Title: Pre–processing

Course: ML

Instructor: Claudio Sartori

Master: Data Science and Business Analytics

Master: Artificial Intelligence and Innovation Management

Academic Year: 2024/2025

BOLOGNA BUSINESS SCHOOL

Alma Mater Studiorum Università di Bologna

Why?

- Problems in data
  - Missing values
  - Skewed distributions
  - Outliers
  - Dimensionality
  - Errors and duplications
- Types not adequate for the processing required
- Necessary to point out particular aspects

Data pre–processing

- Dealing with missing values
- Attribute Type Transformation – Discretization and Binarization
- Sampling
- Dimensionality Reduction
- Feature subset selection
- Feature creation

Dealing with Missing Values in Scikit-learn

Scikit-learn provides various methods to handle missing values:

- **Removing Rows or Columns with Missing Values**
- **Simple Imputation with Mean, Median, or Mode**
- **Iterative Imputation**
- **Using Indicators for Missing Values**

Removing Rows or Columns with Missing Values

- The simplest way to handle missing data is to remove rows or columns containing NaN values.
  - Use `pandas.dropna()` for DataFrame preprocessing:

```
import pandas as pd
data = pd.DataFrame({
           'A': [1, 2, None],
           'B': [4, None, 6]
       })
# Drop columns with missing values
cleaned_columns = data.dropna(axis=1)
# Drop rows with missing values
# default is rows
cleaned_data = data.dropna()
```

- remove the columns with nulls *or*
- remove the rows with nulls
- check the size of reduced dataset

Removing Rows or Columns with Missing Values (cont.)

`pandas.dropna()`

- the default is to drop an entire row/column if there is at least one null
- you can use the the `thresh` integer parameter to indicate the minimum number of nulls that trigger the dropping of the column/row with nulls
- dropping a row with nulls just drops a piece of evidence
- dropping a column with nulls drops an entire "feature" of the data
- simple dropping is meaningful when the amount of rows/columns with nulls is small
- dropping can be substituted or integrated with imputation

Simple Imputation

- `SimpleImputer` replaces missing values with a constant, mean, median, most frequent, pre-defined value.

```
from sklearn.impute import SimpleImputer
# Example dataset with missing values
X = pd.DataFrame({'A':[1,4,7], 'B':['x', None, 'z', 'C':[None, 8, 9]})
numeric_feat=['A','C']
categ_feat = ['B']
# Impute missing values with
# 'mean', 'median', 'most_frequent', 'constant' (specify with fill_value)
imputer_num = SimpleImputer(strategy='mean')
X_imputed[numeric_feat] = imputer.fit_transform(X[numeric_feat])
imputer_categ = SimpleImputer(strategy='constant', fill_value='unknown')
X_imputed[categ_feat] = imputer.fit_transform(X[categ_feat])
```

Integration with ColumnTransformer

Combine imputation strategies for different column types – Examples

- numeric features
  - if the distribution is not skewed, fill nulls with the mean
  - if the distribution is skewed, fill nulls with the median
- discrete features
  - fill nulls with a pre-defined constant, e.g. "unknown"

```
imputer = ColumnTransformer(
        transformers=[
                ("fill-w-median", SimpleImputer(strategy="median"), ["A","B","C"])
                ,("fill-w-mean", SimpleImputer(strategy="mean"), ["D","E"])
                ,("fill-w-constant", SimpleImputer(strategy="constant",
                                fill_value="unknown"), ["F"])
        ]
)
X_imputed = imputer.fit_transform(X)
```

Why do we need type conversion?

- Many algorithms require numeric features
  - categorical features must be transformed into numeric
  - ordinal features must be transformed into numeric, and the order must be preserved
- Classification requires a target with nominal values
  - a numerical target can be discretised
- Discovery of association rules require boolean features
  - a numerical feature can be discretised and transformed int a series of boolean features

Binarization of discrete attributes

Attribute d allowing V values $\Rightarrow$ V binary attributes.

Color
Red
Blue
Green
Orange
Yellow

Color-Red	*Color-Blue*	*Color-Green*	*Color-Orange*	*Color-Yellow*
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Nominal to numeric

- One–Hot–Encoding
  - a feature with V unique values is substituted by V binary features each one corresponding to one of the unique values
  - if object x has value $v$in feature d then the binary feature corresponding to v has `True` for x, all the other binary features have value `False`
  - `True` and `False` are represented as 1 and 0, therefore can be processed by also by procedures working only on numeric data, as is the case for the estimators available in `scikit-learn`
- `sklearn.preprocessing.OneHotEncoder`

link to manual page

Ordinal to numeric

- The ordered sequence is transformed into consecutive integers
  - by default the lexicographic order is assumed
  - The user can specify the proper order of the sequence
  - *awful*, *poor*, *ok*, *good*, *great* $\Rightarrow$ `0,1,2,3,4`
- `sklearn.preprocessing.OrdinalEncoder`

link to manual page

Numeric to binary with threshold

- Not greater than the threshold becomes zero
- Greater than the threshold becomes one
- `sklearn.preprocessing.Binarizer`

link to manual page

Discretization/Reduction of the number of distinct values

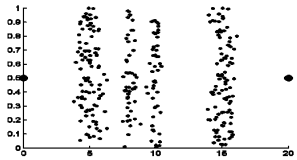
- Some algorithms work better discrete instead of continuous data
- A small number of distinct values can let patterns emerge more clearly
- A small number of distinct values let the algorithms to be less influenced by noise and random effects

Discretization

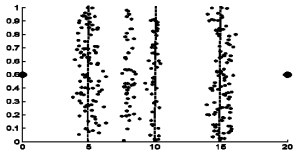
- Continuous ⇒ Discrete
  - thresholds
    - many options
  - binarization ⇒ single threshold
- Discrete with many values ⇒ Discrete with less values
  - guided by domain knowledge

