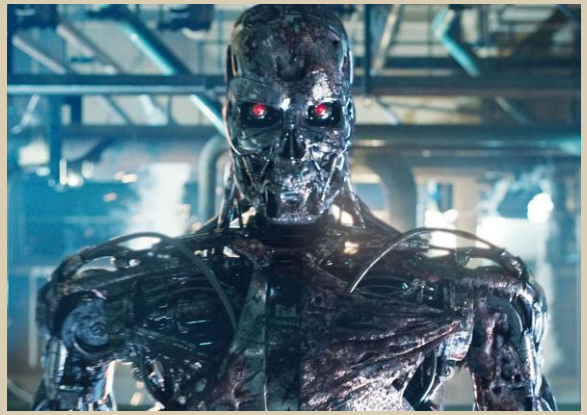
Krakow Applied Physics and Computer Science Summer School '21

ONLINE

July 1 - 28 2021



Machine Learning *Applications*

Tomasz Szumlak

AGH-UST

AGH

Wydział Fizyki i Informatyki Stosowanej

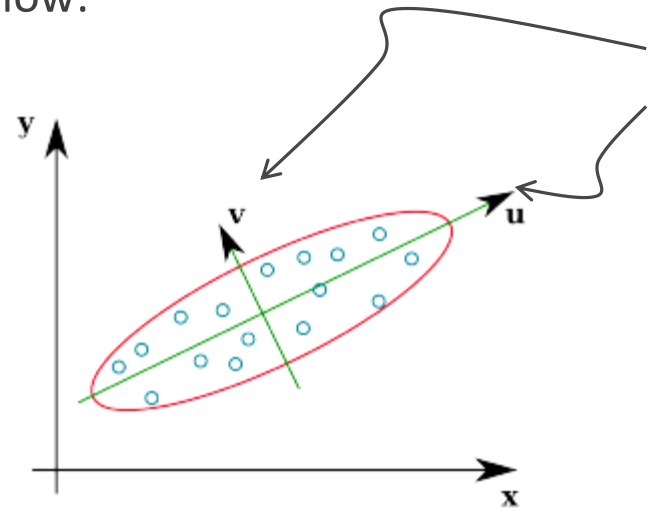
LHCb

ATLAS EXPERIMENT

Intelli-NET

Principal directions

- Let's have a look at the plot below:



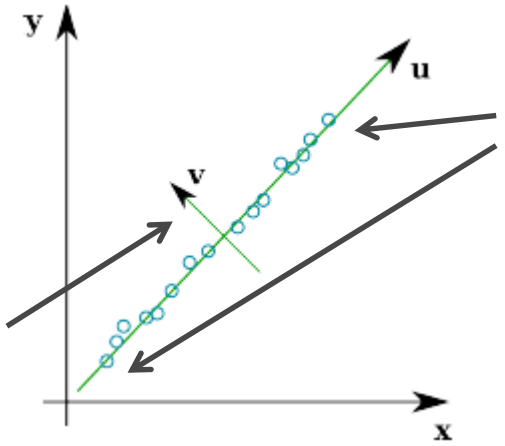
Principal directions

- Here, u and v are called the principal direction of data variation ($\boldsymbol{u}$ is the most important one, $\boldsymbol{v}$ is next and perpendicular to u)
- Anything interesting about the transformation $(X, Y) \rightarrow (U, V)$?
- After the transformation data set is **compact** (mean values are 0) and **decorrelated**

And reduction…?

- Consider this: what if variation in data is caused by a specific relation? For instance:

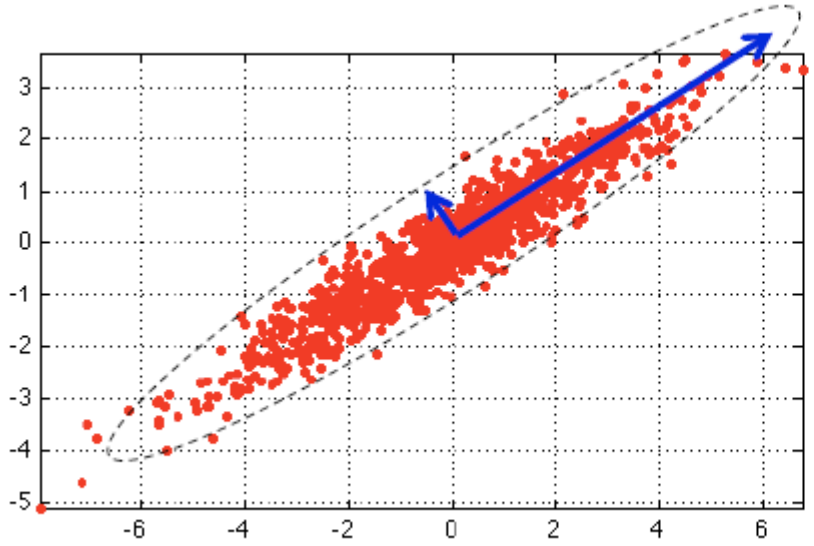
Noise related to measurement in $\boldsymbol{v}$

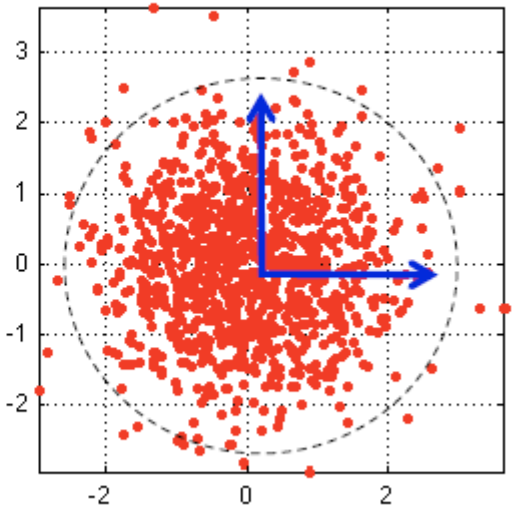


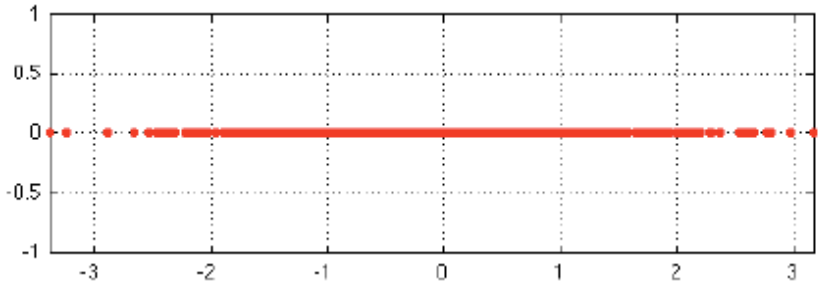
„All data variation” in the principal direction $\boldsymbol{u}$

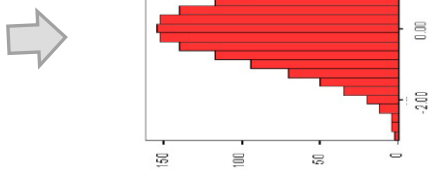
- Actually, we could say, that there is no variation along the second principal direction, i.e., there is no vital information for ML algo.
- Can treat this as $1d$ data set **without compromising** the overall performance of classification

