

Data cleaning and data preprocessing

Čistenie a predspracovanie údajov

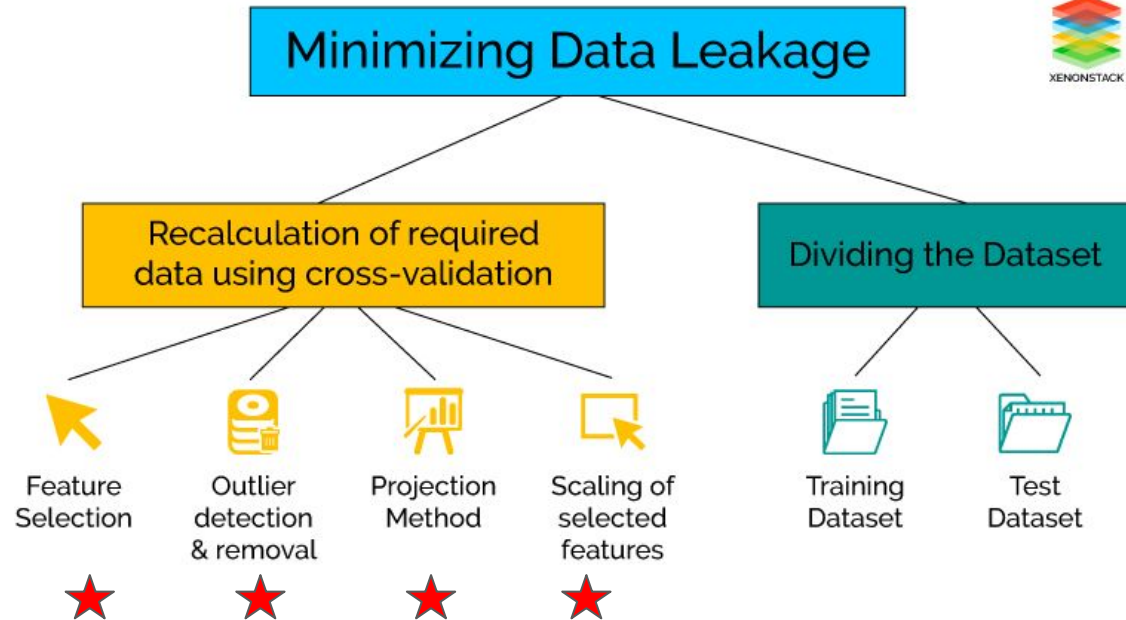
Giang Nguyen - giang.nguyen@stuba.sk

FIIT IAU - ZS 2021/2022



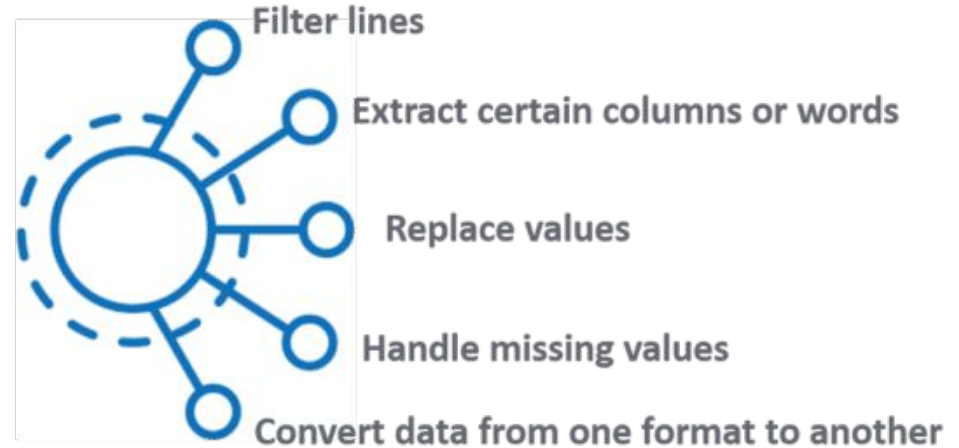
Data preparation without Data Leakage

- **Process of transforming raw data into a form that is appropriate for ML modeling**
- Data preparation must be prepared in order to avoid data leakage
- Data preparation for train-test splits and k-fold cross-validation (4. prednáška)



Data cleaning

Scrubbing or Cleaning Data in Data Science



1. Outlier detection
2. Imputation of missing values
3. Linear column removing *and deduplication* (4. prednáška)

Outlier detection

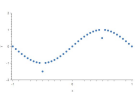
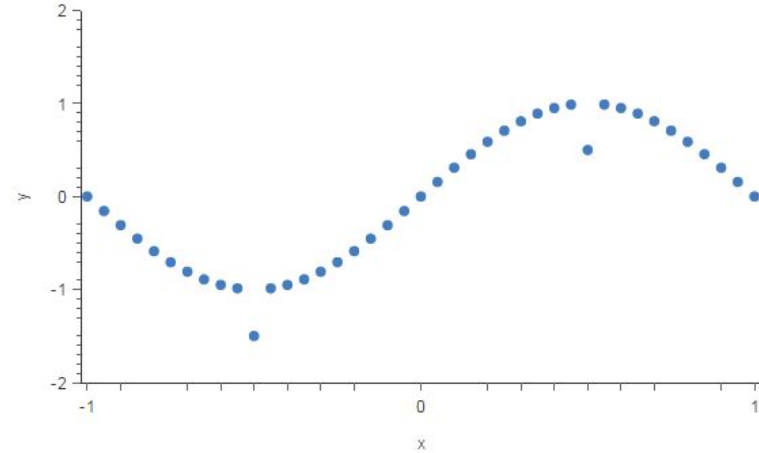
- An outlier is an observation that is unlike the other observations
- They are rare, distinct, or do not fit in some way

Outliers can be

- Measurement or input error
- Data corruption
- True outlier observation !!!

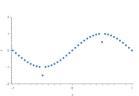
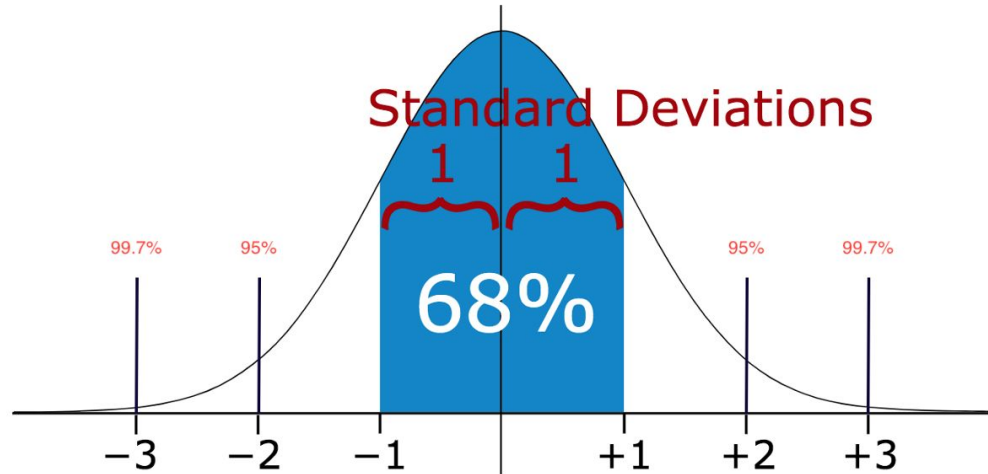
Identifying outliers and bad data is

- One of the most difficult parts of data cleanup
- It can take long time to get right
- They have to be explored cautiously !!!