Continuous ⇒ Discrete

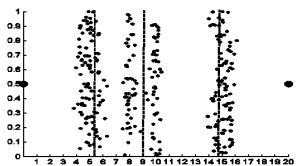
Boundaries on x axis – Unsupervised



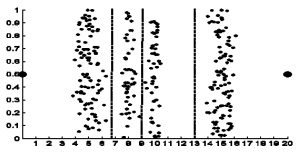
Data



Equal width



Equal frequency



K-means

Numeric to k values

- The numbers are discretised into a sequence of integers 0 to $k-1$
- Several strategies are available
  - {'uniform','quantile','means'}
- `sklearn.preprocessing.KBinsDiscretizer`

link to manual page

Sampling I

- For both preliminary investigation and final data analysis
- Statistician perspective
  - obtaining the entire data set could be impossibile or too expensive
- Data processing perspective
  - processing the entire data set could be too expensive or time consuming

Sampling II

1. using a sample will work almost as well as using the entire data sets, *if the sample is representative*
2. A sample is representative if it has approximately the same property (of interest) as the original set of data

Types of sampling

1. Simple random
   - a single random choice of an object with given probability distribution
2. With replacement
   - repetition of independent extractions of type 1
3. Without replacement
   - repetition of extractions, extracted element is removed from the population
4. Stratified
   - used to split the data set into subsets with homogeneous characteristics
   - the representativity is guaranteed inside each subset
   - typically requested in cross–validation

Sample size

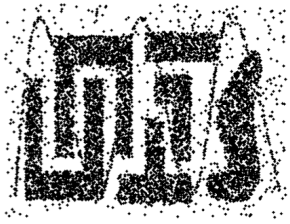
- Statistics provides techniques to assess
  - optimal sample size
  - sample significativity
- Tradeoff between data reduction and precision

Sampling with/without replacement

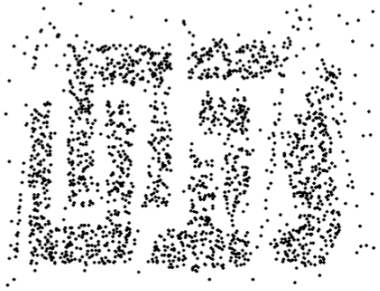
- they are nearly equivalent if sample size is a small fraction of the data set size
- with replacement, in a small population a small subset could be underestimated
- sampling with replacement is
  - much easier to implement
  - much easier to be interpreted from a statistical point of view
    - extractions are statistically independent

Sample size – Example

Loss of information



8000 points



2000 points

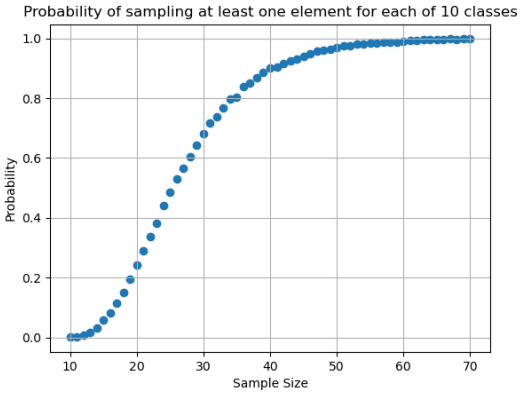
500 points

Sample size – Missing class I[1]

- Probability of sampling at least one element for each class (with replacement)
  - it is independent from the size of the data set!

A B C D E
F G H I J

10 classes



1 It is an instance of the Coupon Collector's Problem

Sample size – Missing class - II

- This aspect becomes relevant, for example, in a supervised dataset with a high number of different values of the target
- If the number data elements is not big enough, it can be difficult to guarantee a stratified partitioning in train/test split or in cross–validation split
- Example:
  - $N = 1000$, $C = 10$, test–set–size $= 300$, cross–validation–folds $= 10$
  - the probability of folds without adequate representation of some classes becomes quite high
- When designing the training processes it is necessary to consider those aspects
  - in the example, one could use only 3 folds in cross–validation

Feature creation

New features can capture more efficiently data characteristics

- Feature extraction
  - pixel picture with a face $\Rightarrow$ eye distance, ...
- Mapping to a new space
  - e.g. signal to frequencies with Fourier transform
- New features
  - e.g. volume and weight to density

Feature engineering, a.k.a. feature creation

- Feature creation is a crucial step in data mining
- It involves transforming raw data into meaningful features that can improve predictive models
- In the financial environment, feature creation plays a vital role in enhancing trading strategies, risk management, and fraud detection

Examples of Feature Creation in Finance I

- **Moving Averages** Calculate the average closing price of a stock over a specific time window (e.g., 10 days, 50 days).
- **Volatility Measures** Compute metrics such as standard deviation or average true range to capture the level of price fluctuations.
- **Relative Strength Index (RSI)** Derive a momentum oscillator indicating overbought or oversold conditions in the market.

Examples of Feature Creation in Finance II

- **Liquidity Ratios** Calculate ratios like bid-ask spread or trading volume to assess market liquidity.
- **Fundamental Analysis Indicators** Include financial metrics such as earnings per share (EPS), price-to-earnings (P/E) ratio, or debt-to-equity ratio.
- **Market Sentiment Analysis** Utilize sentiment scores from news articles, social media, or analyst reports to gauge market sentiment.
- **Technical Analysis Patterns** Identify chart patterns such as head and shoulders, double tops, or flags to predict future price movements.

Feature Creation: closing prices of a stock

```
data = {'Date': ['2022-01-01','2022-01-02','2022-01-03','2022-01-04','2022-01-05']
       ,'Close': [100, 102, 98, 105, 101]}
# Create a DataFrame
df = pd.DataFrame(data)
# Feature 1: Moving Average (10 days)
df['MA_2'] = df['Close'].rolling(window=2).mean()
# Feature 2: Daily Price Change
df['Price_Change'] = df['Close'].diff()
# Feature 3: Daily Percentage Change
df['Pct_Change'] = df['Close'].pct_change()
# Feature 4: Relative Strength Index (RSI)
# Calculation of RSI requires more data points and additional steps,
# so we'll simplify it here for demonstration purposes
df['RSI'] = 100 - (100 / (1 + (df['Close'].\
                              pct_change()).rolling(window=2)\
    .apply(lambda x: x[x > 0].mean() / x[x < 0].mean())))
```

Feature Creation: closing prices of a stock – output

- `rolling()` aggregates a number of consecutive rows specified with `windows`
- `diff()` computes the difference of a feature in consecutive rows
- `pct_change()` computes the percentage of change of a feature in consecutive rows
- the aggregations introduce a number of nulls in the initial rows

Date	Close	MA_2	Price_Change	Pct_Change	RSI
2022-01-01	100	NaN	NaN	NaN	NaN
2022-01-02	102	101.000000	2.000000	0.020000	NaN
2022-01-03	98	100.000000	-4.000000	-0.039216	-104.081633
2022-01-04	105	101.500000	7.000000	0.071429	221.739130
2022-01-05	101	103.000000	-4.000000	-0.038095	214.285714

Examples of Feature Creation in a Housing dataset I

- **Total Area**
  - Calculate the sum of sizes of all rooms in the house.
- **Price per Square Foot**
  - Divide the price of the house by its total area.
- **Age of the House**
  - Calculate the age of each house by subtracting the year it was built from the current year.
- **Neighborhood Median Income**
  - Include data on the median income of households in each neighborhood.