Get some feeling









The math behind PCA

- Most of the times (or even all of the time) we are going to use libraries to do the job! That is fine, however, learning a bit what is under the hood is a good thing!
- First given the data we can compute the **covariance matrix**:

$$\Sigma_{ij} = cov(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)] = E[X_i X_j] - \mu_i \mu_j$$

$$\Sigma = \begin{pmatrix} (X_1 - \mu_1)(X_1 - \mu_1) & \cdots & (X_1 - \mu_1)(X_k - \mu_k) \\ \vdots & \ddots & \vdots \\ (X_k - \mu_k)(X_1 - \mu_1) & \cdots & (X_k - \mu_k)(X_k - \mu_k) \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} var(X_1) & \cdots & cov(X_1, X_k) \\ \vdots & \ddots & \vdots \\ cov(X_k, X_1) & \cdots & var(X_2) \end{pmatrix}$$

The math behind PCA

- Having the covariance matrix one can find the **principal components** by computing its eigen-vectors and eigen-values
- In other words we would say that we want to find a transformation matrix to find the axis system in which the covariance matrix is **diagonal** (or in canonical form)
- **The eigen-vector corresponding to the largest eigen-value is the direction of the greatest variation**
- We start from the characteristic equation (or polynomial) $|(\Sigma - \lambda\mathbb{I})| = 0$, which for Σ matrix of size $n \times n$ has n roots
- Next, we calculate eigen-vectors: $\Sigma \boldsymbol{x}_i = \lambda \boldsymbol{x}_i$
- The eigen-vectors should be normalised: $\boldsymbol{x}_i \cdot \boldsymbol{x}_i^T = \boldsymbol{x}_i^T \cdot \boldsymbol{x}_i = 1$
- We can combine the eigen-vectors and write as a transformation matrix.

The math behind PCA

- The transformation matrix

$$\mathbb{T} = (\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \boldsymbol{x}_3), \mathbb{T}\mathbb{T}^T = \mathbb{T}^T\mathbb{T} = \mathbb{I}$$

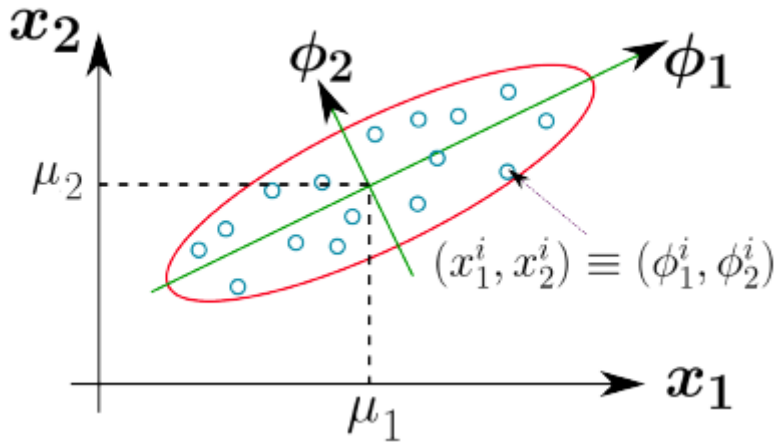
$$\Sigma(\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \boldsymbol{x}_3) = (\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \boldsymbol{x}_3)\begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix}$$

$$\Sigma\mathbb{T} = \mathbb{T}\Lambda \rightarrow \mathbb{T}^T\Sigma\mathbb{T} = \Lambda$$

- So, having calculated e-vectors and e-values, we can use it to **transform all data points** into a data set where the variables are **not correlated**: $(X, Y) \rightarrow (U, V)$
- In this new coordinate system the new correlation matrix is diagonal and can be written as Λ

It is definitely worth considering

- If we have data as follow:

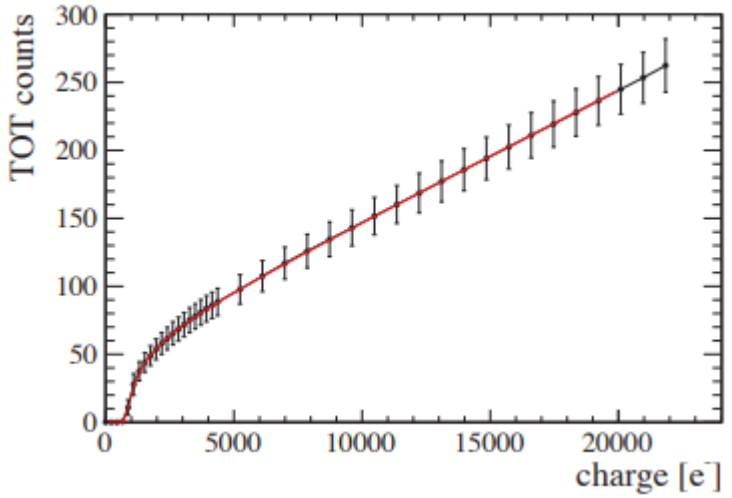


- We do the following:
  - Calculate the mean values: (μ_1, μ_2), Σ and the transformation matrix $\mathbb{T}$
  - Now, each data point can be transformed from $(x_1, x_2) \rightarrow (\phi_1, \phi_2)$ with the equation: $\boldsymbol{p}_\phi = (\boldsymbol{p}_x - \boldsymbol{\mu}_x)\mathbb{T}$
- This kind of data pre-processing is very commonly used for many different types of ML analyses!

LHCb VELO Pixel analysis – activation

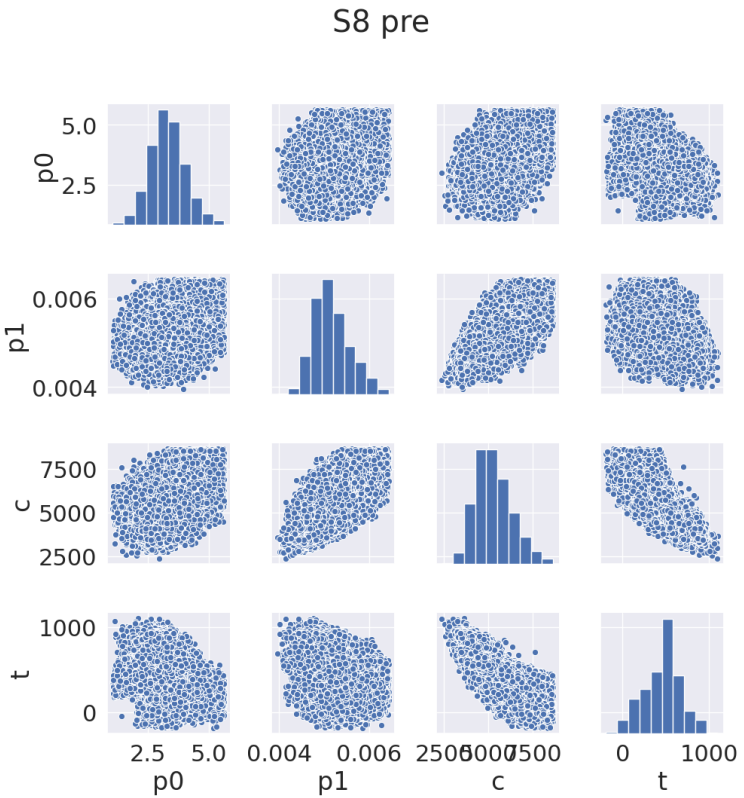
The surrogate function is what we use to calibrate each pixel to be able to translate ToT counts into collected charge (deposited energy)

