Lecture 04:

Linear and Non-Linear Regression

Markus Hohle

University California, Berkeley

Machine Learning Algorithms

MSSE 277B, 3 Units

Fall 2024

Outline

Linear Regression

- Mathematical Notation

- What is Linear?

- Some Statistics

- a Python example

Logistic Regression

Outline

Linear Regression

- Mathematical Notation

- What is Linear?

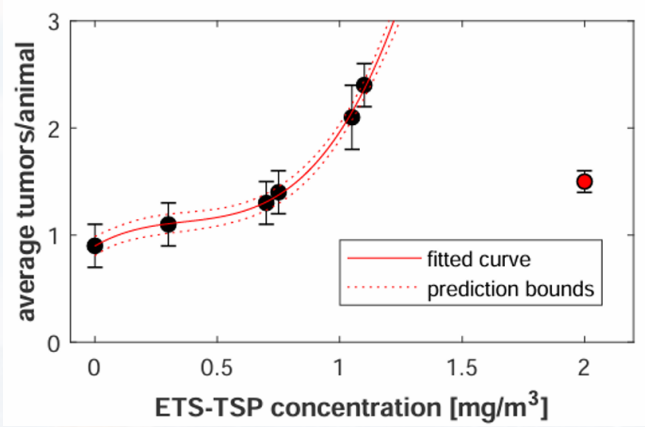
- Some Statistics

- a Python example

Logistic Regression

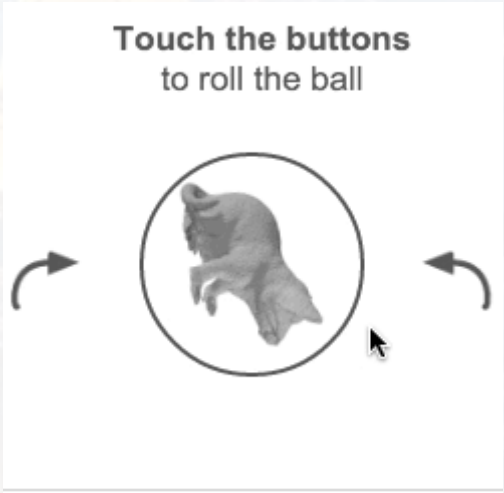
Regression vs **Classification**

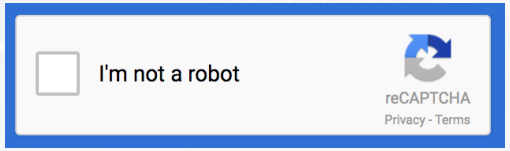
regression



curve fit: finding model parameters by **minimizing** χ^2

$$\chi^2 = \sum_k \frac{(\bar{y}_k - y_k)^2}{\sigma_k^2}$$



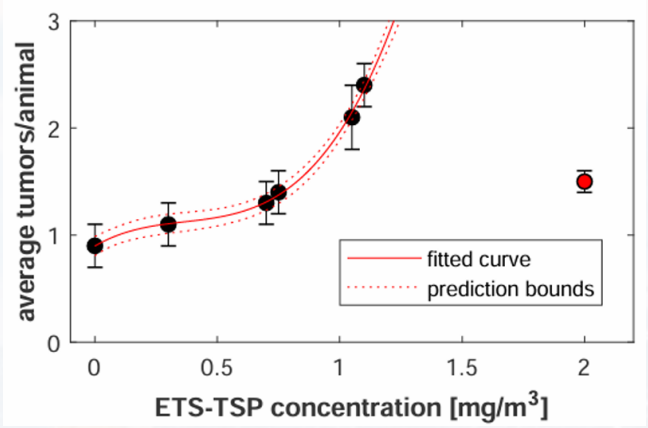


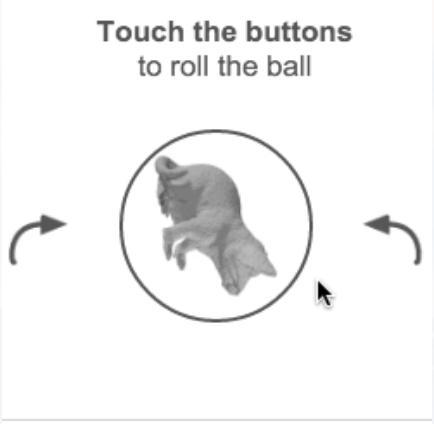
turning an image the right way:

- **maximizing** autocorrelation function
- training an AI

Regression vs **Classification**

regression





classification

cat **dog**

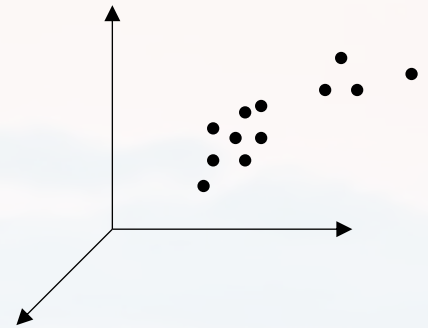
note: we can use (non-linear) regression for classification!

Linear and Non-Linear Regression

Mathematical Notation

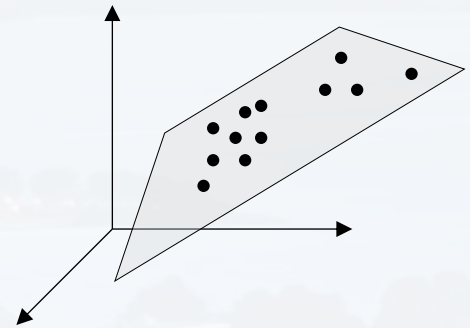
idea: data point y_k in *N* dimensional space

→ $y_k = f(x_1, \dots x_n, \dots x_N) + \epsilon$ for each data point *k*



ansatz: $\boxed{y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon}$ ***linear*** combination

y: response
x: regressors **(assumed to be independent)**
β: factors **(how a regressor contributes to the response)**
β_0: intercept
ε: error **(stochasticity of the data, assumed to be normally dist.)**



Outline

Linear Regression

- Mathematical Notation

- What is Linear?

- Some Statistics

- a Python example

Logistic Regression

Linear and Non-Linear Regression

y:	**response**
x:	**regressors**
β:	**factors**
β_0:	**intercept**
ε:	**error**

linear ≠ not curved

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n \, {x_n}^n + \epsilon$$ ...is still linear

just define: $\bar{x}_n := {x_n}^n$

$$y_k = \beta_1 {x_n}^{\beta_2}$$ ...is still linear

just use log: $\bar{y}_k = \log(y_k) = \log(\beta_1) + \beta_2 \log(x_n)$
$= \bar{\beta}_1 + \beta_2 \bar{x}_n$

<u>As long as we can recover the linear structure by any transformation → it is linear</u>

in part. log scaling is quite common <u>examples:</u>

- **log fold change (DESeq/RNASeq)**
- **log odds ratio (comparing models, HMM)**
- **sound → dB is a log unit**
- **log incidence rates (medical studies)**
- **percentiles (medical studies)**
- **-.....**

Linear and Non-Linear Regression

y:	**response**
x:	**regressors**
β:	**factors**
β_0:	**intercept**
ε:	**error**

...what is **not** linear?

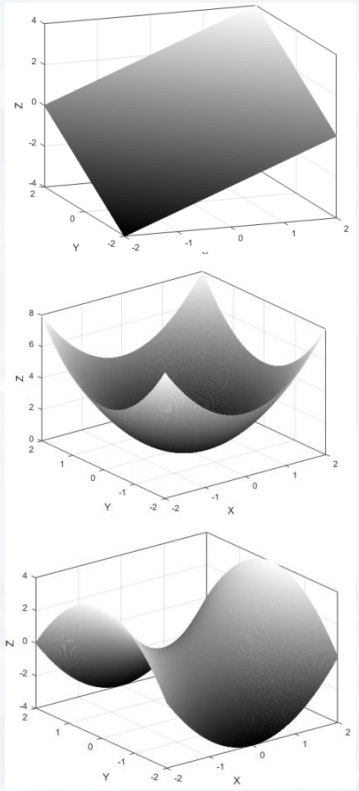
$y_k = \beta_0 + \beta_1 x_n^{\beta_2}$ log trick does not work here

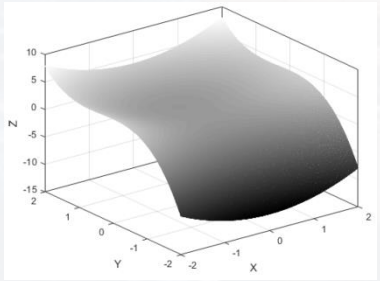
general: linear refers to the **factors**

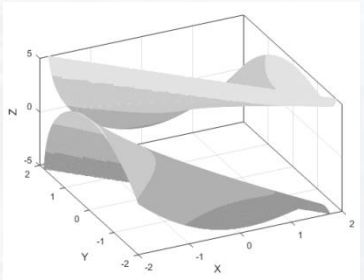
$y_k = \beta_0 + \beta_1 x_1 + \beta_2 x_2$ 2*D* plane in 3*D* space

