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

Table of contents

[1 Prerequisites 6](#_Toc70716696)

[1.1 Python 6](#_Toc70716697)

[1.1.1 Data Structures 6](#_Toc70716698)

[1.1.2 Functions and lambdas 7](#_Toc70716699)

[1.1.3 Error handling 7](#_Toc70716700)

[1.1.4 String operations 8](#_Toc70716701)

[1.1.5 IO 8](#_Toc70716702)

[1.1.6 Classes 8](#_Toc70716703)

[1.1.7 Pickle – serialization framework – 10](#_Toc70716704)

[1.1.8 Generator, yield, async-await and coroutines 10](#_Toc70716705)

[1.1.9 Threading 10](#_Toc70716706)

[1.2 Numpy 10](#_Toc70716707)

[1.2.1 Arrays (vectors and matrices) 10](#_Toc70716708)

[1.2.2 Shaping 11](#_Toc70716709)

[1.2.3 Stacking 11](#_Toc70716710)

[1.2.4 Linalg 11](#_Toc70716711)

[1.2.5 Miscellaneous (random, linspace, math) 11](#_Toc70716712)

[1.3 Pandas 12](#_Toc70716713)

[1.3.1 Series 12](#_Toc70716714)

[1.3.2 DataFrame 12](#_Toc70716715)

[1.4 Matplotlib 14](#_Toc70716716)

[1.4.1 Lines and 2D functions 14](#_Toc70716717)

[1.4.2 Multiple plots on a figure 15](#_Toc70716718)

[1.4.3 Scatter plots, histograms and images 16](#_Toc70716719)

[1.4.4 Texts and Legends 17](#_Toc70716720)

[1.4.5 ... 17](#_Toc70716721)

[1.5 Jupyter notebook 17](#_Toc70716722)

[1.6 R 18](#_Toc70716723)

[1.6.1 Attributes, Basic types, NA and NaN 18](#_Toc70716724)

[1.6.2 Complex types 18](#_Toc70716725)

[1.6.3 Subsetting, indexing 20](#_Toc70716726)

[1.6.4 Date & time 21](#_Toc70716727)

[1.6.5 Reading data, Files 22](#_Toc70716728)

[1.6.6 Control structures 22](#_Toc70716729)

[1.6.7 Functions 23](#_Toc70716730)

[1.6.8 Loop functions (apply-s) 23](#_Toc70716731)

[1.6.9 Misc 24](#_Toc70716732)

[1.7 SQL (Microsoft Server SQL) 25](#_Toc70716733)

[1.7.1 Data types 25](#_Toc70716734)

[1.7.2 Create and alter schemes 26](#_Toc70716735)

[1.7.3 Insert rows into table 26](#_Toc70716736)

[1.7.4 Delete rows from table 26](#_Toc70716737)

[1.7.5 Queries 27](#_Toc70716738)

[1.7.6 Joins 27](#_Toc70716739)

[1.7.7 Data types 27](#_Toc70716740)

[1.7.8 Data types 27](#_Toc70716741)

[1.8 Docker 27](#_Toc70716742)

[1.8.1 Container manipulation 27](#_Toc70716743)

[1.8.2 Creating images (Dockerfile) 28](#_Toc70716744)

[1.8.3 Docker-compose (docker-compose.yml) 28](#_Toc70716745)

[1.8.4 Docker volume 29](#_Toc70716746)

[1.8.5 Docker 29](#_Toc70716747)

[1.9 LaTeX 29](#_Toc70716748)

[1.9.1 Symbols 29](#_Toc70716749)

[1.9.2 Formatting 31](#_Toc70716750)

[2 Machine Learning 32](#_Toc70716751)

[2.1 Types of Machine Learning systems 32](#_Toc70716752)

[2.1.1 Supervised / Unsupervised / Semisupervised / Reinforcement 32](#_Toc70716753)

[2.1.2 Batch / Online 32](#_Toc70716754)

[2.1.3 Instance-based / Model-based 32](#_Toc70716755)

[2.2 Challenges of Machine Learning 32](#_Toc70716756)

[2.3 Testing and Validating 33](#_Toc70716757)

[2.4 Scaling 33](#_Toc70716758)

[3 Data Science workflow 34](#_Toc70716759)

[3.1 Panelek 34](#_Toc70716760)

[3.2 Kényelmi funkciók 34](#_Toc70716761)

[4 Scikit-learn utilities 35](#_Toc70716762)