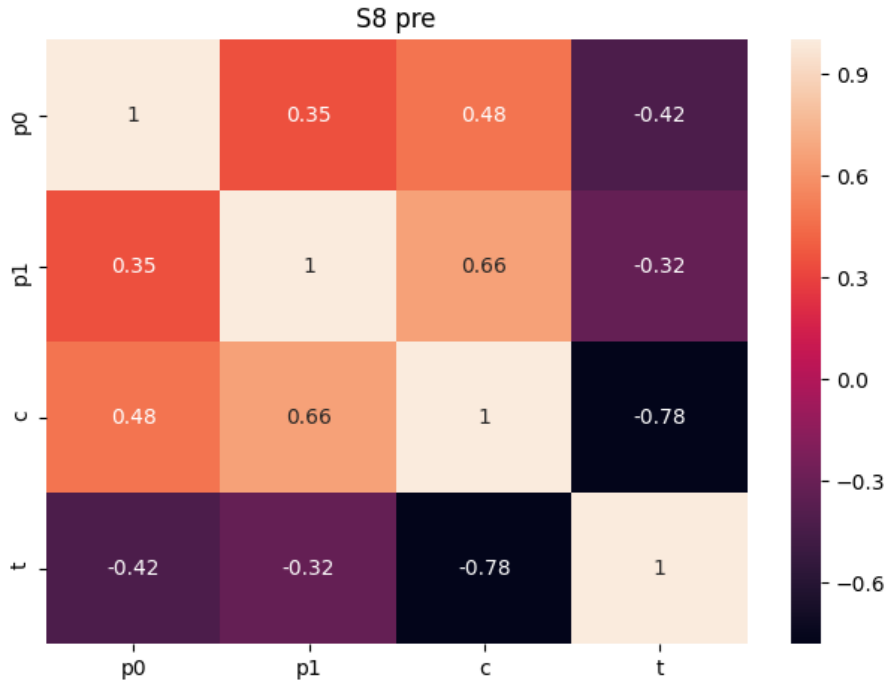
Surrogate function has the following form:



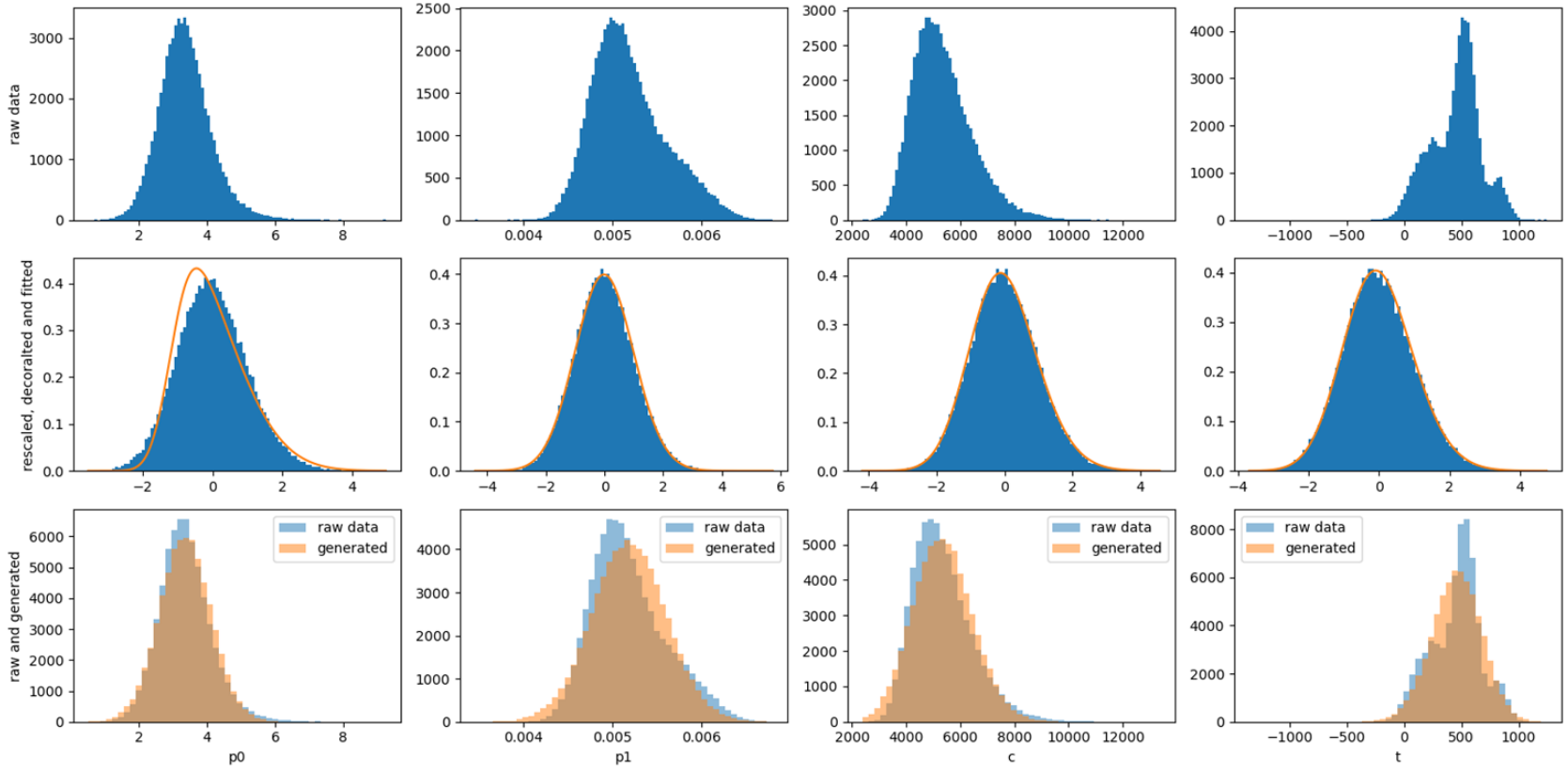
f(x) = p0 + p1*x - c/(x-t)