$y_k = \beta_0 + \beta_1 {x_1}^2 + \beta_2 {x_2}^2$ 2*D* parabolic

$y_k = \beta_0 + \beta_1 {x_1}^2 - \beta_2 {x_2}^2$ 2*D* hyperbolic







...and many more...

all linear

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

Outline

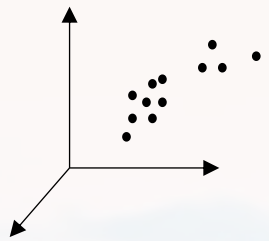
Linear Regression

- Mathematical Notation
- What is Linear?
- Some Statistics
- a Python example

Logistic Regression

for *K* data points in *N* dimensional space

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$



y:	**response**
x:	**regressors**
β:	**factors**
β_0:	**intercept**
ε:	**error**

$$\begin{pmatrix} y_1 \\ \dots \\ y_k \\ \dots \\ y_K \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & x_{12} \dots x_{1n} \dots x_{1N} \\ \dots & \dots & \dots \\ 1 & x_{k1} & x_{kn} \\ 1 & \dots & \dots \\ 1 & x_{K1} & x_{K2} \dots x_{Kn} \dots x_{KN} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \dots \\ \beta_n \\ \dots \\ \beta_N \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \dots \\ \varepsilon_k \\ \dots \\ \varepsilon_K \end{pmatrix}$$

Y X β ε

$$Y = X\beta + \varepsilon$$

<u>fitting:</u> finding the best β in terms of minimizing the errors

$$(Y - X\beta)^T (Y - X\beta) = \sum_k {\varepsilon_k}^2$$

$$\frac{\partial}{\partial \beta} \sum_k {\varepsilon_k}^2 = 0 \longrightarrow \beta_{best} = \hat{\beta} = (X^T X)^{-1} X^T Y \longrightarrow$$

the model

$$\hat{Y} = X\hat{\beta} = X(X^T X)^{-1} X^T Y$$

X and Y are all observables

$$Y = X\beta + \varepsilon$$

y:	**response**
x:	**regressors**
β:	**factors**
β_0:	**intercept**
ε:	**error**

fitting: finding the best β in by minimizing the errors

$$(Y - X\beta)^T (Y - X\beta) = \sum_k {\varepsilon_k}^2$$

$$\frac{\partial}{\partial \beta} \sum_k {\varepsilon_k}^2 = 0 \longrightarrow \beta_{best} = \hat{\beta} = (X^T X)^{-1} X^T Y \longrightarrow$$

the model

$$\hat{Y} = X\hat{\beta} = \underbrace{X(X^T X)^{-1} X^T}_{\text{hat matrix } \boldsymbol{H}} Y$$

some properties of the hat matrix:

- $H = H^T$ (symmetry)
- $HH = H \rightarrow H^n = H$ (idempotency)

evaluating the fit:

$$\hat{\varepsilon} = Y - X\hat{\beta} = Y - \hat{Y} = (I - H)Y$$

$$\rightarrow \hat{\varepsilon}^T \hat{\varepsilon} = [(I - H)Y]^T (I - H)Y = Y^T (I - H)^T (I - H)Y = Y^T (I - H)Y$$

variance $\rightarrow \hat{\sigma}^2 = \frac{\hat{\varepsilon}^T \hat{\varepsilon}}{K - N}$

degrees of freedom

summary:

y:	**response**
x:	**regressors**
β:	**factors**
β_0:	**intercept**
ε:	**error**

the model: $Y = X\beta + \varepsilon$

the fit: $\hat{Y} = X\hat{\beta} = X(X^T X)^{-1} X^T Y$

residual sum of squares: $\hat{\varepsilon}^T \hat{\varepsilon} = Y^T (I - H) Y$
(after the fit)

variance: $\hat{\sigma}^2 = \frac{\hat{\varepsilon}^T \hat{\varepsilon}}{K-N}$
(after the fit)

often fit quality is judged by $R^2 := 1 - \frac{\sum_k (\hat{y}_k - y_k)^2}{\sum_k (y_k - \langle y \rangle)^2}$

or adjusted R^2 $\bar{R}^2 := R^2 - (1 - R^2) \frac{N}{K-N-1}$

and it is said that the fit is good if R^2 is close to one....

...but that is not true...

$$\chi^2_{red} = \frac{1}{df}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{\sigma_i}\right)^2 \qquad df = N - p - 1$$

y_i:	measured value of data point
σ_i:	statistical error of y_i (often aka ey_i)
$\hat{y}_i$:	prediction by the model *after the fit*
N :	number of data points
p:	number of fit parameter

def:

$\bar{y}$: mean of the data point values

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

variance data vs model
(aka residual sum of squares)

variance of the data
(aka total sum of squares)

Note: do not confuse R^2 with Pearsons coefficient: $\rho = \frac{cov(x,y)}{\sqrt{var(x)var(y)}}$

$$\chi^2_{red} = \frac{1}{df}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{\sigma_i}\right)^2$$

$$df = N - p - 1$$

- scales difference between model and data to the error bars
- can be directly translated to a p-value via the Students distribution

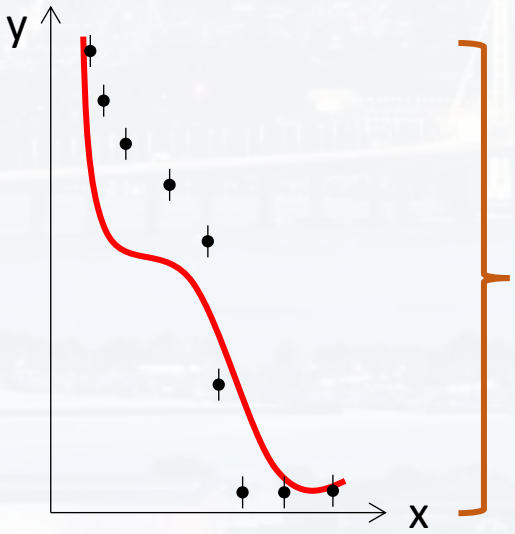
H0: the fitted model has in fact generated the data

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

variance data vs model *(aka residual sum of squares)*

variance of the data *(aka total sum of squares)*

$\bar{y}$: mean of the data point values



data variance can be huge (i. e. exponential functions) → R^2 could be around 1.0 even if fit is completely off!

$$\chi^2_{red} = \frac{1}{df}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{\sigma_i}\right)^2$$

$$df = N - p - 1$$

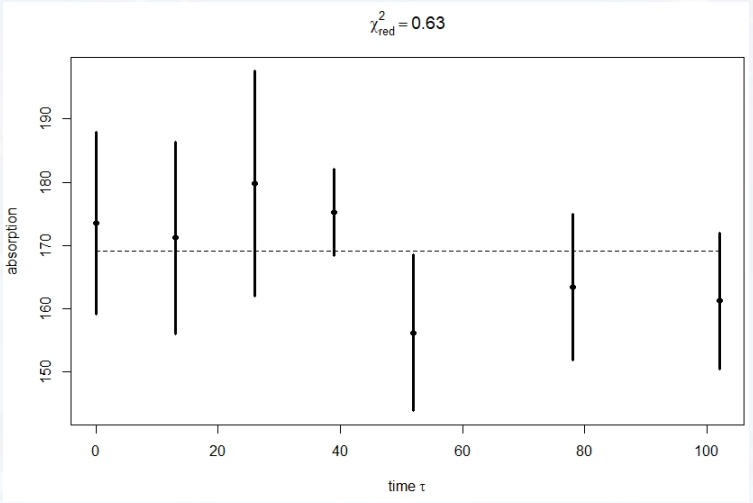
- scales difference between model and data to the error bars
- can be directly translated to a p-value via the Students distribution

H0: the fitted model has in fact generated the data

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

$$\frac{\text{variance data vs model } \textit{(aka residual sum of squares)}}{\text{variance of the data } \textit{(aka total sum of squares)}}$$

$\bar{y}$: mean of the data point values



$$\frac{\text{variance data vs model } \textit{(aka residual sum of squares)}}{\text{variance of the data } \textit{(aka total sum of squares)}} \approx 1 \rightarrow R^2 = 0$$

→ **although the fit is good**

$$\chi^2_{red} = \frac{1}{df}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{\sigma_i}\right)^2$$

$$df = N - p - 1$$

- scales difference between model and data to the error bars
- can be directly translated to a p-value via the Students distribution

