Introduction to Machine Learning with Scikit-Learn

District Data Labs

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

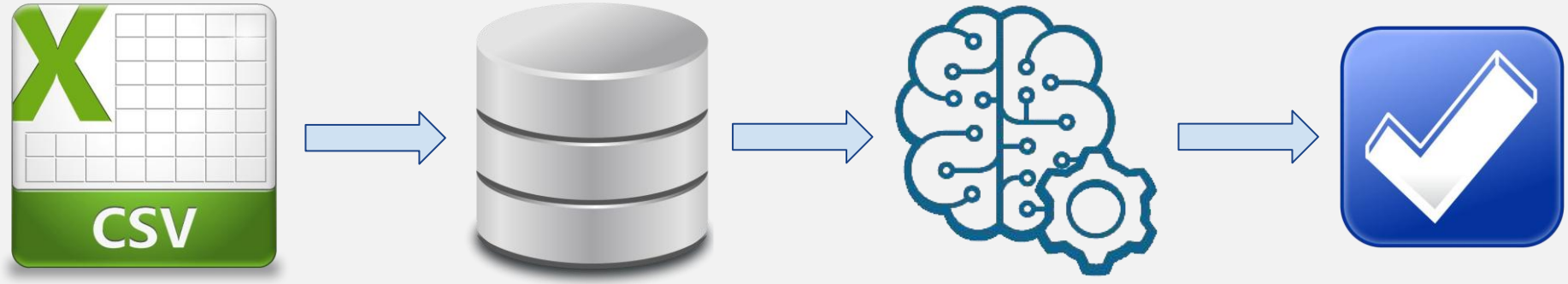
"Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data."

- Jason Brownlee

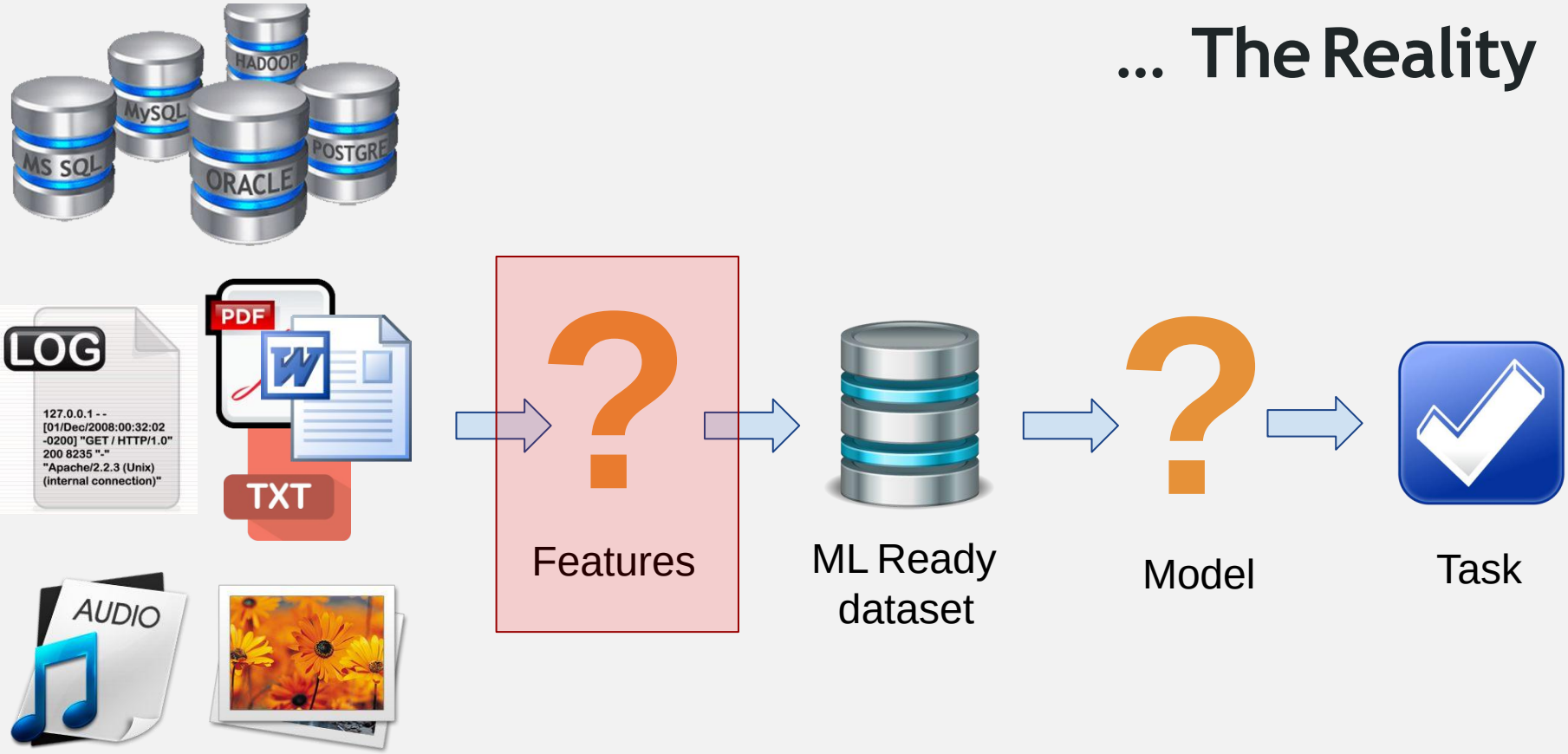
“Coming up with features is difficult,
time-consuming,
requires expert knowledge.
'Applied machine learning' is basically
feature engineering.”