Examples of Feature Creation in a Housing dataset II

- **Distance to City Center**
  - Measure the distance of each house from the city center.
- **Renovation Status**
  - Create a binary feature indicating whether the house has been recently renovated.
- **School District Rating**
  - Include data on the quality of school districts in which the houses are located.
- **Presence of Amenities**
  - Create a categorical feature indicating the presence of amenities such as a swimming pool, garden, garage, etc.

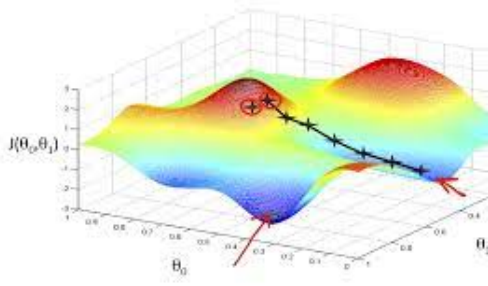
Why Data Transformation

- the features may have different scales
  - this can alterate the results of many learning techniques
  - some machine learning algorithms are sensitive to feature scaling while others are virtually invariant to it
- there can be outliers

Gradient descent

Machine learning algorithms that use *gradient descent* as an optimization technique require data to be scaled

- e.g. linear regression, logistic regression, neural network, etc.
- The presence of feature value X in the formula will affect the step size of the gradient descent
- The difference in ranges of features will cause different step sizes for each feature.
- Similar ranges of the various features ensure that the gradient descent moves smoothly towards the minima and that the steps for gradient descent are updated at the same rate for all the features



Feature transformation

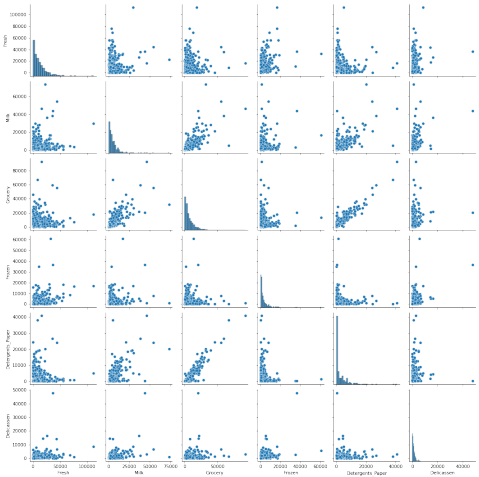
- Map the entire set of values to a new set according to a function
  - $x^k, \log(x), e^x, |x|$
  - in general they change the *distribution of values*
- Standardization: $x \rightarrow \frac{x-\mu}{\sigma}$
  - if the original values have a *gaussian* distribution, the transformed values will have a *standard* gaussian distribution ($\mu = 0, \sigma = 1$)
  - translation and shrinking/stretching, no change in distribution
- MinMax scaling (a.k.a. Rescaling): the domains are mapped to standard ranges

$$x \rightarrow \frac{x - x_{min}}{x_{max} - x_{min}} \quad \text{(0 to 1)} \qquad x \rightarrow \frac{x - \frac{x_{max}+x_{min}}{2}}{\frac{x_{max}-x_{min}}{2}} \quad \text{(-1 to 1)}$$

  - translation and shrinking/stretching, no change in distribution

Feature transformation – Before

Data with skewed distribution

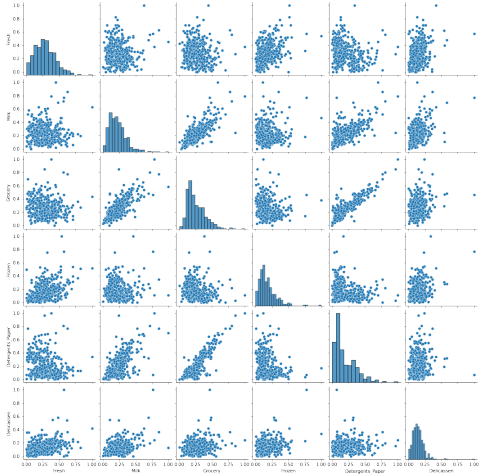


Feature transformation – After

Python code

```
from sklearn.preprocessing
      import PowerTransformer
pt = PowerTransformer(method='box-cox')
X = pd.DataFrame(pt.fit_transform(X0)
                , columns = X0.columns)
```

After the transformation the data are
less skewed



Distance–based algorithms

- KNN, K–Means, SVM, ...
- distances between points are used to determine their similarity

Example

Original data		
Student	**CGPA**	**Salary**
A	3	60
B	3	40
C	4	40
D	4.5	50
E	4.2	52

Scaled data		
Student	**CGPA**	**Salary**
A	-1.18431	1.520013
B	-1.18431	-1.100699
C	0.41612	-1.100699
D	1.21635	0.209657
E	0.736212	0.471728

Distances before and after scaling

$$
\begin{aligned}
distance(A, B) &= \sqrt{(40-60)^2 + (3-3)^2} = 20 \\
distance(B, C) &= \sqrt{(40-40)^2 + (4-3)^2} = 1 \\
distance(A_s, B_s) &= \sqrt{(1.1+1.5)^2 + (1.18-1.18)^2} = 2.6 \\
distance(B_s, C_s) &= \sqrt{(1.1-1.1)^2 + (0.41+1.18)^2} = 1.59
\end{aligned}
$$