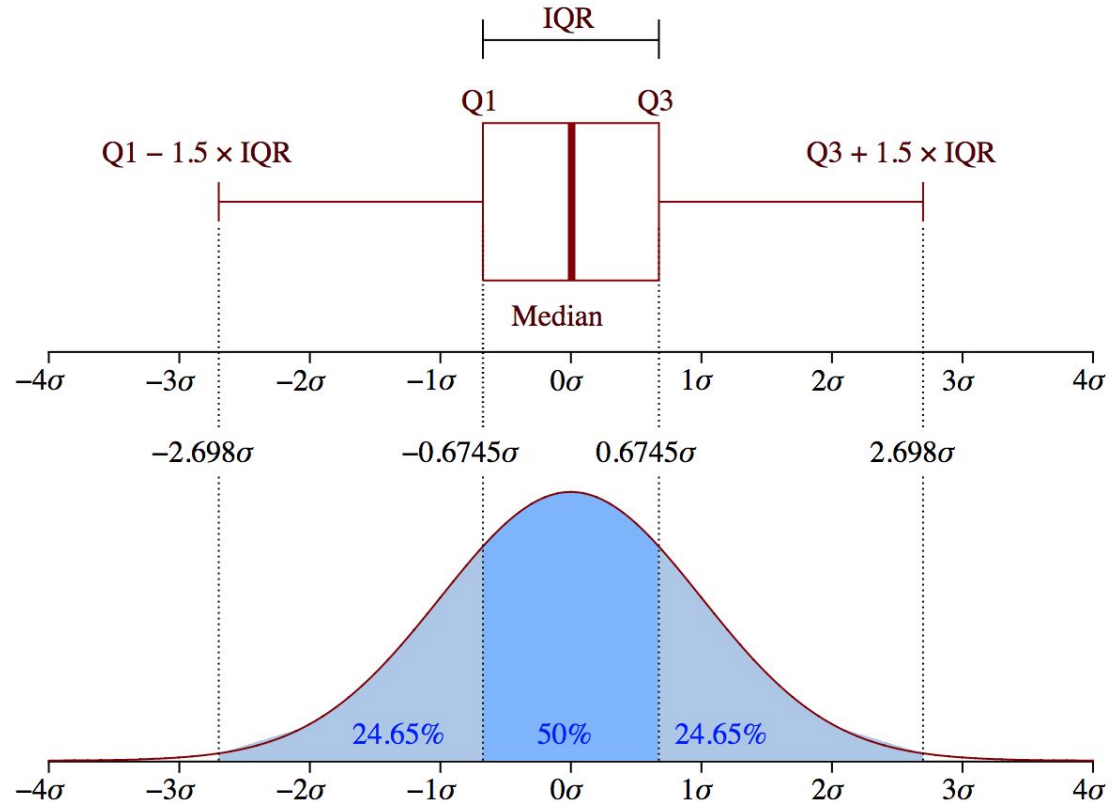
Outlier detection by Standard Deviation

3x standard deviations (σ)
from the **mean** (μ) is a
common cut-off in practice
for identifying outliers in a
Gaussian-like distribution.



Outlier detection by InterQuartile Range (IQR)

Limits on the sample values that are a factor $k=1.5$ of the IQR below the 25% or above 75%

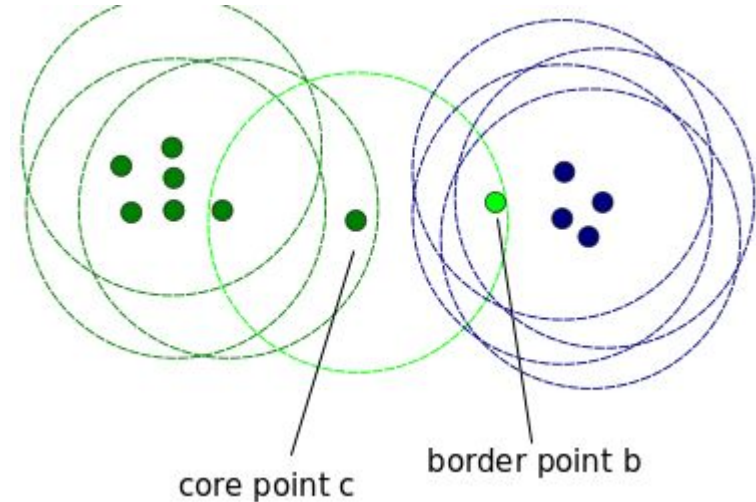


Outlier detection by clustering

Use a simple method to estimate values

- DBScan Clustering
- Isolation Forest
- Random Forest
 - Robust Random Cut Forest

Work well with few features (low dimension)



How to deal with missing data?

It is important to understand that there is **no perfect way** to deal with missing data

Especially, if the number of missing values in your data is big enough **above 5%**



- **Must be marked with NaN**
- Can be replaced with statistical measures to calculate the column of values
- **Use imputation techniques cautiously !!!**

Imputation techniques

- **Delete rows with missing data**
- Mean/Median/Mode imputation
- Assigning a unique value
- Predicting the missing values
- Using an algorithm which supports missing values, like random forests

Imputation of missing values in Python

The imputation strategy

- If “**mean**”, then replace missing values using the mean along each column. Can only be used with numeric data.
- If “**median**”, then replace missing values using the median along each column. Can only be used with numeric data.
- If “**most_frequent**”, then replace missing using the most frequent value along each column. Can be used with strings or numeric data.
- If “**constant**”, then replace missing values with fill_value. Can be used with strings or numeric data.

```
import numpy as np
from sklearn.impute import SimpleImputer

data = [[7, 2, 3], [4, np.nan, 6], [10, 5, 9]]
print(data)

# imputer
imp_mean = SimpleImputer(missing_values=np.nan,
                          strategy='mean')

imp_mean.fit(data)

X = [[np.nan, 2, 3], [4, np.nan, 6], [10, np.nan, 9]]
X_t = imp_mean.transform(X)
print(X_t)
```

KNN Imputation

- The **KNNImputer** is a data transform that is first configured based on the method used to estimate the missing values
- The default distance measure is a **Euclidean distance** measure that is NaN aware
- The **number of neighbors** is set to **5** by default and can be configured by the **n_neighbors** argument.

```
from numpy import isnan
from pandas import read_csv
from sklearn.impute import KNNImputer

dataframe = read_csv('data/horse-colic.csv',
                     header=None, na_values='?')

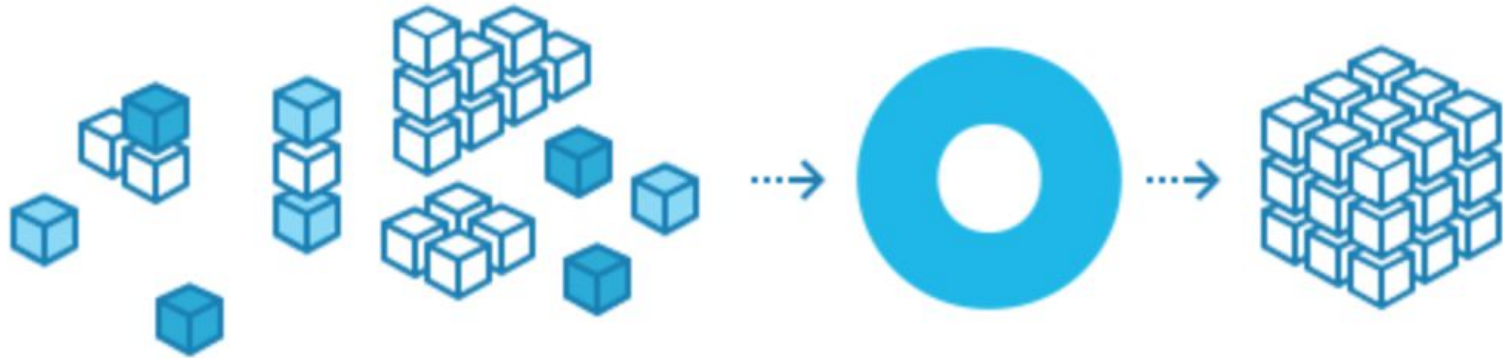
data = dataframe.values
ix = [i for i in range(data.shape[1]) if i != 23]
X, y = data[:, ix], data[:, 23]
print('Missing: %d' % sum(isnan(X).flatten()))

# imputer
imputer = KNNImputer(n_neighbors=5,
                     weights='uniform',
                     metric='nan_euclidean')

imputer.fit(X)
X_t = imputer.transform(X)

# summarize total missing
print('Missing: %d' % sum(isnan(X_t).flatten()))
```

Data Transformations



1. Scale numerical data
2. Make data more Gaussian
3. Encode categorical data

Data Scaling for numerical data

Why data scaling?

- Many ML algorithms perform better or when features are on a relatively similar to normally distributed

Scaling approaches

- Data Normalization
- Data Standardization
- Robust Scaling (Data Standardization)

Examples of such ML algorithms

- Linear and Logistic Regression
- k -nearest neighbors (KNN)
- Neural Networks (NNs)
- Support Vector Machines (SVM)
- Principal Components Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- and many more ...



Data Normalization

Normalizing - transforming numeric data to the same scale as other numeric data.

- **In ML, every dataset does not require normalization.**
- It is required only when **features have different ranges** because this can lead to
 - unstable or poor performance of models
 - increasing sensitivity to inputs
 - higher generalization error
- Normalization is necessary if you have very different values within the same feature (for example, city population).
- Without normalization, training could blow up with NaN if the gradient update is too large.