– Andrew Ng

The Dream…



… The Reality



Raw data

… The Reality

Raw data

Deep Learning

But what architecture?

Task

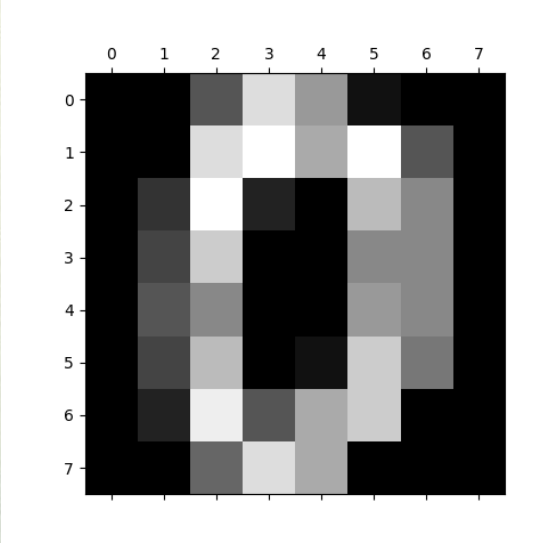
More on scikit-learn

- Data loading and pre-processing
- Feature engineering
- Hyperparameter tuning and Model evaluation
- Pipelines

Load dataset

- sklearn.datasets
  - Some „toy examples“ ready to use
    - datasets.load_iris()
    - datasets.load_wine()
  - Generate artificial data

```
>>> from sklearn.datasets import load_digits
>>> digits = load_digits()
>>> digits.data.shape
(1797, 64)
>>> digits.target_names
[0 1 2 3 4 5 6 7 8 9]
>>> import matplotlib.pyplot as plt
>>> plt.gray()
>>> plt.matshow(digits.images[0])
>>> plt.show()
```



Data pre-processing and transformation

- Inputing for NaNs
- Normalization / standardization
- Transform nominals
- Transform numeric
- Data Aggregation
- Feature extraction / combination

sklearn.preprocessing
sklearn.feature_extraction
sklearn.feature_selection

Exploratory Data Analysis (EDA)

- Get the knowledge about 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?

Exploratory Data Analysis (EDA)

- Get the knowledge about your data

  - get a basic description of the data
  - visualize it
  - identify patterns in it
  - identify challenges of using the data
  - form hypothesis about the data
  - …

Exploratory Data Analysis (EDA)

Basic knowledge

- Describe data (quantiles, med, mean, max, min…)
- head(), tail()
- Pd.Query(), sorting,

Challenges

- NaNs, outliers, numeric -> nominal and vice versa

Patterns

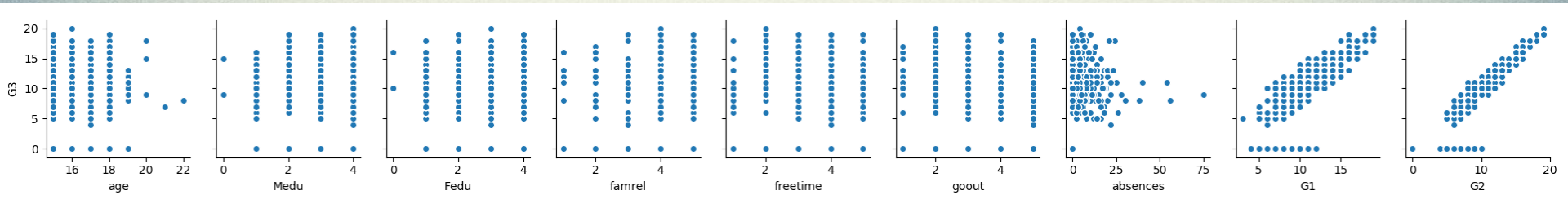
- correlation, covariation, linear dependence, dimensionality reduction…

Visualization

- boxpolots, histograms, cummulative distribution, scatter plots, corr matrices, …