LHCb VELO Pixel analysis – exploration

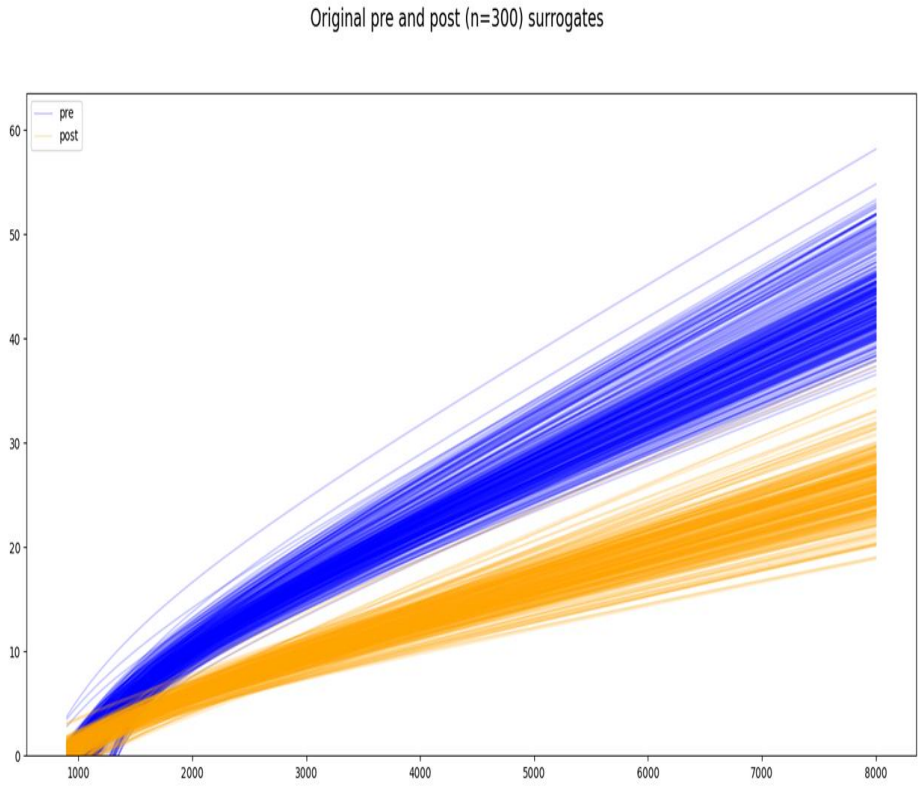


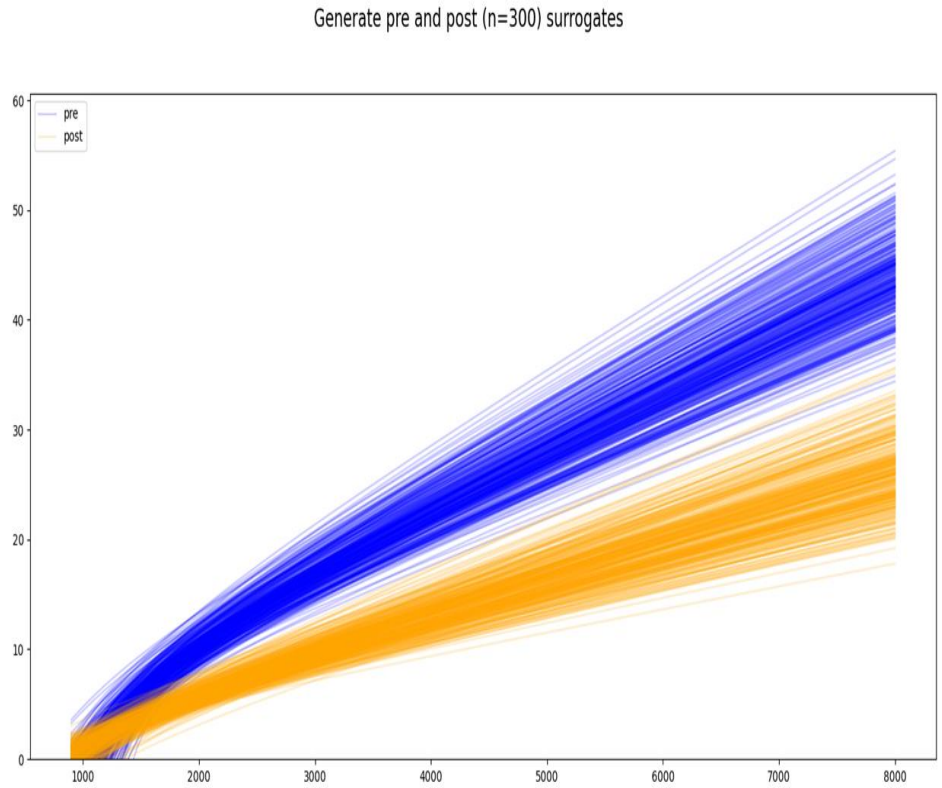


LHCb VELO Pixel analysis – PCA



LHCb VELO Pixel analysis – Model



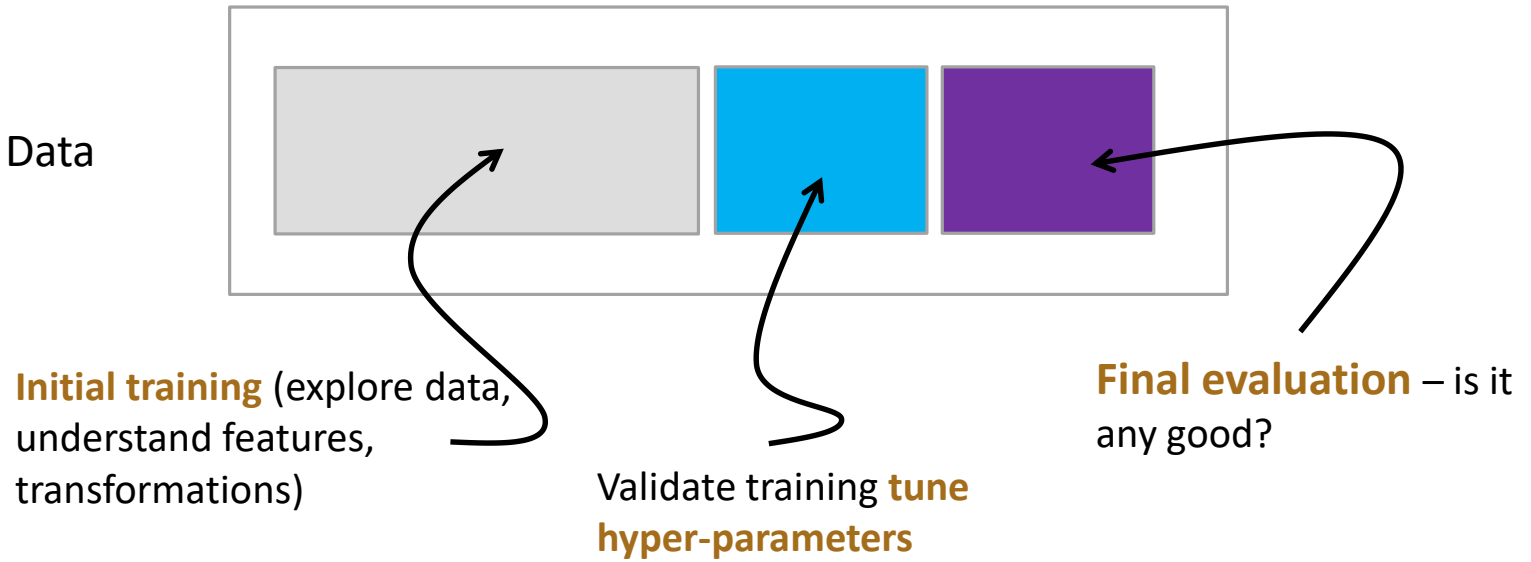


ML your way

- The way you can start to build your own start-to-end projects can be facilitated by tools such, conda, pycharm, git, etc.
- A lot of steps (data pre-processing, feature extraction) can be approached in quite abstract way, thus a set of simple tools can be prepared and shared between different projects
- Key aspect is: always understand your data!
- Elements of statistical data analysis are the key here.

