Hands-on Machine Learning with sklearn, numpy, matplotlib, pandas, keras and TensorFlow 2

Table of contents

[1 Prerequisites 8](#_Toc72192938)

[1.1 Python 8](#_Toc72192939)

[1.1.1 Data Structures 8](#_Toc72192940)

[1.1.2 Functions and lambdas 9](#_Toc72192941)

[1.1.3 Error handling 9](#_Toc72192942)

[1.1.4 String operations 10](#_Toc72192943)

[1.1.5 IO 10](#_Toc72192944)

[1.1.6 Classes 11](#_Toc72192945)

[1.1.7 Pickle – serialization framework – 12](#_Toc72192946)

[1.1.8 Generator, yield, async-await and coroutines 12](#_Toc72192947)

[1.1.9 Threading 12](#_Toc72192948)

[1.2 Numpy 12](#_Toc72192949)

[1.2.1 Arrays (vectors and matrices) 12](#_Toc72192950)

[1.2.2 Shaping 13](#_Toc72192951)

[1.2.3 Stacking 13](#_Toc72192952)

[1.2.4 Linalg 13](#_Toc72192953)

[1.2.5 Miscellaneous (random, linspace, math) 13](#_Toc72192954)

[1.3 Pandas 14](#_Toc72192955)

[1.3.1 Series 14](#_Toc72192956)

[1.3.2 DataFrame 14](#_Toc72192957)

[1.4 Matplotlib 16](#_Toc72192958)

[1.4.1 Lines and 2D functions 16](#_Toc72192959)

[1.4.2 Multiple plots on a figure 17](#_Toc72192960)

[1.4.3 Scatter plots, histograms and images 18](#_Toc72192961)

[1.4.4 Texts and Legends 19](#_Toc72192962)

[1.4.5 ... 19](#_Toc72192963)

[1.5 Jupyter notebook 19](#_Toc72192964)

[1.6 R 20](#_Toc72192965)

[1.6.1 Attributes, Basic types, NA and NaN 20](#_Toc72192966)

[1.6.2 Complex types 20](#_Toc72192967)

[1.6.3 Subsetting, indexing 22](#_Toc72192968)

[1.6.4 Date & time 23](#_Toc72192969)

[1.6.5 Reading data, Files 24](#_Toc72192970)

[1.6.6 Control structures 24](#_Toc72192971)

[1.6.7 Functions 25](#_Toc72192972)

[1.6.8 Loop functions (apply-s) 25](#_Toc72192973)

[1.6.9 Statistics 26](#_Toc72192974)

[1.6.10 Misc 26](#_Toc72192975)

[1.7 SQL (Microsoft Server SQL) 27](#_Toc72192976)

[1.7.1 Data types 28](#_Toc72192977)

[1.7.2 Create and alter schemes 28](#_Toc72192978)

[1.7.3 Insert rows into table 28](#_Toc72192979)

[1.7.4 Delete rows from table 29](#_Toc72192980)

[1.7.5 Queries 29](#_Toc72192981)

[1.7.6 Joins 29](#_Toc72192982)

[1.7.7 Data types 29](#_Toc72192983)

[1.7.8 Data types 29](#_Toc72192984)

[1.8 Docker 29](#_Toc72192985)

[1.8.1 Container manipulation 29](#_Toc72192986)

[1.8.2 Creating images (Dockerfile) 30](#_Toc72192987)

[1.8.3 Docker-compose (docker-compose.yml) 30](#_Toc72192988)

[1.8.4 Docker volume 31](#_Toc72192989)

[1.8.5 Docker 31](#_Toc72192990)

[1.9 LaTeX 31](#_Toc72192991)

[1.9.1 Symbols 31](#_Toc72192992)

[1.9.2 Formatting 33](#_Toc72192993)

[2 Scikit-Learn and book abstract 34](#_Toc72192994)

[2.1 Types of Machine Learning systems 34](#_Toc72192995)

[2.1.1 Supervised / Unsupervised / Semisupervised / Reinforcement 34](#_Toc72192996)

[2.1.2 Batch / Online 34](#_Toc72192997)

[2.1.3 Instance-based / Model-based 34](#_Toc72192998)

[2.2 Challenges of Machine Learning 34](#_Toc72192999)

[2.3 Testing and Validating 35](#_Toc72193000)

[2.4 Bias and variance 35](#_Toc72193001)

[2.5 Scaling 36](#_Toc72193002)

[2.6 Encoding 36](#_Toc72193003)

[2.6.1 Label encoding 36](#_Toc72193004)

[2.6.2 One hot encoding 36](#_Toc72193005)

[2.6.3 Ordinal encoding 36](#_Toc72193006)

[2.7 Transformation pipelines 37](#_Toc72193007)

[2.8 Hyperparameter tuning 37](#_Toc72193008)

[2.8.1 Grid search 37](#_Toc72193009)

[2.8.2 Randomized search 38](#_Toc72193010)

[2.9 Datasets 38](#_Toc72193011)

[2.10 Test-train split and cross validation 38](#_Toc72193012)

[2.11 Clustering 38](#_Toc72193013)

[2.11.1 Agglomerative Clustering 38](#_Toc72193014)

[2.11.2 K-Means Clustering 39](#_Toc72193015)

[2.11.3 DBSCAN (Density-Based) Clustering 41](#_Toc72193016)

[3 Classification 43](#_Toc72193017)

[3.1 Logistic Regression 43](#_Toc72193018)

[3.1.1 Softmax / Multinomial logistic regression 44](#_Toc72193019)

[3.2 Naive Bayes 45](#_Toc72193020)

[3.2.1 Laplace smoothing 46](#_Toc72193021)

[3.3 Support Vector Machine 47](#_Toc72193022)

[3.4 Decision Tree 47](#_Toc72193023)

[3.5 Random Forest 47](#_Toc72193024)

[3.6 Confusion Matrix and ROC Curve 47](#_Toc72193025)

[3.6.1 Rand index 48](#_Toc72193026)

[3.6.2 Jaccard index 48](#_Toc72193027)

[3.6.3 Precision 48](#_Toc72193028)

[3.6.4 Recall 48](#_Toc72193029)

[3.6.5 F1 Score and Precision-Recall tradeoff 49](#_Toc72193030)

[3.7 Multiclass Classification 51](#_Toc72193031)

[3.7.1 One versus All (One versus Rest) 51](#_Toc72193032)

[3.7.2 One versus One 51](#_Toc72193033)

[3.8 Egyéb 51](#_Toc72193034)

[4 Regression 52](#_Toc72193035)

[4.1 Linear Regression 52](#_Toc72193036)

[4.1.1 Ridge regression (L2 regularization) 53](#_Toc72193037)

[4.1.2 Lasso regression (L1 regularization) 53](#_Toc72193038)

[4.1.3 Elasticnet 54](#_Toc72193039)

[4.2 Locally Weighted Linear Regression 54](#_Toc72193040)

[4.3 Polynomial Regression 55](#_Toc72193041)

[4.4 Support Vector Machine 56](#_Toc72193042)

[4.5 Support Vector Machine 56](#_Toc72193043)

[4.6 Support Vector Machine 56](#_Toc72193044)

[4.7 Support Vector Machine 56](#_Toc72193045)

[5 Statistics 57](#_Toc72193046)

[5.1 Permutation, Variation, Combination 57](#_Toc72193047)

[5.2 Probability theory 57](#_Toc72193048)

[5.3 CDF, PDF, PMF 58](#_Toc72193049)

[5.4 Expected value, Variance, Standard deviation 58](#_Toc72193050)

[5.5 E(X) and V(X) of random length sum of iid variables 60](#_Toc72193051)

[5.6 Moments, Skewness, Kurtosis 60](#_Toc72193052)

[5.7 Median, Mode, Quantile (percentile) 61](#_Toc72193053)

[5.8 Multidimension distributions, Covariance, Correlation 61](#_Toc72193054)

[5.9 Popular discrete distributions 63](#_Toc72193055)

[5.9.1 Indicator function (aka characteristic function, Bernoulli distr.) 63](#_Toc72193056)

[5.9.2 Binomial distribution 63](#_Toc72193057)

[5.9.3 Poisson distribution 64](#_Toc72193058)

[5.9.4 Geometric distribution 64](#_Toc72193059)

[5.9.5 Hypergeometric distribution 65](#_Toc72193060)

[5.10 Popular continous distributions 65](#_Toc72193061)

[5.10.1 Uniform distribution 65](#_Toc72193062)

[5.10.2 Exponential distribution 65](#_Toc72193063)

[5.10.3 Normal distribution 66](#_Toc72193064)

[5.10.4 distribution (Chi-square distribution) 66](#_Toc72193065)

[5.10.5 Student distribution (t distribution) 67](#_Toc72193066)

[5.10.6 Fisher distribution (F distribution) 67](#_Toc72193067)

[5.11 Limit theorems 67](#_Toc72193068)

[5.11.1 Law of large numbers 67](#_Toc72193069)

[5.11.2 de Moivre-Laplace 68](#_Toc72193070)

[5.11.3 Central limit (CLT) 68](#_Toc72193071)

[5.11.4 Hoeffding’s inequality 68](#_Toc72193072)

[5.12 Fundamental concepts of statistics 68](#_Toc72193073)

[5.13 Parameter estimation 70](#_Toc72193074)

[5.13.1 Point estimation 71](#_Toc72193075)

[5.13.2 Interval estimation 72](#_Toc72193076)

[5.14 Hypothesis testing 73](#_Toc72193077)

[5.14.1 Z-test (z-próba, u-próba) 75](#_Toc72193078)

[5.14.2 Student’s t-test 76](#_Toc72193079)

[5.14.3 F-test (Fisher-test) 77](#_Toc72193080)

[5.14.4 Welch’s t-test 78](#_Toc72193081)

[5.14.5 Non-parametric tests introduction, common problems 78](#_Toc72193082)

[5.14.6 Kolmogorov-Smirnov test 79](#_Toc72193083)

[5.14.7 Wilcoxon signed-rank test 79](#_Toc72193084)

[5.14.8 Mann-Whitney U test 81](#_Toc72193085)

[5.14.9 Chi-square test 81](#_Toc72193086)

[5.14.10 ANOVA (Analysis of variance) 84](#_Toc72193087)

[5.14.11 Bartlett-test 85](#_Toc72193088)

[5.14.12 Friedman-test 86](#_Toc72193089)

[5.14.13 Kruskal-Wallis test 87](#_Toc72193090)

[5.14.14 Exact tests introduction 87](#_Toc72193091)

[5.14.15 Fisher’s exact test 88](#_Toc72193092)

[5.14.16 Binomial test 88](#_Toc72193093)

[5.15 Regression Analysis 88](#_Toc72193094)

[5.15.1 Multivariable linear regression 93](#_Toc72193095)

[5.15.2 Model building 95](#_Toc72193096)

[5.15.3 Outlier detection 97](#_Toc72193097)

[5.15.4 Binary logistic regression 97](#_Toc72193098)

[5.16 Data reduction 97](#_Toc72193099)

[5.16.1 Factor analysis 97](#_Toc72193100)

[5.16.2 PCA (principal component analysis) 98](#_Toc72193101)

[5.16.3 MDS (multi dimensional scaling) 99](#_Toc72193102)

[5.16.4 Discrimininant analysis 100](#_Toc72193103)

[5.17 Time series data 100](#_Toc72193104)

[5.17.1 Models for time series 101](#_Toc72193105)

[6 Neural Networks 105](#_Toc72193106)

[6.1 Forward and Backpropagation 105](#_Toc72193107)

[6.1.1 Math 105](#_Toc72193108)

[6.1.2 Forward propagation on XOR N.N. 105](#_Toc72193109)

[6.1.3 Backpropagation on XOR N.N. 106](#_Toc72193110)

[6.2 Egyéb 107](#_Toc72193111)

[7 Web scraping 108](#_Toc72193112)

[7.1 BeautifulSoup 108](#_Toc72193113)

[8 Math auxiliary 109](#_Toc72193114)

[8.1 Eigenvalues and eigenvectors 109](#_Toc72193115)

[8.2 Rank of matrix 109](#_Toc72193116)

[8.3 Kernel and image 109](#_Toc72193117)

[9 Standard normal CDF () chart. 111](#_Toc72193118)

[10 Student-t CDF chart 112](#_Toc72193119)

[11 CDF charts in R 113](#_Toc72193120)

[12 114](#_Toc72193121)

# Prerequisites

The book assumes basic programming knowledge as well familiarity with the most often used scientific libraries such as numpy, pandas and matplotlib.

## Python

Modern programming language, where classes passed as *reference copy* and primitives passed by value.

*Reference copy:* Assigning new objects to this won’t affect the original object’s state. Moreover, modifying the object is only feasible by calling operations (add,pop etc.) on this copy.

### Data Structures

#### List

lst = [1, 2, 3]

lst.append(4) #4 to the end

lst.insert(0, 0) #0 to the beginning

lst.remove(obj)

lst.pop(index) #remove element

lst.copy()

lst.extend(lst\_other)

primes = list(filter(lambda num: num.is\_prime, numbers))

#method called list comprehension:

primes = [num.value **for** num **in** numbers **if** num.is\_prime]

lst[-1] gives you the last element

[\*range(a,b+1,s)] every s-th number starting from a ending with b

#### Tuple

Similar to list but unchangeable during runtime.

tpl = (1, 2, 3)

zip() function for pairing lists/tuples into a single tuple

#### Set (unordered)

st = {1, 2, 3}

st.add(4)

st.update(list)

st.remove(obj) #will raise Error if not found

st.discard(obj) #no Error thrown

st3 = st.union(st2)

#### Dictionary

dct = { ”a” : 1, ”b”: 2, ”c” : 3 }

dct[”d”] = 4

dct.pop(”c”)

#no error thrown when key doesnt exists, instead it gets #initialized with 0

**from** collections **import** defaultdict  
dct = defaultdict(int)

**for** key **in** dct:  
 **pass  
  
for** value **in** dct.values():  
 **pass  
  
for** (key, value) **in** dct.items():  
 **pass**

### Functions and lambdas

**def** func(p1, p2, \*args, \*\*kwargs):  
 print(args[0], args[1], kwargs[**"kwparam"**])

#will print 3 4 cc  
  
**def** main():  
 func(1, 2, 3, 4, kwparam=**"cc"**)

lbd = **lambda** x : np.exp(x)

### Error handling

**try**:  
 **raise** NotImplementedError(**"test"**)  
**except** NotImplementedError **as** err:  
 print(err)  
**except**:  
 print(**"wont be printed"**)

### String operations

#### String interpolation

n = **"john"**a = 30  
str1 = **"name: {} age: {:0.2f}"**.format(n, a)  
str2 = **"name: {name} age: {age}"**.format(name=n, age=a)

#### ...

### IO

IO modes:

|  |  |
| --- | --- |
| r | reading  raises error if file’s missing |
| a | append  creates the missing file or writes to the end |
| w | writing  create the missing file or overwrites |
| x | create  error if file exists |

|  |  |
| --- | --- |
| t | text |
| b | binary |

file = open(path, mode) #”rt” ”xb” etc.

for line in file:

**pass**

file.close()

file.write(...)