Before the scaling the two distances seemed to be very different, due to the a big numeric difference in the `Salary` attribute, now they are comparable

Range–based scaling and standardization

operate on single features

- Range–based scaling stretches/shrinks and translates the range, according to the range of the feature (there are some variants)
  - good when we know that the data are not gaussian, or we do not make any assumption on the distribution
  - the base variant, the MinMax scaler, remaps to $0, 1$
- Standardization subtracts the mean and divides by the standard deviation
  - the resulting distribution has mean *zero* and *unitary* standard deviation
  - good when the distribution is gaussian
  - `StandardScaler`

Range–based scalers in Scikit-Learn

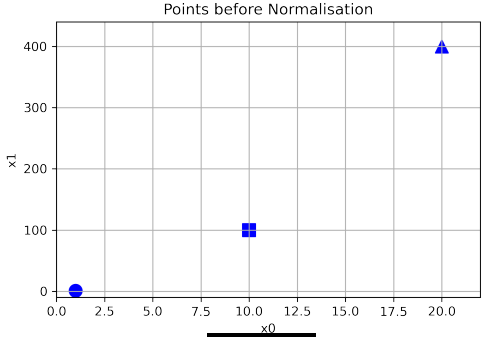
affine transformations: linear transformation plus translation

- `MinMaxScaler` – remaps the feature to $[0, 1]$
- `RobustScaler` – centering and scaling statistics is based on percentiles
  - not influenced by a few number of very large marginal outliers
  - the resulting range of the transformed feature values is larger than the one given by `MinMaxScaler` and `StandardScaler`

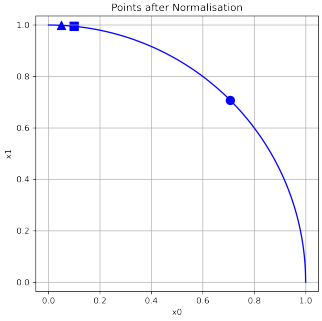
Normalization

- Normalization is mentioned sometimes with different meanings
  - frequently it refers to `MinMaxScaler`
- in `Scikit-learn` the `Normalizer` normalizes each data row to unit norm

Normalizer – Example



X0	X1
1	1
10	100
20	400



X0	X1
0.707107	0.707107
0.099504	0.995037
0.049938	0.998752

Feature rescaling – Summary

Technique	**Advantage**	**Suitable Applications**
Unit Norm	Emphasizes proportions, ignores magnitudes	Cosine similarity, distance-based models
Standardization	Handles outliers, centers data	Models assuming Gaussian distribution
Min-Max Scaling	Scales to bounded ranges	Neural networks, bounded feature spaces

Workflow for feature transformation

1. transform the features as required both for the train and test data
2. fit and optimize the model(s)
3. test
4. possibly, use the original data to plot relevant views (e.g. to plot cluster assignments)

Imbalanced data in classification

- The performance on the minority class (classes) has little impact on standard performance measures
- The optimised model could be less effective on minority class (classes)
- Some estimators allow to weight classes
- Some performance measures allow to take into account the contribution of minority class (classes)

Cost Sensitive learning

Already introduced in *Machine-Learning-classification*

- several classifiers have the parameter `class_weight`
- it changes the cost function to take into account the imbalancing of classes
- in practice it is equivalent to oversampling the minority class, (repeating random examples) in order to produce a balanced training set

Undersampling

- Obtains a balanced training set by randomly reducing the number of examples of the majority class
- Obviously part of the knowledge embedded in the training set is dropped out

Oversampling with SMOTE

Synthetic Minority Oversampling Technique – a type of data augmentation

let the minority class be c_{min}, synthesise new examples of class c_{min}

- choose from the *training set* random example x_r of class c_{min}
- find in the training set the k nearest neighbours of x_r whose class is c_{min}
- choose randomly one of the neighbours, say x_{rn} found above and *create* a new data element chosen randomly from the segment connecting x_r x_{rn} in the feature space $m = r * (x_r + x_{rn})/2$

Theory developed in SMOTE: Synthetic Minority Over-sampling Technique[Bowyer et al.(2011)Bowyer, Chawla, Hall, and Kegelmeyer]

Workflow for *undersampling/oversampling*

- resample the training set
- fit and optimise the estimator
- test the fitted estimator on the test set (untouched)

Why feature selection

Sometimes less is better (by Rohan Rao)

Sometimes:

- It enables the machine learning algorithm to train faster
- It reduces the complexity of a model and makes it easier to interpret
- It improves the accuracy of a model if the right subset is chosen
- It reduces overfitting.

It may be the case that a specific selection action obtain only one of the above effects

Supervised or not?

unsupervised a lot of methods available e.g. for clustering see this:
Feature Selection for Clustering: A Review

- feature transformation techniques, such as PCA, can have the effect of reducing the number of features

supervised consider the relationship between each attribute and the *class*

- Filter methods (i.e. Scheme–Independent Selection)
- Scheme–Dependent Selection
  - Wrapper methods
  - Embedded methods
    have their own built-in feature selection methods (e.g. *Lasso* and *Ridge* regression)

Problems with attributes I

[Witten et al.(2011)Witten, Frank, and Hall] Ch 7 and 11 and also this

the significance of attributes for the purposes of data mining can vary highly

irrelevant alteration they can alter the results of some mining algorithm, in particular in case of insufficient control of overfitting

redundant some attributes can be strongly related to other useful attributes

Problems with attributes II

[Witten et al.(2011)Witten, Frank, and Hall] Ch 7 and 11 and also this

alteration some mining algorithms (e.g. Naive Bayes) are strongly influenced by strong correlations between attributes

Problems with attributes III

[Witten et al.(2011)Witten, Frank, and Hall] Ch 7 and 11 and also this

confounding some attributes can be misleading

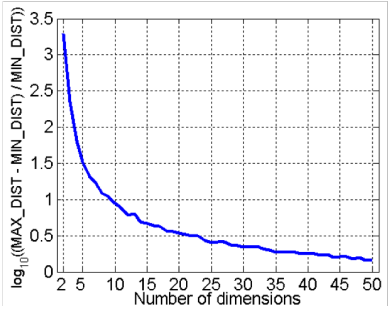
hidden effect on the outcome variable

example in a study on weight gain, physical exercise, age and sex, the sex can be confounding if in the available data the ages of males and females have very different ranges

mixed effect one attribute could be strongly related to the class in 65% of the cases and random in the other cases