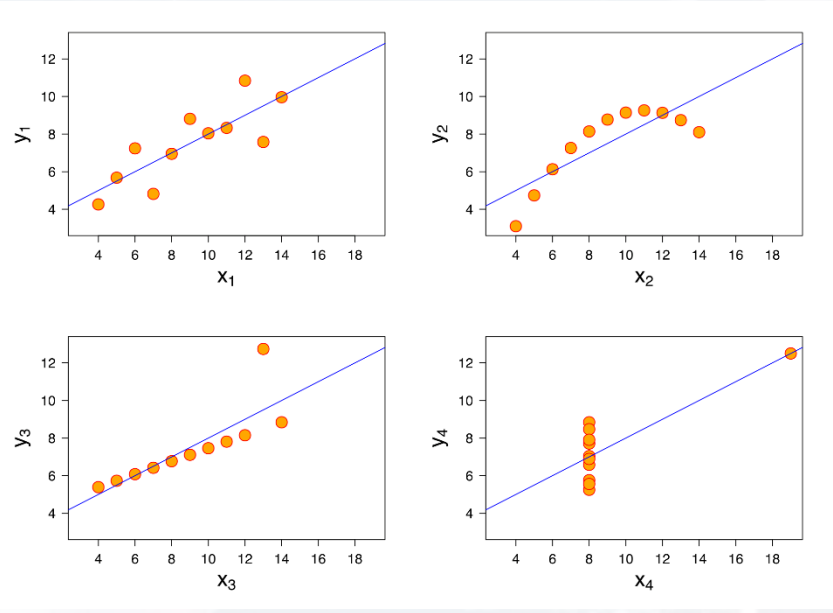
H0: the fitted model has in fact generated the data

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

variance data vs model *(aka residual sum of squares)*

variance of the data *(aka total sum of squares)*

$\bar{y}$: mean of the data point values



all plots: same R^2, but different χ^2_{red}

$$\chi^2_{red} = \frac{1}{df}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{\sigma_i}\right)^2$$

$$df = N - p - 1$$

- scales difference between model and data to the error bars
- can be directly translated to a p-value via the Students distribution

H0: the fitted model has in fact generated the data

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

variance data vs model *(aka residual sum of squares)*

variance of the data *(aka total sum of squares)*

$\bar{y}$: mean of the data point values

conclusion:

- R^2 is not a measure of the fit quality (but χ^2 is)
- error bars are important
- **given a good fit**, R^2 tells how strong the dependent variable responds to the independent variable

Also, Wiki is full of examples…

…and warnings (see "caveats" therein)

Outline

Linear Regression

- Mathematical Notation
- What is Linear?
- Some Statistics
- a Python example

Logistic Regression

Linear and Non-Linear Regression

a Python Example

```
import numpy as np

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pylab
import scipy.stats as stats
import statsmodels.api as sm

from statsmodels.formula.api import ols
from sklearn.preprocessing import MinMaxScaler
```

reading .xlsx
.csv
.txt
…

standard plots

fancy plots:
here a pair-plot

Q-Q plot

the actual super tool for superb data analysis

scaling and normalizing

Linear and Non-Linear Regression

a Python Example

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

```
Train  = pd.read_csv("molecular_train_gbc.csv")
Test   = pd.read_csv("molecular_test_gbc.csv")
```

	x_1	x_2	x_3	x_4	x_5	y_k
Index	molecular_weight	electronegativity	bond_lengths	num_hydrogen_bonds	logP	toxicity_score
0	341.704	2.65585	3.09407	2	9.11147	80.9281
1	335.951	3.22262	2.89039	7	8.92848	83.4911
2	235.203	2.44115	2.48203	1	6.49731	61.8406
3	246.505	2.76656	2.71547	7	7.45089	57.0538
4	437.939	3.4801	3.59569	3	10.9156	131.326

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

y: toxicity_score

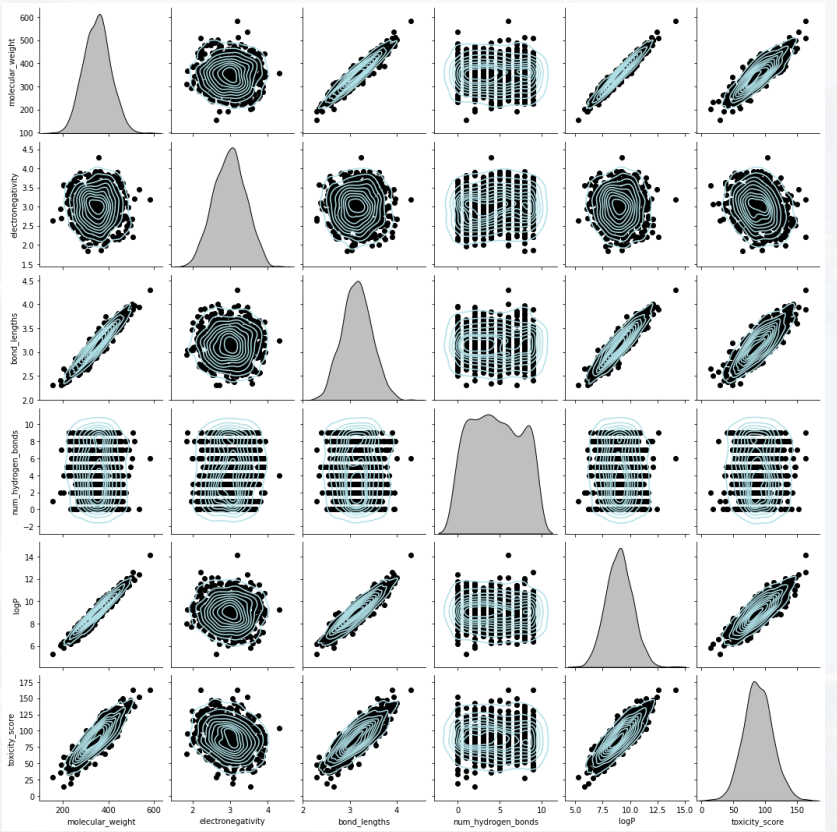
x_n : molecular_weight, electronegativity, bond_lengths, num_hydrogen_bonds, logP

Linear and Non-Linear Regression

a Python Example

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

```
Train  = pd.read_csv("molecular_train_gbc.csv")
Test   = pd.read_csv("molecular_test_gbc.csv")


out = sns.pairplot(Train, kind = "kde", \
                   plot_kws = {'color':[176/255, 224/255, 230/255]},\
                   diag_kws = {'color': 'black'})
out.map_offdiag(plt.scatter, color = 'black')
```



Linear and Non-Linear Regression

a Python Example

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

```
Train  = pd.read_csv("molecular_train_gbc.csv")
Test   = pd.read_csv("molecular_test_gbc.csv")
```

```
out = sns.pairplot(Train, kind = "kde", \
                   plot_kws = {'color':[176/255, 224/255, 230/255]},\
                   diag_kws = {'color': 'black'})
out.map_offdiag(plt.scatter, color = 'black')
```

```
scaler     = MinMaxScaler(feature_range = (0, 1))
All        = pd.concat((Train, Test), axis = 0)
(rows, _)  = Train.shape
AllS       = scaler.fit_transform(All)
```

the scaler returns an `np.array`
→ convert back to `data frame`

```
TrainS = pd.DataFrame(AllS[:rows,:], columns = Train.columns)
TestS  = pd.DataFrame(AllS[rows:,:], columns = Train.columns)
```

Linear and Non-Linear Regression

a Python Example

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

```
TrainS = pd.DataFrame(TrainS, columns = Train.columns)
TestS  = pd.DataFrame(TestS,  columns = Train.columns)
```

```
equation = 'toxicity_score ~ ' + '+'.join(Train.columns[:-1])
print(equation)
```

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

```
toxicity_score ~      molecular_weight + electronegativity +
                      bond_lengths + num_hydrogen_bonds + logP
```

```
my_model = ols(equation, data = TrainS).fit()
my_model.summary()
```

OLS (ordinary least squares)

Linear and Non-Linear Regression

a Python Example

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

```
my_model.summary()
```

not the fit quality!

