/train_fv.vw -f output.model
```

← Hashes values to a feature space of 2^{18} positions (columns)

Feature hashing with Vowpal Wabbit

Bin-counting

- Instead of using the actual categorical value, use a global statistic of this category on historical data.
- Useful for both linear and non-linear algorithms
- May give collisions (same encoding for different categories)
- Be careful about leakage
- Strategies
  - Count
  - Average CTR

Bin-counting

ad_id
423654
123646
68655
54345

ad_views	ad_clicks	ad_CTR
18339	1355	0.074
335	12	0.036
1244	132	0.106
35387	1244	0.035

or or

Counts **Click-Through Rate**

P(click | ad) = ad_clicks / ad_views

LabelCount encoding

- Rank categorical variables by count in train set
- Useful for both linear and non-linear algorithms (eg: decision trees)
- Not sensitive to outliers
- Won’t give same encoding to different variables

ad_id	clicks	ad_rank
54345	35387	1
423654	18339	2
98799	12352	3
68655	9430	4
123646	8232	5

Category Embedding

- Use a Neural Network to create dense embeddings from categorical variables.
- Map categorical variables in a function approximation problem into Euclidean spaces
- Faster model training.
- Less memory overhead.
- Can give better accuracy than 1-hot encoded.

Category Embedding

```
import tensorflow as tf

def get_embedding_size(unique_val_count):
  return int(math.floor(6 * unique_val_count**0.25))