How to make sure we are doing a good job

- 2-point validation works like that (variation on this is k-folding):
  - Take the whole data set and divide it **0.5:0.25:0.25**
  - Use the first subset to make training, use the second to validate and tune the **hyper-parameters**
  - Use the last part to evaluate the results



Hands-on approach

- Below is not a „magic recipe” it is more like **a set of good rules** (may not be possible always to go „exactly like that”)
- **Collect data** (aka experiment), can use structured and unstructured sets
- **Pre-processing** – format data accordingly (algorithm dependent), **missing data and outliers** are delicate to handle
- **Data exploration** and **feature engineering** (this is the most time-consuming part of the ML)
- **Fit the model** using training and validation sets
- **Final tests** on evaluation data set
- **Deploy**! May open a can of worms…

Errors

- Testing is a probabilistic process – the answer is never definite (depends on experiment, significance, etc…)
- When training an algorithm (ML) we need to prepare for the same – not every answer will be perfect!
- In principle errors are related to the fact that we always operate on finite samples – not on populations (regression of information)
- **Type I error** – reject H_0 when it is true (false negative)
- **Type II error** – accept it when it is false (false positive)

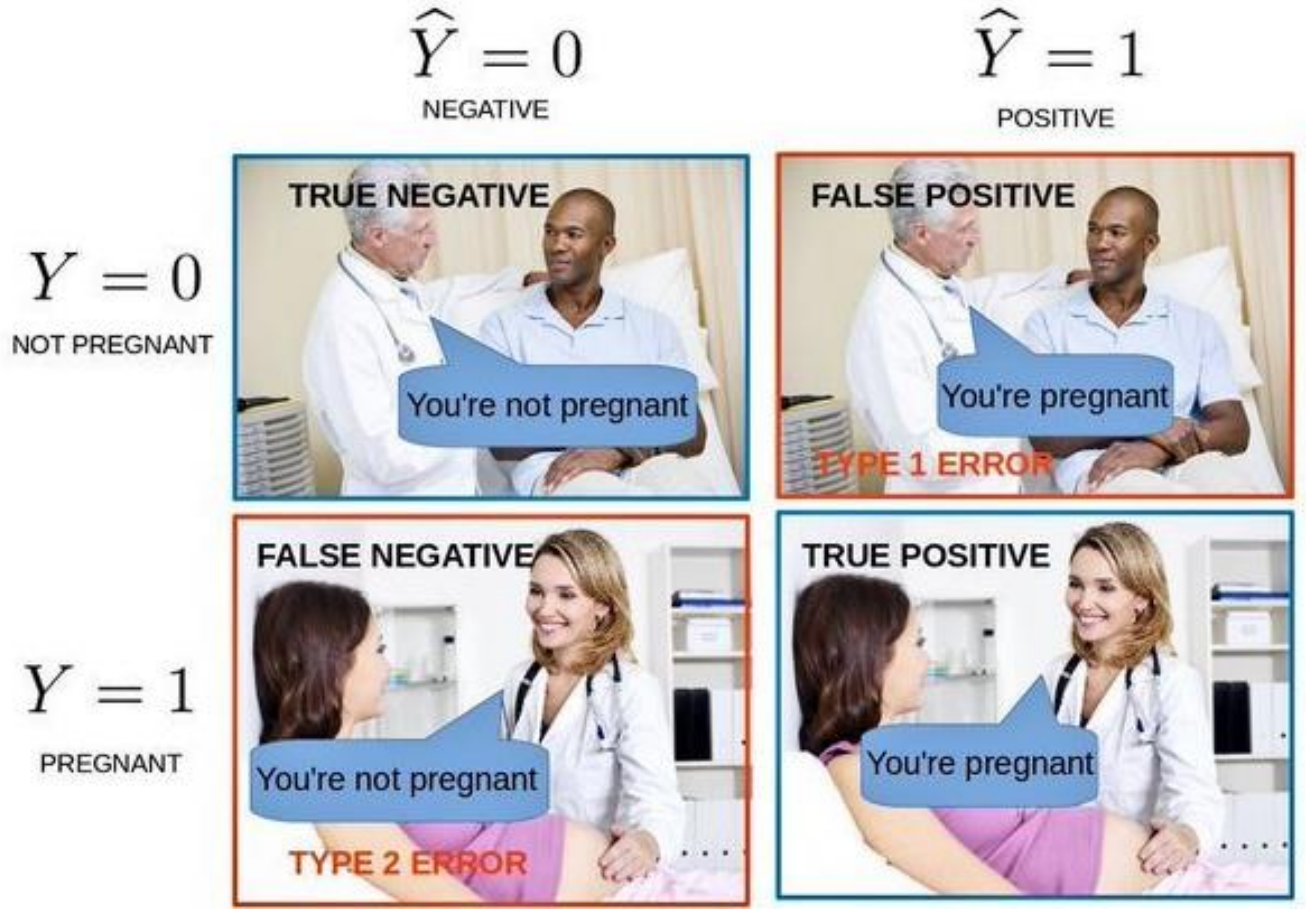
Confusion Matrix (I)

- This is an important tool to asses the quality of our trained model

		Predicted class	
		P	N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

- True positive (TP) – predicted = actual
- True negative (TN) – predicted = actual
- False negative (FN) – predicted ≠ actual
- False positive (FP) – predicted ≠ actual

Confusion Matrix (II)



Confusion Matrix

- Base on the CM we can provide some measures to asses the quality (quantitative!)
- **Precision** (P): or how are we sensitive to true positive hits (how often our signal is predicted correctly)

$$P = \frac{TP}{TP + FP}$$

- **Recall** (R): **sensitivity** (true positive rate), what fraction of true signal was predicted as signal

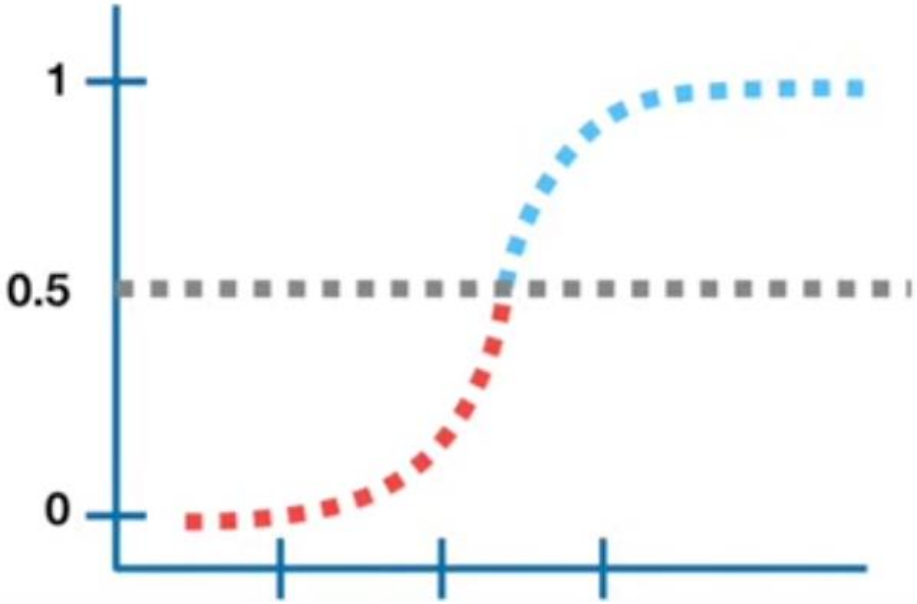
$$R = \frac{TP}{TP + FN}$$

- **F1 score**: (harmonic mean of the precision and recall)
- **Specificity** (S): what fraction of noise was predicted as noise