OLS Regression Results

Dep. Variable:	toxicity_score	R-squared:	0.790
Model:	OLS	Adj. R-squared:	0.789
Method:	Least Squares	F-statistic:	597.5
Date:	Fri, 13 Sep 2024	Prob (F-statistic):	3.34e-266
Time:	20:57:10	Log-Likelihood:	1013.0
No. Observations:	800	AIC:	-2014.
Df Residuals:	794	BIC:	-1986.
Df Model:	5		
Covariance Type:	nonrobust		

p-value for constant model

number of data points is much larger than the number of regressors → degree of freedom approx. no of obs

p-values for factors

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.1494	0.012	12.533	0.000	0.126	0.173
molecular_weight	0.7961	0.089	8.982	0.000	0.622	0.970
electronegativity	-0.1682	0.015	-11.591	0.000	-0.197	-0.140
bond_lengths	0.0204	0.049	0.417	0.677	-0.076	0.116
num_hydrogen_bonds	0.0035	0.008	0.458	0.647	-0.011	0.018
logP	0.1246	0.072	1.723	0.085	-0.017	0.267

2σ conf range of factors

Omnibus:	2.249	Durbin-Watson:	1.984
Prob(Omnibus):	0.325	Jarque-Bera (JB):	2.240
Skew:	-0.129	Prob(JB):	0.326
Kurtosis:	2.980	Cond. No.	65.6

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

more accurate: determining **the p-values for the factors using ANOVA** for the corresponding residuals

```
table     = sm.stats.anova_lm(my_model, typ = 1)
print(table)
```

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

	df	sum_sq	mean_sq	F	PR(>F)
molecular_weight	1.0	13.346285	13.346285	2847.525516	8.024085e-265
electronegativity	1.0	0.640388	0.640388	136.631363	3.085962e-29
bond_lengths	1.0	0.000684	0.000684	0.145954	7.025342e-01
num_hydrogen_bonds	1.0	0.000703	0.000703	0.150055	6.985866e-01
logP	1.0	0.013917	0.013917	2.969353	8.524510e-02
Residual	794.0	3.721459	0.004687	NaN	NaN

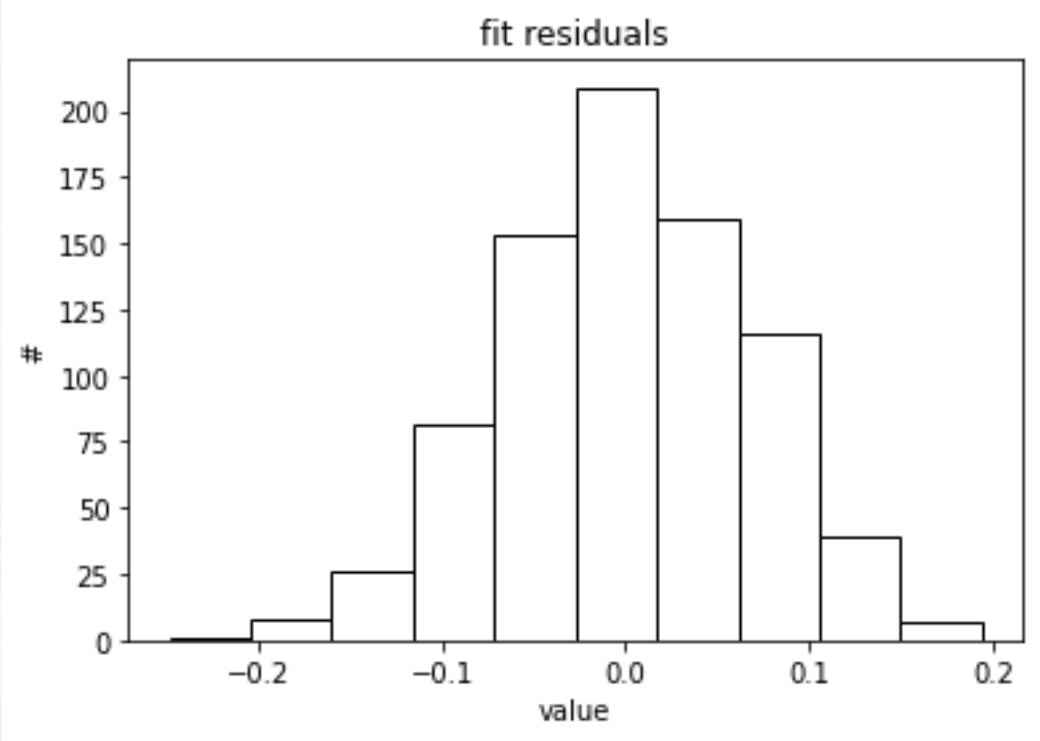
vs from t-test

0.0000
0.0000
0.6766
0.6473
0.0852

Linear and Non-Linear Regression

a Python Example

```
residuals = my_model.resid

plt.hist(residuals, color = 'w', edgecolor = 'black')
plt.title('fit residuals')
plt.ylabel('#')
plt.xlabel('value')
plt.show()
```



1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

residuals approx. normally distributed around μ = 0

Linear and Non-Linear Regression

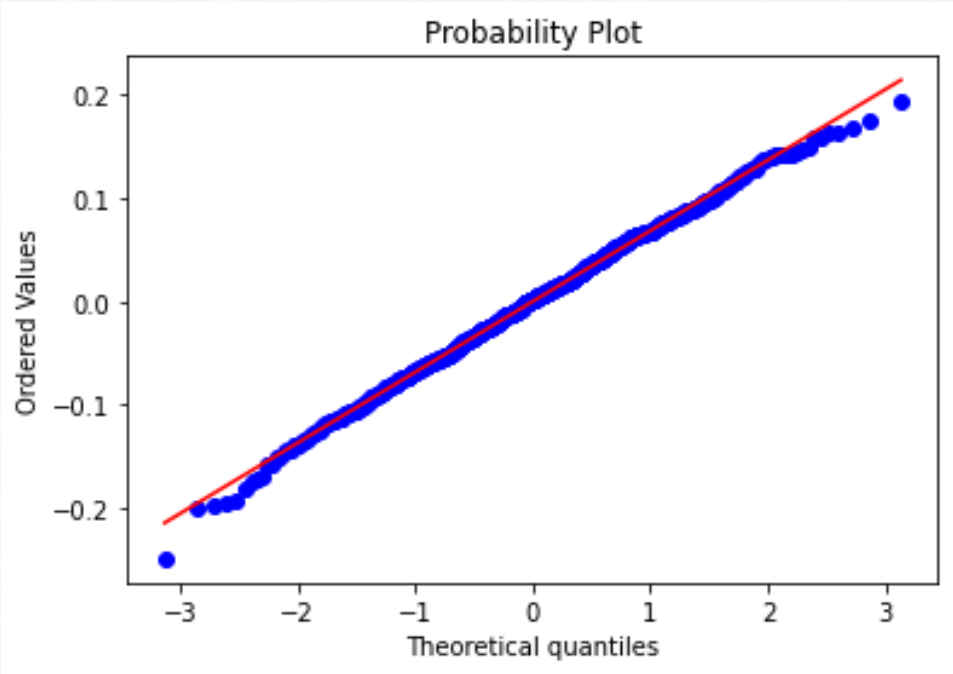
a Python Example

```
residuals = my_model.resid

plt.hist(residuals, color = 'w', edgecolor = 'black')
plt.title('fit residuals')
plt.ylabel('#')
plt.xlabel('value')
plt.show()

stats.probplot(residuals, dist = "norm", plot = pylab)
pylab.show()
```

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

residuals approx. normally distributed around μ = 0

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$



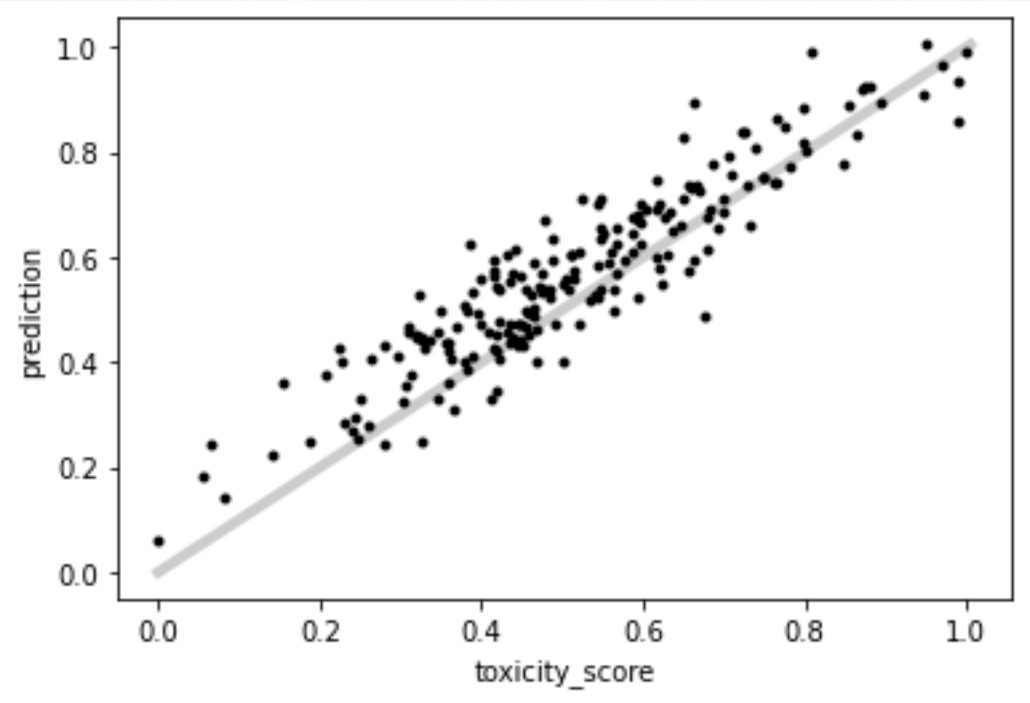
Linear and Non-Linear Regression

a Python Example

```
Ypred  = my_model.predict(TestS)

higher = np.max([Ypred, TestS.toxicity_score])
lower  = np.min([Ypred, TestS.toxicity_score])

plt.plot([lower, higher], [lower, higher], c = [0, 0, 0, 0.2],\
         linewidth = 4)
plt.scatter(TestS.toxicity_score, Ypred, marker = '.', c = 'k')
plt.ylabel('prediction')
plt.xlabel('toxicity score')
plt.show()
```