ad_id_hashed_feature =
tf.contrib.layers.sparse_column_with_hash_bucket('ad_id',
                                hash_bucket_size=250000, dtype=tf.int64, combiner="sum")
embedding_size = get_embedding_size( ad_id_hashed_feature.length)
ad_embedding_feature = tf.contrib.layers.embedding_column(
                    ad_id_hashed_feature, dimension=embedding_size, combiner="sum")
```

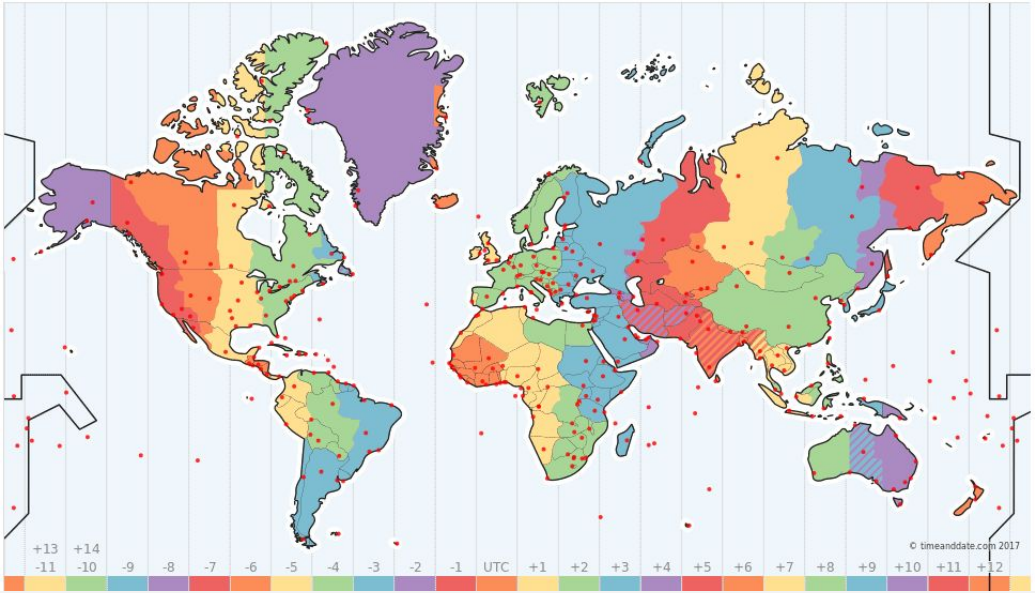
Category Embedding using TensorFlow

Temporal Features

Time Zone conversion

Factors to consider:

- Multiple time zones in some countries
- Daylight Saving Time (DST)
  - Start and end DST dates



	country_name	utc_time_offset	dst_time_offset
0	Afghanistan	+04:30	-
1	Aaland Islands	+02:00	+03:00
2	Albania	+01:00	+02:00
3	Algeria	+01:00	-
4	Samoa (American)	−11:00	-
5	Andorra	+01:00	+02:00
6	Angola	+01:00	-
7	Anguilla (UK)	−04:00	-
8	Antigua & Barbuda	−04:00	-
9	Argentina	−03:00	-

Time binning

- Apply binning on time data to make it categorial and more general.
- Binning a time in hours or periods of day, like below.

Hour range	Bin ID	Bin Description
[5, 8)	1	Early Morning
[8, 11)	2	Morning
[11, 14)	3	Midday
[14, 19)	4	Afternoon
[19, 22)	5	Evening
[22-24) and (00-05]	6	Night

- Extraction: weekday/weekend, weeks, months, quarters, years...

Trendlines

- Instead of encoding: total spend, encode things like: Spend in last week, spend in last month, spend in last year.
- Gives a trend to the algorithm: two customers with equal spend, can have wildly different behavior — one customer may be starting to spend more, while the other is starting to decline spending.

Closeness to major events

- Hardcode categorical features from dates