[4.1 Datasets 35](#_Toc70716763)

[4.2 Test-train split and cross validation 35](#_Toc70716764)

[4.3 Clustering 35](#_Toc70716765)

[4.3.1 Agglomerative Clustering 35](#_Toc70716766)

[4.3.2 K-Means Clustering 35](#_Toc70716767)

[4.3.3 DBSCAN (Density-Based) Clustering 35](#_Toc70716768)

[4.4 Egyéb 35](#_Toc70716769)

[4.5 Egyéb 35](#_Toc70716770)

[4.6 Egyéb 35](#_Toc70716771)

[5 Classification 36](#_Toc70716772)

[5.1 Linear model 36](#_Toc70716773)

[5.2 Confusion Matrix and ROC Curve 36](#_Toc70716774)

[5.2.1 Precision 37](#_Toc70716775)

[5.2.2 Recall 37](#_Toc70716776)

[5.2.3 F1 Score and Precision-Recall tradeoff 37](#_Toc70716777)

[5.3 Multiclass Classification 39](#_Toc70716778)

[5.3.1 One versus All (One versus Rest) 39](#_Toc70716779)

[5.3.2 One versus One 39](#_Toc70716780)

[5.4 Egyéb 39](#_Toc70716781)

[6 Statistics 41](#_Toc70716782)

[6.1 Permutation, Variance, Combination 41](#_Toc70716783)

[6.2 Probability theory 41](#_Toc70716784)

[6.3 CDF, PDF, PMF 42](#_Toc70716785)

[6.4 Expected value, Variance, Standard deviation 42](#_Toc70716786)

[6.5 Moments, Skewness, Kurtosis 43](#_Toc70716787)

[6.6 Median, Mode, Quantile (percentile) 44](#_Toc70716788)

[6.7 Multidimension distributions, Covariance, Correlation 44](#_Toc70716789)

[6.8 Popular discrete distributions 46](#_Toc70716790)

[6.8.1 Indicator function (aka characteristic function) 46](#_Toc70716791)

[6.8.2 Binomial distribution 46](#_Toc70716792)

[6.8.3 Poisson distribution 46](#_Toc70716793)

[6.8.4 Geometric distribution 46](#_Toc70716794)

[6.8.5 Hypergeometric distribution 47](#_Toc70716795)

[6.9 Popular continous distributions 47](#_Toc70716796)

[6.9.1 Uniform distribution 47](#_Toc70716797)

[6.9.2 Exponential distribution 47](#_Toc70716798)

[6.9.3 Normal distribution 47](#_Toc70716799)

[6.9.4 Hypergeometric distribution 48](#_Toc70716800)

[6.9.5 Hypergeometric distribution 48](#_Toc70716801)

[6.10 Popular discrete distributions 48](#_Toc70716802)

[7 Neural Networks 49](#_Toc70716803)

[7.1 Forward and Backpropagation 49](#_Toc70716804)

[7.1.1 Math 49](#_Toc70716805)

[7.1.2 Forward propagation on XOR N.N. 49](#_Toc70716806)

[7.1.3 Backpropagation on XOR N.N. 50](#_Toc70716807)

[7.2 Egyéb 51](#_Toc70716808)

[8 Web scraping 52](#_Toc70716809)

[8.1 BeautifulSoup 52](#_Toc70716810)

[9 Classification 53](#_Toc70716811)

# 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

#### 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))

### 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.

### 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}

# Machine Learning

## 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.

## Scaling

Machine learning models work better when features are on a similar scale. 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)

# Data Science workflow

## Panelek

## Kényelmi funkciók

# Scikit-learn utilities

## 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

// TODO

### K-Means Clustering

// TODO

### DBSCAN (Density-Based) Clustering

// TODO

## Egyéb

## Egyéb

## Egyéb

# Classification

During classification we are predicting classes.

## Linear model

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

sgd\_classifier = SGDClassifier()

# Y\_train is an array of True-Falses

sgd\_classifier.fit(X\_train, Y\_train)

Y\_preds = sgd\_classifier.predict(X\_test)

## 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.

### 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

# Statistics

## Permutation, Variance, 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 accuring 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.

**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:**

**Standard deviation:**

Identities:

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

V(aX + b) = a2V(X)

## 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.

## Multidimension distributions, Covariance, Correlation

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

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

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)

### 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:

### Hypergeometric distribution

### Hypergeometric distribution

## Popular discrete distributions

# 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**

# Classification