The curse of dimensionality

- When dimensionality is very high the occupation of the space becomes very sparse
- Discrimination on the basis of the distance becomes uneffective
- Experiment:
  - random generation of 500 points
  - plot the relative difference between the maximum and the minimum distance between pairs of points



Dimensionality reduction

- Purposes:
  - avoid the *curse of dimensionality*
  - noise reduction
  - reduce time and memory complexity of the mining algorithms
  - visualization
- Techniques:
  - principal component analysis
  - singular values decomposition
  - supervised techniques
  - non–linear techniques

Aggregation

- Combining two or more attributes (or objects) into a single attribute (or object)
- Purpose
  - Data reduction
    - Reduce the number of attributes or objects
  - Change of scale
    - Cities aggregated into regions, states, countries, etc
  - *More stable* data
    - Aggregated data tends to have less variability

Feature subset selection I

A *local* way to reduce dimensionality

- Redundant attributes
  - duplicate most of the information contained in other attributes
    - e.g. price and v.a.t. amount
- Irrelevant attributes
  - do not contain any information useful for analysis
    - e.g. SSN is not relevant to predict wealth

Feature subset selection II

1. Brute force
   - try all possible feature subsets as input to data mining algorithm and measure the effectiveness of the algorithm with the reduced dataset
2. Embedded approach
   - Feature selection occurs naturally as part of the data mining algorithm
     - e.g. decision trees
3. Filter approach
   - Features are selected before data mining algorithm is run
4. Wrapper approaches
   - A data mining algorithm can choose the best set of attributes
     - try as in 1, but without exhaustive search

Filter methods (Scheme–Independent Selection)

- Assessment based on general characteristics of data
- Select the subset of attributes independently from the mining model that will be used
  - e.g. build a decision tree and consider the attributes near the root of the tree, then use the selected attributes for building a classifier with another method
  - e.g. select a subset of attributes that individually correlate to the class, but but have a little intercorrelation[2]

2 See Symmetric Uncertainty for correlation between nominal attributes

Some filter methods

Pearson's Correlation A measure for quantifying linear dependence between two continuous variables X and Y; value from -1 to $+1$

LDA Linear Discriminant Analysis is used to find a linear combination of features that characterizes or separates two or more classes

ANOVA Analysis Of VAriance is similar to LDA except for the fact that it is operated using one or more categorical independent features and one continuous dependent feature

Chi-Square Is a statistical test applied to the groups of categorical features to evaluate the likelihood of correlation or association between them using their frequency distribution

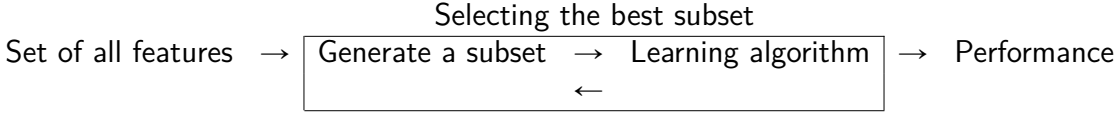
Filter Methods – Synopsis

Feature Response	*Continuous*	*Categorical*
Continuous	Pearson's Correlation	LDA
Categorical	ANOVA	Chi-Square

Set of all features → Selecting the best subset → Learning algorithm → Performance

Wrapper methods

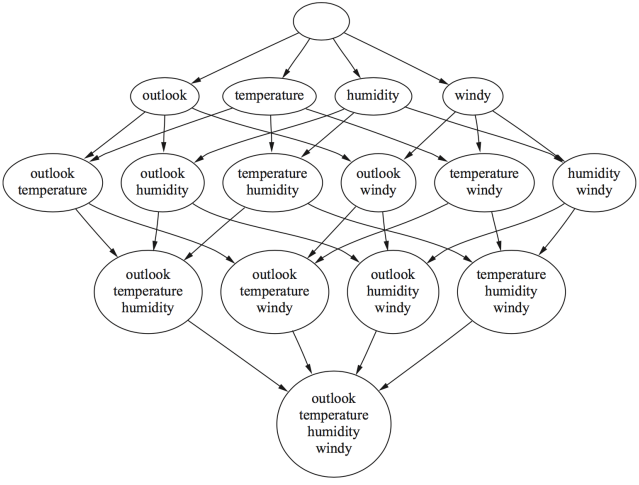
- Try to use a subset of features and train a model using them
- Based on the inferences that we draw from the previous model, we decide to add or remove features from your subset
- The problem is essentially reduced to a search problem



One wrapper method

Search the Attribute Space

- e.g. Weather dataset
- Search greedily the space
- For each subset test the performance of the chosen classification model
- Computation intensive



Difference between Filter and Wrapper methods

- Filter methods measure the relevance of features by their correlation with dependent variable while wrapper methods measure the usefulness of a subset of feature by actually training a model on it
- Filter methods are much faster compared to wrapper methods as they do not involve training the models. On the other hand, wrapper methods are computationally very expensive as well.
- Filter methods use statistical methods for evaluation of a subset of features while wrapper methods use cross validation
- Filter methods might fail to find the best subset of features in many occasions but wrapper methods can always provide the best subset of features
- Using the subset of features from the wrapper methods make the model more prone to overfitting as compared to using subset of features from the filter methods

Correlation of quantitative data (Pearson's)

OPTIONAL

Measure of the linear relationship between a pair of attributes

- Standardize the values
- For two given attributes p and q, consider as vectors the ordered lists of the values over all the data records
- Compute the dot product of the vectors

$$\mathbf{p} = [p_1, \ldots, p_N] \xrightarrow{standardize} \mathbf{p}'$$

$$\mathbf{q} = [q_1, \ldots, q_N] \xrightarrow{standardize} \mathbf{q}'$$

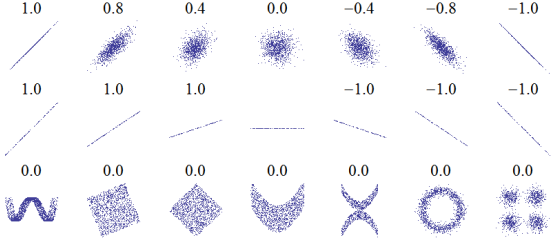
$$\mathrm{corr}(p, q) = \mathbf{p}' \bullet \mathbf{q}'$$