- Example: Factors that might have major influence on spending behavior
- Proximity to major events (holidays, major sports events)
    - Eg. date_X_days_before_holidays
- Proximity to wages payment date (monthly seasonality)
    - Eg. first_saturday_of_the_month

Time differences

- Differences between dates might be relevant
- Examples:
  - **user_interaction_date - published_doc_date**
    To model how recent was the ad when the user viewed it.
    Hypothesis: user interests on a topic may decay over time
  - **last_user_interaction_date - user_interaction_date**
    To model how old was a given user interaction compared to his last interaction

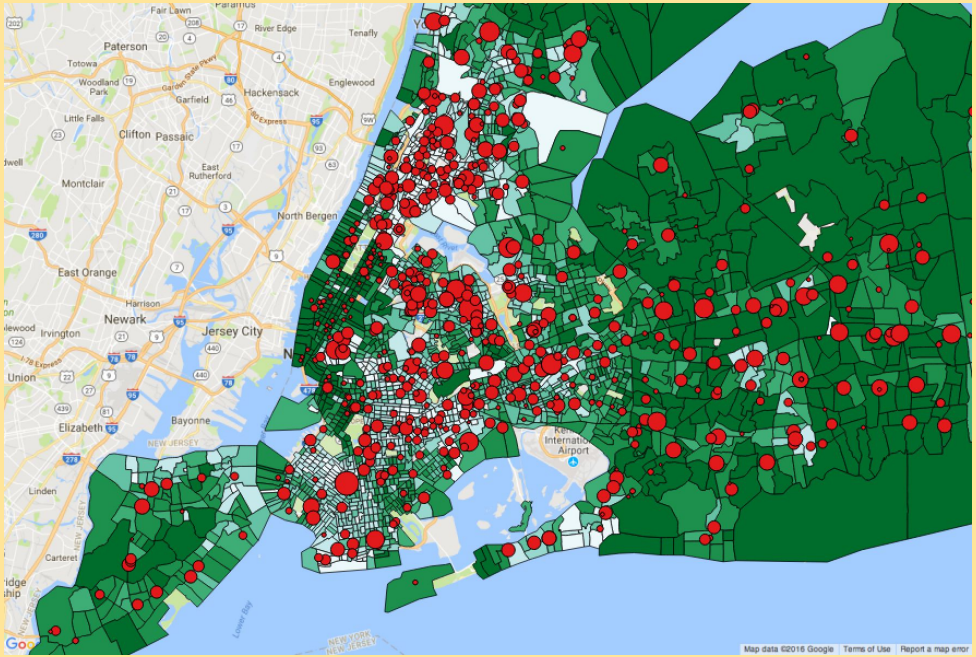
Spatial Features

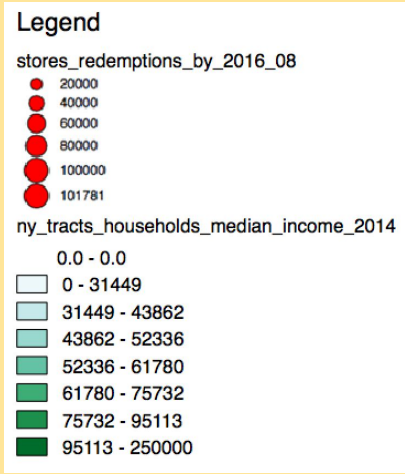
Spatial Variables

- Spatial variables encode a location in space, like:
  - GPS-coordinates (lat. / long.) - sometimes require projection to a different coordinate system
  - Street Addresses - require geocoding
  - ZipCodes, Cities, States, Countries - usually enriched with the centroid coordinate of the polygon (from external GIS data)
- Derived features
  - Distance between a user location and searched hotels (Expedia competition)
  - Impossible travel speed (fraud detection)

Spatial Enrichment

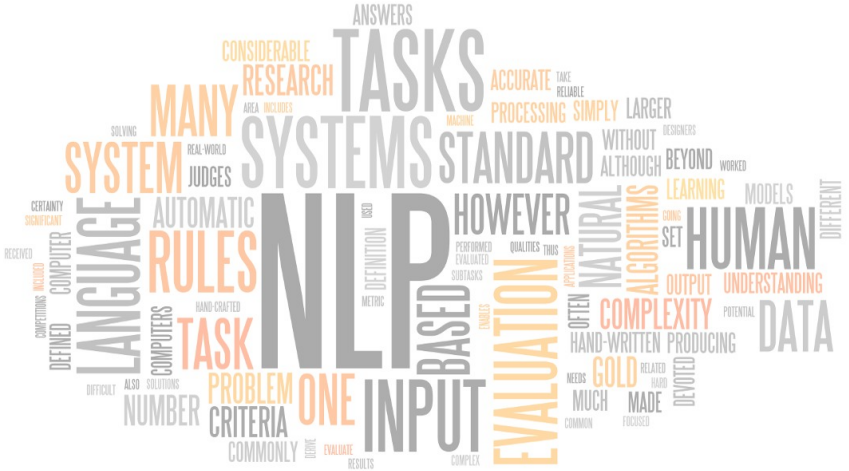
Usually useful to enrich with external geographic data (eg. Census demographics)





Beverage Containers Redemption Fraud Detection: Usage of # containers redeemed (red circles) by store and Census households median income by Census Tracts

Textual data



Natural Language Processing

Cleaning
- Lowercasing
- Convert accented characters
- Removing non-alphanumeric
- Repairing

Tokenizing