RAII for IO (will automatically close, even on raised errors):

**with** open(**"path"**, **"mode"**) **as** f:  
 **pass**

### Classes

#### Defining a class and achieve inheritance

**class** BASE:  
 **def** \_\_init\_\_(self, x):  
 self.x = x  
 **def** GetX(self):  
 **return** self.x  
  
**class** DERIVED(BASE):  
 **def** \_\_init\_\_(self, x, y):  
 super().\_\_init\_\_(x) #BASE.\_\_initself.y = y  
 **def** GetY(self):  
 **return** self.y  
  
**def** main():  
 d = DERIVED(10, 20)  
 print(d.GetX())  
 print(d.GetY())  
  
main()

#### Static variables

**class** C:  
 static = **"ABC"  
  
 def** \_\_init\_\_(self):  
 **pass**

#### Class and Static methods

**class** C:  
 static = **"ABC"  
  
 def** \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
  
 @classmethod  
 **def** John(cls):  
 **return** cls(**"John"**, 32)  
  
 @classmethod  
 **def** Eva(cls):  
 **return** cls(**"Eva"**, 22)  
  
 @staticmethod  
 **def** set\_static(val):  
 C.static = val  
  
**def** main():  
 c\_john = C.John()

**@classmethod** decorator is used for returning objects constructed by a set of predefined parameters. Also, it is the most pythonic way to achive multiple constructors as you can create only one \_\_init\_\_ definition.

**@staticmethod** decorator marks the function as a static one.

#### ...

### Pickle – serialization framework –

**if** os.path.exists(**"ser.data"**):  
 obj = pickle.load(open(**"ser.data"**, **"rb"**))  
**else**:  
 obj = **None** *# some long processing task..* pickle.dump(obj, open(**"ser.data"**, **"wb"**))

### Generator, yield, async-await and coroutines

### Threading

## Numpy

Math library with operations faster than standard library.

**import** numpy **as** np

### Arrays (vectors and matrices)

np.empty(5)

np.zeros((4,4))

np.full((4,4), np.pi)

np.empty((0, 2), int) #ideal for loops

np.array([[...], [...], ..., [...]])

np.arange(lower\_bound\_inc, upper\_bound\_exc, step=1) #iota

np.random.rand(4) #4 long vector with rands between 0,1

np.random.randn(4) #4 long vector with N(0,1) distribution

np.fromfunction(**lambda** i, j: i\*j, (4, 4)) #ij can be 0

### Shaping

arr.shape = (4,4)

arr = np.reshape(arr, (16, 1)) #-1 calculates the other dim

x = np.atleast\_2d(x) #converts lower dimensions to 2D, useful for reshaping vector type np.arrays to matrix type (for dotprod)

arr.ravel() #to vector

arr.flatten() #copies, and converts copy to vector

### Stacking

qv1, qv2: [[1 1 1]], [[2 2 2] [2 2 2]]

qv3 = np.vstack((qv1, qv2))

qv3 = [[1 1 1] [2 2 2] [2 2 2]]

qh1, qh2: [[1 1 1] [1 1 1]], [[2 2] [2 2]]

qh3 = np.hstack((qh1, qh2))

qh3 = [[1 1 1 2 2] [1 1 1 2 2]]

### Linalg

**import** numpy.linalg **as** linalg

mtx.transpose()

mtx.T

mtx\_mult = mtx1.dot(mtx2)

inverse = linalg.inv(mtx)

pinverse = linalg.pinv(mtx)

identity\_mtx = np.eye(n)

det = linalg.det(mtx)

eigenvalues, eigenvectors = linalg.eig(mtx)

### Miscellaneous (random, linspace, math)

axis=0 refers to row

axis=1 refers to column

np.linspace(min, max, num): splits the interval (min, max) into num equal parts

r1, r2, r3 = np.random.rand(3, 100): creates a 3x100 random matrix, ri are the ith row

np.random.normal(0, 1, 100): creates a 100 long vector with normally distributed vals

a[i, j] indexing a multidiminensional array

np.sum(), np.max(), np.argmax() and other unary operators

np.squeeze(X, axis=1): removes the axis if it is single dimensioned

np.append(X, y): append element y to array X, returns a new object

## Pandas

Library for messing around with data.

**import** pandas **as** pd

### Series

Equivalent of Excel’s column.

|  |  |
| --- | --- |
| s\_indexed | |
| „a” | 30 |
| „b” | 27 |
| „c” | 22 |
| „d” | 41 |

#### Creating Series

|  |  |
| --- | --- |
| s\_base | |
| 0 | 2 |
| 1 | 3 |
| 2 | 4 |
| 3 | 5 |

predefined\_dic = {**"c"** : 22, **"d"** : 40}  
  
s\_base = pd.Series([2, 3, 4, 5])

s = np.exp(s\_base)  
s = s\_base + pd.Series([4, 3, 2, 1])  
s = s\_base + 1  
s = s\_base <= 3

s = pd.Series([30, 27, 22, 41], index=[**"a"**, **"b"**, **"c"**, **"d"**])  
s = pd.Series(predefined\_dic, index=[**"c"**]) #filtering by indices

s = pd.Series([22, 23], index=[**"alice"**, **"bob"**], name=**"ages"**)

s = pd.Series(np.nan, index=[**"def1"**, **"def2"**]) #def1: Nan, def2: NaN

#### Plotting Series

plt.scatter(s.index, s.values)   
#plt.plot would connect the points with linesplt.show()

### DataFrame

Equivalent of Excel’s spreadsheet.

#### Creating DataFrames

predefined\_dic = {  
 **"in letters"** : pd.Series([1, 2, 3]),  
 **"w numbers"** : pd.Series([**"ein"**, **"zwei"**, **"drei"**]),  
 **"rand"**: pd.Series(np.full(3, np.nan))  
 }  
df = pd.DataFrame(predefined\_dic) #if some columns couldn't be matched, those will become NaNsprint(df[[**"in letters"**, **"w numbers"**]])

#filtering  
df = pd.DataFrame(predefined\_dic, columns=[**"w numbers"**], index=[1, 2]) np\_datamatrix = np.array([  
 [**"alice"**, 33, **"mercedes"**],  
 [**"bob"**, 40, **"cadillac"**],  
 [**"chad"**, 25, **"bmw"**],  
])  
df = pd.DataFrame(np\_datamatrix, columns=[**"name"**, **"age"**, **"car"**], index=[**"a"**, **"b"**, **"c"**])

#### Accessing rows

|  |  |  |  |
| --- | --- | --- | --- |
|  | name | age | car |
| a (iloc 0) | alice | 33 | mercedes |
| b (iloc 1) | bob | 40 | cadillac |
| c (iloc 2) | chad | 25 | bmw |

#row of chaddf.loc[**"c"**]  
df.iloc[2]  
#row of alice and chaddf[np.array([**True**, **False**, **True**])]  
#row of alice and chaddf[pd.to\_numeric(df[**"age"**], errors=**"raise"**) <= 35]  
#inserting a new rowdf.loc[**"d"**] = np.array([**"dave"**, 17, np.nan])

df.drop(index=[**"d"**], inplace=**True**) #deleting the new row

#### Accessing columns

df = df.astype({**"age"** : **"int32"**})  
df[**"over 18"**] = [**False**] \* len(df.index)  
df[**"over 18"**] = pd.Series({**"a"** : **True**}) *#other will default to np.nan*df[**"over 18"**] = df[**"age"**] >= 18  
popped\_col = df.pop(**"over 18"**)  
df.drop(columns=[**"over 18"**])  
df.insert(0, **"ID card num"**, pd.Series({**"a"** : **"0x00"**, **"b"** : **"0xcf"**, **"c"** : **"0xfd"**}))  
evdf1 = df.eval(**"age\*\*2 + sin(age)"**)  
age\_limit = 21  
evdf2 = df.eval(**"age > @age\_limit"**)

#### Querying

queried\_df = df.query(**"age > @age\_limit and name.str.contains('a')"**)

#### Sorting

df.sort\_values(**"car"**, ascending=**True**, inplace=**True**)  
df.sort\_values([**"car"**, **"age"**], ascending=[**True**, **True**], inplace=**True**)  
df.sort\_index(ascending=**False**)

#### Handle missing data

df.fillna(**"no data"**, inplace=**True**)  
df.dropna(axis=0, how=**"any"**, inplace=**True**) #drop row if any of its value is np.nan

#### Pandas Utilities for reading structured data and misc

axis 0: rows

axis 1: columns

housing = pd.read\_csv(**"housing.dat"**, sep=**"..."**)  
housing[**"age\_category"**] = pd.cut(  
 housing[**"house age"**],  
 bins = (5, 20, 40, 80, np.inf),  
 labels = (**"new"**, **"renovated"**, **"mid"**, **"old"**, **"for demolition"**)  
)

#for printing a subset of the dataframe

df.head(num)

df.tail(num)

## Matplotlib

Scientific python library for showing graphs.

**import** matplotlib.pyplot **as** plt

### Lines and 2D functions

plt.figure(**"figure\_title"**, figsize=(10, 12))

plt.axis(**"off"**)



plt.plot([2, 1, 2, 4, 1])  
plt.show()



plt.plot([-3,-2, 1, -1], [2, 4, 1, 2])  
plt.axis([-3, 1, 1, 5]) *# xe[-3, 1], ye[1, 5]*plt.show()



x = np.linspace(-1.4, 1.4, 30)  
plt.plot(x, x, **"g--"**, linewidth=3)  
plt.plot(x, x\*\*2, **"r:"**)  
plt.plot(x, x\*\*3, **"b^"**, alpha=0.1)  
plt.show()



x = np.linspace(-2, 2, 500)  
y = x\*\*2  
plt.plot(x, y)  
plt.title(**"x squared"**)  
plt.xlabel(**"x"**)  
plt.ylabel(**"y"**)  
plt.grid(**True**)  
plt.show()

### Multiple plots on a figure



x = np.linspace(-1.4, 1.4, 30)  
plt.subplot(2,2,1)  
plt.plot(x,x)  
plt.subplot(2,2,2)  
plt.plot(x, x\*\*2)  
plt.subplot(2,1,2)  
plt.plot(x, x\*\*3)  
plt.show()

*subplot(2,2,2):*

*A table made up by two rows and two columns. We put the figure in the 2nd „cell”.*

### Scatter plots, histograms and images



mu, sigma = 100, 15  
x = mu + sigma \* np.random.randn(10000)  
plt.grid(**True**)  
plt.hist(x, bins=50, facecolor=**'green'**, edgecolor=**"black"**, alpha=0.75)  
plt.show()



x = np.linspace(-1.4, 1.4, 30)  
y = x\*\*2  
plt.scatter(x, y, s=200.0, c=**"red"**, alpha=0.9, edgecolors=**"green"**)  
plt.show()

plt.imshow(pic, cmap=**"binary"**)

plt.axis(**"off"**)  
plt.show() *# pic: NxM bitmap*

### Texts and Legends



x = np.linspace(-1.4, 1.4, 30)  
y = x\*\*2  
px = 0  
py = px\*\*2  
  
plt.plot(x, y)  
plt.plot(px, py, **"ro"**)  
plt.text(px, py + 0.05, **"minima"**, fontsize=10, color=**"red"**, horizontalalignment=**"center"**)  
plt.show()



plt.scatter([0.1, 0.15], [0.1, 0.08], color=**"red"**, marker=**"x"**, label=**"safe"**)  
plt.scatter([1, 0.89], [0.95, 0.99], color=**"green"**, marker=**"s"**, label=**"dangerous"**)  
plt.legend(loc=**"upper left"**)  
plt.show()

### ...

## Jupyter notebook

*pip install notebook*

*jupyter notebook (localhost:8888)*



## R

In R console we can include libraries (e.g.: library(datasets), library(**"BSDA"**)) and can install new ones:

install.packages("BoutrosLab.plotting.general")

### Attributes, Basic types, NA and NaN

atomic types: character, numeric, integer, complex, logical

x <- 1 #numeric

x <- 1L #integer

x <- 1/0 #numeric, Inf

x <- 0/0 #numeric, NaN

types gets printed on using class(object) and can be checked with is.numeric(obj)..

type casts: as.numeric(x), as.character(x)

operations with non-ordinary notations: %% (mod), %/% (integer division), ^

logical operations (and, or) are single digit: & |

check if two objects are identical: identical(x, y)

format string (3 to 003): sprintf("%03d", 3)

get size of an object: object.size(o)

NA: not available, missing value

NaN: not a number, counts as an NA as well

|  |  |  |
| --- | --- | --- |
|  | **NA** | **NaN** |
| is.na(x) | TRUE | TRUE |
| is.nan(x) | FALSE | TRUE |

### Complex types

vector (for storing elements with the same class):

x <- rep(NA, times=5) #NA NA NA NA NA

x <- rep(c(1, 2), each=2) #1 1 2 2

x <- c("numeric", length=5) #0 0 0 0 0

x <- c(1 + 0i, 2 + 3i) #explicitly telling the elements

x <- c(x, 3 + 2i) #appending to vector

x <- c(col1 = 1, col2 = 2) #named vector (names(object))

x <- 1:5 #int sequence: 1 2 3 4 5

x <- seq(0, 2, by=0.5) #0.0 0.5 1.0 1.5 2.0

x <- seq(0, 1, length=5) #0.0 0.25 0.5 0.75 1

x <- seq\_along(x) #1 2 3 ... length(x)

x <- rnorm(1000) #1000 normally distributed value

x <- sample(x, 100) #selects 100 elements randomly

length of vector: length(x)

list (for storing elements even with different types):

x <- list(1, TRUE, 1+2i)

x <- list(c1=1, c2=TRUE, c3=1+2i)

matrix (vector with set dimension attribute):

x <- matrix(nrow=2, ncol=3) # [NA NA NA]

# [NA NA NA]

x <- matrix(1:6, nrow=2, ncol=3) #column major filling

# [1 3 5]

# [2 4 6]

x <- 1:3 #1 2 3

y <- 6:8 #6 7 8 [1 6]

m1 <- cbind(x, y) #column-wise merge [2 7]

[3 8]

m2 <- rbind(x, yÖ #row-wise merge [1 2 3]