$$S = \frac{TN}{TN + FP}$$

Tuning the decision threshold

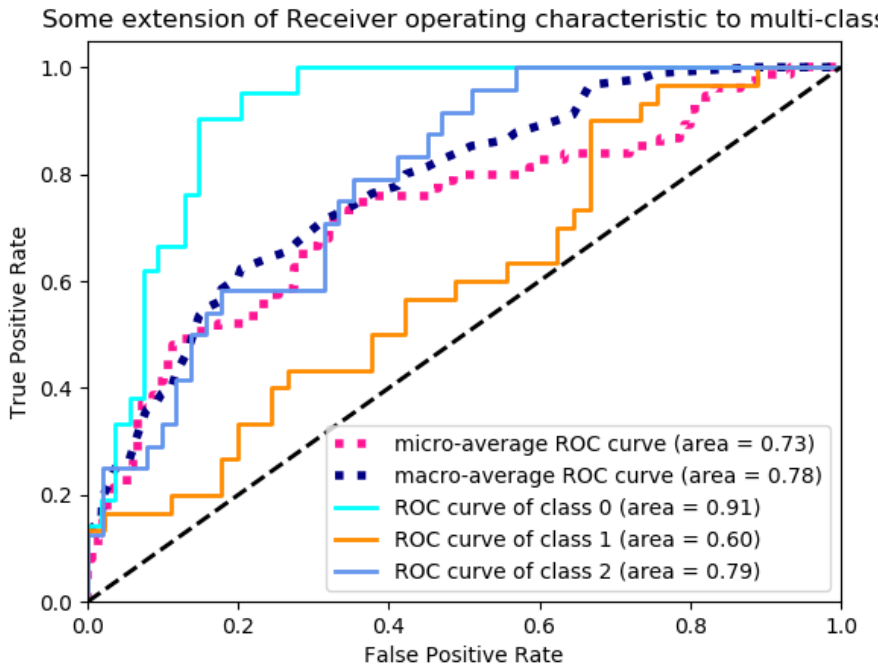
- Say, we trained a model (the plot below may represent a logistic regression problem)
- What decision threshold should we choose? Are there any other stipulations than the loss?



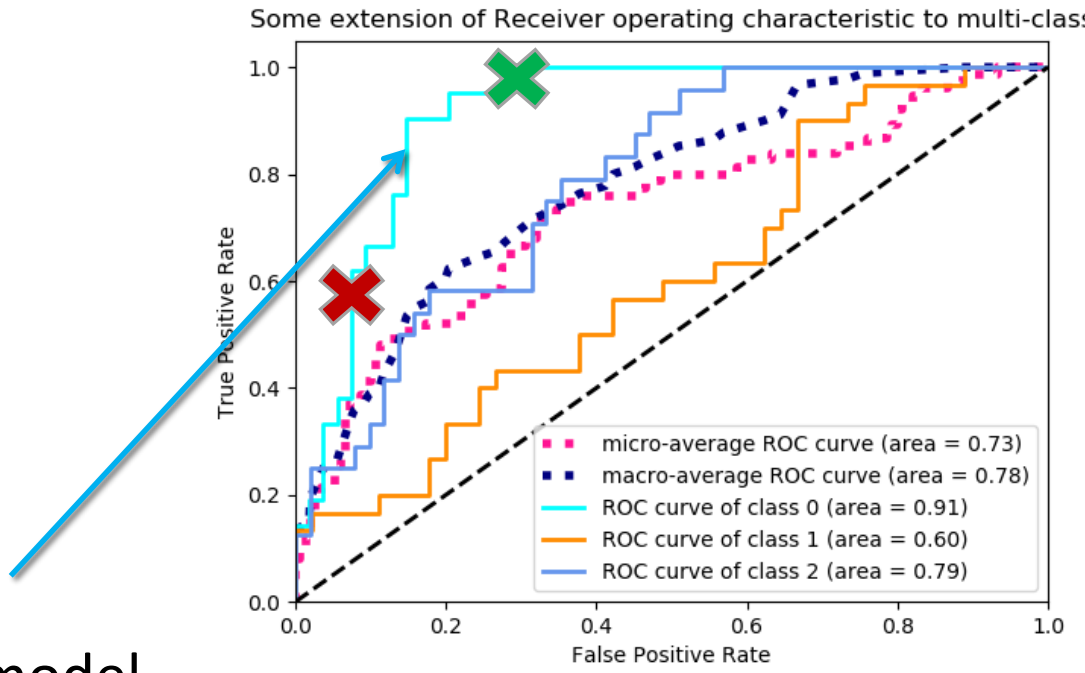
ROC

- **Receiver operating characteristic curve**
- It expresses the dependence of TP events rate (sensitivity) versus FP events rate (aka **1-Specificity**)

This basically shows how kind of a trade-off we are willing to accept – the measure of goodness is area under ROC - A_{ROC} (AUC)



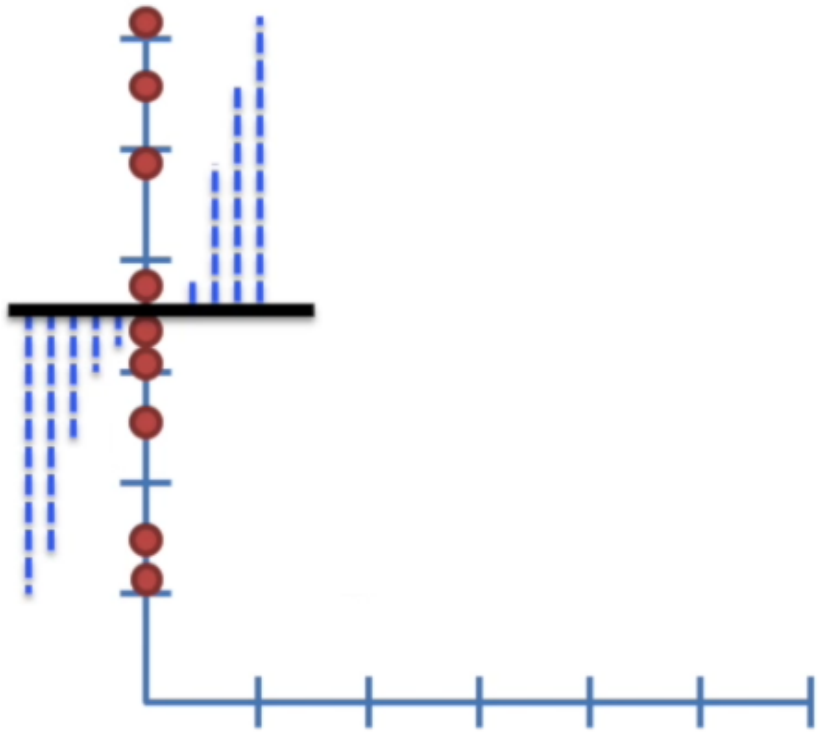
ROC



Green threshold is better than **red**

The best model

R-squared

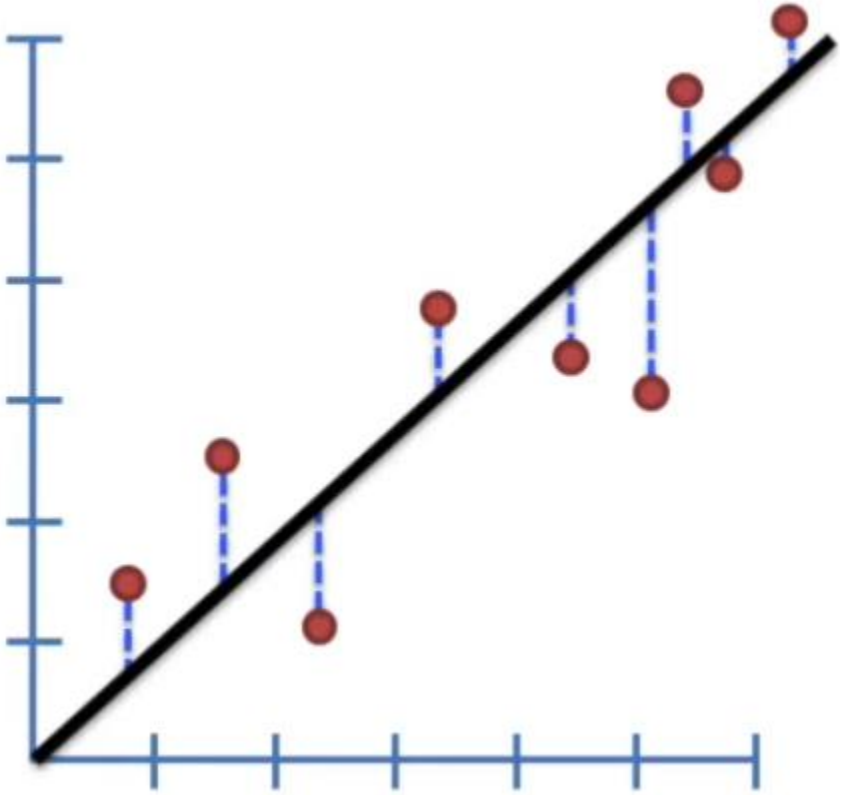


We collected a data sample and want to understand its variance
For the moment we just consider the data set as 1-dim one