- Encode punctuation marks
- Tokenize
- N-Grams
- Skip-grams
- Char-grams
- Affixes

Removing
- Stopwords
- Rare words
- Common words

Roots
- Spelling correction
- Chop
- Stem
- Lemmatize

Enrich
- Entity Insertion / Extraction
- Parse Trees
- Reading Level

Text vectorization

Represent each document as a feature vector in the vector space, where each position represents a word (token) and the contained value is its relevance in the document.

- ***BoW*** (*Bag of words*)
- ***TF-IDF*** (*Term Frequency - Inverse Document Frequency*)
- **Embeddings** (eg. Word2Vec, Glove)
- **Topic models** (e.g LDA)

	linux	modern	the	system	steering	petrol
D1	3	4	3	0	2	0
D2	4	3	4	1	0	1
D3	1	0	4	1	0	1
D4	0	1	3	3	3	4

Document Term Matrix - Bag of Words

Text vectorization

TF-IDF - More "relevant" terms in a document are frequent terms in the document and rare in other documents

f_{ij} = frequency of term (feature) i in doc (item) j

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

n_i = number of docs that mention term i

N = total number of docs

$$IDF_i = \log \frac{N}{n_i}$$

TF-IDF score: $w_{ij} = TF_{ij} \times IDF_i$

Doc profile = set of words with highest **TF-IDF** scores, together with their scores

Text vectorization - TF-IDF

tokens

documents	face	person	guide	lock	cat	dog	sleep	micro	pool	gym	...
	0	1	2	3	4	5	6	7	8	9	
D1		0.05					0.25				
D2	0.02			0.32					0.45		
...											

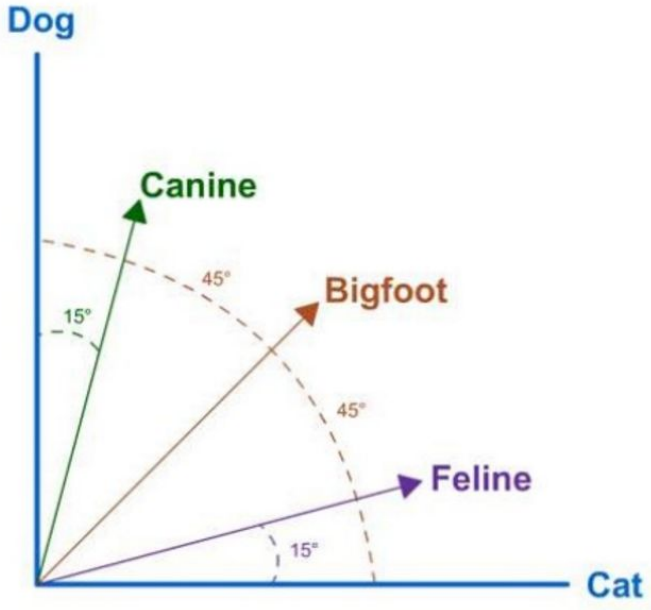
TF-IDF sparse matrix example

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(max_df=0.5, max_features=1000,
                             min_df=2, stop_words='english')
tfidf_corpus = vectorizer.fit_transform(text_corpus)
```