There is an alternative definition based on covariance and variances

Correlation – Discussion

OPTIONAL

- Independent variables ⇒ correlation is zero
  - the inverse is not valid *in general*
- Correlation zero ⇒ absence of *linear relationship* between the variables
- Positive values imply positive linear relationship



From "Correlation and Dependence" on Wikipedia

Correlation between nominal attributes OPTIONAL

Symmetric Uncertainty [Witten et al.(2011)Witten, Frank, and Hall]

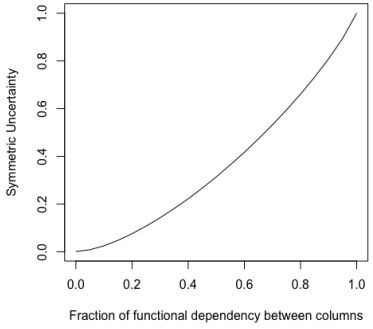
$$U(p,q) = 2\frac{H(p) + H(q) - H(p,q)}{H(p) + H(q)}$$

- where $H()$ is the entropy of a single attribute while $H(,)$ is the joint entropy, computed from the joint probabilities
- is always between 0 and 1

Correlation between nominal attributes – Experiment

OPTIONAL

- Behavior of SU for two independent uniformly distributed discrete attributes, say p and q
  - in a variable fraction of records the value of p is copied to q
- from complete independence (left) to complete biunivocal correspondence (right)
- when there is independence, the joint entropy is the sum of the individual entropies, and SU is zero
- when there is complete correspondence, the individual entropies and the joint entropy are equal and SU is one



Role of Correlation in Feature Selection

- Identifying Redundant Features
  - Features highly correlated with each other contain overlapping information
  - Retain one feature from such groups to reduce dimensionality
- Identifying Relevant Features
  - High correlation with the target variable helps identify features with high predictive power
- Caveat: low Pearson's correlation between a feature and the target can sometimes hide a non–linear correlation, therefore the mere use of low Pearson's correlation for feature filtering can be dangerous

Computational Complexity of Correlation–based Feature Selection

OPTIONAL

- Calculating Correlation
  - For D features and N samples, computing $r(f_i, y)$ for all features is:

$$O(N \cdot D)$$

- Pairwise Feature Correlation
  - Requires computing correlations for $\binom{D}{2}$ feature pairs:

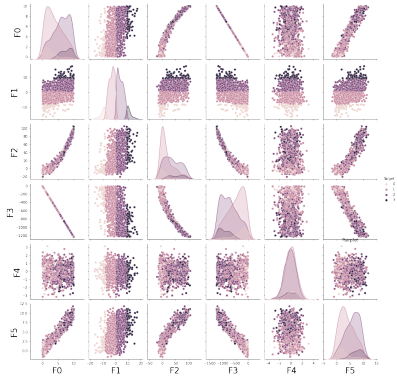
$$O(D^2 \cdot N)$$

- Overall Complexity
  - Linear in the number of samples, quadratic in the number of features.

Advantages and Limitations

☺ Simple and interpretable method for feature selection.
☺ Reduces overfitting by removing irrelevant/redundant features.
☹ Only considers linear relationships between variables.
☹ Fails to detect nonlinear dependencies.

Comparing pairplot and correlation heathmap



Pairplot

Correlation Matrix

Summary on correlation

- Correlation is a simple and effective tool for feature selection.
- Helps reduce dimensionality by removing redundant and irrelevant features.
- Should be complemented with other techniques to handle nonlinear relationships.

Dimensionality reduction

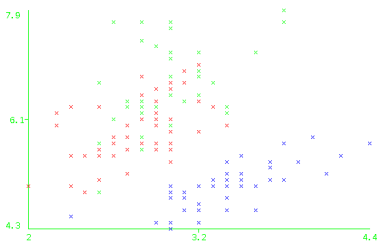
Instead of considering which subset of attributes is to be ignored it is possible to map the dataset into a new space with fewer attributes

PCA Principal Component Analysis

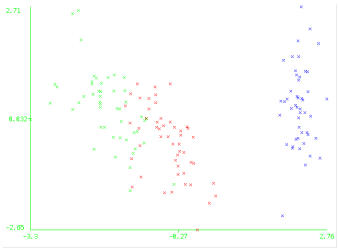
PCA

- Find a new (ordered) set of dimensions that better captures the variability of the data
  - the first one captures most of the variability
  - the second one is orthogonal to the first one and captures most of the remaining variability
  - ...
- The fraction of variance in data captured by each new variable is measured
- A small number of new variables can capture most of the variability

Iris dataset

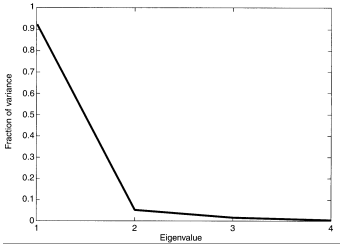


Sepalwidth/Sepallength Plot



PCA - first two components

95% variance



Fraction of variance for each principal component

A few mathematical details

- Covariance matrix (positive semidefinite)
- Eigenvalue analysis
- Eigenvalues are positive and can be sorted in decreasing order
- Eigenvectors are sorted according to the eigenvalue order

Overview of PCA's Computational Complexity

- PCA involves several steps, each contributing to the overall complexity:
  - Data centering.
  - Covariance matrix computation.
  - Eigenvalue decomposition or Singular Value Decomposition (SVD).
- Complexity depends on:
  - Number of samples N
  - Number of features D
  - Method used (e.g., full decomposition vs truncated SVD).