$$s_{Tot} = \sum_{i/1}^{n} (y_i - \bar{y})^2$$

$$Var_{Tot} = \frac{1}{n} \sum_{i/1}^{n} (y_i - \bar{y})^2$$

R-squared



Then, we decided to use another variable to try to understand our data better
And, now we can even fit the model to the data points

$$s_{Res} = \sum_{i/1}^{n} (y_i - f_i)^2 \, , f_i - \text{model}$$

$$Var_{Res} = \frac{1}{n} \sum_{i/1}^{n} (y_i - f_i)^2$$

R-squared

- This is a convenient measure to check how our **model performs in explaining the variation in our data sample**
- In other words – how well our model minimises the variance compare to calculating just a simple mean
- Can be negative – if our model is not useful at all…

$$\bar{y} = \frac{1}{n}\sum_{i/1}^{n} y_i \quad ss_{Tot} = \sum_{i/1}^{n} (y_i - \bar{y})^2$$

$$ss_{Res} = \sum_{i/1}^{n} (y_i - f_i)^2 \, , f_i - \text{model}$$

$$R^2 = 1 - \frac{SS_{Res}}{SS_{Tot}}$$

Practical part

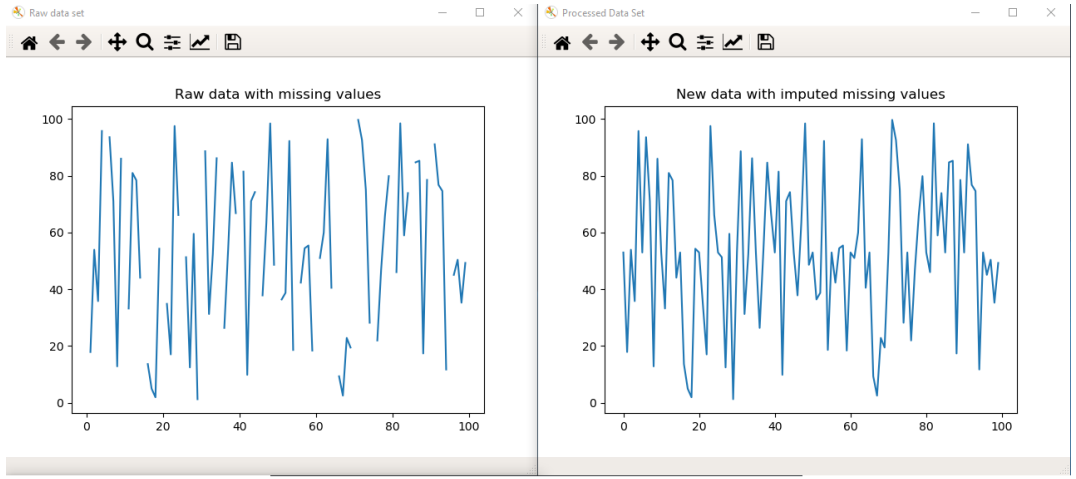
- Toy data pre-processing
- Training the perceptron
- Training the perceptron with pre-processed data (PCA)
- GAN model – training a sine wave generator

Toy pre-processing

```
# generic script for data exploration - add missing values
# with a strategy NAN for missing and and mean for the values
# to be imputed

import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer as Imputer
from matplotlib import pyplot as plt

data = np.load('sample.npy')

# Plot raw data.
plt.figure('Raw data set')
plt.title('Raw data with missing values')
plt.plot(data)

# Impute missing values.
imputer = Imputer()
data = imputer.fit_transform(data)

plt.figure('Processed Data Set')
plt.title('New data with imputed missing values')
plt.plot(data)

# Scale data.
scaler = StandardScaler()
data = scaler.fit_transform(data)

plt.figure('Scaled data set')
plt.title('Scaled data set - ready for the training')
plt.plot(data)

plt.show()
```

Write these lines, the data will be provided for you

Toy pre-processing



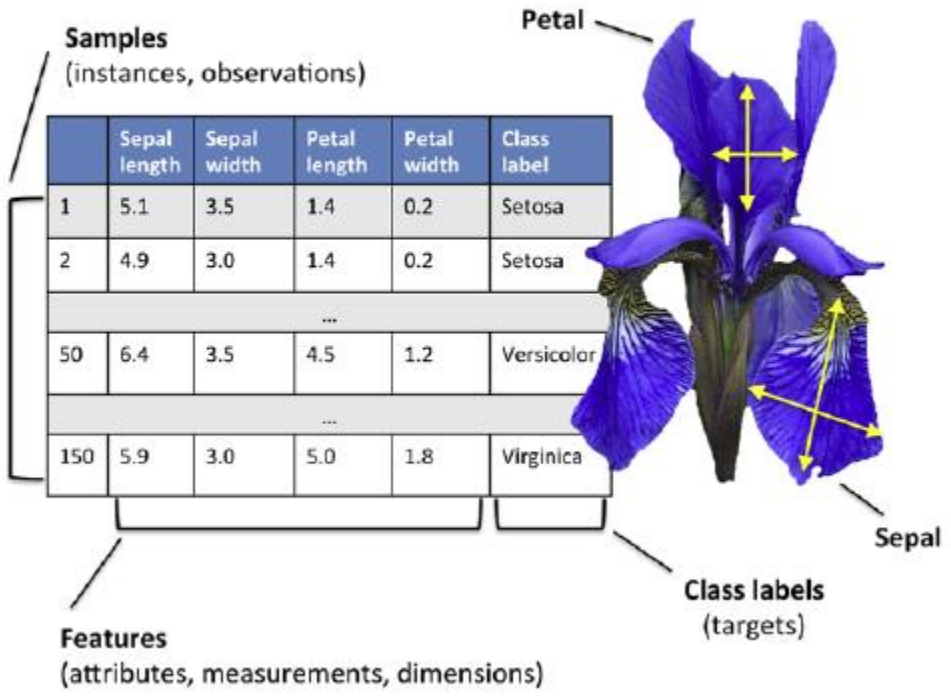
Filling the gaps and scaling are good tricks for training data preparation