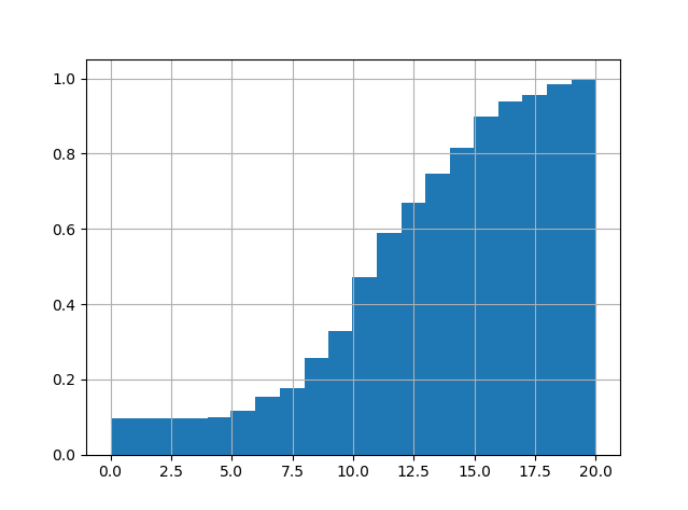
Exploratory Data Analysis (EDA) - pairplot

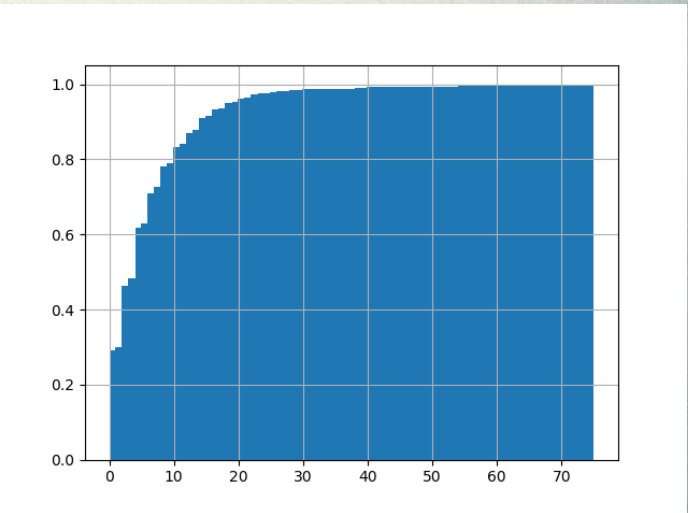
```
import pandas as pd
import seaborn as sns #statistical data visualization, based on matplotlib
import matplotlib.pyplot as plt

df = pd.read_csv("C:/Users/peska/Documents/uceni/2017_2018/NPRG065/cviceni/student-mat.csv", delimiter=";")

sns.pairplot(data=df,
                x_vars= ["age","Medu","Fedu","famrel","freetime","goout","absences","G1","G2"],
                y_vars= ['G3'])
plt.show()
```



Exploratory Data Analysis (EDA) - CDF

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("C:/Users/peska/Documents/uceni/2017_2018/NPRG065/cviceni/student-mat.csv", delimiter=";")


df.G3.hist(cumulative=True, normed=1, bins=len(df.G3))
#df.absences.hist(cumulative=True, normed=1, bins=len(df.absences))
plt.show()
```





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

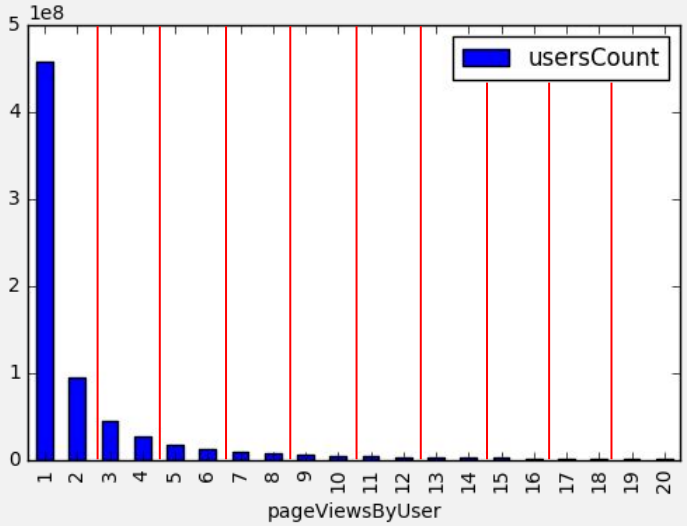
```
>>> from sklearn import preprocessing
>>> X = [[ 1., -1.,  2.],
...      [ 2.,  0.,  0.],
...      [ 0.,  1., -1.]]