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$



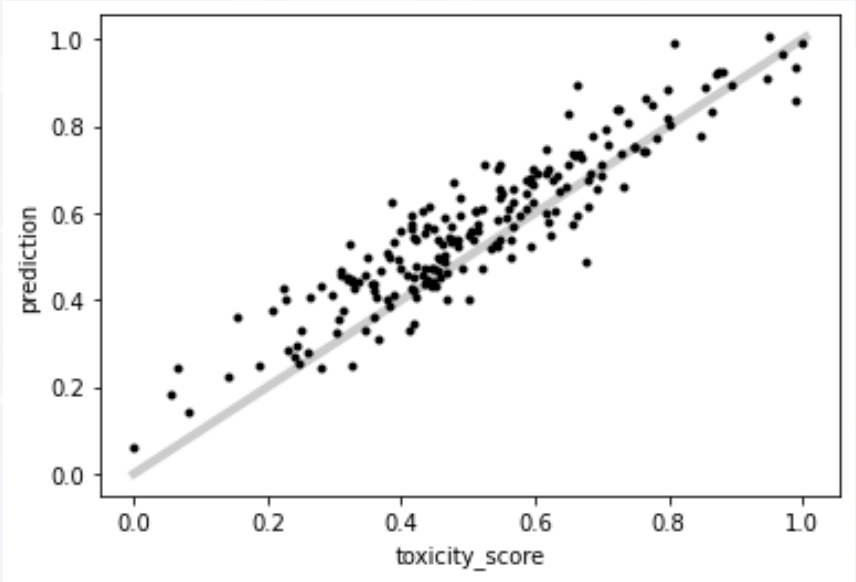
Linear and Non-Linear Regression

a Python Example

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

```
Ypred  = my_model.predict(TestS)

higher = np.max([Ypred, TestS.toxicity_score])
lower  = np.min([Ypred, TestS.toxicity_score])

plt.plot([lower, higher], [lower, higher], c = [0, 0, 0, 0.2], linewidth = 4)
plt.scatter(TestS.toxicity_score, Ypred, marker = '.', c = 'k')
plt.ylabel('prediction')
plt.xlabel('toxicity score')
plt.show()
```




```
mean_dev = np.sum( abs(TestS.toxicity_score - Ypred) )/len(Ypred)
print(mean_dev)
```

5%

Outline

Linear Regression

- Mathematical Notation
- What is Linear?
- Some Statistics
- a Python example

Logistic Regression

Linear and Non-Linear Regression

linear model: regressors are continuous or categorical, response is continuous

logistic model: response is **categorical**

y:	response
x:	regressors **(assumed to be independent)**
β:	factors
β_0:	intercept
ε:	error **(stochasticity of the data, assumed to be normally dist.)**

Index	molecular_weight	electronegativity	bond_lengths	num_hydrogen_bonds	logP	label
0	341.704	2.65585	3.09407	2	9.11147	Toxic
1	335.951	3.22262	2.89039	7	8.92848	Toxic
2	235.203	2.44115	2.48203	1	6.49731	Non-Toxic
3	246.505	2.76656	2.71547	7	7.45089	Non-Toxic
4	437.939	3.4801	3.59569	3	10.9156	Non-Toxic

Linear and Non-Linear Regression

linear model: regressors are continuous or categorical, response is continuous

logistic model: response is **categorical**

y:	response
x:	regressors **(assumed to be independent)**
β:	factors
β_0:	intercept
ε:	error **(stochasticity of the data, assumed to be normally dist.)**

dichotomic model:

probability to be in state A) → ***p***

probability to be in state B) → ***1 - p***

label
Toxic
Toxic
Non-Toxic
Non-Toxic
Non-Toxic

ansatz:

$$log\left(\frac{p}{1-p}\right) = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

log odds ratio: linear model

Linear and Non-Linear Regression

dichotomic model:

probability to be in state A) → ***p***

probability to be in state B) → ***1 - p***

label
Toxic
Toxic
Non-Toxic
Non-Toxic
Non-Toxic

y:	response
x:	regressors **(assumed to be independent)**
β:	factors
β_0:	intercept
ε:	error **(stochasticity of the data, assumed to be normally dist.)**

ansatz:

$$log\left(\frac{p}{1-p}\right) = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

log odds ratio: linear model

→ probability for being in a certain state

$$p = \frac{e^{\beta_0+\beta_1 x_1+\cdots}}{1 + e^{\beta_0+\beta_1 x_1+\cdots}}$$

often:

$$logit(p) = log\left(\frac{p}{1-p}\right)$$

examples:

- probability that a gene has been mutated
- probability of being diseased (cancer, alzheimer etc) as function of age, environmental influence etc ...
- Verhulst equation: $N(t) = N_0 \frac{e^{rt}}{C+e^{rt}}$
- activation functions in ANNs

$$log\left(\frac{p}{1-p}\right) = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

y:	response
x:	regressors **(assumed to be independent)**
β:	factors
β_0:	intercept
ε:	error **(stochasticity of the data, assumed to be normally dist.)**

Note: one can derive the logit function from max. entropy too!

$$p = \frac{e^{\beta_0+\beta_1 x_1+\cdots}}{1+e^{\beta_0+\beta_1 x_1+\cdots}} = \frac{1}{1+e^{-\beta_0-\beta_1 x_1-\cdots}}$$

onset of Alzheimer's disease (AD) is a function of ***age*** and years spent in ***education***
(and other risk factors we ignore here for the sake of simplicity)

education: $\mathrm{d} = x_1$ [yrs]
age: $a = x_2$ [yrs]

model: $p_{AD} = \frac{1}{1+e^{-\beta_0-\beta_1 d-\beta_2 a}}$

+ data set + fit → $\beta_0 = +0.1$
$\beta_1 = -1.5$
$\beta_2 = +0.12$

- *positive* value → *increasing p*
- *negative* value → *decreasing p*
- intercept: "background" prevalence, not related to environmental/internal conditions

Linear and Non-Linear Regression

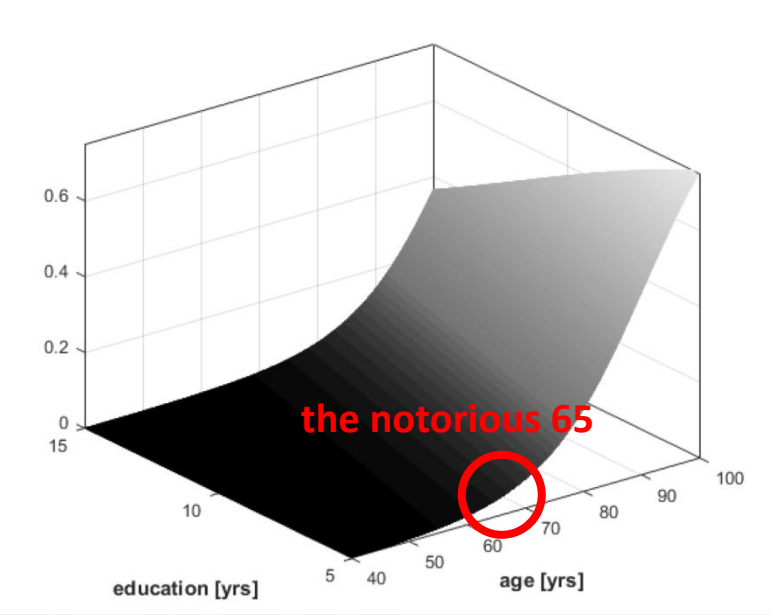
y:	response
x:	regressors **(assumed to be independent)**
β:	factors
β_0:	intercept
ε:	error **(stochasticity of the data, assumed to be normally dist.)**