Data Standardization (z-normalization)

$$z = \frac{x - \mu}{\sigma}$$

where:

μ is the mean of the population.

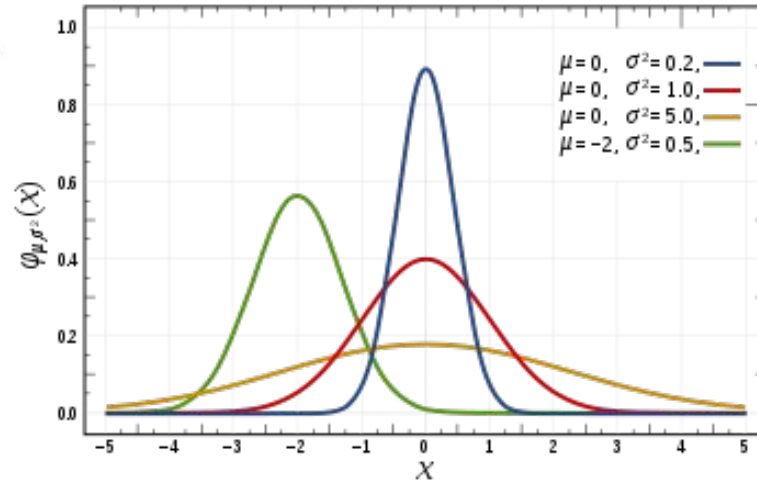
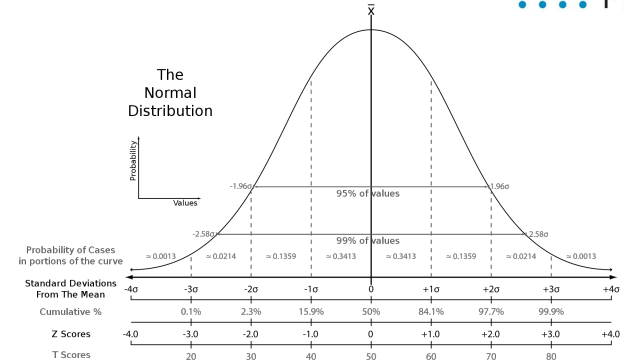
σ is the standard deviation of the population.

$$z = \frac{x - \bar{x}}{S}$$

where:

\bar{x} is the mean of the sample.

S is the standard deviation of the sample.



Data Normalization and Standardization

1. Data Normalization

- MinMaxScaler Transform

$$x_{normalization} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

2. Data Standardization

- z-normalization
- StandardScaler Transform

$$x_{standardized} = \frac{x - \mu}{\sigma}$$

where

- μ is the mean of x
- σ is the standard deviation of x

```
from numpy import asarray
from sklearn.preprocessing import MinMaxScaler
# from sklearn.preprocessing import StandardScaler
```

```
data = asarray([[100, 0.001],
                [8, 0.05],
                [50, 0.005],
                [88, 0.07],
                [4, 0.1]])
```

```
print(data)
```

```
scaler = MinMaxScaler()
# scaler = StandardScaler()
```

```
scaled = scaler.fit_transform(data)
print(scaled)
```



Robust Scaling

Robust scaler removes the median and scales the data according to the quantile range.

The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).

$$x_{robust} = \frac{x - median}{p_{75} - p_{25}}$$

where

- *median* is the median of x
- p_{25} and p_{75} are 25th and 75th interquartile range values (IQR).

```
from numpy import asarray
from sklearn.preprocessing import RobustScaler

data = asarray([[100, 0.001],
                 [8, 0.05],
                 [50, 0.005],
                 [88, 0.07],
                 [4, 0.1]])

print(data)

# define robust scaler
scaler = RobustScaler()

# transform data
scaled = RobustScaler().fit_transform(data)
print(scaled)
```

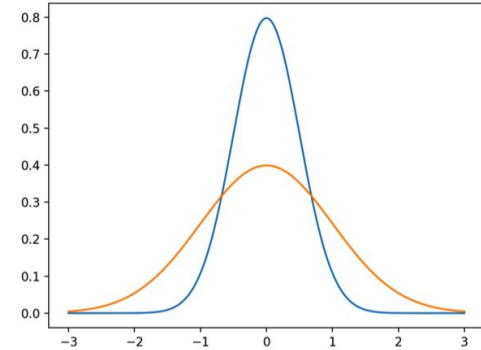


Make data more Gaussian: Power Transformer

Many ML algorithms perform better or when features are on a relatively similar to normally (Gaussian) distributed.

Power Transformer

- **Yeo-Johnson transform: works with \pm values**
- **Box-Cox transform: only works with strictly positive values**
- **Replacing the data with the log, square root, or inverse to remove skew**
 - $\lambda = -1.0$ is a **reciprocal** transform ($1/x$)
 - $\lambda = -0.5$ is a **reciprocal square root** transform ($1/\sqrt{x}$)
 - $\lambda = 0.0$ is a **log** transform $\log(x)$
 - $\lambda = 0.5$ is a **square root** transform (\sqrt{x})
 - $\lambda = 1.0$ is no transform.

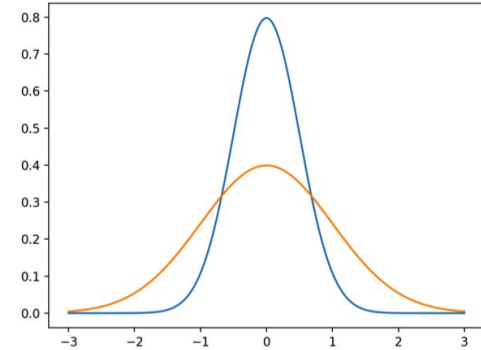


Make data more Gaussian: Quantile Transformer

Many ML algorithms perform better or when features are on a relatively similar to normally (Gaussian) distributed.

Quantile Transformer

- The transformation is applied on each feature independently, transforms the features to follow a uniform or a normal distribution
- Non-linear transform, which may distort linear correlations between variables



Power Transformer

```
from matplotlib import pyplot
from numpy import exp
from numpy.random import randn
from sklearn.preprocessing import PowerTransformer

# data
data = randn(1000)
data = exp(data)
pyplot.hist(data, bins=25)

# reshape data to have rows and columns
data = data.reshape((len(data), 1))

# power transform the raw data
power = PowerTransformer(
    method='yeo-johnson',
    standardize=True)
data_trans = power.fit_transform(data)

# histogram of the transformed data
pyplot.hist(data_trans, bins=25)
```

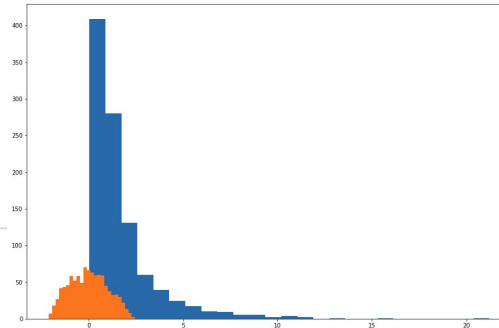
Quantile Transformer

```
from matplotlib import pyplot
import numpy as np
from sklearn.preprocessing import QuantileTransformer

rng = np.random.RandomState(0)
X = np.sort(rng.normal( loc=0.5,
                        scale=0.25,
                        size=(100, 1)),
            axis=0)

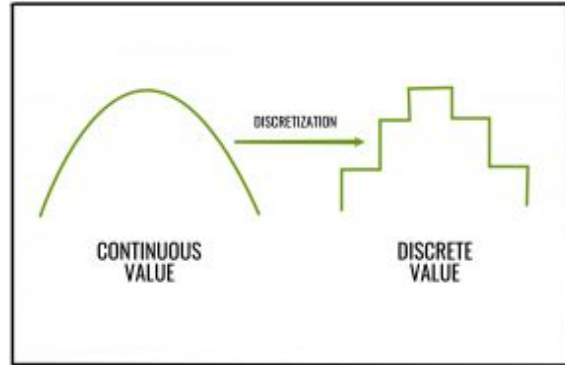
pyplot.hist(X, bins=10)

qt = QuantileTransformer(n_quantiles=10,
                        random_state=0)
X_t = qt.fit_transform(X)
```



Data Bucketing

Data Bucketing (also known as Data Binning or Data Discretization) is used to reduce the effects of minor observation errors, e.g., the original **data values** fall into a **given small interval**, are replaced by a **value representative** of that interval, often the central value.