[6 7 8]

x <- 1:10 #redefining the dimension of a vector

dim(x) <- c(2, 5) [1 2 3 4 5]

[6 7 8 9 10]

inverse matrix: solve(m)

transpose matrix: t(m)

true matrix multiplication: m1 %\*% m2

factor (for categorizing data):

dirs <- c("left", "up", "up", "down", "up", "right", "left")

fdirs <- factor(dirs) # left up up down up right left

# Levels: down left right up

# levels are automatically ordered

fdirs2 <- factor(dirs, levels=c("up", "down", "left", "right")

# levels are manually ordered

Factor level generation, gl function (num\_leves, replications, seq):

f <- gl(3, 2, c("NY", "BP", "LA"))

# NY NY BP BP LA LA

# Levels: NY BP LA

On loading tabular data, columns of strings will be interpreted as factors.

data frame (handling tabular data with different types):

empty\_df <- data.frame(matrix(nrow=0, col=3))

colnames(empty\_df) <- c("col1", "col2", "col3")

df <- data.frame(col1 = 1:4, col2 = c(T,F,F,F))

add rows by calling: df <- rbind(df, list(col1 = 1, col2 = F))

add columns by calling: df <- cbind(df, col3 = c("a", "b"))

printing dataframe: head(df, n) tail(df, n)

dimensions of dataframe: nrow(df) ncol(df)

### Subsetting, indexing

[] : return an object of the same class, can be used to select multiple elements

[[]] : can only extract a single element from a list or dataframe, might return another type of object than is was called on

$: same as [[]], can helps us extract a single element by name

vector examples:

x <- c("a", "b", "c", "b", "d", "a")

x[1] #a, indexing starts at 1 in R!

x[1:3] #a b c

y[x %in% 1:3] #every element in y in [1, 3]

x[x < "c"] #a b b a

list examples:

x <- list(foo=1:4, bar=0.6)

x[1] #returns 1:4 sequence as LIST!

x[[1]] #returns 1:4 sequence as raw vector

x$foo #returns 1:4 sequence as raw vector

x[c(1,2)] #returns foo and bar as list

x[c("foo", "bar")] #returns foo and bar as list

x$f #partial matching, autocompleted to foo

matrix examples:

x <- matrix(rep(1, 6), 3, 2)

x[1,1] #returns 1 as numeric

x[1, ] #returns the first row as vector of numerics

x[1, , drop=FALSE] #returns the first row as a matrix

x[c(1, 2), ] #returns the 1st and 2nd row as a matrix

data frame examples:

df <- df[!is.na(df), c1] #returns every non-NA elem under c1

df <- df[complete.cases(df),] #returns NA-free rows

### Date & time

time:

t <- Sys.time() #class(t) POSIXct ("2021-03-08 22:43:50 CET") #unclass(t) integer (seconds since 1970)

t <- as.POSIXlt(t) #class(t) POSIXlt ("2021-03-08 22:43:50 CET")

#unclass(t) list with names: sec, min, ...

t <- strptime("October 17, 1999 08:00", "%B %d, %y %H:%M")

#will create a POSIXlt object

date:

d <- Sys.Date() #class(d) Date ("2021-03-08")

#unclass(d) integer (days since 1970)

d <- as.Date("1969-01-01) #unclass(d) integer (-365)

weekdays(d/t): "Monday"

months(d/t) "March"

quarters(d/t) "Q1"

### Reading data, Files

file.path(path1, path2) #will join path1 and path2

paste(str1, str2, ...) #concatenates strings

read.table ( filename, header=FALSE, sep="\tab", colClasses, nrows,

commentChar="#", skipBegin=0, stringAsFactors=TRUE)

read.csv is the same as read.table but sep is ",".

getwd() return the currect working directory.

source – dump:

y <- data.frame(a=1, b="str")

dput(y, "ser.R")

y <- dget("ser.R")

dget – dput:

Can handle multiple data.

x <- "foo"

y <- c(1, 2, 3)

dump(c(x, y), file="ser.R") #serialization

rm(x, y) #clears variable from enviroment

source("ser.R") #deserialization

### Control structures

if() {} else if {} else {}

y <- if (x < 3) { 10 } else { -10 }

for (i in 1:10)

for (elem in x)

for (i in seq\_along(x)) { next }

while(cnt < 10) { cnt <- cnt + 1}

repeat { ... break ... } #creates infinite loop

### Functions

Usually functions are written in a .R file which is then included to the current environment with sourcing (source("XYZ.R")). The last row of function is the return statement.

Complex examples:

f <- function(x=NA, fun, ...) {

args <- list(...)

y <- if (args[[using.fun]] == TRUE) { fun(args[[data]]) }

else { 0 }

x + y

}

f(c(1, 2), mean, using.func=TRUE, data=c(1, 1))

% bin % <- function(lhs, rhs) {

lhs %% rhs

}

x %bin% y

### Loop functions (apply-s)

lapply (x, fun, ...): Evaluates function on a list’s/vector’s each element.

Always returns a list.

sapply (x, fun, ...): Same as lapply but return the simplest possible class. By simplest class we mean either a single number or a vector.

vapply (x, fun, template, ...): Same as sapply, but checks whether the return type matches the template. If not it halts the program with an error. Template can be numeric(5) for example.

apply (x, margin, fun, ...): Allows us to call functions other than on arrays. The margin parameter’s counterpart in pandas is axis.

apply(m, 1, mean): mean of every row in matrix m (use rowMeans!)

mapply (fun, ..., more.args=NULL, simplitfy=TRUE, use.names=TRUE):

With mapply we can use a function with multiple sets of arguments.

mapply(rep, 1:3, 3:1) #list(rep(1, 3), rep(2, 2), rep(3, 1))

tapply (x, index, fun, ..., simplify=TRUE):

Useful when we need to break up the vector to groups defined by some classifying factor.

x <- c(rnorm(10), runif(10), rnorm(10, 1))

f <- gl(3, 10)

tapply(x, f, mean) #mean of rnorm, runif, rnorm1 in a list

split (x, f, drop=FALSE):

Will break up the vector to grups defined by the classifying factor f.

Will return a list of (# of levels in f) vectors. Can be used to simulate the group by statement.

table (x,f):

Maps the factor levels to their number of appearance in a tabular form.

### Statistics

empirical corrected variance: var(x)

### Misc

str: Compatly displays the internal structure of an object, alternative to summary

set.seed: important to set beforehand for reproducibility

sample(1:10, replace=FALSE): will make a permutation of 1:10

Random number generation: (with norm, gamma, poiss, etc. postfixes)

d: density (evaluates f(x), Probability DF)

r: random

p: cumulative distribution (returns F(x), CDF)

q: quantily function (returns F-1(x))

rbinom(100, size=1, prob = 0.7): gives the result of 100 runs of 1 coin toss each.

Profiling: system.type(expr)

elapsed time (”wall clock”): time you experience

user time: time charged to the CPU(s)

often: elapsed time = user time

if (elapsed time > user time): CPU waits for resources without running code

if (elapsed time < user time): In case of multiple processors

Rprof(), summaryRprof(): Prints out function call stack every 0.02s

#Rprof’s output will be just a bunch of function names

#summaryRprof will tabulate this output and will calculate how much

#time is spend in which function

by.total: Divides the spent time in each function by the total runtime.

by.self: Doest the same but subtracts the lower level function calls.

LETTERS: predefined vector of every english letter

plot(x, y, xlab, ylab, xlim, ylim, sub, main, pch ...)

(pch: point shape: triangle, filled circle, etc.)

## SQL (Microsoft Server SQL)

SQL is a Data Query/Definition/Control/Manipulation language which allows us to issue Create, Read, Update, Delete (CRUD) commands to a relational database system.

### Data types

INT(size=255)

DEC(size=10, precision=0)

BOOL

VARCHAR(length)

BLOB # Binary Large OBject

DATA # YYYY-MM-DD

TIMESTAMP # YYYY-MM-DD HH:MM:SS

### Create and alter schemes

CREATE TABLE student (

id INT, -- IDENTITY(1,1) for autoincrement

-- teacher\_id INT,

name VARCHAR(20), -- NOT NULL, UNIQUE

major VARCHAR(20), -- DEFAULT ’unknown’

PRIMARY KEY(student\_id),

-- FOREIGN KEY (teacher\_id) REFERENCES teacher(id)

-- ON DELETE SET NULL

-- ON DELETE CASCADE

);

ALTER TABLE ADD gpa DEC(3,2);

ALTER TABLE DROP COLUMN gpa;

DROP TABLE student;

primary key: A minimal set of attributes that uniquely specifies a row.

foreign key: A primary key of another scheme.

### Insert rows into table

INSERT INTO student VALUES(1, ’Jack’, ’CS’);

INSERT INTO student(id, name) VALUES(2, ’Kate’);

INSERT INTO student(id) VALUES (3), (4), (5), ... ; --multiple rows

### Delete rows from table

DELETE FROM student WHERE major=’Unknown’

### Queries

### Joins

### Data types

### Data types

## Docker

Docker is an ecosystem aroun creating and running containers. In a docker image we can have dependencies and other configs (FileSystem Snapshot) from which we can launch running containers.

Docker is a virtual machine with linux OS and by that it allows us to use Namespacing (define the resources that the container can use: Memory/Networking, ...) and Control Groups which helps limiting the usage of these resources. The docker image can contain a default (startup) command (e.g: run hello-world).

### Container manipulation

docker run <image name> // docker create + docker start

docker run <image name> <overriden default command>

docker create <image name> <odc> // return a unique id

docker start –a <unique\_id> // runs the image, -a: output to console

docker logs <unique\_id> // gets every log that was emitted from the run

docker ps // lists every running container

docker ps –all // every container that used to run (exited can be restarted)

docker system prune // deletes every stopped containers and caches (-f force)

docker stop <unique\_id> // SIGTERM (give time for cleanup and save)

docker kill <unique\_id> // SIGKILL (instantenaous stop of the process)

// gets called automatically after 10s of docker stop

docker exec –it <unique\_id> <command> // input text, additional command

docker exec –it <unique\_id> sh // open shell in the context of container

docker run –it busybox sh // alternative

### Creating images (Dockerfile)

**FROM --- COPY --- RUN --- CMD**

FROM node:alpine (hub.docker.com/explore to see the popular ones)

WORKDIR /usr/app (against overwriting the root directory)

COPY ./ ./ (copy everything in the wd to the container’s filesystem, SPLIT!)

RUN npm install (will look for package.json to install)

CMD [”npm”, ”start”] (will look for package.json to start)

Dockerfile // docker build . in its folder

docker build <unique\_id> // if we changed anything besides Dockerfile!

docker build –f Dockerfile.dev . // allows custom dockerfile name

docker build –t mkis98/test:latest . // this way we don’t need id-s, img tagging

docker run –p 8080:8080 <unique\_id> // every incoming request on localhost 8080 will be forwarded to the container’s 8080 port

### Docker-compose (docker-compose.yml)

version: ’3’

services:

s1:

image: ’redis’

s2:

restart: always # on crash policy

build: . # Built from dockerfile, image defined there

ports:

- ”4001:8080” # array

docker-compose up (= docker run <image>)

docker-compose up –build (= docker build ., docker run <image>)

docker-compose up –d (open service in the background)

docker-compose down (stop services facilitating using ’docker stop’s)

On crash restart policies: **”no”**, **always**, **unless-stopped** (always restart except on forced close, **on-failure** (exit code is not 0)

docker-compose ps (will look for docker-compose.yml and finds the running containters defined there)

### Docker volume

### Docker

## LaTeX

Editor can be found here: <https://www.overleaf.com/>

useful packages to include:

\usepackage{amsmath} % advanced math symbols, spaces

\usepackage{physics} % for prettier vector symbols

### Symbols

Inline and complex formulas: $ e = 2.71... $ $$ complex formula $$

Greek alphabet: \alpha \gamma \Alpha \Gamma \pi

Partial derivative letter: \partial

Basic arithmetic: \cdot (mult dot) \dots (...) ^{x + 1} v\_{index} \vdots colv \cross

Fractions: \frac{a}{b} \xfrac{a}{b} (tilted line)

Sum and product: \sum\_{}^{} \prod\_{}^{} % bounds are omittable

Integral: \int\_{-\infty}^{\infty}{f(x) \text{ } dx}

Root: \sqrt[n]{...}

Limit: \lim\_{n \to \infty}

Autoscaling bracket: \left(1 + \frac{1}{n}\right)^n

Sets: \in \notin \subset \subseteq \cap (intersection) \cup \setminus

Vectors: \vb{A\_{n\cross n}} % physics library for bold vectors

Matrix:

$$

\begin{pmatrix/bmatrix}

1 & 2 & 3 \\

4 & 5 & a+1 \\

\end{pmatrix/bmatrix}^T

$$

\def\A{

\begin{bmatrix}

x\_1 & x\_2 & \cdots & x\_N

\end{bmatrix}}

\def\B{

\begin{bmatrix}

ax\_0 + bx\_1 \\

ax\_1 + bx\_2 \\

\vdots \\

x\_{N-1} + x\_N

\end{bmatrix}}

\def\C{

\begin{bmatrix}

z\_1 \\

z\_2 \\

\vdots \\

z\_N

\end{bmatrix}}

$$ y =\A \left(\B - \C\right) $$



### Formatting

**bold:** \textbf{...}

**underline:** \underline{}

**italic:** \textit{...}

**emphasize:** \emph{...} % effect depends on the context

\chapter{Intro} \section{...} \subsection{...} \section\*{no #}

\begin{itemize/enumerate}

\item ... % will insert bullett/number

\item ... % will insert bullett/number

\end{itemize/enumerate}

# Scikit-Learn and book abstract

## Types of Machine Learning systems

### Supervised / Unsupervised / Semisupervised / Reinforcement

Supervised: Training set includes the solutions (aka the labels) – Classification –

Unsupervised: Training set is unlabeled – Clustering & Anomaly Detection –

Semisupervised: Combined version of the previous types

Reinforcement learning: Reward based training of an agent

### Batch / Online

Batch: The system must be retrained from scratch with the full dataset on every new version

Online: The system can be trained incrementally by feeding it data instances either individually or in mini-batches

### Instance-based / Model-based

Instance-based: Similarity to the already learnt examples is measured on new cases

Model-based: Build a model from the learnt examples and make predictions according to the model.

## Challenges of Machine Learning

Insufficient quantity of training data

Nonrepresentative training data: training data must be representative of the new cases

Poor-quality: outliers, noise, errors, n/a-s

Redundant features

Overfitting training data (regularization helps)

Underfitting training data (model with more params helps)

## Testing and Validating

Testing how well the model will generalize to new cases is done by splitting the data set into a *training set* and a *test set*. We then train our model with the training set and test its error rate on never previously seen cases from the testing set. This error rate is called the *generalization error*. If we evaluate our model on the training set and get a much lower training error than the generalization error of the corresponding test set then we have the problem of overfitting, meaning that our model only performs well on the training data and the handling of new instances is quite flawed.