model: $p_{AD} = \frac{1}{1+e^{-\beta_0-\beta_1 d-\beta_2 a}}$

education: $d = x_1$ [yrs] $\quad \beta_0 = +0.1$

age: $a = x_2$ [yrs] $\quad \beta_1 = -1.5$

$\beta_2 = +0.12$



example: ***65yrs*** old person, ***8yrs*** spent in education

→ $p_{AD} = 1.6\%$

65yrs old person, ***13yrs*** spent in education

→ $p_{AD} = 0.001\%$

<u>How does education compensate aging?</u>

$$p_{AD}(d + \bar{d}, a + \bar{a}) = p_{AD}(d, a)$$

→ $\bar{a} = 12.5\ \bar{d}$

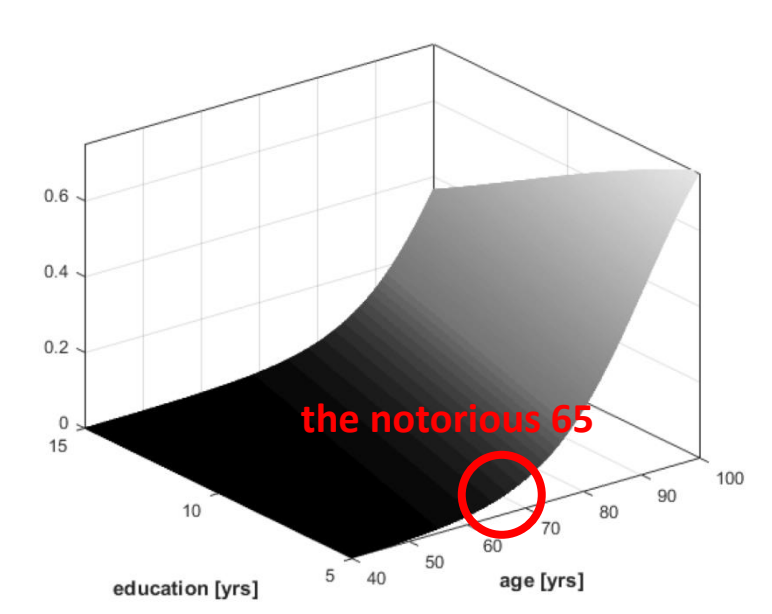
hence, one more year prolonged education compensates 12.5 years of aging

(warning: don't confuse correlation with causation here!)

model: $p_{AD} = \frac{1}{1+e^{-\beta_0-\beta_1 d-\beta_2 a}}$

education:	$d = x_1$ [yrs]	$\beta_0 = +0.1$
age:	$a = x_2$ [yrs]	$\beta_1 = -1.5$
		$\beta_2 = +0.12$

y:	response
x:	regressors **(assumed to be independent)**
β:	factors
β_0:	intercept
ε:	error **(stochasticity of the data, assumed to be normally dist.)**



<u>How does the risk of onset changes *per year*?</u>

relative change:

$$\frac{p_{AD}(a+1) - p_{AD}(a)}{p_{AD}(a)} \approx e^{\beta_2} - 1 \approx 12.7\%$$

$p_{AD} \ll 1$ (hence, for small Δa and "young" ages, i. e. below ≈ 80yrs)

the risk of getting AD increases by 12.7% every year

(warning: does not mean that it increases by 127% in ten yrs – we made an approximation!)

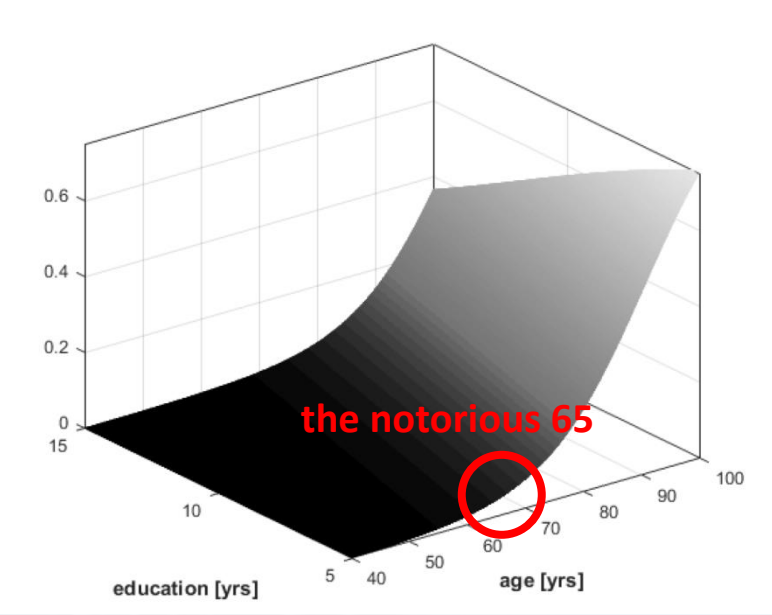
y:	response
x:	regressors **(assumed to be independent)**
β:	factors
β_0:	intercept
ε:	error **(stochasticity of the data, assumed to be normally dist.)**

model: $p_{AD} = \frac{1}{1+e^{-\beta_0-\beta_1 d-\beta_2 a}}$

education: $d = x_1$ [yrs] $\quad \beta_0 = +0.1$

age: $a = x_2$ [yrs] $\quad \beta_1 = -1.5$

$\beta_2 = +0.12$



How does the risk of onset changes *per year*?

more precise: relative change of the odds ratio

$$\frac{\frac{\partial}{\partial x_i}\left(\frac{p_{AD}}{1-p_{AD}}\right)}{\frac{p_{AD}}{1-p_{AD}}} = \beta_i$$

x_i is the desired regressor, for example, age again (x_2)

the factors β_i indicate how strong (and in which direction) *p* changes wrt a regressor x_i

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

let us return to the molecule data set:

```
Train  = pd.read_csv("molecular_train_gbc_cat.csv")
Test   = pd.read_csv("molecular_test_gbc_cat.csv")
```

Index	molecular_weight	electronegativity	bond_lengths	num_hydrogen_bonds	logP	label
0	341.704	2.65585	3.09407	2	9.11147	Toxic
1	335.951	3.22262	2.89039	7	8.92848	Toxic
2	235.203	2.44115	2.48203	1	6.49731	Non-Toxic
3	246.505	2.76656	2.71547	7	7.45089	Non-Toxic
4	437.939	3.4801	3.59569	3	10.9156	Non-Toxic

Linear and Non-Linear Regression

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

it is the same data set → plotting and scaling is as before

```
X = sm.add_constant(TrainS)
```

adding the intercept

```
Y = pd.get_dummies(Train['label'])
```

Python needs True/False as categorical

```
In [48]: print(Y)
     Non-Toxic  Toxic
0        False   True
1        False   True
2         True  False
3         True  False
4         True  False
```

$$p = \frac{e^{\beta_0+\beta_1 x_1+\cdots}}{1+e^{\beta_0+\beta_1 x_1+\cdots}}$$

we have two states: toxic / non-toxic

```
my_model = sm.GLM(Y, X, family = sm.families.Binomial()).fit()
```

```
my_model.summary()
```

GLM: general linear model

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

it is the same data set → plotting and scaling is as before

```
X = sm.add_constant(TrainS)
Y = pd.get_dummies(Train['label'])

my_model = sm.GLM(Y, X, family = sm.families.Binomial()).fit()
my_model.summary()
```

Generalized Linear Model Regression Results

Dep. Variable:	['Non-Toxic', 'Toxic']	No. Observations:	800
Model:	GLM	Df Residuals:	794
Model Family:	Binomial	Df Model:	5
Link Function:	Logit	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-332.82
Date:	Sat, 14 Sep 2024	Deviance:	665.64
Time:	20:59:18	Pearson chi2:	1.14e+03
No. Iterations:	6	Pseudo R-squ. (CS):	0.4243
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	6.1641	0.585	10.536	0.000	5.017	7.311
molecular_weight	-10.4920	3.626	-2.893	0.004	-17.599	-3.385
electronegativity	3.2874	0.599	5.492	0.000	2.114	4.461
bond_lengths	0.6736	1.913	0.352	0.725	-3.075	4.422
num_hydrogen_bonds	-0.3082	0.303	-1.018	0.309	-0.902	0.285
logP	-7.6090	2.978	-2.555	0.011	-13.447	-1.771

$$p = \frac{e^{\beta_0+\beta_1 x_1+\cdots}}{1+e^{\beta_0+\beta_1 x_1+\cdots}}$$

p-value for constant model

p-values for factors

2σ conf range of factors

accuracy: How ***often*** did the model make the correct prediction.
cross-entropy: How ***certain*** was the model when making the prediction.

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

accuracy: How ***often*** did the model make the correct prediction.

cross-entropy: How ***certain*** was the model when making the prediction.