TF-IDF with scikit-learn

Cosine Similarity

Similarity metric between two vectors is cosine among the angle between them



$$\vec{a} \cdot \vec{b} = \|\vec{a}\| \|\vec{b}\| \cos\theta$$

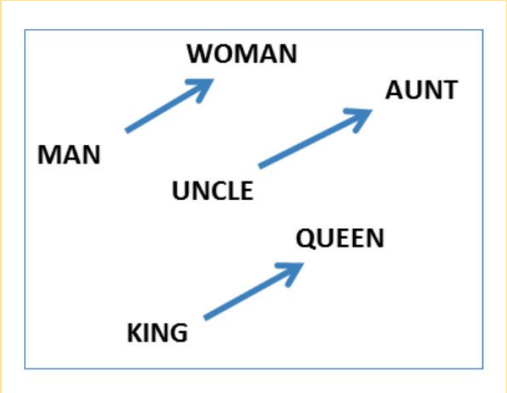
$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$$

```
from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity(tfidf_matrix[0:1], tfidf_matrix)
```

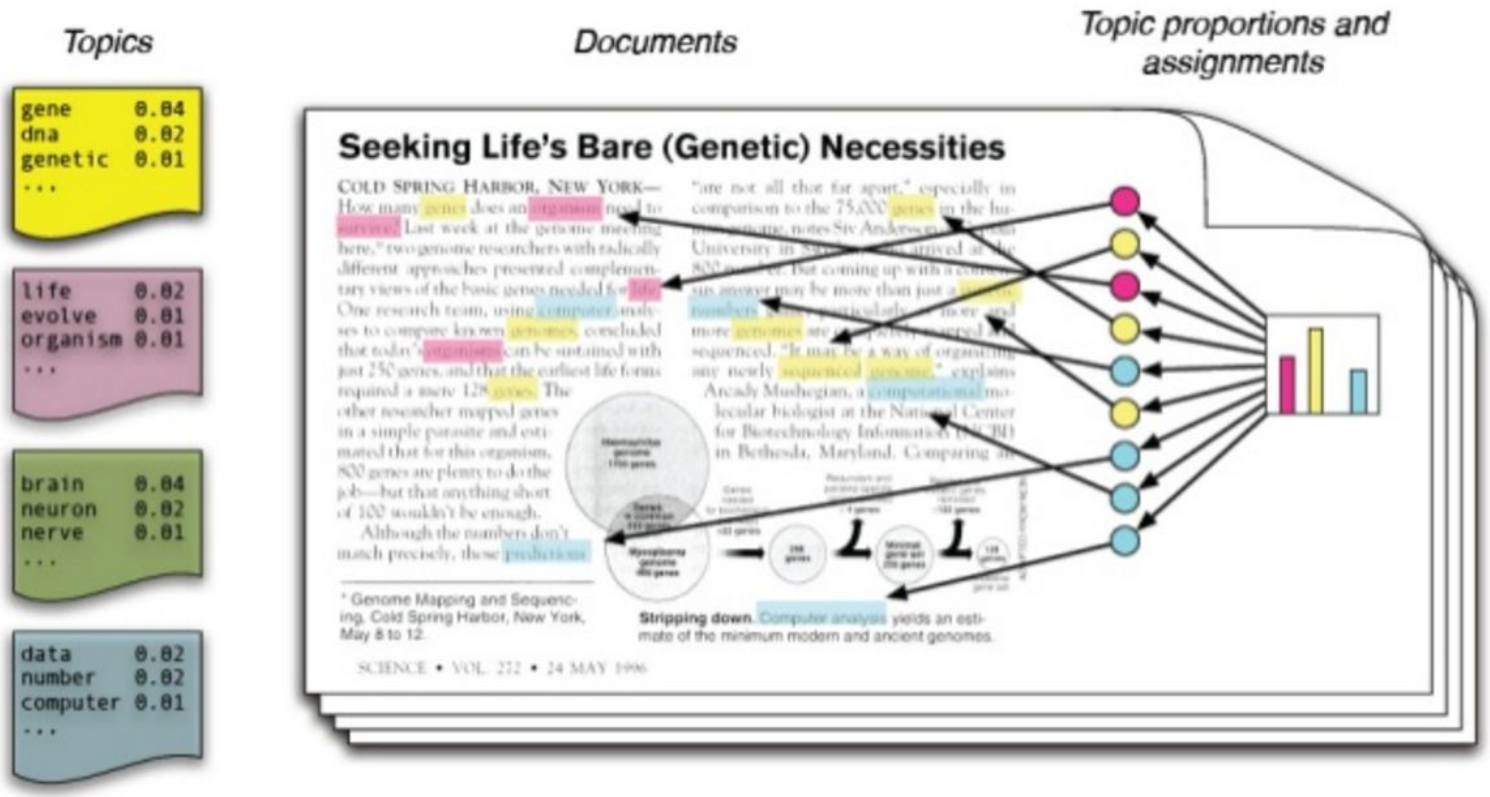
Cosine Similarity with scikit-learn

Textual Similarities

- **Token similarity**: Count number of tokens that appear in two texts.
- **Levenshtein/Hamming/Jaccard Distance**: Check similarity between two strings, by looking at number of operations needed to transform one in the other.
- **Word2Vec / Glove**: Check cosine similarity between two word embedding vectors