Often we hold out another set called the *validation set* to avoid refining (experimenting with regularization hyperparameters) a model to perform well solely on a particular training set. We train the model with the reduced training set (full traning set minus the validation set) and choose the model that produces the lowest generalization error on the validation set. After this process we retrain this model with the full training set and lastly evaluate the final model on the test set to get an estimate of the generalization error.

Cross-validation: The idea is to use multiple validation sets. Each model candidate (SVM, logistic regression etc) is evaluated once per validation set and trained on the rest of the training set. By averaging the evaulation results we get a realistic picture of the performance of out model at the cost of training time.

## Bias and variance

**bias:** Measures how much the model differs from the true labelling function.

**variance:** Measures how sensitive the model is to small deviations in the training data.

**Underfitting:** The model has high bias and low variance which means that it makes too general predictions (good instead of great, but can be worse) but at least it does it consistently. Solution: make the model more complex.

**Overfitting**: The model has low bias but high variance which means that it makes too specific predictions so it will generalize bad and will make extremely large errors on new instances. Solution: regularization, simplified model.

## Scaling

Machine learning models work better when features are on a similar scale (gradient descent can use larger learning rate, svm can draw a larger margin boundary). The most often used scaling method are MinMax scaling [0, 1] and Standard scaling N(0, 1).

**from** sklearn.preprocessing **import** MinMaxScaler, StandardScaler  
  
X = np.array([[1], [2], [3], [4]])

scaler = MinMaxScaler()  
X = scaler.fit\_transform(X)

## Encoding

### Label encoding

Converts a 1 dimensional vector of strings/numbers into categorical values (ints). Ideal to transform **y**-s that are in string format and **has ranking**. (bad/average/good)

**from** sklearn.preprocessing **import** LabelEncoder

labels = np.array([**"Tokyo"**, **"London"**, **"London"**, **"Frankfurt"**])

label\_encoder = LabelEncoder()

cat\_labels = label\_encoder.fit\_transform(labels)

print(label\_encoder.inverse\_transform(cat\_labels)) #prints labels

### One hot encoding

Converts a 2 dimensional array of strings/numbers into one-hot encoded format. Ideal to transform **y**-s for softmax regression and **y-**s that are unranked (nominal).

**from** sklearn.preprocessing **import** OneHotEncoder

labels = np.array([**"Tokyo"**, **"London"**, **"London"**, **"Frankfurt"**])

labels = np.reshape(labels, (-1, 1))

oh\_encoder = OneHotEncoder(dtype=np.int32)

oh\_labels = oh\_encoder.fit\_transform(labels).toarray()

print(oh\_encoder.inverse\_transform(oh\_labels)) #prints labels

### Ordinal encoding

Converts a 2 dimensional array of strings/number into categorical values (floats). Ideal to transform features that are strings (city/car brand).

**from** sklearn.preprocessing **import** OrdinalEncoder

#two column of cities

features = np.array([**"Tokyo"**, **"London"**, **"London"**, **"Frankfurt"**],

[**"London"**, **"Los Angeles"**, **"Frankfurt"**, **"Frankfurt"**])

features = features.T

ordinal\_encoder = OrdinalEncoder(dtype=np.int32)

cat\_features = ordinal\_encoder.fit\_transform(features)

print(ordinal\_encoder.inverse\_transform(cat\_labels)) #prints features

## Transformation pipelines

With pipelines we can define a series of data transformations to be executed consecutively (in the example: PolynomialFeatures with LinearRegression).

**from** sklearn.pipeline **import** Pipeline

**from** sklearn.preprocessing **import** StandardScaler

**from** sklearn.preprocessing **import** PolynomialFeatures

**from** sklearn.linear\_model **import** LinearRegession

polynomial\_regression = Pipeline([

(**"poly\_features"**, PolynomialFeatures(degree=4, include\_bias=**False**)),

(**"lin\_regression"**,LinearRegression())

])

polynomial\_regression.fit(x, y)

Fit will call each estimator’s (=member) *fit\_transform(...)* function except on the last one where it calls the regular *fit*function.

**from** sklearn.compose **import** ColumnTransformer

**from** sklearn.impute **import** SimpleImputer

missing\_embark = Pipeline([  
 (**"fill\_na"**, SimpleImputer(strategy=**"constant"**, fill\_value=**"other"**)),  
 (**"one\_hot"**, OneHotEncoder(dtype=np.int32))  
])  
  
missing\_agefare = Pipeline([  
 (**"fill\_na"**, SimpleImputer(strategy=**"median"**)),  
 (**"one\_hot"**, StandardScaler())  
])  
  
column\_preprocessing = ColumnTransformer([  
 (**"sex\_encoder"**, OrdinalEncoder(dtype=np.int32), [**"Sex"**]),  
 (**"pclass\_encoder"**, OrdinalEncoder(dtype=np.int32), [**"Pclass"**]),  
 (**"missing\_embark"**, missing\_embark, [**"Embarked"**]),  
 (**"missing\_agefare"**, missing\_agefare, [**"Age"**, **"Fare"**]),  
 (**"min\_max\_scaler"**, MinMaxScaler(), [**"SibSp"**, **"Parch"**])  
])

Column transformers are similar to pipelines. They are responsible for doing the standard column modifications (scaling/handling missings/encoding). The input should be a pandas dataframe. The output will be a numpy matrix.

## Hyperparameter tuning

### Grid search

Grid search helps with finding the best performing hyperparameters. As grid search is slow we should first start with coarse-grained searching and fine-grain it overtime.

**from** sklearn.model\_selection **import** GridSearchCV

param\_grid = [

{**"n\_estimators"**:[3, 10, 30], **"max\_features"**:[3, 4, 5] },

{**"n\_estimators"**:[3, 10], **"max\_features"**:[2, 3], **"bootstrap"**=[**False**] }

]

forest\_reg = RandomForestRegressor()

# cv: cv-fold cross validation

grid\_search = GridSearchCV(forest\_reg, param\_grid, cv=5)

grid\_search.fit(X, y)

grid\_search.best\_params\_ # optimal params in a dict

grid\_search.best\_estimator\_ # matching RandomForestRegressor object

### Randomized search

Random search will not try out every combination of hyperparameters but instead will choose randomly from the intervals we’re trying to pick our hyperparamters from.

**from** sklearn.model\_selection **import** RandomizedSearchCV

**import** scipy.stats **as** stats

# np.linspace, np.arange should work too

param\_dist = [{  
 **"n\_estimators"**: [\*range(15, 150, 5)],  
 **"max\_depth"**: [5,6,7,8,9,10,11,12],  
 **"max\_features"**: [3,4,5,6,7,8,9],  
 **"min\_samples\_split"**: [2,5,10,15,25],  
 **"ccp\_alpha"**: stats.uniform(0, 3)  
}]

forest\_reg = RandomForestRegressor()

# cv: cv-fold cross validation

# n\_iter: how many combinations of params we want to sample

rnd\_search = RandomizedSearchCV(forest\_reg, param\_dist, cv=5, n\_iter=20)

rnd\_search.fit(X, y)

rnd\_search.best\_params\_ # optimal params in a dict

rnd\_search.best\_estimator\_ # matching RandomForestRegressor object

## Datasets

**from** sklearn.datasets **import** fetch\_xxx / load\_xxx

After fetching use the suitable keys (keys()) to extract the data and the labels.

## Test-train split and cross validation

**from** sklearn.model\_selection **import** cross\_val\_score  
**from** sklearn.model\_selection **import** train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
val\_scores = cross\_val\_score(sgd\_classifier, X\_train, y\_train, cv=3, scoring=**"accuracy"**)

## Clustering

### Agglomerative Clustering

The method:

As initialization we put our **k** datapoint into separate clusters and in every step we will choose the two clusters that are closest (up to a point defined by the *distance\_threshold*) defined by a linkage criterion and merge them. The method will not solve the problem of choosing the number of wanted clusters but we can utilize dendrograms to tackle this problem.

The linkage criterion between two clusters has various forms:

**single linkage:**

**complete linkage:**

**average linkage:**

**ward linkage:** complex method for minimum variance within clusters

**from** sklearn.cluster **import** AgglomerativeClustering

clustering = AgglomerativeClustering (n\_clusters=5, ...)

clustering.fit(x)

clustering.labels\_

### K-Means Clustering

An algorithm that will split the datapoints into **k** (predefined number) clusters. The position of the cluster-means are updated dynamically until a stable state. The default positions of the cluster-means can be predefined or chosen randomly. A point is belonging to a specific cluster if the distance between them is the less than the distance between the point and any other cluster mean. The distance metrics here is the eucledian distance.

Algorithm:

* We have the defult positions of cluster means
* We go through every data point and append to the cluster for which the cluster-mean-datapoint distance is minimal.
* We recalculate the claster-means by averaging the data points belonging to the same cluster
* We keep on doing this until convergence

Note that the cluster centers are not necessarily real data points. If we want our cluster centers to represent real data then we should just overwrite the average with the closest real data point. In this case the method is called **K-Medoid**.

K-Means algorithm is not ideal for density based clustering or when the expected clusters are concave. sklearn’s KMeans implementation uses an updated version of the method mentioned above outputting suboptimal solutions less likely.

**from** sklearn.cluster **import** KMeans

**from** sklearn.cluster **import** MiniBatchKMeans

kmeans = KMeans(n\_clusters=5, init, n\_init, ...)

kmeans.fit(x)

kmeans.labels\_

Evaluation of clusters:

**Silhouette**

is x’s average distance from every other point in its cluster.

is the minimum of x’s average distance from every other point in foreign clusters.

refers to the class of x, return the numeric class label of x.

Silhoutte is a number between -1 and 1. Silhoutte of 1 means that x is far from the neighbouring clusters. Silhoutte of 0 means that the data point is on the border of two clusters. Silhoutte -1 is an indicator that the point is in the wrong cluster.

**Elbow curve**

Elbow curve can help us finding the ideal number of clusters for the K-Means algorithm. The elbow curve is defined the following way:



We choose the highest k value for which the error significantly dropped. Note, that if we choose the number of cluster equal to the number of datapoints the error will be zero. The W(k) value is stored in the *inertia\_*member of the KMeans class.

**from** sklearn.cluster **import** KMeans

**from** matplotlib **import** pyplot **as** plt

X = # data

distorsions = []

for k in range(2, 20):

kmeans = KMeans(n\_clusters=k)

kmeans.fit(X)

distorsions.append(kmeans.inertia\_)

fig = plt.figure(figsize=(15, 5))

plt.plot(range(2, 20), distorsions)

plt.grid(**True**)

plt.title('Elbow curve')

### DBSCAN (Density-Based) Clustering

DBSCAN is a computationally intensive clustering technique that can handle concave (non-convex) groups by using a density-based approach.

The *eps* neighbourhood of a data point **p** are the points that are at most *eps* far from **p**:

We say that **p** is directly density reachable from **q** with respect to *eps, min\_samples* if:

**q** in this case is called a core point.

We say that **p** is density reachable from **q** with respect to *eps, min\_samples* if there is a series of where each is directly density reachable from .

We say that **p** is density connected to **q** with respect to *eps, min\_samples* if there is an **o** point such that both **p** and **q** are density reachable from **o**.

If a point **p** is in cluster **C** and **q** is density reachable from **p** then **q** will be a point of the cluster too. In the cluster every point is density connected. The DBSCAN method might not classify points. Those unclassified points are considered noise.

**from** sklearn.cluster **import** DBSCAN

dbscan = DBSCAN(eps=3, min\_samples=5)

dbscan.fit(x)

dbscan.labels\_

# Classification

During classification we are predicting classes.

## Logistic Regression

Logistic regression is used to estimate the probability of an instance failling into a particular class. Logistic regression is a binary classifier: we decide whether the instance belongs to a certain class or not.

is the **sigmoid**/logistic function that clamps its argument to the (0, 1) interval. The derivative of sigmoid is :



This mixed results in the following formula:

With maximum likelihood estimation we can estimate for which the given classification outputs (talking about the training set) appeared with the highest probablity:

The optimal can be found with the gradient ascent algorithm:

**from** sklearn.linear\_model **import** LogisticRegression

clf = LogisticRegression(...)

clf.fit(X\_train, Y\_train)

Y\_preds = clf.predict(X\_test)

By specifying *multi\_class=’multinomial’* we got to do multiclass classification with the so-called Softmax regression method. C parameter will be the inverse of the regularization constant.

### Softmax / Multinomial logistic regression

Let **K** denote the number of classes we are trying to classify our datapoints to, the outputs become one hot encoded.

For each instance we give several softmax scores corresponding to a class k. Note that each class will have its own parameters.

Now, we can predict the probability of **x** belonging to the class k the following way:

To determine the parameters we will use the categorical cross-entropy loss function:

Here is the probability of the ith instance falling into the class k. It is either 0 or 1 defined by the digit on the kth index of the one hot encoded vector. on the other side is a matrix containing each class’ parameters in the following structure:

The loss function can be minimized knowing:

## Naive Bayes

Naive Bayes has a strong assumption on the datapoint’s features which is that they are conditionally independent given y (the class they are in).

Since P() is a constant we want to classify the example **x** to the y for which the is maximal.

P(y) can be calculated from the training set’s labels:

The naive Bayes classifiers differ in calculating . It depends on our assumption regarding the distribution of .

Note that this method might not work in case of Bernoulli features (a word was present in an email or not). Suppose we have a spam email (y=1) and the word „Bye” in its text.

will return 0 if there were no training examples that were spams and included the word „Bye” which messes up the classification, even if we had several evidence that this mail was indeed spam. This is resolved by the smoothing techniques.

**from** sklearn.naive\_bayes **import** GaussianNB, MultinomialNB, BernoulliNB

clf = BernoulliNB(alpha=1.0) #1.0 for laplace smoothing

clf.fit(X\_train, Y\_train)

Y\_preds = clf.predict(X\_test)

Use naive Bayes as a baseline as it is fast and sometimes works well.

### Laplace smoothing

Let denote the probability of the jth word being present in a spam email (calculated from the training set):

If we haven’t seen a word appearing in either spam or non-spam emails during training time it’s a bad assumption that it won’t ever show up (0 probability). The correction:

+2 refers to the number of possible classes (spam, non-spam).