Bucketing - transforming numeric (usually continuous) data to categorical data.

- Be clear about how you are setting the boundaries and which type of bucketing you're applying
- **Buckets with equally spaced boundaries:** the boundaries are fixed and encompass the same range
- **Buckets with quantile boundaries:** each bucket has the same number of points. The boundaries are not fixed and could encompass a narrow or wide span of values.



Transform and Inverse Transform

Data Transformation

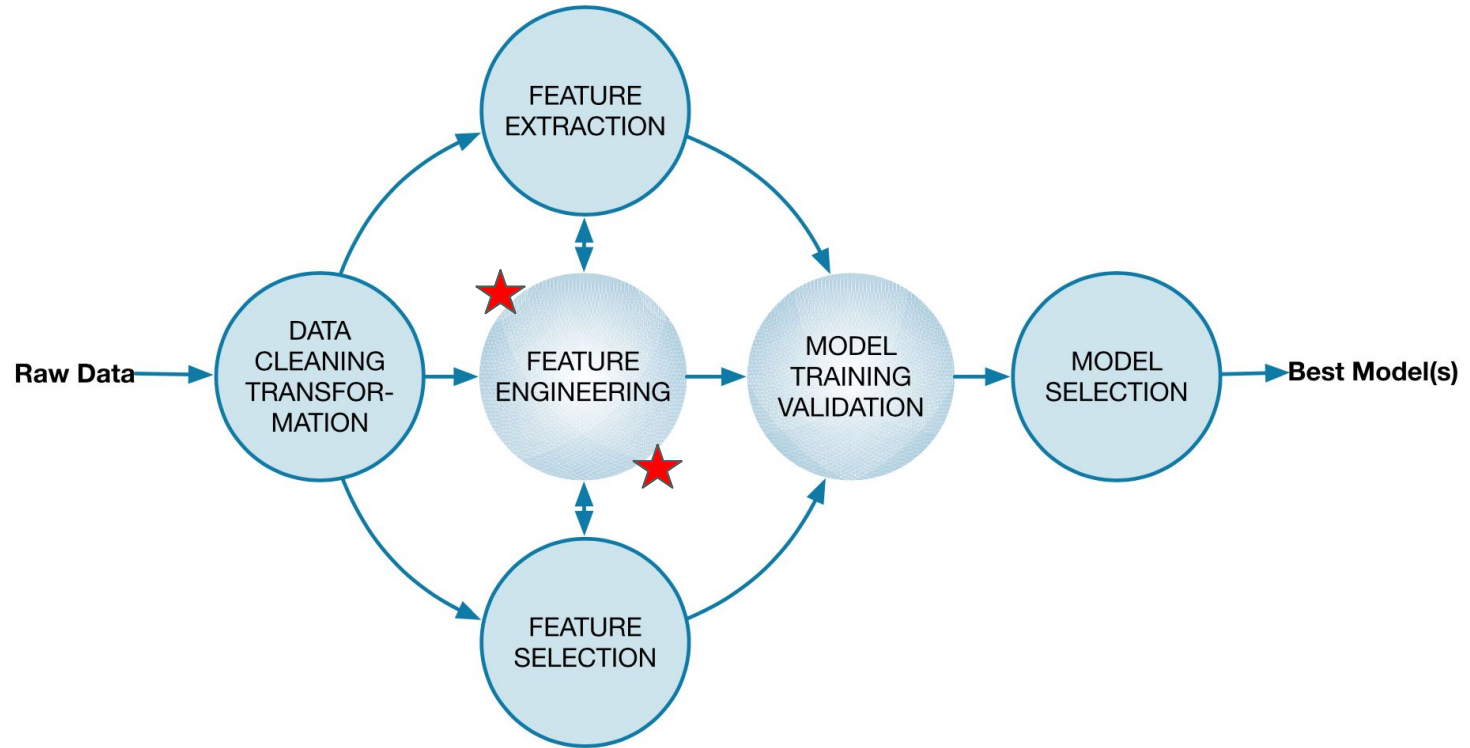
- Applies deterministic mathematical function $f_{transform}$ to each datapoint x_i in dataset \mathcal{X} by replacing it with the transformed value $y_i = f_{transform}(x_i)$

Inverse Transformation

- If desired, the output from ML modelling can be transformed back to the **original scale** using the **inverse** f_{invert} **of the transformation** $f_{transform}$ that was applied to the data



Data Preprocessing



Feature Extraction and Feature Encoding

Feature extraction

- **7. prednáška**
- Feature extraction from text source
- Brief on automatic feature extraction from structured data like images











Feature encoding

- Most of ML libraries works (only) with numbers
- Encode categorical data



Encoding categorical data

1. Ordinal Encoding or Label Encoding
2. **One Hot Encoding**
3. Binary Encoding
4. BaseN Encoding
5. **Hashing (md5 algorithm)**
6. Target Encoding
7. Leave One Out

Gender		Is_Male	Is_Female	Tree		Type
	→	0	1		→	1
	→	0	1		→	2
	→	1	0		→	1
	→	0	1		→	2
	→	1	0		→	3



Encoding categorical data in Python

```
import pandas as pd
import category_encoders as ce

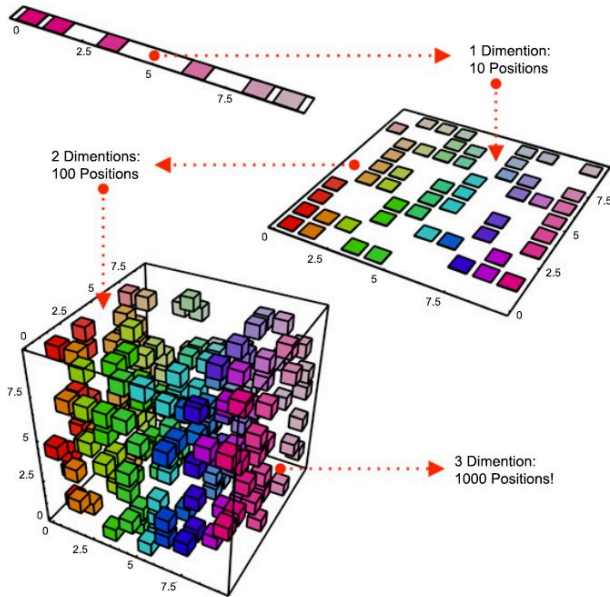
# make some data
data = pd.DataFrame({
    'class' : ['a', 'b', 'a', 'b', 'd', 'e', 'd', 'f', 'g', 'h', 'h', 'k', 'h', 'i', 's', 'p', 'z']})

# data = pd.DataFrame({
#     'color' : ['Blue', 'Black', 'Black', 'Blue', 'Blue'],
#     'outcome' : [2, 1, 1, 1, 2] })

# create object of Encoder
# ce_encoder = ce.OrdinalEncoder(cols = ['class'])
# ce_encoder = ce.OneHotEncoder(cols = ['class'])
# ce_encoder = ce.BinaryEncoder(cols = ['class'])
# ce_encoder = ce.BaseNEncoder(cols=['class'], base=4)
ce_encoder = ce.HashingEncoder(cols=['color'], n_components=5)
# ce_encoder = ce.TargetEncoder(cols=['color'])
# ce_encoder = ce.LeaveOneOutEncoder(cols=['color'])

# fit and transform to get the encoded data
ce_encoder.fit_transform(data)
# ce_encoder.fit_transform(X,Y)
```

Curse of dimensionality

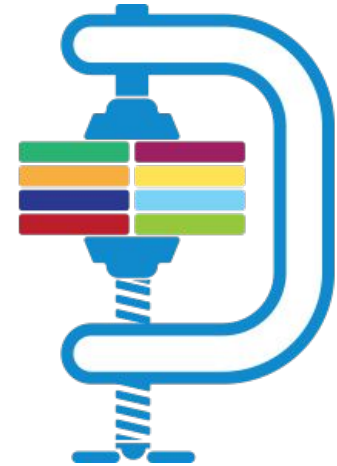


- The **fundamental reason** for the **curse of dimensionality** is that high-dimensional functions have the potential to be **much more complicated** than low-dimensional ones
- **High-dimensionality** might mean hundreds, thousands, or even millions of input variables
- The presence of non-informative variables can add uncertainty to the predictions and reduce the overall effectiveness of the model

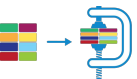
Data reduction

Obtains reduced representation in volume but produces the same or similar analytical results