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

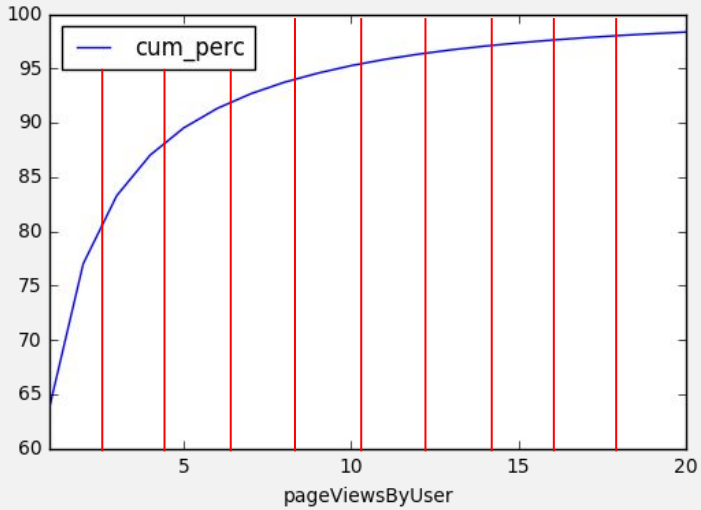
Binarization with scikit-learn

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?





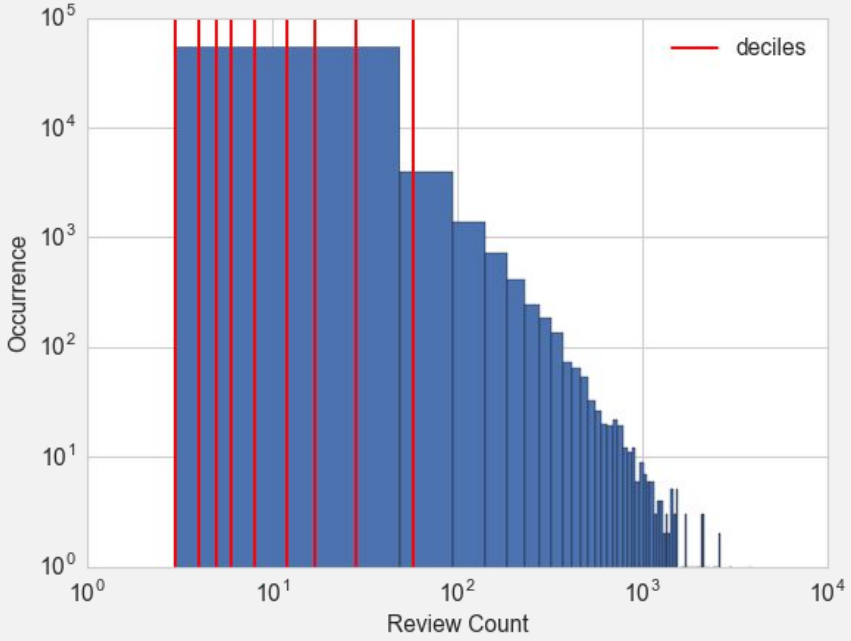
Most users (458,234,809 ~ 5*10^8) had only 1 pageview during the period.

numpy.digitize

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
```

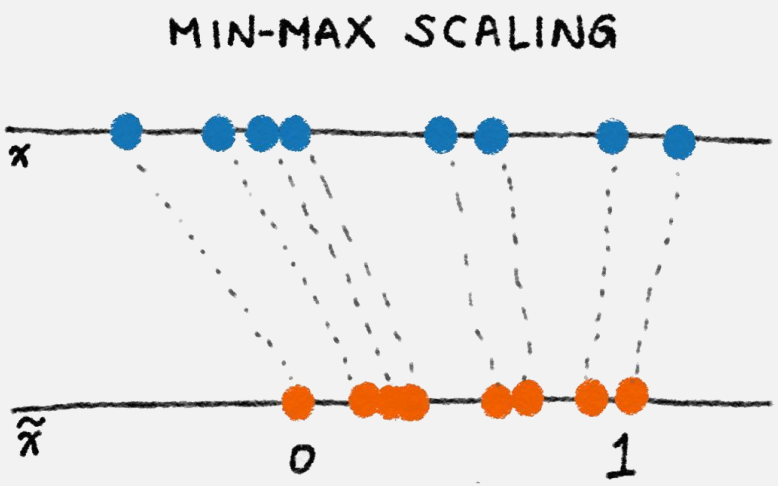
Quantile binning with Pandas

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.



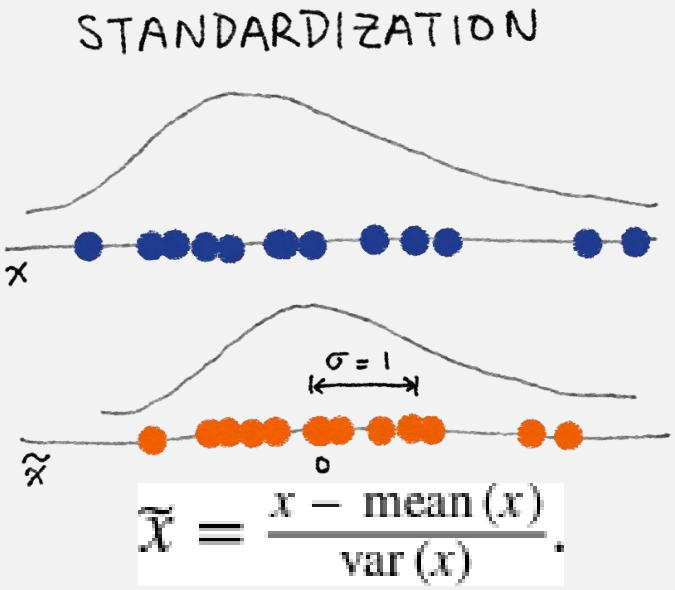
$$\tilde{x} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