## Support Vector Machine

## Decision Tree

## Random Forest

## Confusion Matrix and ROC Curve

**from** sklearn.metrics **import** confusion\_matrix  
**from** sklearn.metrics **import** precision\_score  
**from** sklearn.metrics **import** recall\_score  
**from** sklearn.metrics **import** f1\_score  
  
cfm = confusion\_matrix(y\_train, y\_preds)  
ps = precision\_score(y\_train, y\_preds)  
rs = recall\_score(y\_train, y\_preds)  
fs = f1\_score(y\_train, y\_preds)

Given the task of deciding whether a handwritten digit is 5 or not:

**Precision (3 / 4)**

**Recall (3 / 5)**

\*TN: The letter is actually other than 5 and fortunately the model says the same.

\*\*FP: The letter is actually other 5 but we accidentally classify it as 5.

\*\*\*FN: The letter is actually 5, but we accidentally classify it as other than 5.

\*\*\*\*TP: The letter is actually 5 and luckily the model says so as well.

### Rand index

We divide the correctly clustered/classified instances with the number of every index.

### Jaccard index

Jaccard index is mostly used in computer vision when using bounding boxes to identify objects. Jaccard index in computer vision will be the area of overlap divided by the area of union often called the IoU metrics (Intersection over Union).



### Precision

Amongst all the cases the model predicted positive, how many of them is actually positive. (Collecting games that can be played by children. Gather less, but guarantee that those games wont show any gore / adult content)

### Recall

How many amongst all the positive cases in the real world got found by our model. (For example we’d like to get all the patients with viral infections from a group of people. It’s definitely better if someone has to go to the hospital redundatly rather than missing someone with dangerous disease.)

### F1 Score and Precision-Recall tradeoff

Usually, the lower the precision is, the higher the recall is. Thus, we combined these two measures into ’F1 Score’ by taking the harmonic mean of them.

Ideally, we’d like to maximize this.

**from** sklearn.metrics **import** precision\_recall\_curve  
  
y\_scores = sgd\_classifier.decision\_function(X\_test)  
precisions, recalls, thresholds = precision\_recall\_curve(Y\_test\_5, y\_scores)  
plt.title(**"PR curve"**)  
plt.plot(thresholds, recalls[:-1], **"g-"**, label=**"Recall"**)  
plt.plot(thresholds, precisions[:-1], **"b--"**, label=**"Precision"**)  
plt.legend(loc=**"center right"**)  
plt.ylim([0, 1])  
plt.xlabel(**"Threshold"**)  
plt.grid(**"on"**)  
plt.show()

*Receiver operation characteristic curve* (**ROC**) is another tool used with binary classifiers.

ROC curve plots the True Positive Rate (TPR a.k.a. recall or sensitivity) against the False Positive Rate (FPR). FPR is equal to (1 – True Negative Rate). True Negative Rate (specificity) is the ratio of the actually negatively marked instances over all the negative instances of the real world.



We can see the tradeoff here as well. As we find more and more actually positive

cases the false positive rate increases simultaneously. One way to compare classifiers is to measure the AUC (area under the curve). An ideal classifier’s AUC would be close to 1 because we want a classifier that almost touches the upper left corner.

**from** sklearn.metrics **import** roc\_curve  
**from** sklearn.metrics **import** roc\_auc\_score  
  
fpr, tpr, thresholds = roc\_curve(Y\_test\_5, Y\_preds\_5)  
auc\_score = roc\_auc\_score(Y\_test\_5, Y\_preds\_5)  
plt.title(**"Receiver Operating Characteristic"**)  
plt.plot(fpr, tpr, **"b"**, label=**"AUC = {:0.2f}"**.format(auc\_score))  
plt.legend(loc=**"lower right"**)  
plt.plot([0, 1], [0, 1], **"r--"**)  
plt.xlim([0, 1])  
plt.ylim([0, 1])  
plt.ylabel(**"True Positive Rate"**)  
plt.xlabel(**"False Positive Rate"**)  
plt.grid(**"on"**)  
plt.show()

## Multiclass Classification

Doing multiclass classification is not always supported natively by the most frequent classification algorithms. Thus, the problem of multiclass classificiation is often reduced to multiple binary classification problems.

The examples below will show two different approach to solve the 10-digit handwriting recognition problem.

### One versus All (One versus Rest)

Make 10 binary classifier that tells whether the given digit is 0, 1, ..., 9 or not.

On a newly introduced digit we run all these 10 classifiers and select the class whose classifier outputs the highest score.

### One versus One

This method trains a binary classifier for all possible pairs (45 for the current problem)

Run all 45 classifiers on a digit never seen before and choose the class that wins the most.

## Egyéb

# Regression

While doing regression we try to predict a continous outcome variable (y) based on some input data **x**.

## Linear Regression

Linear regression assumes that y is a linear function of **x**’s attributes.

We can rearrange this into a matrix form (we prepend **x** with the intercept term 1):

The task from now on is to find the parameters that minimize the loss function (here m is the number of training examples):

We can solve the minimization problem by gradient descent (iterative) or normal equation, for which we define the desing matrix **X** where the rows are the training examples without the values (**y** column vector):

The normal equation:

**from** sklearn.linear\_model **import** LinearRegression

regression = SGDClassifier()

regression.fit(X\_train, Y\_train)

Y\_preds = regression.predict(X\_test)

regression.intercept\_

regression.coeff\_

### Ridge regression (L2 regularization)

The extra term in the cost function forces the learning algorithm to not only fit the data but also keep the model’s weight small.

adjusts the power of the regularization. An close to zero will result in the classic linear regression, on the other hand a significantly large will result a flat line passing through the data’s mean. Note that the bias term is not regularized.

For ridge linear regression we have a closed form (non-iterative) solution as well:

**from** sklearn.linear\_model **import** Ridge

regression = Ridge(alpha=1.0)

regression.fit(X\_train, Y\_train)

Y\_preds = regression.predict(X\_test)

L2 (2nd norm) regularization works well with polynomial regression.

### Lasso regression (L1 regularization)

Least absolute shrinkage and selection operator.

It works similarly as the L2 regularization but here we are using the first norm. This type of regularization excels at completely eliminating weights connected with the less important features.

**from** sklearn.linear\_model **import** Lasso

regression = Lasso(alpha=1.0)

regression.fit(X\_train, Y\_train)

Y\_preds = regression.predict(X\_test)

### Elasticnet

Elasticnet is a middle ground between Ridge and Lasso regression (or L1 and L2 regularization). The mix of the two methods can be controlled with a hyperparamter r as follows:

Elasticnet is slightly more preferred over Lasso, even if you suspect that there will be some useless features.

**from** sklearn.linear\_model **import** ElasticNet

regression = ElasticNet(alpha=1.0, l1\_ratio=0.5)

regression.fit(X\_train, Y\_train)

Y\_preds = regression.predict(X\_test)

## Locally Weighted Linear Regression

LWLR is a non-parametric learning algorithm which means there is no fixed to the data, but every prediction will require a completely new parametrization. This essentially means that we have to keep our data in the memory. Locally weighted linear regression can help in problems where classic linear regression fails.



When doing LWLR the loss function what we’d like to minimalize with the proper looks like the following:

w is the weighting function, a common choice for w is:

is the bandwidth parameter which controls the wideness of the significant neighbourhood (larger tau implies larger non-zero impact neighbourhood).

If is close to **x** then the weight is large (close to 1) and has large impact on the output. On the other hand when is further away then the weight will be close to 0.

LWLR has no sklearn implementation.

Use LWLR for low dimensional dataset (at most 10 features) where we have a lot of data and we don’t want to do feature selection.

## Polynomial Regression

Quite often a straigth line won’t be able to fit the data properly as the data is clearly nonlinear. When doing polynomial regression we add certain powers of the original features as new features and with that we use the classic linear regression algorithm.

**from** sklearn.preprocessing **import** PolynomialFeatures

poly = PolynomialFeatures(degree=2, include\_bias=False)

X\_train = poly.fit\_transform(X\_train)

Here if X\_train was a single training example with features [a, b] then the fit\_transform will do the following expansion of input features (with include\_bias turned off, thus missing a 1 from the beginning):

With the new training data we can do linear regression to determine :

**from** sklearn.linear\_model **import** LinearRegression

regression = SGDClassifier()

regression.fit(X\_train, Y\_train)

Y\_preds = regression.predict(X\_test)

regression.intercept\_

regression.coeff\_

## Support Vector Machine

## Support Vector Machine

## Support Vector Machine

## Support Vector Machine

# Statistics

## Permutation, Variation, Combination

P = 1·2·...·n = n! (number of orderings of n different elements)

P = n! / (k1!·k2!·...·kn!) (number of orderings if we have k1, k2, ..., kn pieces of the same instance)

V = n(n-1)...(n-k+1) (numbers of ordering of k diff. elements chosen from n diff. elements)

V = n·n·...·n = nk (if we allow repetition during the selection of the k elements)

C = n! / k!(n-k)! (same as the first V but we dont order, that’s why the k! division)

C = (n-k-1 k) (same as C but we allow repetition)

## Probability theory

P(A+B) = P(A) + P(B) – P(AB)

P(AB) = P(A|B)P(B) = P(B|A)P(A)

if A and B are independent events then P(AB) = P(A)P(B), P(A|B)=P(A)

if A and B are mutually exclusive (aka disjoint) then P(AB) = 0

Mutually exclusive events are those events that cannot occur at the same time. In case of independent events one event remains unaffected by the occurrence of the other event. Note, that two mutually exclusive event are never independent: one occuring anticipates the occurrence of the other (if A happened B will not).

Events A1, A2, ..., An are fully indenpendent if they are all pairwise independent.

We examine the probability of A happening if we know that B already did. P(B) cannot be 0 for this reason.

**Chain rule of probality:**

**Law of total probability:**

A1, A2, ..., AN are pairwise disjoint and their union is the whole sample space (their probabilities summed up is 1). P(Ai) cannot be zero.

**Bayes’ rule:**

We have the same assumptions as before. We substitute P(B) with the formula given above.

## CDF, PDF, PMF

**Cumulative distribution function (CDF):**

**Probability density function (PDF):**

**Probability mass function (PMF):**

It is the same as PDF, but in discrete case. P(X = x) for example.

## Expected value, Variance, Standard deviation

**Expected value:**

Discrete:

Continous:

**Variance (the average squared discrepancy from the mean):**

**Standard deviation:**

Identities:

E(aX + b) = aE(X) + b

V(aX + b) = a2V(X)

By substituting with a, we got an equivalent form:

z-score of X:

**Conditional expectation:**

Discrete:

Continous:

**Law of total expectation:**

The expected value of the X|Y conditional expectation is the expected value of X.

**Law of unconscious statistician (LOTUS):**

Discrete:

Continous:

## E(X) and V(X) of random length sum of iid variables

iid: Identically and independently distributed.

are iid. N is a random variable coming from a specific distribution.

## Moments, Skewness, Kurtosis

kth moment: E(Xk)

kth central moment: E[(X-E(X))k]

kth abs. moment: E(|X|k)

kth abs. central moment: E[|X-E(X)|k]

The 1st moment is the expected value. The 2nd central moment is the variance. With little modification the 3rd c. moment is the skewness (assimetry of the random variable about its mean) and the 4th is the kurtosis.

## Median, Mode, Quantile (percentile)

The median is a single number that separates the data sample to a lower half and a higher half. In discrete case median is the middle element of the ordered dataset or the average of the 2 middle elements.

The modes are the most frequent elements, so the ones that have the highest probabilities. For continous distributions the modes are the x values for which the PDF has a local maximum.

The p-quantile of the random variable X is the number for which the X is below with p chance and above with 1-p chance. The percentile is basically the same concept except we grant the p chance in percents.

Quantile () for the probablity.

## Multidimension distributions, Covariance, Correlation

E(X + Y) = E(X) + E(Y)

V(X + Y) = V(X - Y) = V(X) + V(Y) + 2cov(X, Y)

E(XY) = E(X)E(Y) if X and Y are independent

V(X+Y) = V(X-Y) = V(X) + V(Y) if X and Y are independent

**Joint CDF**:

Discrete: (xi, yj) are the pairs that can be assigned to (X, Y)

Continous:

**Marginal CDFs:**

Events X and Y are independent if (discrete) or (continous).

Discrete:

marginal CDF of X:

marginal CDF of Y:

Continous:

marginal CDF of X:

marginal CDF of Y:

**Marginal PDFs:**

marginal PDF of X:

marginal PDF of Y:

**Law of total probability and Bayes’ rule in continous case:**

**Covariance:**

Covariance is the unscaled version of correlation. Covariance indicates the direction of the linear relationship between variables.

**Correlation:**

Correlation on the other hand both measures the strength and the direction of the linear relationship between two variables. R is in [-1, 1]. If X and Y are independent, then R is 0.

## Popular discrete distributions

### Indicator function (aka characteristic function, Bernoulli distr.)

### Binomial distribution

We make n independent experiments and we know that the event will happen with probability p. The goal is to find out the probability of the event happening k times during the n trials. X will be the number of successful trials when the event went off. It can be considered as the sum on n indicators.

Expected value and variance:

### Poisson distribution

If n is large and p is really low the binomial distribution transforms into the so-called Poisson distribution. is the expected value (= np).

Expected value and variance:

### Geometric distribution

We keep on experimenting until we finally make the event A (p = P(A)) go off. X is the number of experiments we need to do until the first appearance of A.

Expected value and variance:

### Hypergeometric distribution

We have N product in which M is faulty. The goal is to find out the probability of us choosing n product (without replacement) we got k faulty in our hands.

Expected value:

## Popular continous distributions

### Uniform distribution

X is uniformly distributed in I=[a,b] if the probability of X being in any subset [a0, b0] is equal with the ratio between the width of the subset and I.

Expected value and variance:

### Exponential distribution

X follows an exponential distribution with parameter if the following holds:

Expected value and variance:

Expected value equation can be proven using parcial integration and L’Hopital rule.

### Normal distribution

X is normally distributed if there are parameters ) for which the PDF is:

X is standard normally distributed if . In this case the PDF and the CDF are:

Relation between and :

values can be found in tables. Also = 1 - because is symmetric.

Expected value and variance:

### distribution (Chi-square distribution)

If X1, X2, ..., Xn are all standard normally distributed fully independent random variables then we say that the random variable

is following a distribution with n degrees of freedom.

### Student distribution (t distribution)

If X1, X2, ..., Xn, Y are all standard normally distributed fully independent random variables then we say that the random variable

is following a Student (or tn) distribution with n degrees of freedom.

### Fisher distribution (F distribution)

If X is a distributed random variable with n degrees of freedom and Y is a

distributed random variable with k degrees of freedom we say that the random variable

is following a Fisher distribution with k and n parameters.

## Limit theorems

### Law of large numbers

The relative frequency (empirical probability) will converge to the actual probablity as we increase the number of trials.