Data discretization, data aggregation, dimensionality reduction, data compression, feature reduction, generalization



Data preparation technique performed on data prior to modeling

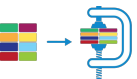


Sparse data

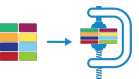
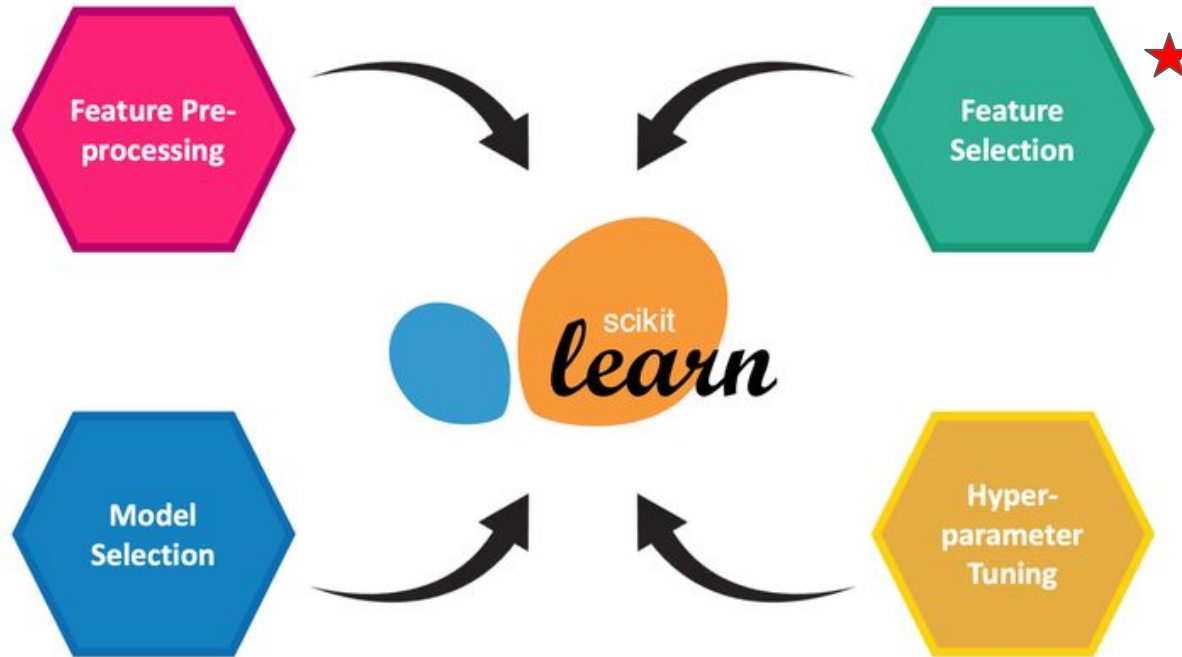
- **Sparse data** refers to rows of data where **many** of values are **zero**
- **Examples of sparse data**
 - Text Classification
 - Recommender Systems
 - Customer-Product purchases
 - User-Song Listen Counts
 - User-Movie Ratings
 - Bag-of-Words
 - One Hot Encoding

Feature selection

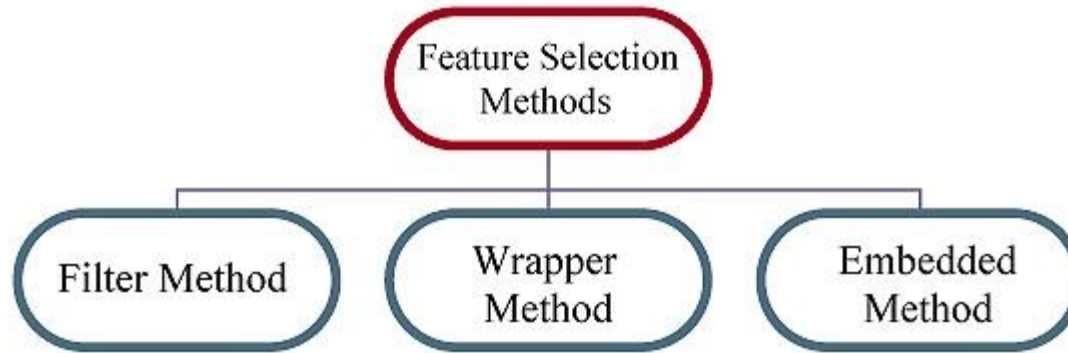
- **Selecting a subset of input features** that are most relevant to the target variable that is being predicted.
- Process of reducing the number of input variables when **developing** a predictive model.
- **Reduce the number of input variables to**
 - **Reduce the computational cost of modeling**
 - **Improve the ML model performance**



Feature Selection



Feature Selection Methods



Filter

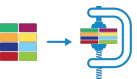
- Variance Threshold
- Mutual Information
- Chi-Squared (χ^2)
- F-value, etc.

Wrapper

- RFE
- Selection from model, etc.

Embedded

- LASSO
- Elastic Net
- Ridge Regression, etc.



Filter feature selection methods



- Use statistical techniques to **evaluate the relationship between each input variable and the target variable**,
- These scores are used **to rank and choose** those input variables to be used in the model.

**Set of all
Features**



**Selecting the
Best Subset**



**Learning
Algorithm**



Performance



Variance Threshold

Simple method, which removes all features whose variance doesn't meet some threshold (linear column removing) **!!! cautiously !!!**

```
from sklearn.feature_selection import VarianceThreshold
```

```
X = [ [0, 0, 1], [0, 1, 0], [1, 0, 0],  
       [0, 1, 1], [0, 1, 0], [0, 1, 1]]  
selector = VarianceThreshold(threshold=(0.8*(1-0.8)))  
X_new = selector.fit_transform(X)  
print(X_new)
```

```
array([[0, 1],  
       [1, 0],  
       [0, 0],  
       [1, 1],  
       [1, 0],  
       [1, 1]])
```



Mutual Information

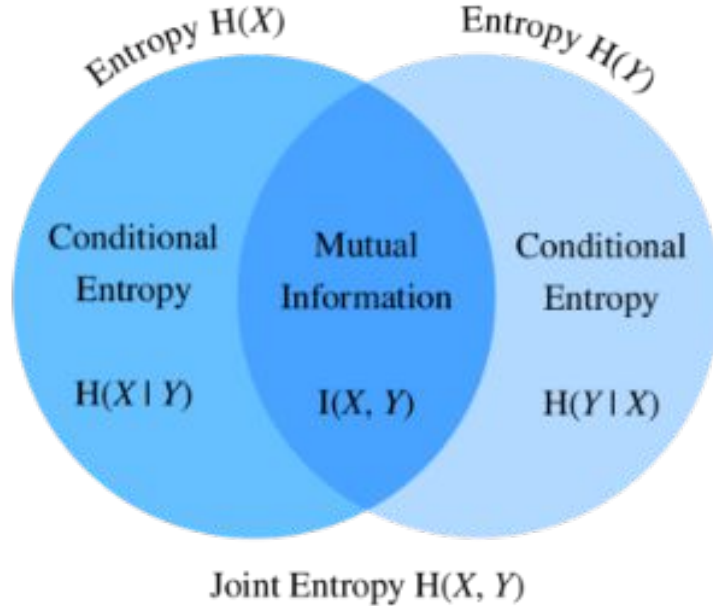
Mutual information (MI) between two random variables is a **non-negative** value, which measures the dependency between the variables. *MI* is linked to **Information Entropy**, it is also closely related to *Information Gain (IG)*.

$$\text{Entropy} = \sum_{i=1}^C -p_i * \log_2(p_i)$$

It is equal to zero if and only if two random variables are independent
Higher values mean higher dependency.



Mutual Information and Information Entropy



The concept of *MI* is linked to information theory and **information entropy** (H).

MI is non-negative and symmetric.

$$H(X) = - \int dx \mu(x) \log \mu(x)$$

$$I(X, Y) = - \int \int dx dy \mu(x, y) \log \frac{\mu(x, y)}{\mu_x(x) \mu_y(y)}$$

For discrete variables, $H(X)$ is calculated as:

$$H(X) = - \sum_i P(x_i) \log P(x_i)$$

$$I(X; Y) = H(X) - H(X|Y)$$

The entropy of X after observing values of Y is formulated by

$$H(X|Y) = - \sum_j P(y_j) \sum_i P(x_i|y_j) \log_2 P(x_i|y_j)$$

where

- $P(x_i)$ is the prior probabilities for all values of X and
- $P(x_i|y_j)$ is the posterior probabilities of X given the values of Y .