```
>>> 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)



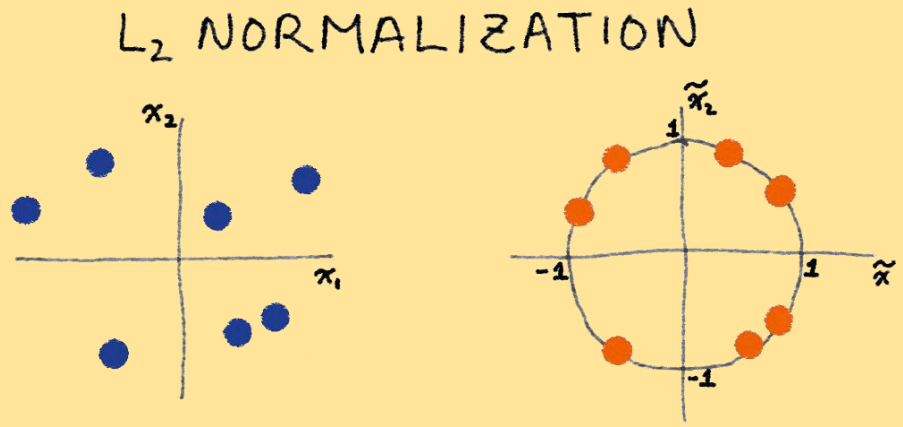
$$\tilde{x} = \frac{x - \text{mean}(x)}{\text{var}(x)}.$$

```
>>> 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...]])
```

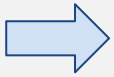
Normalization with scikit-learn

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.

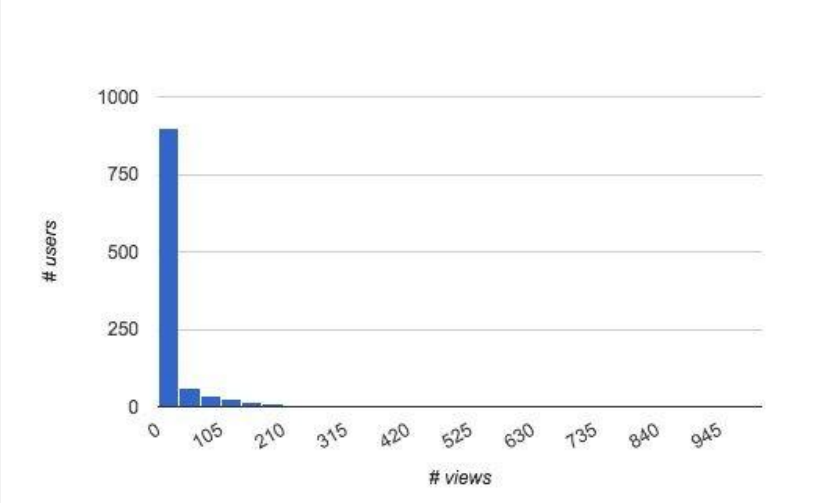
user_id	views_count
a	1000
b	500
c	300
d	200
e	150
f	100
g	70
h	50
i	30
j	20
k	10
l	5
m	1



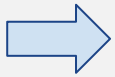
log(1+views_count)
6.91
6.22
5.71
5.30
5.02
4.62
4.26
3.93
3.43
3.04
2.40
1.79
0.69

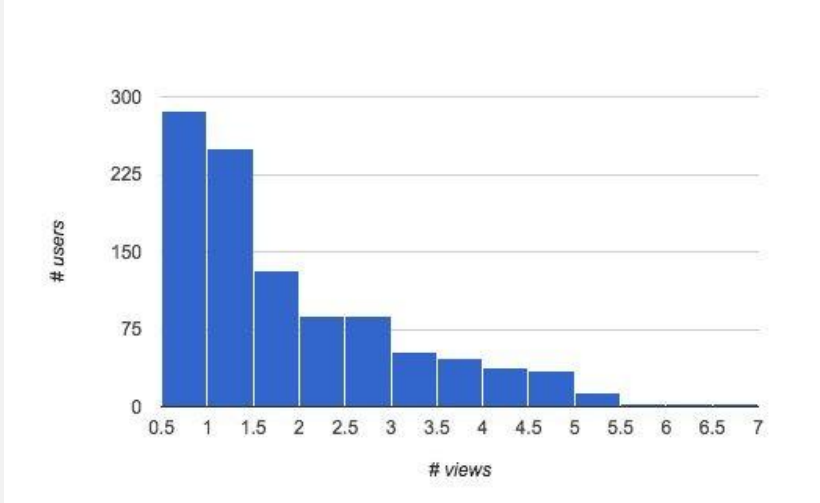
Log transformation

Smoothing long-tailed data with log



Histogram of # views by user





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

Feature extraction

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

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
- sklearn.feature_extraction.DictVectorizer

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	...
...	...	...	...	...