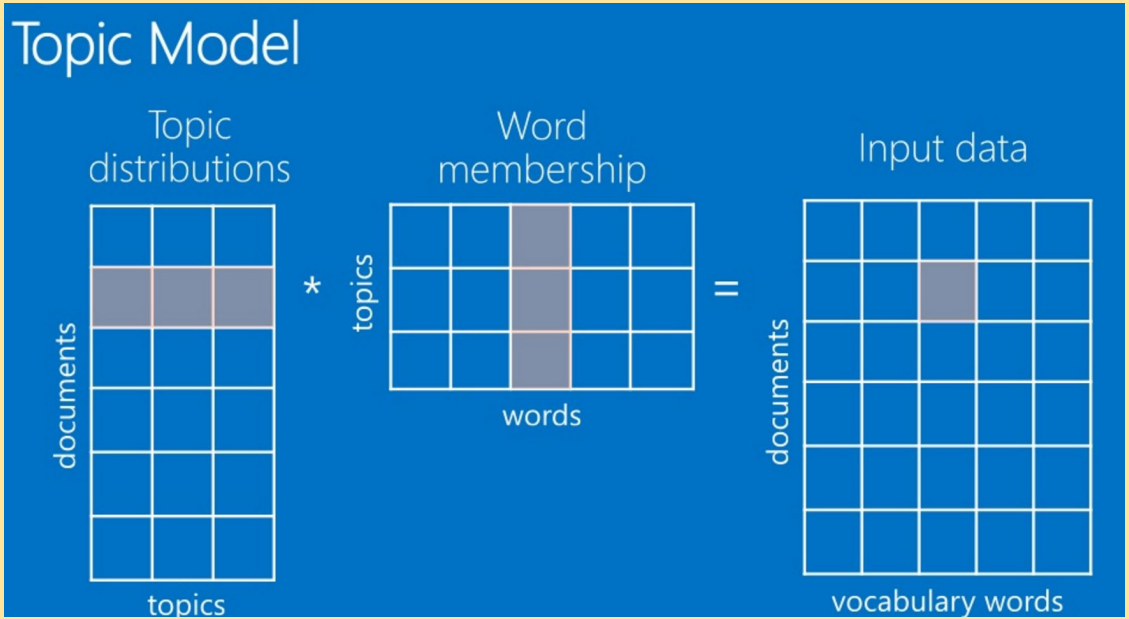


Topic Modeling



Topic Modeling

- ***Latent Dirichlet Allocation (LDA)*** -> Probabilistic
- ***Latent Semantic Indexing / Analysis (LSI / LSA)*** -> Matrix Factorization
- ***Non-Negative Matrix Factorization (NMF)*** -> Matrix Factorization



Feature Selection

Feature Selection

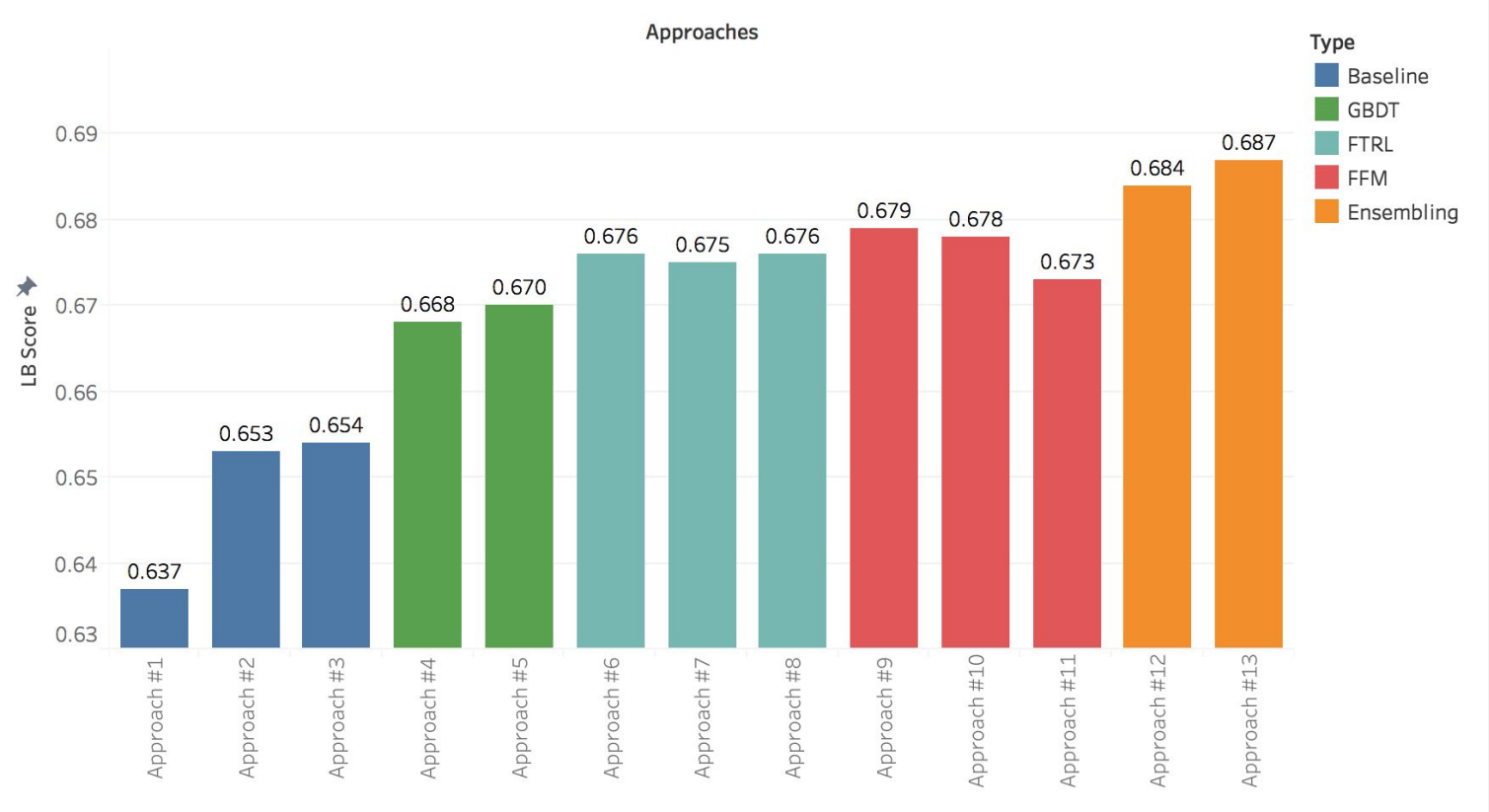
Reduces model complexity and training time