Chi-Squared and F-value

- The **chi-square (χ^2) statistic** compares the size any discrepancies between the expected results and the actual results, given the size of the sample and the number of variables in the relationship.
- An **F statistic is a value** you get when you run an ANOVA test or a regression analysis to find out if the **means between two populations** are significantly different. It's similar to a T statistic from a T-Test

$$\chi_{df}^2 = \sum \frac{(X_i - Y_i)^2}{Y_i}$$

where:

- df = Degrees of freedom
- X = Observed value(s)
- Y = Expected value(s)

$$F_{value} = \frac{variance_{dataset1}}{variance_{dataset2}} = \frac{\sigma_1^2}{\sigma_2^2}$$



Mutual Information, Chi-Squared, F-value in Python

```
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_regression
# from sklearn.feature_selection import chi2
# from sklearn.feature_selection import f_regression

X, y = load_iris(return_X_y=True)
print(X.shape)

(150, 4)

X_new = X_new = SelectKBest(mutual_info_regression, k=2).fit_transform(X, y)
# X_new = SelectKBest(chi2, k=2).fit_transform(X, y)
# X_new = SelectKBest(f_regression, k=2).fit_transform(X, y)
print(X_new.shape)

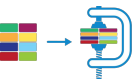
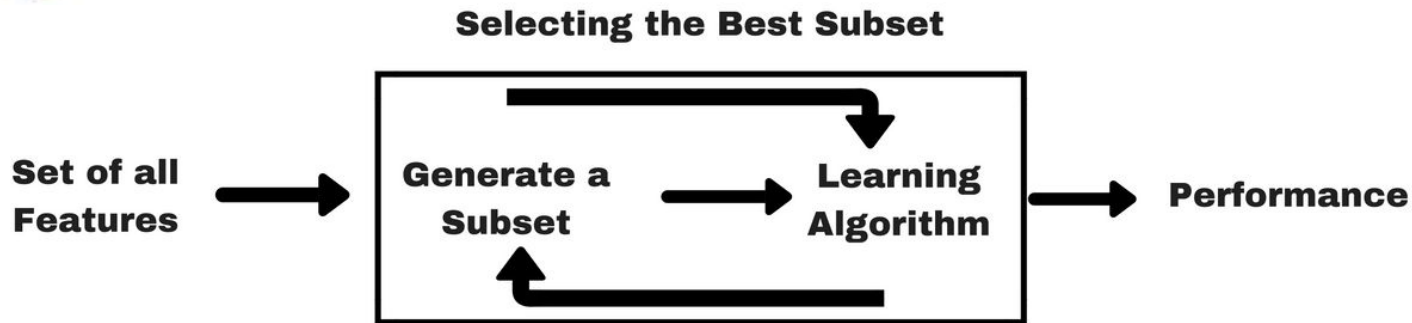
(150, 2)
```



Wrapper feature selection methods



- Create **many models with different subsets** of input features and
- Select those features that result in the best performing model according to a performance metric.
- **Can be computationally expensive.**



Recursive Feature Elimination (RFE)

Recurrently remove less important features from the set

```
from sklearn.datasets import make_friedman1
from sklearn.feature_selection import RFE
from sklearn.svm import SVR

X, y = make_friedman1(n_samples=50, n_features=10, random_state=0)

estimator = SVR(kernel="linear")
selector = RFE(estimator, n_features_to_select=5, step=1)
selector = selector.fit(X, y)

print(selector.support_)

[ True  True  True  True  True False False False False False]

print(selector.ranking_)

[1 1 1 1 1 6 4 3 2 5]
```



Selection from model: L1-based feature selection

Use a simple model (SVC) to calculate importance weights

```
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectFromModel
from sklearn.svm import LinearSVC

X, y = load_iris(return_X_y=True)
print(X.shape)
(150, 4)

lsvc = LinearSVC(C=0.01, penalty="l1", dual=False).fit(X, y)
selector = SelectFromModel(lsvc, prefit=True)
print(selector.get_support())
[ True  True  True False]

X_new = selector.transform(X)
print(X_new.shape)
(150, 3)
```

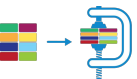


Embedded Methods

Embedded methods check the different training iterations of the ML model and evaluate the importance of each feature. Just certain ML methods (e.g. linear and logistic regression) have this option.

- Lasso (L1)
- Ridge (L2)
- Elastic Net: linear combination of Lasso and Ridge

(9. prednáška)

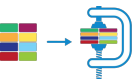


Sparse data

- **Sparse data** refers to rows of data where **many** of values are **zero**
- **Examples of sparse data**
 - Text Classification
 - Recommender Systems
 - Customer-Product purchases
 - User-Song Listen Counts
 - User-Movie Ratings
 - Bag-of-Words
 - **One Hot Encoding**

Dimensionality reduction

- Dimensionality reduction refers to **reducing the number of input variables** for a dataset.
- The resulting dataset, the projection, can then be used as **input to train a ML model**.
- A popular approach to dimensionality reduction is to use techniques from the field of linear algebra. This is often called **feature projection** and the algorithms used are referred to as projection methods.

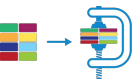


Feature selection

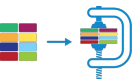
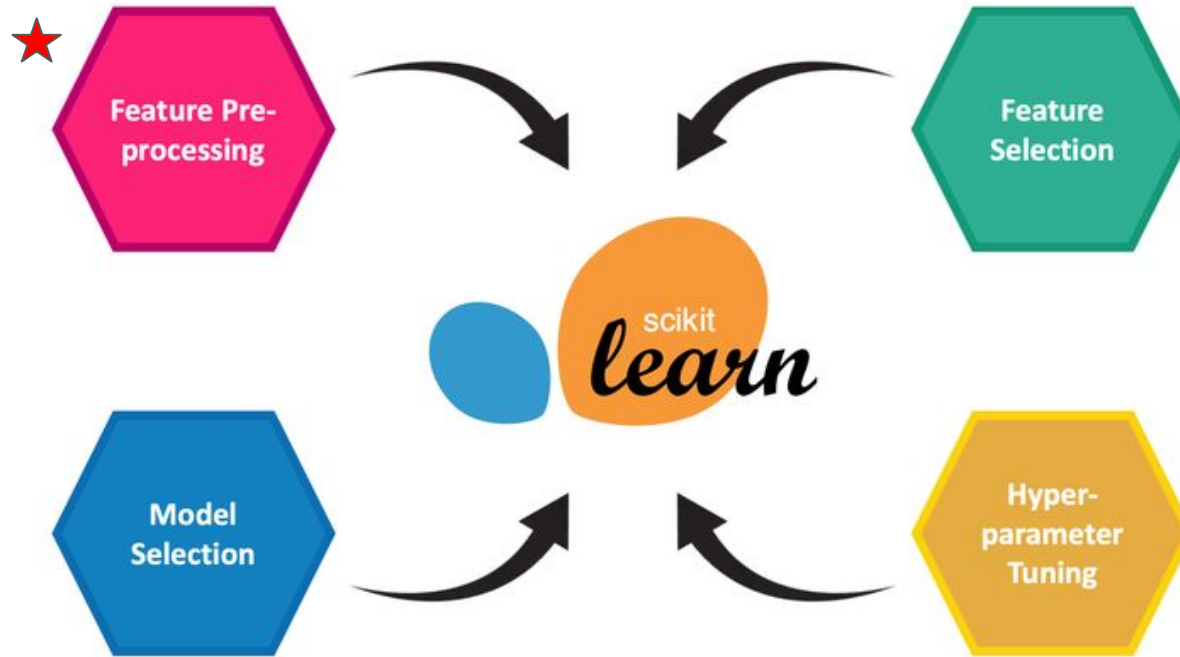
- **Selecting a subset of input features** that are most relevant to the target variable that is being predicted.
- Process of reducing the number of input variables when **developing** a predictive model.
- Reduce the number of input variables to
 - **Reduce the computational cost of modeling**
 - **Improve the ML model performance**

Dimensionality reduction

- Dimensionality reduction refers to **reducing the number of input variables** for a dataset.
- The resulting dataset, the projection, can then be used as **input to train a ML model**.
- A popular approach to dimensionality reduction is to use techniques from the field of linear algebra. This is often called **feature projection** and the algorithms used are referred to as projection methods.