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
18339
335
1244
35387

or

ad_clicks
1355
12
132
1244

or

ad_CTR
0.074
0.036
0.106
0.035

Counts

Click-Through Rate

P(click | ad) = ad_clicks / ad_views

Textual features

- sklearn.feature_extraction.text
- Count, Tf-IDF,

Image features

- sklearn.feature_extraction.image
- Patches, img to graph,…

Feature selection

Feature selection

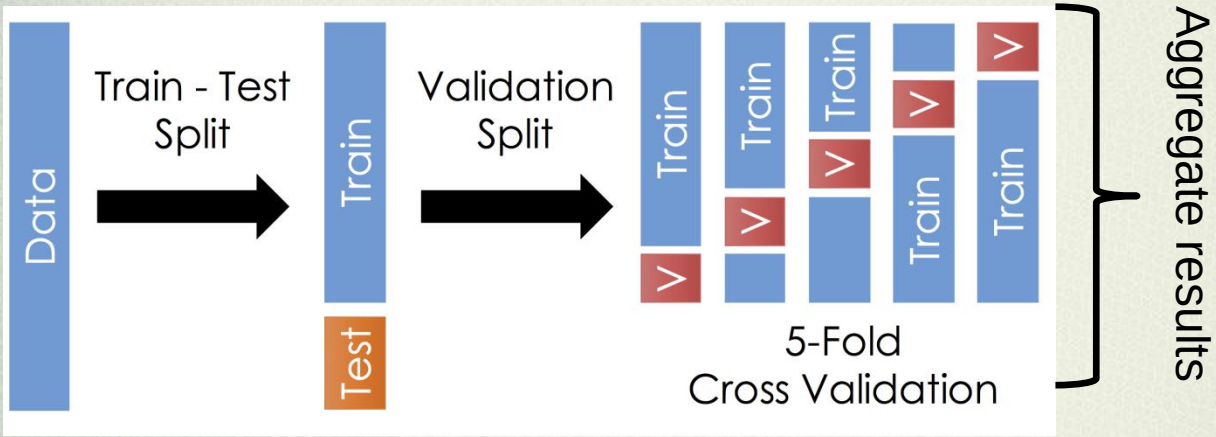
- Decrease complexity of model
- Remove redundant and irrelevant features

- sklearn.feature_selection.SelectKBest
  - Scoring function
  - K features
  - Threshold
  - …

Model Fitting & Evaluation

Model Fitting Protocol

How to learn model's hyperparameters: grid search & cross-validation



Istead of Train – Test split, you may use additional „outer" cross-validation

- Get results from all parts of the dataset

Never use any knowledge of the test set data

- E.g. For mean ratings, object similarities etc

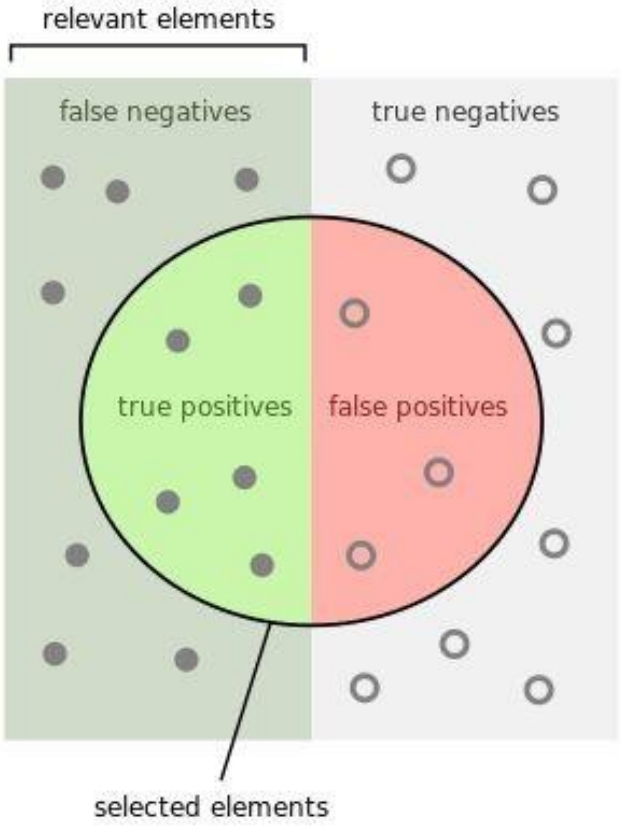
Model Fitting Protocol

Further variants:

- Monte-Carlo cross validation:
  - random splits, arbitrary size (cold start problems)