- **Filtering -** Eg. Correlation our Mutual Information between each feature and the response variable
- **Wrapper methods -** Expensive, trying to optimize the best subset of features (eg. Stepwise Regression)
- **Embedded methods -** Feature selection as part of model training process (eg. Feature Importances of Decision Trees or Trees Ensembles)

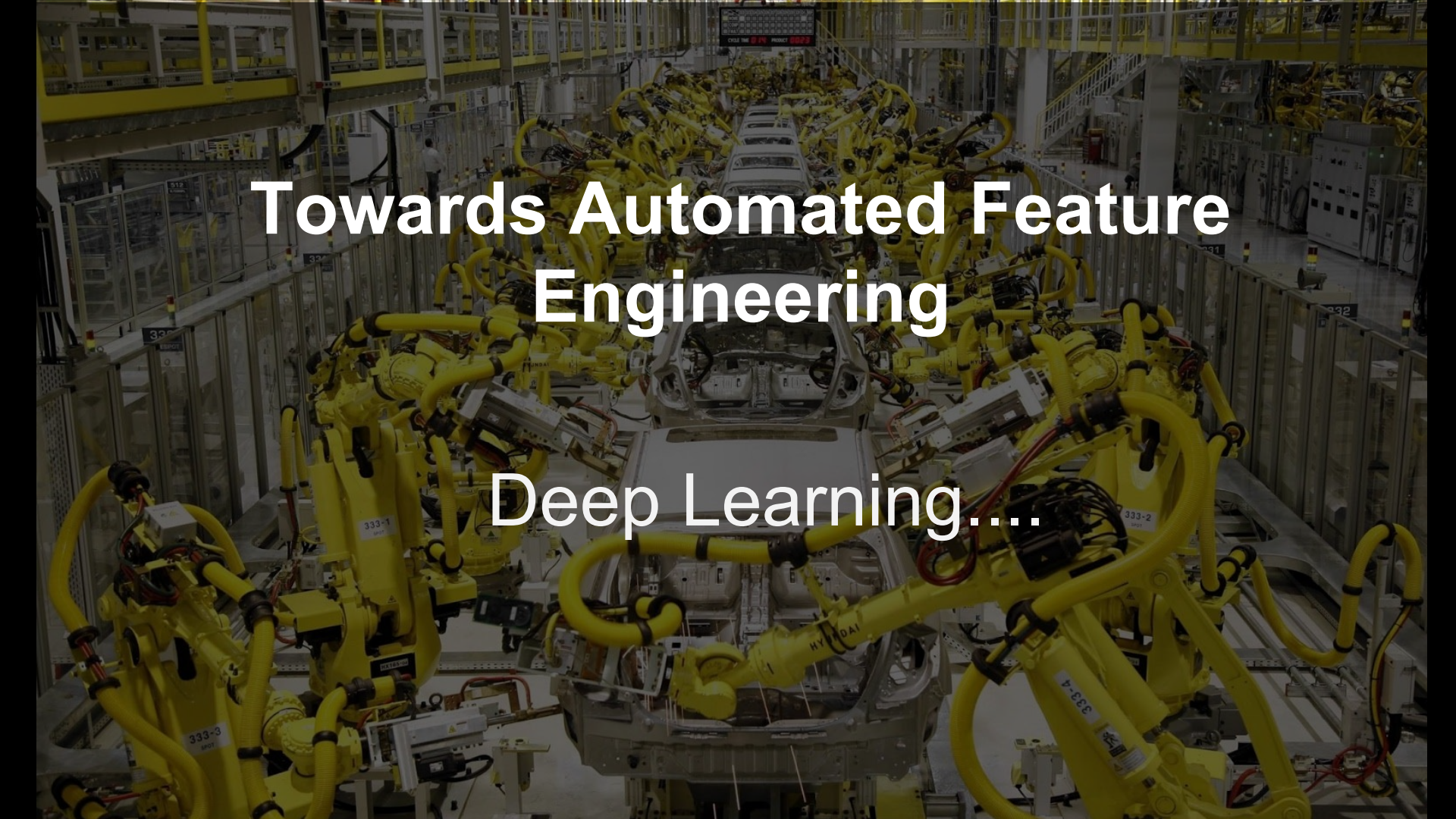
“More data beats clever algorithms, but **better data** beats more data.”