Dimensionality reduction



Dimensionality reduction approaches

Matrix factorization

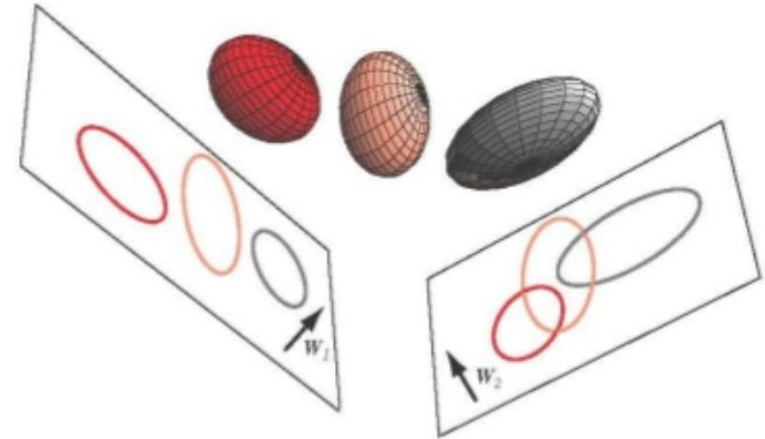
- Singular Value Decomposition (SVD)
- Principal Components Analysis (PCA)

Model-based projection

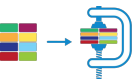
- Linear Discriminant Analysis (LDA)

Manifold Learning

- MDS, Isomap, t-SNE, ...

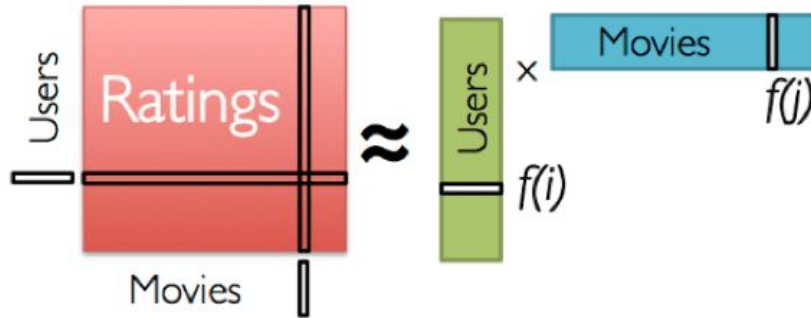


There is no best technique for dimensionality reduction.



Matrix Factorization

Low-Rank Matrix Factorization:



Reduce a **dataset matrix** into **constituent parts**

- EigenDecomposition
- **Singular Value Decomposition (SVD)**

These constituent parts can be **ranked** and then **selected** that best captures the salient structure of the matrix

- **Principal Components Analysis (PCA)**

Singular Value Decomposition (SVD)

$$\begin{array}{c}
 \boxed{\begin{array}{c} A \\ n \times d \end{array}} = \boxed{\begin{array}{c} \hat{U} \\ n \times r \end{array}} \boxed{\begin{array}{c} \hat{\Sigma} \\ r \times r \end{array}} \boxed{\begin{array}{c} \hat{V}^T \\ r \times d \end{array}} \\
 \begin{array}{ccc} U & \Sigma & V^T \\ n \times d & n \times d & d \times d \end{array}
 \end{array}$$

Singular Value Decomposition (SVD) in Python

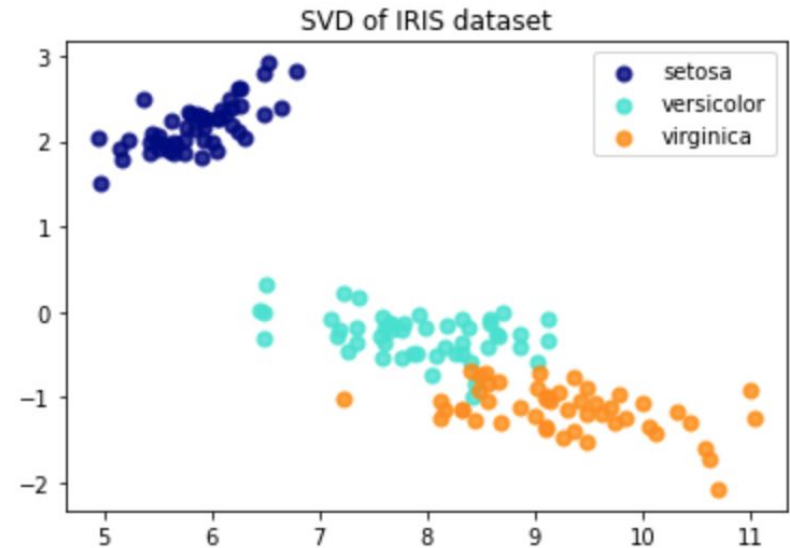
```
from sklearn import datasets
from sklearn.decomposition import TruncatedSVD
```

```
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

```
svd = TruncatedSVD(n_components=2)
X_svd = svd.fit(X).transform(X)
```

SVD: reduced shape (150, 2)

Text(0.5, 1.0, 'SVD of IRIS dataset')



Principal Components Analysis (PCA) in Python

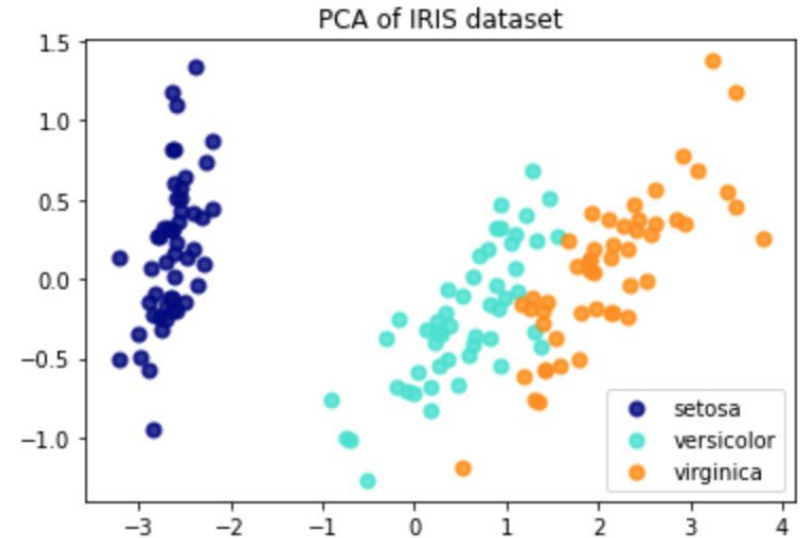
```
from sklearn import datasets  
from sklearn.decomposition import PCA
```

```
iris = datasets.load_iris()  
X = iris.data  
y = iris.target
```

```
pca = PCA(n_components=2)  
X_pca = pca.fit(X).transform(X)
```

PCA: reduced shape (150, 2)

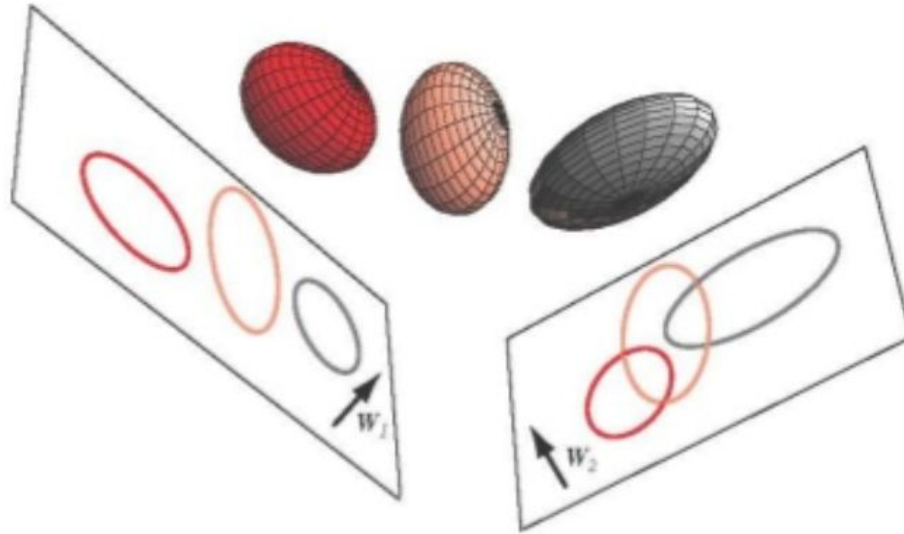
Text(0.5, 1.0, 'PCA of IRIS dataset')



Model-based projections

Linear Discriminant Analysis (LDA)

is most commonly used as dimensionality reduction technique



Linear Discriminant Analysis (LDA) in Python

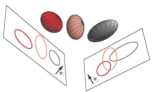
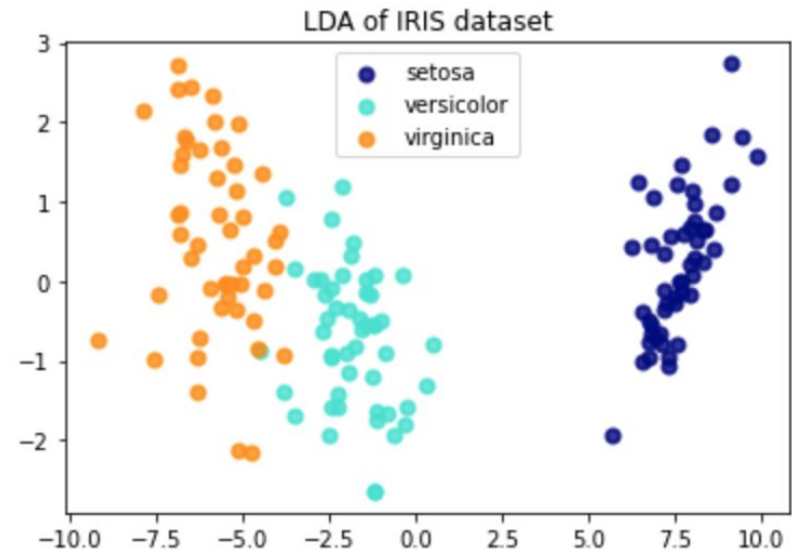
```
from sklearn import datasets
from sklearn.discriminant_analysis import
    LinearDiscriminantAnalysis