- Bootstrap validation:
  -only one split (if you have abundant data)

-Temporal splits:
  -if domain changes over time



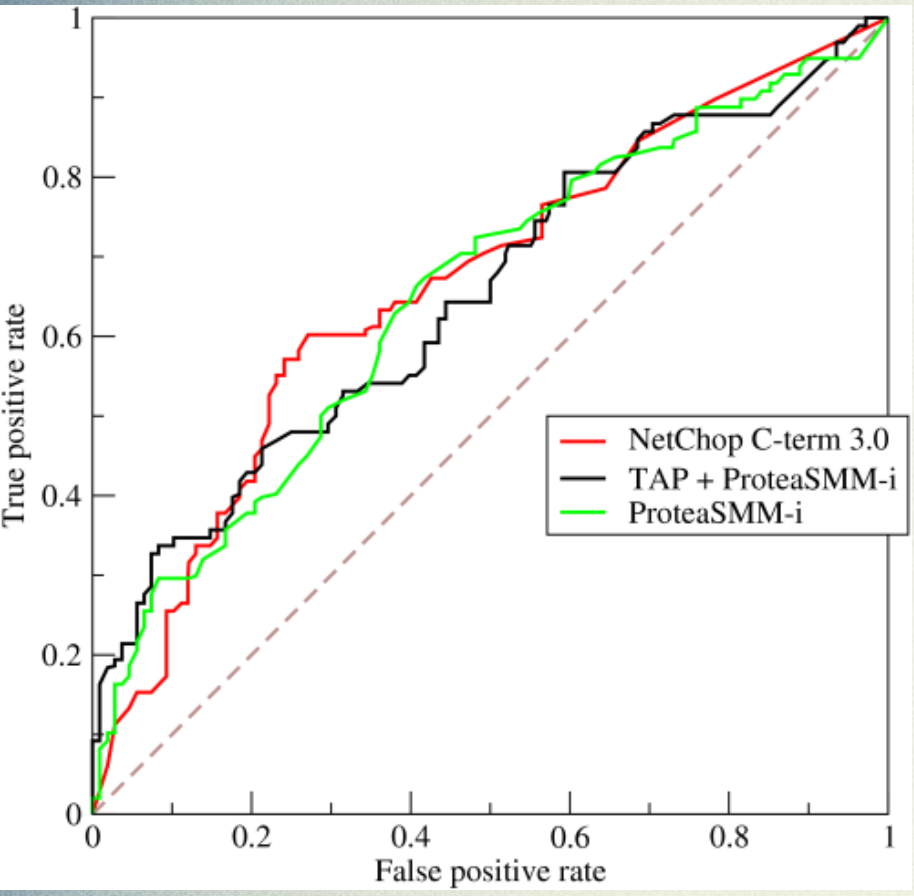
accuracy =
true positives + true negatives / total

precision =
true positives / (true positives + falsepositives)

recall =
true positives / (false negatives + true positives)

F1 score =
2 * ((**precision** * **recall**) / (**precision** + **recall**))

https://en.wikipedia.org/wiki/Precision_and_recall



ROC (reciever-operator curve)

True Positive Rate /
False Positive Rate

AUC (AUROC) = area under ROC

- *probability that random positive example will be ranked higher than random negative*
- calculate empirically from test results

AUPR (area under precision-recall curve)

https://en.wikipedia.org/wiki/Receiver_operating_characteristic
http://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html#sklearn.metrics.auc

MSE & Coefficient of Determination

MAE / MSE / RMSE

MSE = np.mean((predicted-expected)**2)

Coefficient of determination

R^2 is a predictor of "goodness of fit" and is a value $\in$ [0,1] where 1 is perfectfit.

-the proportion of the variance in the dependent variable that is predictable from the independent variable(s)

-independent from the scale of the output feature

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2,$$

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}.$$

Evaluation - sklearn

sklearn.model_selection
-GridSearchCV
-train_test_split()
-LeaveOneOut (predict for each example separately)

- Split data / Learn hyperparameters / Validate metrics

Evaluation - sklearn

sklearn.metrics: metric(true, predicted/rank,…)
Classification
-classification_report()
-precision, recall, auc, f1_score, jaccard,…
-confusion matrix