Key Steps in PCA

- Data Centering:
  - Subtract the mean of each feature to center the data around the origin.
  - Complexity: $\mathcal{O}(ND)$, where N is the number of samples and D is the number of features.
- Covariance Matrix Computation:
  - If $D \leqslant N$: Compute the $D \times D$ covariance matrix.
    - Complexity: $\mathcal{O}(ND^2)$.
  - If $D > N$: Compute the $N \times N$ covariance matrix (dual PCA approach).
    - Complexity: $\mathcal{O}(N^2D)$.
- Eigenvalue Decomposition or SVD:
  - Eigenvalue decomposition of a $D \times D$ covariance matrix:
    - Complexity: $\mathcal{O}(D^3)$.
  - SVD of the $N \times D$ centered data matrix:
    - Complexity: $\mathcal{O}(N^2D)$ if $N \leqslant D$.
    - Complexity: $\mathcal{O}(ND^2)$ if $D \leqslant N$.

Overall Computational Complexity

- Small feature set ($d \ll n$):
  - Dominated by SVD or eigenvalue decomposition of the covariance matrix.
  - Overall complexity: $\mathcal{O}(nd^2 + d^3)$.
- Large feature set ($d \gg n$):
  - Dual PCA approach (eigenvalue decomposition of $n \times n$ covariance matrix).
  - Overall complexity: $\mathcal{O}(n^2 d + n^3)$.

Optimizations for PCA

OPTIONAL

- Truncated SVD:
  - Computes only the top k singular values/vectors.
  - Complexity: $\mathcal{O}(kND)$, where k is the number of principal components.
  - Particularly useful when $k \ll \min(N, D)$.
- Sparse Data:
  - Sparse matrix methods reduce complexity.
  - Exploits data sparsity for efficient computation.

Practical Implications

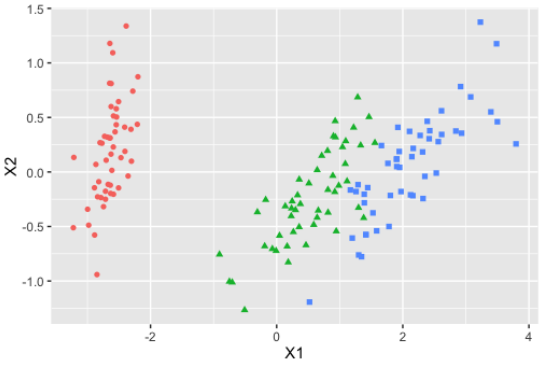
OPTIONAL

- PCA is computationally expensive for large N and D, especially with full SVD.
- Alternatives:
  - Dimensionality reduction techniques (e.g., random projections).
  - Iterative methods (e.g., power iteration) to approximate principal components.
- Use cases:
  - Small datasets: Full PCA is feasible.
  - Large datasets: Approximation methods are recommended.

MDS – Multi–Dimensional Scaling

A presentation technique

- Starting from the distances among the elements of the dataset
- Fits the projection of the elements into a m dimensional space in such a way that the distances among the elements are preserved
- Versions for non–metric and metric spaces



2D scaling for the Iris dataset

Introduction to Multi-Dimensional Scaling (MDS)

- MDS is a technique used to visualize high-dimensional data in a low-dimensional space.
- Goals:
  - Preserve pairwise distances or dissimilarities between points.
  - Provide a geometric representation of data in 2D or 3D for interpretation.
- Applications:
  - Exploratory data analysis.
  - Visualizing relationships in datasets like genetics, psychology, or marketing.

Problem Setup

- Given:
  - A set of n objects or points.
  - A dissimilarity matrix $\Delta = (\delta_{ij})$, where δ_{ij} is the distance or dissimilarity between points i and j.
- Objective:
  - Find a configuration of points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ in a low-dimensional space $\mathbb{R}^p$ ($p \ll n$) such that:

$$d_{ij} \approx \delta_{ij}, \quad \forall i, j$$

where d_{ij} is the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ in $\mathbb{R}^p$.

Mathematical Formulation

- Types of MDS:
  - Classical MDS:
    - Assumes δ_{ij} are Euclidean distances.
  - Non-metric MDS:
    - Focuses on preserving the rank order of distances.
- Cost Function:
  - Minimize a stress function S:

$$S(\mathbf{X}) = \sum_{i<j} w_{ij} \left(d_{ij} - \delta_{ij}\right)^2,$$

    where:
    - w_{ij} are weights (often set to 1).
    - $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$.
- Optimization:
  - Minimize $S(\mathbf{X})$ over the coordinates $\mathbf{X}$.

Classical MDS and Eigenvalue Decomposition

- Classical MDS Steps:
  - Start with a distance matrix Δ.
  - Convert distances to a similarity matrix:

$$\mathbf{B} = -\frac{1}{2}\mathbf{H}\Delta^2\mathbf{H},$$

    where $\mathbf{H} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$ centers the data.
  - Perform eigenvalue decomposition on $\mathbf{B}$:

$$\mathbf{B} = \mathbf{Q}\Lambda\mathbf{Q}^\top,$$

    where $\mathbf{Q}$ contains eigenvectors and Λ is diagonal with eigenvalues.
  - Obtain low-dimensional embedding:

$$\mathbf{X} = \mathbf{Q}_p\Lambda_p^{1/2},$$

    where $\mathbf{Q}_p$ and Λ_p correspond to the largest p eigenvalues.

Computational Complexity

- Key Steps and Their Complexity:
  - Computing pairwise distances: $\mathcal{O}(n^2)$ for n objects.
  - Eigenvalue decomposition in classical MDS:
    - $\mathcal{O}(n^3)$ for dense matrices.
- Scalability:
  - Classical MDS is computationally expensive for large n.
  - Approximation methods, such as Landmark MDS, can reduce complexity.

Advantages and Limitations

- Advantages:
  - Provides an interpretable low-dimensional representation.
  - Works well when distances are approximately Euclidean.
- Limitations:
  - Sensitive to noise and outliers in the distance matrix.
  - Computationally intensive for large datasets.