If X1, X2, ..., Xn are independent random variables from the same distribution (samples) then we can say that the mean of those variables will converge to the expected value.

### de Moivre-Laplace

We make a lot of independent experiments. Let Xi the indicator that tells whether the ith trial was successful. (X is 1 if head, 0 if tail for example in case of a coin tossing game). For a successful trial (A goes off) we have p = P(A) chance.

or if we consider X as a binomially distributed random variable :

### Central limit (CLT)

X1, X2, ..., Xn are independent random variables that are following the same distribution. Assuming expected value E(X)and standard deviation we got the following formula:

### Hoeffding’s inequality

X1, X2, ..., Xn are independent random variables that are following the same distribution and can take values in between [a, b].

The Hoeffding inequality holds for small n-s (< 30) unline CLT:

## Fundamental concepts of statistics

**Population**: Set of similar items on which we’d like to make experiments. (Every FB user)

**Sample**: Set of individual items selected from the population.

**Types of statistical variables:**

Quantitative/Numerical: the number represents real amounts

Qualitative/Categorical data: the number represents grouping

Nominal var: The groups has no order (Man/Woman, Brand)

Ordinal var: The groups are ranked (rating)

We model the sample with n random independent variables with the same distribution. We can image that we are given n random instances of the population and from that, we’d like to know more about the population. The sampling should be representative.

Sample mean:

kth smallest:

Sample standard deviation:

Uncorrected:

Corrected (prefer this as it gives precise estimate!):

Sample range:

Empirical CDF:

**Glivenko-Cantelli’s theorem:**

As we increase the number of samples we can reconstruct the CDF of the population fully.

**sup** of a set is the lowest value for which every set-element is lower. The max must the be element of the set, the sup doen’t have to.

## Parameter estimation

We have an assumption on the distribution family that the population follows. From the samples we’d like to get the parameters of that distribution. (e.g.: mean and std for a distribution assumed Gaussian/normal)

Statistics (Tn) is a function of the samples X1, X2, ..., Xn. We say that the parameter is unbiased if

An estimation is consistent if .

An estimation is strongly consistent if the variance converges to zero:

An estimation T is unbiased (torzítatlan) if

An estimation T is asymptotically unbiased if

Between two unbiased estimators the more performant is the one with the faster convergence to zero variance. (namely, for fixed semi-large n the variance is lower).

**Cramer-Rao inequality:**

We saw that in case of a strongly consistent estimator we have 0 as variance in limit. The Cramer-Rao bound defines a boundary for fixed n for which the variance cannot fall below.

I is the Fisher information defined as following:

is the log likelihood function.

### Point estimation

We exactly determine the parameter that best fits our sample.

Maximum likelihood estimation:

We look for the parameters for which we have the highest chance to get our n samples.

is the parametrized PMF or PDF. We choose such a way that the (log)likelihood becomes maximal:

To determine we derivate the log(likelihood) function with respect to and make it equal with 0. This may give the stationary point for the maximum.

Method of moments:

It is a more general method than MLE, as it can find multiparametered distributions’ parameters. is the definition of the jth moment. A requirement to use MoM is that the parameters we’re looking for should be constructed from the moments in some way.

empirical moment (to model the expected value of Xj):

With empirical moments we can give estimates to the parameters (remember we have the assumption that each parameter is a function of moments )

**Example for MoM:**

Let’s make an estimation on the expected value () and the standard deviation () of a normal distribution with the method of moments.

First check out whether those parameters can be constructed from moments:

Yes they can, so we calculate the empirical moments:

From this we can have our estimations ():

### Interval estimation

In case of continous distributions we cannot assure that the point estimation gives us the real parameters (actually we have 0 chance for that), so we’d like an interval in which we will likely find the real parameters of the underlying distribution of the population.

In case of interval estimation, we use two sample statistics (Ln and Un) for which

In this case we call the interval (ln, un) the 100% confidence interval for .

**Example for interval estimation**

Let’s give a confidence interval for the expected value () of the normal distribution if the standard deviation () is known.

We know that follows a standard normal distribution.

Find the critical y for which:

From this we have the confidence interval for with rearranging:

To determine y we need to do the following ( is the std. normal CDF):

## Hypothesis testing

We’d like to review the truthfulness of statements. Usually these statement are referring to the overall population (null hypothesis) but we only have data samples. Also, often these statement are not completely true. We’d like to discover what is the contradiction level between the statement and our data when we don’t believe what the statement claims. Where’s the point where we sense significant level of discrepancy between the data we see and the statement? How can we measure significance level?

By default we accept every statement unless the data clearly says the opposite.

During hypothesis testing the goal is not to find the adequate mean and other parameters and compare with the one the null hypothesis says (e.g.: all car have 5,4 +/- 0.1 liter consumption) but rather to measure how much our data sample differs from the null hypothesis and what is the difference we can still tolerate.

We accept/reject a statement on a given significance level. (Significance level is the probability that that the statement is true but our data says the opposite)

H0 (null hypothesis) will say the distribution is in the set of .

H1 (alternative hypothesis) will say the distribution is the set of .

and are non-overlapping sets.

|  |  |  |
| --- | --- | --- |
|  | H0 accepted | H0 rejected |
| H0 is true | TN – correct decision | Type 1 error: False positive |
| H0 is false | Type2 error: False negative | TP – correct decision |

Number of false positives can be measure with the significance level (usually 0.05 - 0.1). Number of false negatives cannot be controlled and overall we have hard time estimating it.

**p-value:** p-value is the probability of getting a data sample at least as extreme as the sample statistics assuming that the null hypothesis is true. If the p value is lower than the significance level we have to reject the claims of the null hypothesis. Imagine having a data sample with mean of 25 whereas the null hypothesis stated that the mean is 20. Then the p value is:

p-value is the significance level of the test statistics where we just reject the null hypothesis. If this p-value is lower than the predefined significance value we reject the null hypothesis. It is another method besides critical values to test hypotheses.

If we reject the null hypothesis on a significance level (0.1 for example -> 90%) then we reject it on **every larger** significance level as well (0.15 -> 85%). So if our predefined significance level was 0.15 and we get that the p-value is 0.1, then we have to reject the null hypothesis.

### Z-test (z-próba, u-próba)

We have X1, X2, ..., Xn samples and we know that each of them is coming from a normal distribution with known standard deviation () but unknown mean ().

The null hypothesis says that the mean is . is the chosen significance level.

If H0 is true than u follows a standard normal distribution,

We substitute the Xi-s with our real numeric data and if we get that the then we accept the null hypothesis on the significance level . Once again if we accept the null hypothesis then that means that between the sample mean and the proposed mean there were no significant differences to reject the claim.

|  |  |
| --- | --- |
|  | 0.9 |
|  | 0.95 |
|  | 0.975 |

We use Z-tests on-non normally distributed data too if we have a lot of samples. You can justify the legitimacy of this method with the central limit theorem (CLT).

**Meaning of two-tailed (two-sided) test:**

A two-tailed z-test will check both if the mean is significantly greater than the one the null hypothesis claims or significantly lower. On the other side a **one-tail test** will only check if the sample mean is lower or higher than the claimed one.

two sample case:

We are given X1, X2, ..., Xn and Y1, Y2, ..., Ym independent and normally distributed samples. We know the distributions’ standard deviation.

The null hypothesis in this case claims that the expected values are the same. ()

By using the null hypothesis () and standardizing we get:

And from this point we can follow the previously introduced method:

### Student’s t-test

We are given X1, X2, ..., Xn  independent and normally distributed samples, but we don’t know the distributions’ standard deviation.

The null hypothesis says that the expected value is m0.

t follows a Student (t) distribution with n-1 degrees of freedom.

can be found in one of the charts on the last page.

two sample case:

We are given X1, X2, ..., Xn and Y1, Y2, ..., Ym independent and normally distributed samples. We don’t know the distributions’ standard deviation but we know that they can be considered equal (this should be reviewed with F-test).

The null hypothesis in this case claims that the expected values are the same. ()

This time t follows a Student (t) distribution with n+m-2 degrees of freedom.

paired sample t-test:

We have (X1, Y1), ... (Xn, Yn) samples (we take samples from the same objects twice for like comparing „before-after” or similar purposes). The assumption here is that is normally distributed:

The standard deviation () is unknown.

The null hypothesis in this case is that the expected value is zero ( or .

t follows a Student (t) distribution with n-1 degrees of freedom.

### F-test (Fisher-test)

We are given X1, X2, ..., Xn and Y1, Y2, ..., Ym independent and normally distributed samples. We don’t know neither the underlying distributions’ mean nor their standard deviation.

The null hypothesis questions whether we can consider .

We take the corrected variances , and our test statistics will be:

f follows a Fisher distribution with parameters n-1 and m-1.

can be found in one of the charts on the last page.

### Welch’s t-test

If F-test is rejected then we cannot use the t-test for two samples checking whether their expected value is the same.

We are given X1, X2, ..., Xn and Y1, Y2, ..., Ym independent and normally distributed samples. We don’t know neither the underlying distributions’ mean nor their standard deviation.

The null hypothesis in this case claims that the expected values are the same. ()

is Student distributed with f degrees of freedom where for f the following holds:

can be found in one of the charts on the last page.

### Non-parametric tests introduction, common problems

non-parametric tests:We consider the distribution completely unknown.

Due to the lack of information non-parametric tests only works on large data. Our information is really general in this case (standard deviation is finite, the distribution is continous, etc.)

* **Goodness of fit (illeszkedésvizsgálat):** When measuring goodness of fit, we’d like to assure that our sample is following some hypothetical distribution. (null hypothesis). For this we can use Chi-Square test or the single-sample Kolmogorov-Smirnov test.
* **Test of independence (függetlenségvizsgálat):** We check whether the examined variables are independent. For this we can use Chi-Square test.
* **Test of homogenity (homogenitásvizsgálat):** We have two samples and we’d like to check whether they come from the same distribution. For this we can use Chi-Square test, two sample Kolmogorov-Smirnov test, Mann-Whitney test, Wilcoxon test, Kruskal-Wallis test, Friedman test

### Kolmogorov-Smirnov test

One sample for checking goodness of fit:

We have a sample of X1, X2, ..., Xn  random variables which come from a continous distribution. The null hypothesis say that the sample follows the hypothetical CDF .

We can omit the term, so we must only compare the supremum values.

The critical values can be found with the linked code snippet in RStudio.

Two sample for test of homogenity:

We have a sample of X1, X2, ..., Xn and Y1, Y2, ..., Ym random variables which come from a continous distribution. We’d like to know whether they have the same distribution.

Where and are the empirical CDFs of X and Y.

### Wilcoxon signed-rank test

Homogenity test that can be used for ordinal variables (aka variables that are ranked, and ranking contains additional information).

We have a sample of X1, X2, ..., Xn  random ordinal variables which come from a continous distribution. We need to decide whether we can accept the proposed median of:

H0 says that the median should be med0:

(**remove the zero differences!**)

is the ranking of in the ordered samples of a-s ().

*In case of identical values (1 1 3 5) we will substitute the rankings of identical values with the ranking average so instead of (1 2 3 4) the result will be (1.5 1.5 3 4). In R programming this effect can be achieved by using rank(x, ties.method=”average”).*

ad

When the amount of samples is low (at most 20 elements):

We check the critical values for from a table (qsignrank). The critical value depends on the number of samples and the level of significance. We accept the null hypothesis if:

When we have sufficiently large amount of samples:

Paired sample Wilcoxon test:

We have (X1, Y1), ... (Xn, Yn) samples of ordinals. We’d like to decide whether their distributions are the same. In other words the null hypothesis questions if the difference of the pairwise samples has the median of 0 or not on some significance level.

This problem can be transformed into the problem we saw during the one sample case by redefining the following way:

### Mann-Whitney U test

We have a sample of X1, X2, ..., Xn and Y1, Y2, ..., Ym random ordinal variables. We’d like to know whether they have the same distribution (or the same median).

First we create the sample Z: by forming a union between X and Y. Then we rank the samples by their absolute rank in the ordered Z sample.

Let C be the critical value for the n and m on the given significance level.

If then we accept that their distribution is the same.

For large samples (over 20) the following statistics could be considered standard normal:

The uc values can be found with the help of the proper table or qnorm(1-eps/2)

### Chi-square test

goodness of fit:

We have X1, X2, ..., Xn samples and we’d like to determine whether they follow the hypotethical CDF (null hypothesis says that they do!)

As a preparation we split the real numbers into **r** disjoint subset:

Then we calculate the probability of an X falling into the interval in case of the hypothetical CDF .

will be the number of data samples falling into the interval . We would expect it to be approximately .

The test statistics will be:

T follows a chi-square distribution with degrees of freedom. We can find the critical value by plugging in the significance level and degrees of freedom. Approximated or calculated parameters decrease the degrees of freedom (empirical mean, empirical variance from the samples).

If then we accept the null hypothesis.

test of indepence:

We have (X1, Y1), ... (Xn, Yn) paired samples. We’d like to check whether the components of the samples are independent or there is a significant bound.

claims that

We split the space of real numbers into k and l partitions:

Define to be the number of (X,Y) pairs for which .

Define the empirical probabilities the following way:

Our test statistics will be:

Tn follows a chi-square distribution with degrees of freedom. We can find the critical value by plugging in the significance level and degrees of freedom.

If then we accept the null hypothesis and by that, we consider X and Y independent.

test of homogenity:

We have independent samples of X1, X2, ..., Xn and Y1, Y2, ..., Ym random variables. We’d like to check if their distributions are the same (null hypothesis).

We split the space of real numbers into r partitions:

Define as the number of samples from X and Y (respectively) falling into the partition .

The constructed test statistics (T) will be:

T follows a chi-square distribution with degrees of freedom. We can find the critical value by plugging in the significance level and degrees of freedom.

If then we accept the null hypothesis and by that, we consider X identically distributed as Y.

### ANOVA (Analysis of variance)

ANOVA is a method for comparing several normally distributed populations’ mean. An important criteria here is that the variance should be equal for every distribution.

two sample Student (t) test is a special case of the ANOVA method (two groups).

ANOVA checks whether the deviation is caused by pure randomness or there is some connection between the groups (eg.: difference between means). Doing t-tests pairwise is feasible to do on low group count, but the computational requirement increases quadratically as we try to use it on more and more groups. Also, we have high chance to get Type 1 errors while doing independent t-tests rendering the significance level useless.

We have the samples coming from the distributions respectively. Also, we know that .

Define m as:

In this case (model of ANOVA):

here is the deviation of the group (group effect)

is the error term.

One-way ANOVA (egyszempontos varianciaanalízis):

H0: The null hypothesis claims that .