Regression
-r2_score(), MAE, MSE

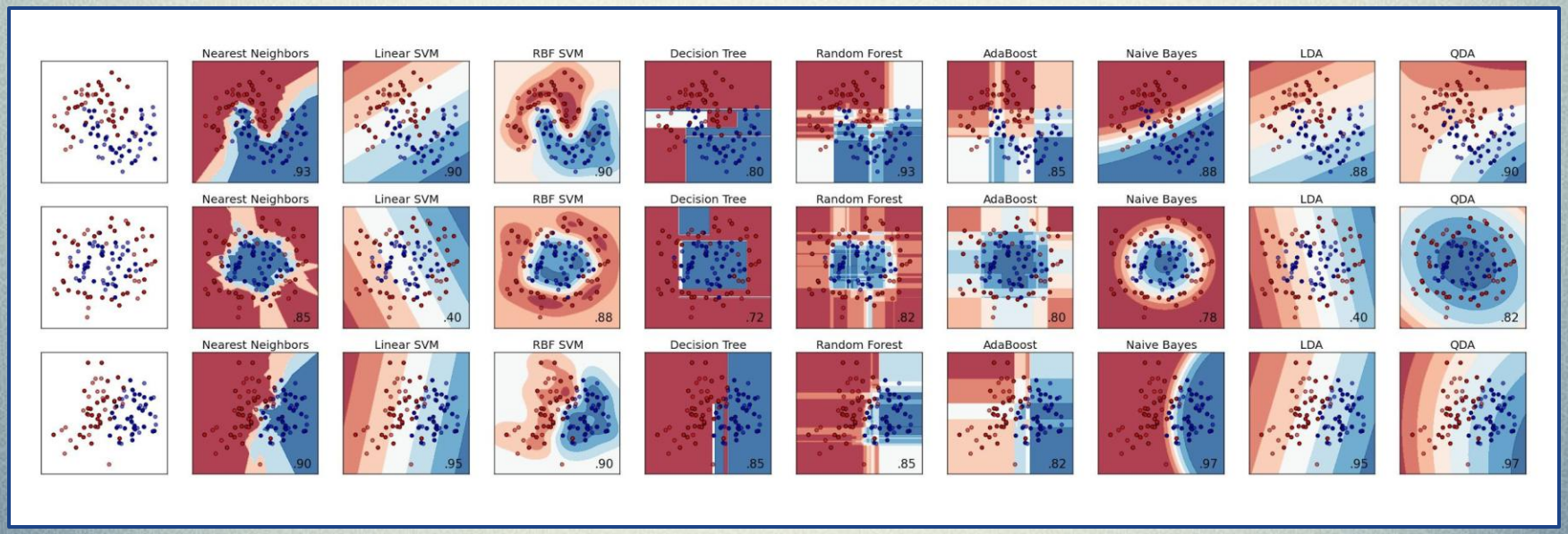
Clustering
Pairwise
- cosine / euklidean distance, …
…
-http://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

```
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import classification_report

iris = datasets.load_iris()
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
svc = svm.SVC()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)

clf = GridSearchCV(svc, parameters)
clf.fit(X,y)
print(clf.cv_results_['mean_test_score'])
print(clf.cv_results_['params'])
print(clf.best_params_)

y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

Other Evaluation

How to evaluate clusters?

Visualization (but only in 2D)



Unpredictable Future

Machine learning models attempt to predict the future as new inputs come in - but human systems and processes are subject to change.

Solution: Precision/Recall tracking over time

Pipelines

Pipelines

`sklearn.pipeline.Pipeline(steps)`

- Sequentially apply *repeatable* transformations to final estimator that can be validated at every step.
- Each step (except for the last) must implement `Transformer`, e.g. `fit` and `transform` methods.
- Pipeline itself implements both methods of `Transformer` and `Estimator` interfaces.

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> from sklearn.pipeline import make_pipeline
>>> model = make_pipeline(PolynomialFeatures(2), linear_model.
Ridge())
>>> model.fit(X_train, y_train)

>>> mean_squared_error(y_test, model.predict(X_test))
3.1498887586451594

>>> model.score(X_test, y_test)
0.97090576345108104
```

Pipelined Model

Task 7

Select dataset (wines / student performance)

- Run train_test_split() @ 20% test set
- Select learning method
  - Select best hyperparameters based on cross-validation

Return R^2, MAE and RMSE based on scipy methods

- vizualize CDF and independent/dependent vars

-try, e.g. StandardScaler() and PolynomialFeatures() for extraction

- http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html

Semestrální projekt – varianta 3

- Identifikace / autentizace osob dle keystroke dynamics
  - Vzdělávací servery / online výuka atp.
  - Soutěž na http://biointelligence.hu/typing-challenge/index.php
    - dataset není v současné době veřejný (GDPR), zájemcům bude předán
    - malý dataset (12 osob, 5 trénovacích a 50 testovacích vzorků / osobu)
    - JavaScript events keyup/down/press, keycode, timestamp
    - Autentizace (potvrzení očekávané osoby)
    - Identifikace (která z osob odpovídá danému vzorku)
  - Evaluace:
    - Odeslat řešení v daném formátu na web soutěže
    - Accuracy (TP+TN)/ALL
    - „Rozumné“ výsledky alespoň okolo ECKNN
  - Typické algoritmy:
    - Dynamic Time Warping (https://en.wikipedia.org/wiki/Dynamic_time_warping)
    - Rychlosti lokálních přechodů / dimensionality reduction
    - RNN? asi příliš malý dataset. Možná document2vec.
  - http://biointelligence.hu/typing.html