The *Scikit-learn* solution for feature selection

General structure

The main methods (there are more, somewhat different for the various estimators)

- `.fit`
  - Learn empirical variances from X
- `.fit_transform`
  - Fit to data, then transform it
- `.transform`
  - Reduce X to the selected features
- The main argument is X, the dataset

The baseline estimator

- `VarianceThreshold` = removing features with low variance
  - unsupervised
  - Example:
    - dataset with binary attributes
    - we decide to eliminate the features with a proportion 80-20 or more, $p = .8$ or more
    - a *bernoullian* experiment has variance $p * (1 - p)$
    - the threshold will be $.08 * (1 - .8) = .16$

Univariate feature selection

- Select the best set of features based on univariate statistical tests
- Consider the *original set of features* and the *target*
- For each feature, return a score and a pvalue
- Among the selection methods:
  - `SelectKBest`
    - removes all but the k highest scoring features
  - `SelectPercentile`
    - removes all but a user-specified highest scoring percentage of features

Interpreting the Output of SelectKBest Estimator OPTIONAL

- **'score_' (Test Statistic)** – Several score functions available
  - **ANOVA F-value (for continuous target variables)**
    - Measures the F-statistic, which indicates the variance between groups compared to within-group variance.
    - A higher F-statistic suggests a stronger relationship with the target.
  - **Chi-Squared (for categorical target variables)**
    - Measures the independence between each feature and the target.
    - A higher score indicates a stronger association between the feature and the target.
  - **Mutual Information**
    - Measures how much information a feature contains about the target.
    - A higher score indicates more informative features.

Understanding p-values

- **p-values**
  - It is the *probability that the null hypothesis is acceptable*
    - the *null hypothesis* is that there is no relationship between the feature and the target
  - Measures the statistical significance of the test statistic.
  - Low p-value (typically < 0.05) indicates statistical significance.
  - High p-value (typically > 0.05) suggests the feature is not significant.
- **Interpreting p-values**
  - Low p-value: Strong relationship between the feature and the target.
  - High p-value: Weak or no significant relationship with the target.

Example Interpretation

- **Example 1: ANOVA F-value**
  - **Score:** 100, **p-value:** 0.001
    - High F-statistic indicates strong discriminatory power.
    - Low p-value suggests statistical significance.
- **Example 2: Chi-Squared**
  - **Score:** 0.5, **p-value:** 0.5
    - Low score indicates weak discriminatory power.
    - High p-value suggests no significant relationship with the target.

Algorithm for SelectKBest

Algorithm SelectKBest Feature Selection

1: **Input:** X (features), y (target)
2: **Output:** Selected features
3: Compute scores and p-values for each feature
4: **for** each feature i in X **do**
5: **if** score of feature i is high and p-value is low **then**
6: Select feature i
7: **else**
8: Discard feature i

Summary of Interpreting SelectKBest Output

- **High scores and low p-values:** Features with high importance and statistical significance.
- **Low scores and high p-values:** Features with weak relationships to the target.
- **Choosing Features:** Combine both score and p-value to select the best features for the model.

Score functions

Are used by the feature selector to evaluate how much a feature is useful to predict the target

- `mutual_info_classif` computes the Mutual Information, which is a generalisation of the *Information Gain*
- `f_classif`: Fisher test with ANOVA (analysis of variance)

Recursive Feature Elimination - RFE

click to see the manual page

Feature ranking with recursive feature elimination

- Uses an external estimator to assign weights to features
- Considers smaller and smaller sets of features
- The estimator is trained on the initial set of features and the importance of each feature is obtained
- The least important features are pruned
- Stops when the desired number of features is reached

Recursive Feature Elimination (RFE)

- Purpose:
  - To identify the best subset of features that contribute most to the model's prediction.
- Steps:
  - Rank features by importance.
  - Eliminate the least important features.
  - Repeat until the desired number of features is reached.

Steps in Recursive Feature Elimination

RFE involves the following steps:

- Step 1: Train a model on all features
  - The model is trained on all features of the dataset.
  - Feature importance is evaluated (e.g., using model coefficients or feature importance scores).
- Step 2: Rank features by importance
  - Based on the trained model, rank all features by their importance.
  - Features with lower importance are considered for removal.
- Step 3: Remove the least important feature(s)
  - Eliminate one or more features with the lowest importance.
  - The model is retrained with the remaining features.
- Step 4: Repeat steps 1–3 until the desired number of features is reached.

Recursive Feature Elimination Algorithm

Algorithm Recursive Feature Elimination (RFE)

1: **Input:** Dataset X, Target y, Model M, k features to select
2: **Output:** Subset of selected features S
3: Initialize feature set $F \leftarrow \{\text{all features in } X\}$
4: **while** the number of features in F is greater than k **do**
5: Train model M on the current feature set F
6: Evaluate feature importance (e.g., using coefficients or feature importances)
7: Remove the least important feature(s) from F
8: Return the selected features S

Example of RFE Application

Let's consider an example using the Iris dataset and the logistic regression model.

- Step 1: Load the dataset
  - The Iris dataset has 150 samples and 4 features.
- Step 2: Apply RFE with Logistic Regression
  - Using 'sklearn.feature_selection.RFE' to select the best 2 features.
- Step 3: Evaluate the performance
  - Train the model with the selected features and evaluate accuracy.

Python Code Example for RFE

.

```python
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE

# Load the dataset
data = load_iris()
X = data.data
y = data.target

# Initialize the model
model = LogisticRegression(max_iter=200)

# Initialize the RFE selector
selector = RFE(model, n_features_to_select=2)

# Fit the RFE model
selector.fit(X, y)

# Get the selected features
print("Selected features:", selector.support_)
print("Ranking of features:", selector.ranking_)
```

Advantages and Limitations of RFE

- Advantages:
  - Provides a clear method for selecting the most important features.
  - Can be used with any machine learning model.
  - Reduces overfitting by eliminating irrelevant features.
- Limitations:
  - Computationally expensive, especially for large datasets.
  - May not be effective when feature importance is not clearly defined.
  - Requires a large number of iterations for feature selection.

Bibliography

- Kevin W. Bowyer, Nitesh V. Chawla, Lawrence O. Hall, and W. Philip Kegelmeyer.
  SMOTE: synthetic minority over-sampling technique.
  CoRR, abs/1106.1813, 2011.
  URL http://arxiv.org/abs/1106.1813.

- Ian H. Witten, Eibe Frank, and Mark Hall.
  Data Mining – Practical Machine Learning Tools and Techniques.
  Morgan Kaufman, 2011.
  ISBN 978-0-12-374856-0.