**Can be proved that**

Q sums up n normal variables but computes one: with n-1 degrees of freedom.

sums up p normal variable but computes one: with p-1 degrees of freedom.

sums up n normal variable but computes p: with n-p degrees of freedom.

Our test statistics will be:

T follow a Fisher distribution with (p-1), (n-p) parameters.

If this is fulfilled for T then we accept that .

### Bartlett-test

To do ANOVA we have to ensure that the variances are identical.

We are given the samples coming from the distributions respectively.

H0: The null hypothesis claims that the variances are the same: .

Define the in-group empirical variance and total empirical variance the following way:

The test statistics will be:

This T approximately distributed with (p-1) degrees of freedom.

If this is fulfilled for T then we accept that .

### Friedman-test

The Friedman-test is the generalization of the paired sample Wilcoxon sign-rank test. For various **ordinal** groups (samples) we’d like to determine if they are following the same distribution (basically means that they have identical medians) or not.

We are given the samples . (n groups, each containing p random variables (threatments))

We order each sample in increasing order and make a ranking. Let be the ranking of the ith variable in the jth sample group.

For each ’column’ we sum up the ranks:

Our test statistics will be:

In case of low amount of samples we look for the critical values in a table.

Otherwise we know that T is distributed with (p-1) degrees of freedom.

### Kruskal-Wallis test

Kruskal-Wallist test is the generalization of the two sample Wilcoxon (Mann-Whitney) test.We’re given the samples . (n groups, each containing p random variables (threatments)). In this case the **ordinal** samples are completely **independent** from each other. And the null hypothesis claims that they are all coming from the same distribution (aka they have the same median).

H0:

First order the concatenated samples and define the rank numbers. The concatenated sample will contain N element, where:

is the sum of the ranks of the random variables in the ith sample. Our final test statistics will be:

~~If the amount of samples is low, we use the Kruskal-Wallis table to find the critical values~~, otherwise we know that T follows a distribution with p-1 degrees of freedom.

### Exact tests introduction

In case of non-parametric tests we only know the asymptotical CDF and by that we can make huge errors on low sample count. Exact tests try to tackle this problem.

In case of exact test we try to calculate the chance of Type 1 errors with combinatorical methods ().

### Fisher’s exact test

If we have too little samples to do the test of independence, we prefer proving independence with manual probability calculation, mostly with hypergeometric distribution.

Gives lower (thus better) significance level than test and helps us avoiding Type 1 errors.

### Binomial test

We have X1, X2, ..., Xn samples all of them having the indicator distribution with p probability (parameter). This type of exact test is useful when it is costy to make samples (check the quality of ammunition for example).

The null hypothesis (H0) says that:

We sum up how many times our events happened.

We check the binomial table for the critical value corresponding to the significance level . On large samples (30+) we can use z-test (u-test):

T is approximately a standard normal distribution. We can find its critical values by taking a look at the proper CDF. Also, as an alternative solution we can calculate the p-value (getting a result this extreme) and compare it with the significance level. If the p-value is less then the significance level then we reject the null hypothesis.

## Regression Analysis

If we find out that two variables are dependent from each other then we’d like to measure the bound between them. First time with a number, then describing it with a function.

By applying Cauchy-Bunyakovszky-Schwarz\* inequality ():

\*

If X and Y are independent the correlation is 0 but the reversed case doesn’t hold.

If corr(X, Y) is 1 or -1, then the connection between X and Y is linear:

If we have the (X1, Y1), (X2, Y2), ..., (Xn, Yn) statistical samples from the X, Y random variable we can define the empirical covariance and correlation:

If we get a non-zero correlation from the data first we need to guarantee that is not by randomness. So we make up a null hypothesis claiming that the correlation was 0. And test the following statistics:

Which is Student (t) distributed with n-2 degrees of freedom.

The goal of regression: We have a random variable Y and we would like to approximate it by a function of the X1, X2, ..., Xp variables. By approximation we mean an f function for which the following expression is minimal:

f is coming from a family of functions (linear, quadratic, logarithmic, exponential).

linear regression between X and Y:

The minimum of the h function can be at that point where the gradient is the zero vector:

Solving this we get that the regressor line is:

The task can be explained the following way too:

We are looking for a and b such that the expected value of the squared error term () is minimal:

In practice we have the samples (X1, Y1), (X2, Y2), ..., (Xn, Yn) from the X and Y random variables. Let:

is coming from an specific distribution what is independent from every has 0 mean and standard deviation. By constructing a summed error function (loss function) we can describe our new goal, which is to minimize:

Quality measures:

Coefficient of determination:

Coefficient of determination will qualify how good the linear regression explains data. It measures how good we can guess the value of knowing compared to our naive guess which would be the average of the Y-s we got so far. If this coefficient is 0.95 then the explains 95% of the ’s value.

SST (total sum of squares)

SSE (sum of squares of errors / residuals)

SSR (sum of squares of regression)

**SST** = **SSR** + **SSE**

**Theorem**: The parameters a, b are approximated by and in an unbiased manner.

**Theorem** (Gauss-Markov): The approximations for a,b we get from the least squares method are the most efficient ones (aka they have to lowest standard deviation)

**Theorem**: An approximation for the error’s variance:

If we assume that the error is normally distributed then we can estimate the standard errors of parameters:

By knowing the empirical variances we can define a confidence interval for estimating a,b:

The same way we can define a confidence interval around using :

We’d like to ensure that between X and Y there’s an actual bound so we hypothesis test and on a proper significance level:

H0 claims that .

With the same approach we can accept or reject the null hypothesis for .

A special case when we test the hypothesis claiming that . If we fail to reject this one that would mean that there are no linear connection between X and Y.

H0 claims that .

T is Fisher distributed with parameters 1 and (n-2).

Quite often our data does not follow a linear trend. So the function f is not neccessarily an aX+b type function but can be:

* polynomial:
* exponential I:
* exponential II:

Some of these function families can be converted into the previously introduced linear kind of function:

So if we name the random variable ln(Y) as Z we got a classic linear regression task between Z~X and a, ln(b) can be found.

### Multivariable linear regression

Y is the dependent variable (eredményváltozó).

**Dependent variable:** Upon manipulating the X independent variables we expect change in the value of the dependent variable. (effect)

**Independent variable:** The variable is stable and unaffected by other independent variables. (cause, predictors)

are the independent (**uncorrelated**) variables (magyarázó változók).

The model:

Given the data

Here is the ith attribute in the jth row.

Where **X** is a sized matrix where the first column is filled with ones.

We’d like to define -s such that the sum of -s are minimal. The errors are coming from various sources: missed independent variable, redundant independent variable, wrong function family, measurement errors, random unexplainable effects. The error can be considered normally distributed.

The loss function is:

Can be computed that it will have a minimum at:

We can determines which independent variable had the greatest contribution (needs abs) to the value of the dependent variable by normalizing :

Where is the uncorrected sd of Y-s and is the uncorrected sd of -s (standard deviation of the ith attribute/column).

Coefficient of determination():

SST (total sum of squares)

SSE (sum of squares of errors / residuals)

SSR (sum of squares of regression)

**SST** = **SSR** + **SSE**

Regular coefficient of determination would be close to one when adding a lot of extra unneeded independent variables. By punishing this kind of addition we get a more correct measure which is called the adjusted coefficient of determination:

This is less then or equal with 1, but might go below 0.

Before utilizing the result, we’d like to check whether the a factors can be considered different than 0:

We use a global ANOVA to accept or reject the null hypothesis.

T follows a Fisher distribution with parameters k, (n-k-1).

If it turns out that our model is not a null model (so we previously rejected ), then we can one by one check the relevance of the variables:

This time T follows a Student (t) distribution with n-k degrees of freedom.

The variance of errors:

The error of parameters:

### Model building

How do we decide which independent variables should we add to the mode?

SPSS and statistical softwares use partial F-test in which we check whether introducing a new (pth) independent variable increases the coefficient of determination or not.

Out test statistics:

T is Fisher distributed with parameters 1, n-p-1.

Automatic model building techniques:

* ALL VAR. ENTERED: We enter all variables in a block.
* FORWARD SELECTION: The variables added sequentially depending on a criterion (the one with the lowest rejected significance level under PIN).
* BACKWARD ELIMINATION: We enter all variables and then sequentially erase variables (the one with the highest rejected significance level over POUT)
* STEPWISE SELECTION: A method that both use PIN and POUT.

Multicollinearity:

Unwanted linear connection between independent variables.

Quality indicators of multicollinearity:

**Tolerance**: Measures how tightly the ith independent variable is defined by others. A tolerance close to zero is bad, it would mean that there is an association between the variables in a form of a function. For each independent variable we construct this function like we previously did by finding the parameters that minimizes the loss funcion. For every regression made we calculate the coefficient of determination (denoted as for the regression of ). The value of tolerance is 1-. Tolerance above 0.1 is acceptable.

**Variance inflation factor (VIF)**: This measure is the reciprocal of Tolerance. The ideal value of VIF is 1 but until 10 it can be accepted.

**Condition index (CI)**: A statistics derived from the correlation matrix of the s. CI is the square root of the ratio of the largest and the smallest eigenvalue. CI > 15 means a strong correlation so C < 15 is acceptable only.

### Outlier detection

Removing outliers are as important as removing unneeded s. To detect outliers we computer the so-called levarage matrix the following way:

The diagonal elements () defines the strongness of the ith data point’s impact on the regressor’s estimation. The ith data point is casual if and significant if .

The state of the data point depending on :

* > 0.5: Outlier
* > 0.2: Suspicious and risky to involve in the loss function
* < 0.2: Can be added to our analysis with no worried

### Binary logistic regression

We use logistic regression to anticipate an event that happend with p probablity. Binary regression assigns 0 to probablity values and 1 to the ones which are . This is equalent if we examine the sign of the expression (negative 0, positive 1).

## Data reduction

Helps reducting the size of the data matrix, thus increasing the computational efficiency. To achieve this we might introduce new hidden variables assembled from various, somehow dependent attributes. We do this a way that our statistical explorations will be still valid for the underlying population. Data reduction can help eliminating columns (attributes/variables) as well as rows (cases).

### Factor analysis

We have large amount of dependent variables containing superflous (and redundant) information. We’d like to represent the connection with low number of uncorrelated factor variables.

Correlation matrix:

Anti image matrix:

The element other than the diagonal show how independent two variables’ variance are. The lower, the better. The diagonal elements (MSA) show how much the variance of a random variable is determined by the others. The higher, the better as we couldn’t compress independent enough variables.

Kaiser-Meyer-Olkin (KMO) value:

A complex measure made up from empirical correlations and empirical partial correlations which tells how much the variables depend from each other. This has to be over a limit (0.5, but rather 0.8) to even consider using factor analysis or PCA.

Bartlett test of sphericity:

The null hypothesis will claim that the correlation matrix can be considered the identity matrix. We reject this hypothesis if the value of the test statistics is large so the significance level for approval is super low. If we cannot reject H0, then we should forget about using factor analysis or PCA once again.

model of factor analysis:

We have X1, X2, ..., Xn variables put into the observation (X) matrix.

A: Loading matrix (p, k)

F: Factor matrix (k, n)

U: Unique factor vector (error term) (p, n)

### PCA (principal component analysis)

An algorith that will determine the component of the factor analysis model.

Let be the covariance matrix of X1, X2, ..., Xn. By knowing that is symmetric we can spectrally decompose it.

are the eigenvectors and is an orthonormal basis. In this case will be the X’s principal component where U is a matrix of column vectors. Y[1] is the direction with the largest variance, then comes Y[2], etc.

**Theorem:** Dimensionality reduction done with PCA algorithm minimizes the information loss.

Rotation of the load matrix:

QUARTIMAX: The number of variables will the lowest possible.

VARIMAX: The number of variables for which we have heavily weighted factors will be the low.

### MDS (multi dimensional scaling)

When doing PCA we assumed that our variables are not categorical or ordinal.

In case of MDS the goal is to make a geometrical representation of similarities that would display the hierarchy between the variables the most obvious and clear way.

So we have data points in the n-dimensional space and we’d like to find point in 2D or 3D that has matching distance metrics than the given data points. In short: we’d like to represent out data in low dimensional space (possibly 3D) with minimal distortion in the representation.

We have **n** data points: each of the has **p** attributes (can be said that we have n point in the p dimensional space). We’d like a lower dimension representation.

Distance matrix:

d is the distance function in the p-dimensional space.

We want points in a lower dimensional space for which:

is low. Here, is the distance function in the lower dimensional space.

To measure the quality of the MDS procedure we have 3 metrics.

**s-stress:**

**stess:**

Stresses must be low for a good representation.

**RSQ:**

The squared correlation of D matrix and the representation’s distance matrix. If large then all the variances can be respresented pretty well.

### Discrimininant analysis

We have the k datapoint already clustered (each datapoint has p attribute). We’d like to linearly separate (linear combination of the attributes) those cluesters with functions. Doing this can help classify new, not yet seen, datapoints.

## Time series data

The random variables are indexed by timesteps.

Can be used for prediction or to complement missing timesteps.

The CDF becomes 2 parametered:

Autocovariance & autocorrelation:

Autocovariance & autocorrelation function (ACF):

A time series is autouncorrelated (has no memory) if:

Partial autocorrelation function (PACF): Correlation between without knowing the variables of the timesteps in between.

First task what we should do when analysing time series is to prove that the time series has actually autocorrelating elements, so there is explainable value difference depending on the timestep.

Define as the indicator indicating 1 if , 0 otherwise. If -s are independent then the measure standardized is standard normally distributed:

The null hypothesis says that the is indeed , so we all the X-es are indepenedent, so there are no time series to analyze. It would be preferred if we could just reject this null hypothesis. (**Váltakozásmódszer,** csúcsmódszer, előjelmódszer /\* TODO: need english translations! \*/)

### Models for time series

**Deterministic model**:

The time series can be fully described by a function of t, the randomness is only present at the error term.

Decomposition model: The time series is decomposed to different components which are either summed up or multiplied together to reconstruct the time series.

* Additive:
* Multiplicative:

T, C, S and I are the trend, cyclical, seasonal and irregular (noise) component.

trend: A long-term impact on the whole timeline.

seasonality: Short-term periodical effect (increasing beer sales during summer).

cyclicity: Long-term periodical effect that happens irregularly (economical crisis).

irregularity (noise): has 0/1 expected value and low standard deviation.

**trend analysis**:

The goal is to find the trend an subtract it from the time series.

Moving average:

We define the trend with the help of the moving average of the original time series. Suppose that our time series it T long and our sampling window is k wide. The trend will be:

By doing averaging we get rid of the seasonal fluctuation as well as the irregularity.

Linear trend:

We look for a trend in the following format:

With the method of least square we are trying to find the value of a and b for which the:

expression is minimal (similarly to regression).