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

```
predProbs   = my_model.predict(sm.add_constant(TestS))

Pred        = np.round(predProbs).astype(int)
predictions = ['Non-Toxic' if i==1 else 'Toxic' for i in Pred]
```

```
Dep. Variable:      ['Non-Toxic', 'Toxic']
```

```
In [51]: predictions
Out[51]:
['Toxic',
 'Toxic',
 'Non-Toxic',
 'Non-Toxic',
 'Toxic',
 'Toxic',
 'Toxic',
```

```
TestY       = Test['label']
accuracy    = 100*(TestY == predictions).sum()/len(predictions)
print(f'accuracy = {accuracy: .2f}%')
```

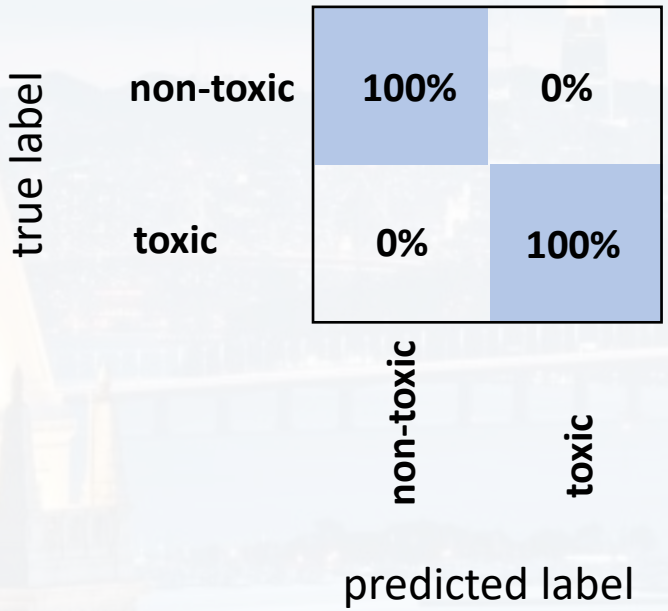
```
accuracy =  80.50%
```

accuracy: How ***often*** did the model make the correct prediction.

cross-entropy: How ***certain*** was the model when making the prediction.

accuracy is ≈ 80% But does it depend on the class? → **confusion matrix**

ideal world:




```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

accuracy: How ***often*** did the model make the correct prediction.

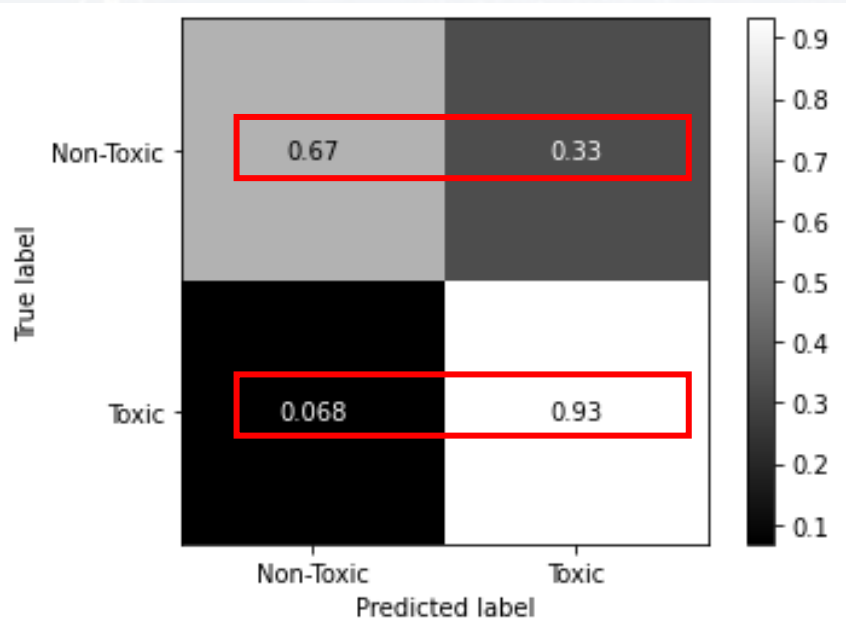
cross-entropy: How ***certain*** was the model when making the prediction.

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

accuracy is ≈ 80% But does it depend on the class? → **confusion matrix**

two labels

```
L = ['Non-Toxic', 'Toxic']