– Peter Norvig

Diverse set of Features and Models leads to different results!



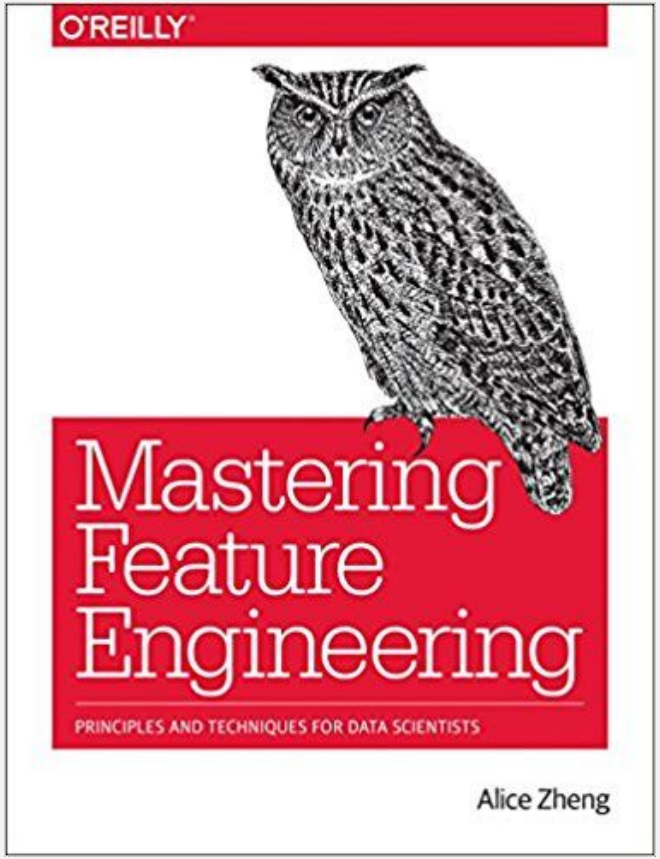
Outbrain Click Prediction - Leaderboard score of my approaches


Towards Automated Feature Engineering
Deep Learning....

“...some machine learning projects succeed and some fail. Where is the difference? Easily the most important factor is the features used.”

– Pedro Domingos

References



Scikit-learn - Preprocessing data

Spark ML - Feature extraction

Discover Feature Engineering...

ciandt.com

Data Scientists wanted!
bit.ly/ds4cit

Slides: bit.ly/feature_eng

Blog: medium.com/unstructured

Gabriel Moreira
Lead Data Scientist