**seasonality:**

Assume we already found the trend and remove from the time series, so we have a simplified dataset where there are no trend influence. The time series can be written formally as follows:

Where d is the seasonal component an X is a stationary random influence in the given year and month and is:

Here n is the number of years and m is the number of months or any other hierarchical metric. To calculate the seasonal component d (seasonal index) we first define the seasonal average (for every month we compute the averages of yearly data):

**ciclycity:**

Advanced! TODO

**Stochastic model:**

The time series is not only the function of time but the previous timesteps has major impact on the wheregoings. Randomness has reponsibility forming the time series, randomness is not only noise. So while the deterministic model can be written as:

The stochastic model has recursion in its formal definition and the variance is not fixed but will increase along with the time:

// TODO ARMA ARIMA

# Neural Networks

## Forward and Backpropagation

### Math

chain rule:

### Forward propagation on XOR N.N.



### Backpropagation on XOR N.N.

## Egyéb

# Web scraping

Web scraping is used for extracting data from websites directly from the raw HTML code.

Online JS beautifier (<https://beautifier.io/>) can help to go over unformatted HTML codes.

## BeautifulSoup

pip install beautifulsoup4

pip install lxml # to parse non-perfect/broken HTML codes as well

**from** bs4 **import** BeautifulSoup

**from** urllib.request **import** urlopen

**import** re

connection = urlopen(URL\_PAGE1)  
raw\_html = connection.read()  
connection.close()

soup = BeautifulSoup(raw\_html, **"lxml"**)  
cars = soup.find\_all(**"div"**, {**"class"**: re.compile(**"row talalati-sor\*"**)})x = cars[0].find(**"div"**, {**"class"** : **"vetelar"**})  
x.text **#will return the text from the div**

x.div.div.a.img[**"title"**] **#will return the title attr of the image tag**

# Math auxiliary

## Eigenvalues and eigenvectors

**A** is a square matrix, **x** is a non null vector. is a scalar called eigenvalue. The is the eigenvector associated with the eigenvalue .

Trace are the sum of the diagonal entries of a square matrix.

**Theorem:** The sum of eigenvalues is exactly the trace.

**Theorem:** The product of eigenvalues will result in the matrix’ determinant.

## Rank of matrix

Row rank: number of linearly independent rows. ().

Column rank: number of linearly independent columns. ().

Often the row rank is equal to the column rank, so we simply refer to this common value as the rank of the matrix.

A matrix full of zeros has a rank of 0. Zero vectors do not increase the rank.

An matrix is invertible if and only if the rank of the matrix is .

## Kernel and image

Kernel (nullspace) is the set of vectors for which a given linear mapping outputs the zero vector.

// TODO

# Standard normal CDF () chart.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***z*** | **+ 0.00** | **+ 0.01** | **+ 0.02** | **+ 0.03** | **+ 0.04** | **+ 0.05** | **+ 0.06** | **+ 0.07** | **+ 0.08** | **+ 0.09** |
| **0.0** | 0.50000 | 0.50399 | 0.50798 | 0.51197 | 0.51595 | 0.51994 | 0.52392 | 0.52790 | 0.53188 | 0.53586 |
| **0.1** | 0.53983 | 0.54380 | 0.54776 | 0.55172 | 0.55567 | 0.55962 | 0.56360 | 0.56749 | 0.57142 | 0.57535 |
| **0.2** | 0.57926 | 0.58317 | 0.58706 | 0.59095 | 0.59483 | 0.59871 | 0.60257 | 0.60642 | 0.61026 | 0.61409 |
| **0.3** | 0.61791 | 0.62172 | 0.62552 | 0.62930 | 0.63307 | 0.63683 | 0.64058 | 0.64431 | 0.64803 | 0.65173 |
| **0.4** | 0.65542 | 0.65910 | 0.66276 | 0.66640 | 0.67003 | 0.67364 | 0.67724 | 0.68082 | 0.68439 | 0.68793 |
|  |  |  |  |  |  |  |  |  |  |  |
| **0.5** | 0.69146 | 0.69497 | 0.69847 | 0.70194 | 0.70540 | 0.70884 | 0.71226 | 0.71566 | 0.71904 | 0.72240 |
| **0.6** | 0.72575 | 0.72907 | 0.73237 | 0.73565 | 0.73891 | 0.74215 | 0.74537 | 0.74857 | 0.75175 | 0.75490 |
| **0.7** | 0.75804 | 0.76115 | 0.76424 | 0.76730 | 0.77035 | 0.77337 | 0.77637 | 0.77935 | 0.78230 | 0.78524 |
| **0.8** | 0.78814 | 0.79103 | 0.79389 | 0.79673 | 0.79955 | 0.80234 | 0.80511 | 0.80785 | 0.81057 | 0.81327 |
| **0.9** | 0.81594 | 0.81859 | 0.82121 | 0.82381 | 0.82639 | 0.82894 | 0.83147 | 0.83398 | 0.83646 | 0.83891 |
|  |  |  |  |  |  |  |  |  |  |  |
| **1.0** | 0.84134 | 0.84375 | 0.84614 | 0.84849 | 0.85083 | 0.85314 | 0.85543 | 0.85769 | 0.85993 | 0.86214 |
| **1.1** | 0.86433 | 0.86650 | 0.86864 | 0.87076 | 0.87286 | 0.87493 | 0.87698 | 0.87900 | 0.88100 | 0.88298 |
| **1.2** | 0.88493 | 0.88686 | 0.88877 | 0.89065 | 0.89251 | 0.89435 | 0.89617 | 0.89796 | 0.89973 | 0.90147 |
| **1.3** | 0.90320 | 0.90490 | 0.90658 | 0.90824 | 0.90988 | 0.91149 | 0.91308 | 0.91466 | 0.91621 | 0.91774 |
| **1.4** | 0.91924 | 0.92073 | 0.92220 | 0.92364 | 0.92507 | 0.92647 | 0.92785 | 0.92922 | 0.93056 | 0.93189 |
|  |  |  |  |  |  |  |  |  |  |  |
| **1.5** | 0.93319 | 0.93448 | 0.93574 | 0.93699 | 0.93822 | 0.93943 | 0.94062 | 0.94179 | 0.94295 | 0.94408 |
| **1.6** | 0.94520 | 0.94630 | 0.94738 | 0.94845 | 0.94950 | 0.95053 | 0.95154 | 0.95254 | 0.95352 | 0.95449 |
| **1.7** | 0.95543 | 0.95637 | 0.95728 | 0.95818 | 0.95907 | 0.95994 | 0.96080 | 0.96164 | 0.96246 | 0.96327 |
| **1.8** | 0.96407 | 0.96485 | 0.96562 | 0.96638 | 0.96712 | 0.96784 | 0.96856 | 0.96926 | 0.96995 | 0.97062 |
| **1.9** | 0.97128 | 0.97193 | 0.97257 | 0.97320 | 0.97381 | 0.97441 | 0.97500 | 0.97558 | 0.97615 | 0.97670 |
|  |  |  |  |  |  |  |  |  |  |  |
| **2.0** | 0.97725 | 0.97778 | 0.97831 | 0.97882 | 0.97932 | 0.97982 | 0.98030 | 0.98077 | 0.98124 | 0.98169 |
| **2.1** | 0.98214 | 0.98257 | 0.98300 | 0.98341 | 0.98382 | 0.98422 | 0.98461 | 0.98500 | 0.98537 | 0.98574 |
| **2.2** | 0.98610 | 0.98645 | 0.98679 | 0.98713 | 0.98745 | 0.98778 | 0.98809 | 0.98840 | 0.98870 | 0.98899 |
| **2.3** | 0.98928 | 0.98956 | 0.98983 | 0.99010 | 0.99036 | 0.99061 | 0.99086 | 0.99111 | 0.99134 | 0.99158 |
| **2.4** | 0.99180 | 0.99202 | 0.99224 | 0.99245 | 0.99266 | 0.99286 | 0.99305 | 0.99324 | 0.99343 | 0.99361 |
|  |  |  |  |  |  |  |  |  |  |  |
| **2.5** | 0.99379 | 0.99396 | 0.99413 | 0.99430 | 0.99446 | 0.99461 | 0.99477 | 0.99492 | 0.99506 | 0.99520 |
| **2.6** | 0.99534 | 0.99547 | 0.99560 | 0.99573 | 0.99585 | 0.99598 | 0.99609 | 0.99621 | 0.99632 | 0.99643 |
| **2.7** | 0.99653 | 0.99664 | 0.99674 | 0.99683 | 0.99693 | 0.99702 | 0.99711 | 0.99720 | 0.99728 | 0.99736 |
| **2.8** | 0.99744 | 0.99752 | 0.99760 | 0.99767 | 0.99774 | 0.99781 | 0.99788 | 0.99795 | 0.99801 | 0.99807 |
| **2.9** | 0.99813 | 0.99819 | 0.99825 | 0.99831 | 0.99836 | 0.99841 | 0.99846 | 0.99851 | 0.99856 | 0.99861 |
|  |  |  |  |  |  |  |  |  |  |  |
| **3.0** | 0.99865 | 0.99869 | 0.99874 | 0.99878 | 0.99882 | 0.99886 | 0.99889 | 0.99893 | 0.99896 | 0.99900 |
| **3.1** | 0.99903 | 0.99906 | 0.99910 | 0.99913 | 0.99916 | 0.99918 | 0.99921 | 0.99924 | 0.99926 | 0.99929 |
| **3.2** | 0.99931 | 0.99934 | 0.99936 | 0.99938 | 0.99940 | 0.99942 | 0.99944 | 0.99946 | 0.99948 | 0.99950 |
| **3.3** | 0.99952 | 0.99953 | 0.99955 | 0.99957 | 0.99958 | 0.99960 | 0.99961 | 0.99962 | 0.99964 | 0.99965 |
| **3.4** | 0.99966 | 0.99968 | 0.99969 | 0.99970 | 0.99971 | 0.99972 | 0.99973 | 0.99974 | 0.99975 | 0.99976 |
|  |  |  |  |  |  |  |  |  |  |  |
| **3.5** | 0.99977 | 0.99978 | 0.99978 | 0.99979 | 0.99980 | 0.99981 | 0.99981 | 0.99982 | 0.99983 | 0.99983 |
| **3.6** | 0.99984 | 0.99985 | 0.99985 | 0.99986 | 0.99986 | 0.99987 | 0.99987 | 0.99988 | 0.99988 | 0.99989 |
| **3.7** | 0.99989 | 0.99990 | 0.99990 | 0.99990 | 0.99991 | 0.99991 | 0.99992 | 0.99992 | 0.99992 | 0.99992 |
| **3.8** | 0.99993 | 0.99993 | 0.99993 | 0.99994 | 0.99994 | 0.99994 | 0.99994 | 0.99995 | 0.99995 | 0.99995 |
| **3.9** | 0.99995 | 0.99995 | 0.99996 | 0.99996 | 0.99996 | 0.99996 | 0.99996 | 0.99996 | 0.99997 | 0.99997 |
|  |  |  |  |  |  |  |  |  |  |  |
| **4.0** | 0.99997 | 0.99997 | 0.99997 | 0.99997 | 0.99997 | 0.99997 | 0.99998 | 0.99998 | 0.99998 | 0.9999 |

# Student-t CDF chart

| **Conf. Level** | **90%** | **95%** | **98%** | **99%** |
| --- | --- | --- | --- | --- |
| **One Tail** | **0.050** | **0.025** | **0.010** | **0.005** |
| **Two Tail** | **0.100** | **0.050** | **0.020** | **0.010** |
| **df = 1** | 6.314 | 12.706 | 31.821 | 63.657 |
| **2** | 2.920 | 4.303 | 6.965 | 9.925 |
| **3** | 2.353 | 3.182 | 4.541 | 5.841 |
| **4** | 2.132 | 2.776 | 3.747 | 4.604 |
| **5** | 2.015 | 2.571 | 3.365 | 4.032 |
| **6** | 1.943 | 2.447 | 3.143 | 3.707 |
| **7** | 1.895 | 2.365 | 2.998 | 3.499 |
| **8** | 1.860 | 2.306 | 2.896 | 3.355 |
| **9** | 1.833 | 2.262 | 2.821 | 3.250 |
| **10** | 1.812 | 2.228 | 2.764 | 3.169 |
| **11** | 1.796 | 2.201 | 2.718 | 3.106 |
| **12** | 1.782 | 2.179 | 2.681 | 3.055 |
| **13** | 1.771 | 2.160 | 2.650 | 3.012 |
| **14** | 1.761 | 2.145 | 2.624 | 2.977 |
| **15** | 1.753 | 2.131 | 2.602 | 2.947 |
| **16** | 1.746 | 2.120 | 2.583 | 2.921 |
| **17** | 1.740 | 2.110 | 2.567 | 2.898 |
| **18** | 1.734 | 2.101 | 2.552 | 2.878 |
| **19** | 1.729 | 2.093 | 2.539 | 2.861 |
| **20** | 1.725 | 2.086 | 2.528 | 2.845 |
| **21** | 1.721 | 2.080 | 2.518 | 2.831 |
| **22** | 1.717 | 2.074 | 2.508 | 2.819 |
| **23** | 1.714 | 2.069 | 2.500 | 2.807 |
| **24** | 1.711 | 2.064 | 2.492 | 2.797 |
| **25** | 1.708 | 2.060 | 2.485 | 2.787 |
| **26** | 1.706 | 2.056 | 2.479 | 2.779 |
| **27** | 1.703 | 2.052 | 2.473 | 2.771 |
| **28** | 1.701 | 2.048 | 2.467 | 2.763 |
| **29** | 1.699 | 2.045 | 2.462 | 2.756 |
| **30** | 1.697 | 2.042 | 2.457 | 2.750 |
| **40** | 1.684 | 2.021 | 2.423 | 2.704 |
| **50** | 1.676 | 2.009 | 2.403 | 2.678 |
| **60** | 1.671 | 2.000 | 2.390 | 2.660 |
| **70** | 1.667 | 1.994 | 2.381 | 2.648 |
| **80** | 1.664 | 1.990 | 2.374 | 2.639 |
| **90** | 1.662 | 1.987 | 2.368 | 2.632 |
| **100** | 1.660 | 1.984 | 2.364 | 2.626 |
| **+inf** | 1.645 | 1.960 | 2.326 | 2.576 |

# CDF charts in R

Critical U values:

Critical T values:

Critical F values:

Critical KS values: =

(needs: "BoutrosLab.plotting.general")

Critical values for low sample Wilocoxon test:

=

Critical values for low sample Mann-Whitney test:

=

Critical CSq values:

Critical Friedman value:

(needs: 'SuppDists')

Critical binom value:

# 