cm   = confusion_matrix(TestY, predictions, labels = L, normalize = 'true')
disp = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = L)
disp.plot(cmap = 'gray')
plt.show()
```



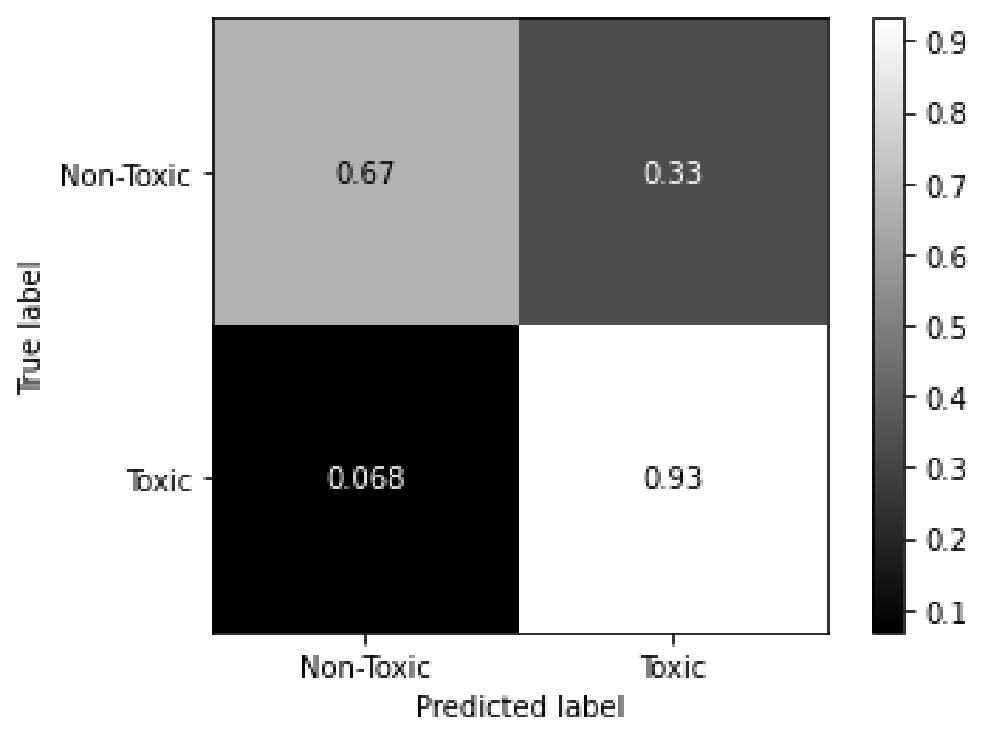
accuracy: How ***often*** did the model make the correct prediction.

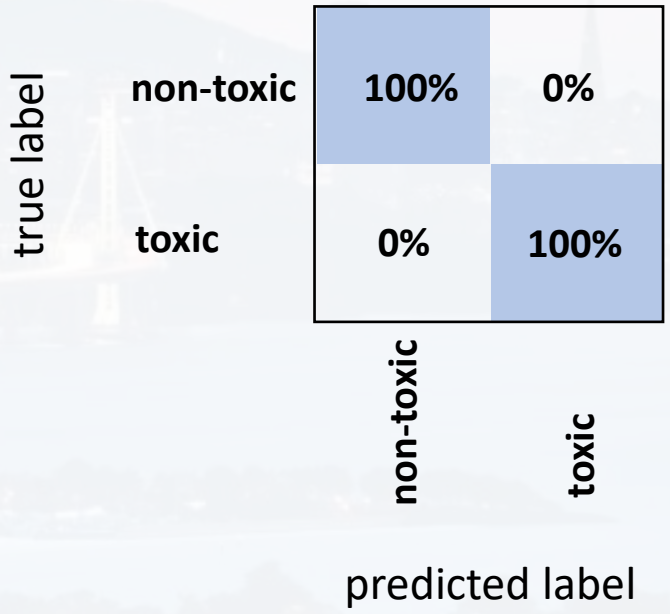
cross-entropy: How ***certain*** was the model when making the prediction.

accuracy is ≈ 80%

But does it depend on the class? → **confusion matrix**

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model



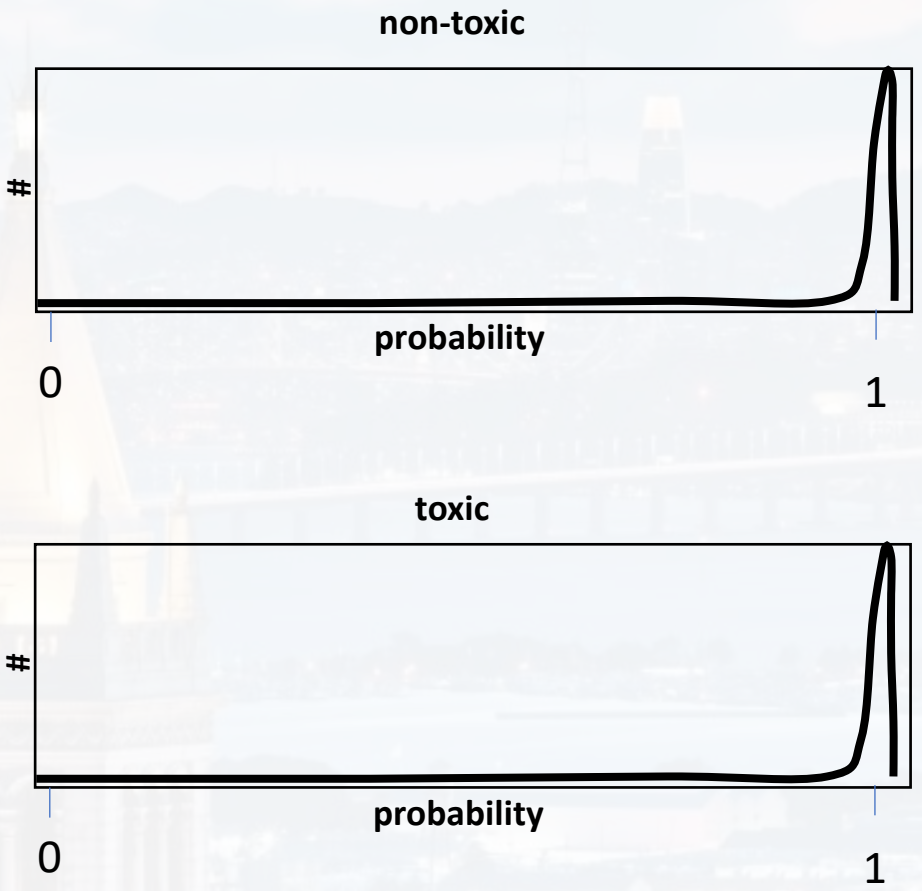


accuracy: How ***often*** did the model make the correct prediction.

cross-entropy: How ***certain*** was the model when making the prediction.

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

ideal world:



accuracy: How *often* did the model make the correct prediction.

cross-entropy: How ***certain*** was the model when making the prediction.

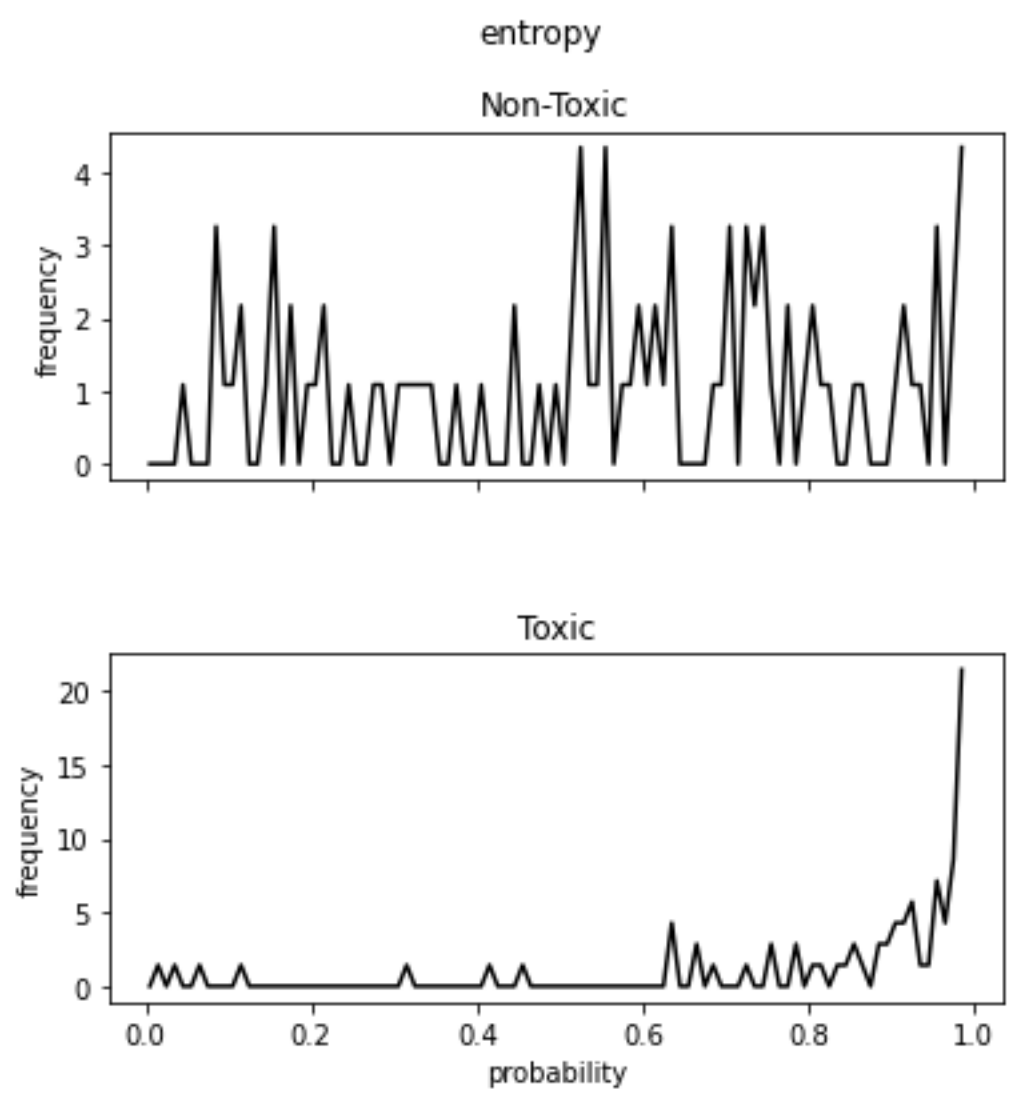
1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

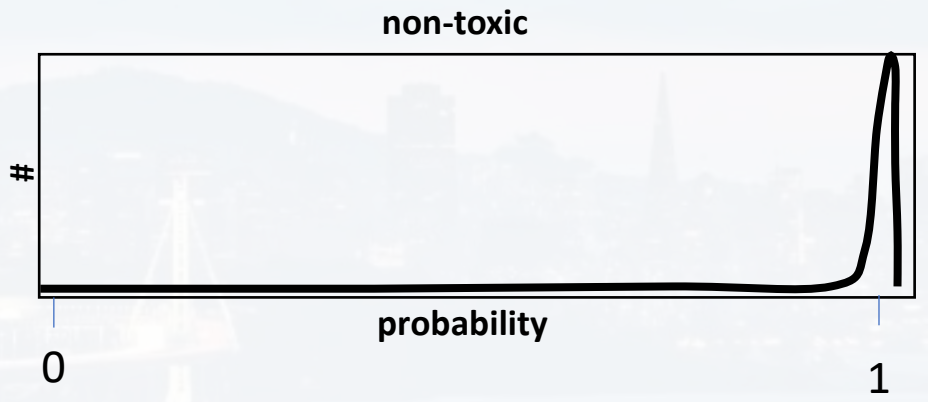
```
PredProbs = np.vstack((predProbs, 1 - predProbs))

fig, ax = plt.subplots(len(L), 1, sharex = True)
fig.set_figheight(6)
fig.subplots_adjust(hspace = 0.5)
fig.suptitle('entropy')
for i, l in enumerate(L):
    idx = [k for k, y in enumerate(TestY) if y == l]
    idx = np.array(idx)
    (value, where) = np.histogram(PredProbs[i,idx],\
                                  bins = np.arange(0, 1, 0.01),\
                                  density = True)
    w = 0.5*(where[1:] + where[:-1])
    ax[i].plot(w, value, 'k-')
    ax[i].set_ylabel('frequency')
    ax[i].set_title(l)
ax[len(L)-1].set_xlabel('probability')
plt.show()
```

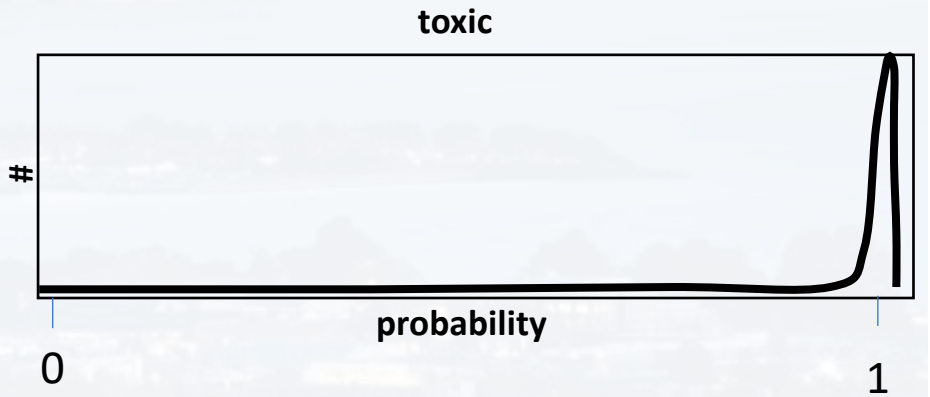
accuracy: How ***often*** did the model make the correct prediction.

cross-entropy: How ***certain*** was the model when making the prediction.

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model







Thank you very much for your attention!