iris = datasets.load_iris()
X = iris.data
y = iris.target

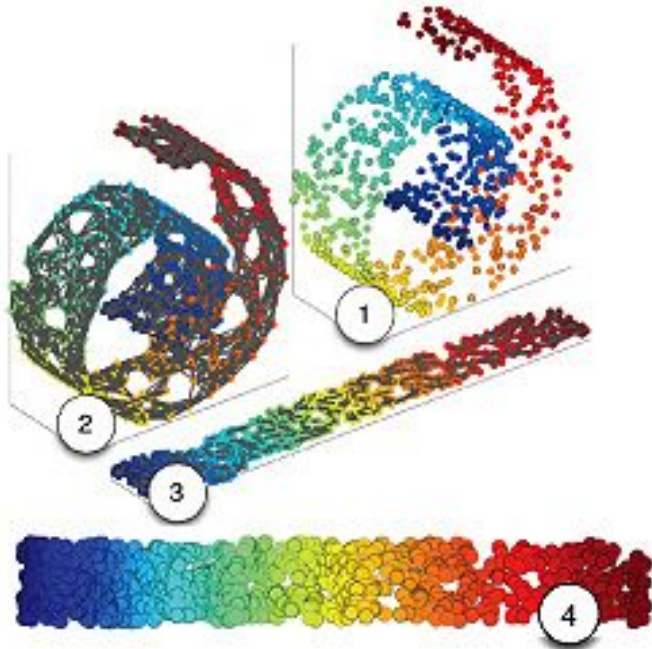
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit(X, y).transform(X)
```

LDA: reduced shape (150, 2)

Text(0.5, 1.0, 'LDA of IRIS dataset')



Dimensionality reduction by Manifold Learning



- **Manifold Learning** are techniques used to **create a low-dimensional projection of high-dimensional data**, often for data visualization
- Can be viewed as the non-linear version of PCA
- **The features in the projection often have little relationship with the original columns**

Manifold Learning methods in Python

```
from sklearn import datasets
from sklearn.manifold import MDS
```

```
X, y = load_digits(return_X_y=True)
print(X.shape)
```

```
(1797, 64)
```

```
embedding = MDS(n_components=2)
X_t = embedding.fit_transform(X[:100])
print(X_t.shape)
```

```
(100, 2)
```

- **MultiDimensional Scaling (MDS)**
- **Isomap Embedding**
- **t-distributed Stochastic Neighbor Embedding (t-SNE)**
- Kohonen Self-Organizing Map (SOM)
popularity failed since 2005 due to efficiency concerns



A note on data preprocessing

- **Scikit-learn is modern library, which implements many trendy and optimized algorithms**
- It is not only ML library but also contains other aspects such as feature selection, **data preprocessing**, model evaluation, optimizations, as well as codepipeline
- However, in real world, we work with many other intelligent libraries, e.g., TensorFlow, Keras, PyTorch, ...
- Various intelligent libraries have various requirements on data preprocessing
- **Therefore, the work and the study of other data processing tools like Pandas, SQL are essential, important and needed.**





Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.



A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M', 'M', 'F', 'F', 'M', 'F', 'M', 'M', 'F', 'F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    random_state=0)
```

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning

```
>>> lr.fit(X, y)
>>> knn.fit(X_train, y_train)
>>> svc.fit(X_train, y_train)
```

Fit the model to the data

Unsupervised Learning

```
>>> k_means.fit(X_train)
>>> pca_model = pca.fit_transform(X_train)
```

Fit the model to the data

Fit to data, then transform it

Prediction

Supervised Estimators

```
>>> y_pred = svc.predict(np.random.random((2,5)))
>>> y_pred = lr.predict(X_test)
>>> y_pred = knn.predict_proba(X_test)
```

Predict labels

Predict labels

Estimate probability of a label

Unsupervised Estimators

```
>>> y_pred = k_means.predict(X_test)
```

Predict labels in clustering algos

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```

Estimator score method
Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```

Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_true, y_pred)
```

R² Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

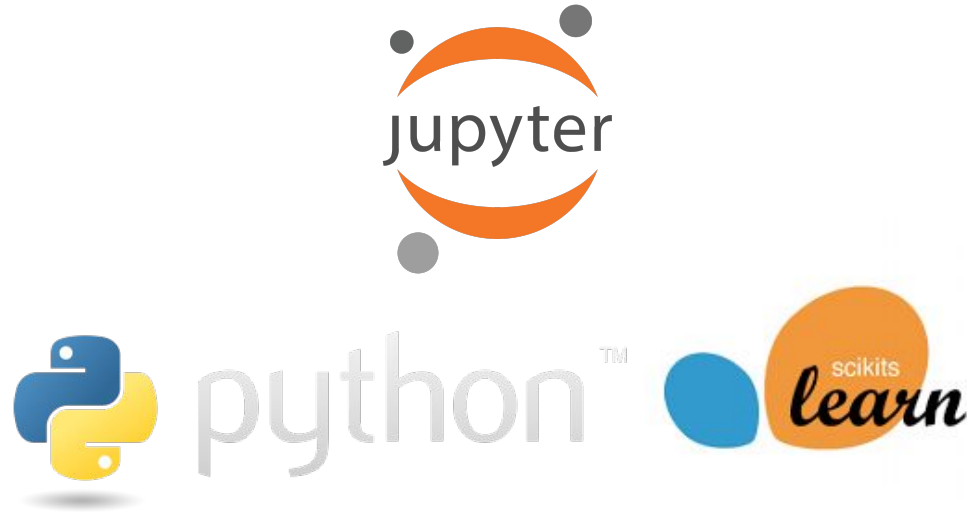
Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
>>>           "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
>>>                     param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
>>>           "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
>>>                               param_distributions=params,
>>>                               cv=4,
>>>                               n_iter=8,
>>>                               random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```





Jupyter notebooks

Data transformation,

Feature selection, Dimensionality reduction

References

- Basics of Data Preprocessing
- Data Preparation for Machine Learning
- Data Preparation Process, Preprocessing and Data Wrangling
- Data Preparation and Feature Engineering for Machine Learning
- What is the Team Data Science Process?
- 6 Steps For Ultimate Data Cleaning That Can Drastically Boost Your Business
- 5 Ways to Detect Outliers/Anomalies That Every Data Scientist Should Know
- Visualizing cross-validation behavior in scikit-learn
- Dive into Deep Learning: Information Theory
- Chi-Square (χ^2) Statistic Definition
- The 7 Variants of Matrix Factorization
- Types of Categorical Data Encoding Schemes
- What Is Manifold Learning?

Next time:

Predspracovanie textových dát

Preprocessing of textual data

SENTIMENT ANALYSIS



Discovering people opinions, emotions and feelings about
a product or service