Perceptron training – raw data

Iris setosa

Iris versicolor

Perceptron training – raw data

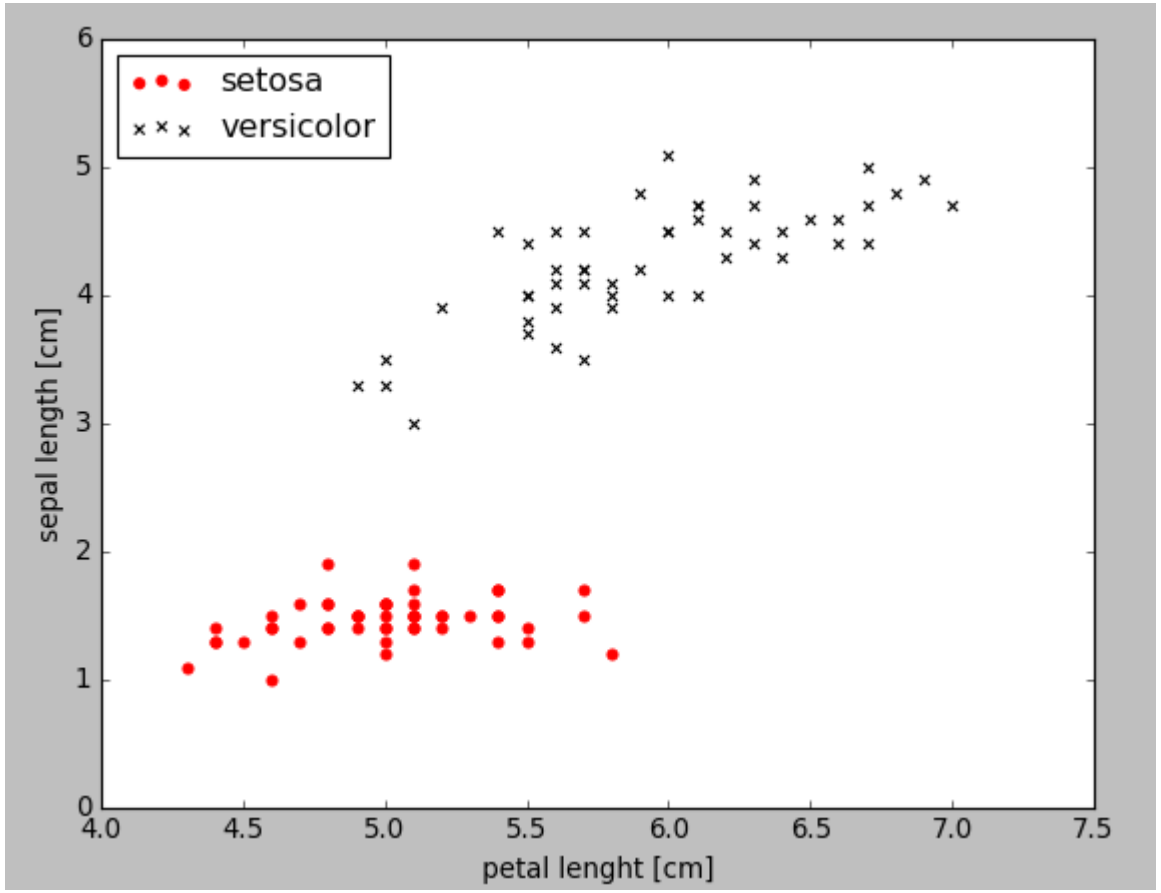


	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

$$x_j = \begin{pmatrix} x_j^{(1)} \\ . \\ . \\ . \\ x_j^{(m)} \end{pmatrix}$$

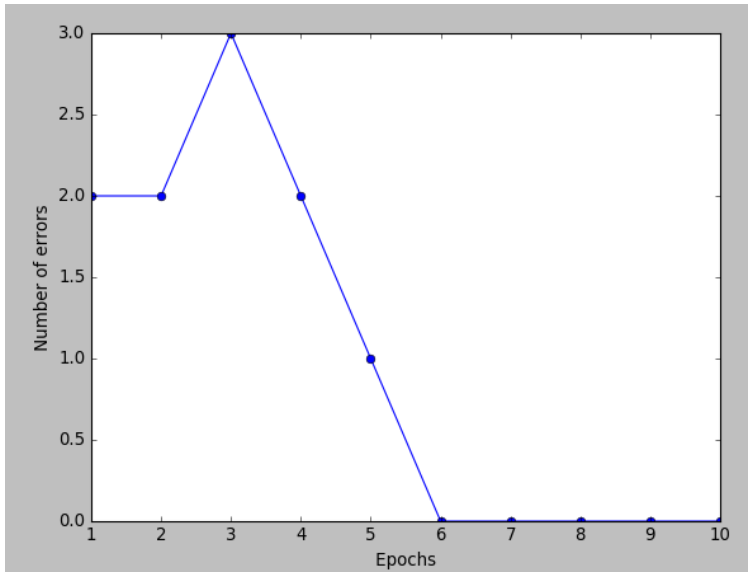
$x^{(i)} = \left(x_1^{(i)}, \ldots, x_n^{(i)}\right)$ - one instance

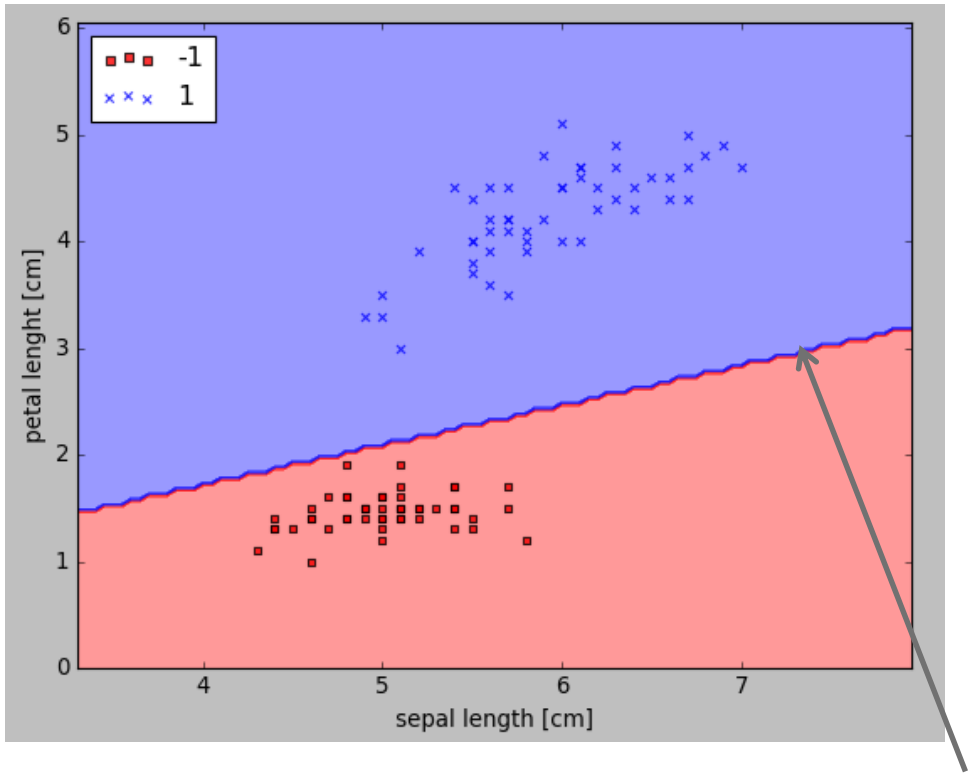
Perceptron training – raw data



Perceptron training – raw data

Loss function evolution





Our algorithm learned this decision boundary line

Perceptron training – PCA

```
# decomposition sequence
from sklearn import decomposition
pca = decomposition.PCA()

# here we do all the math...
iris_pca = pca.fit_transform(iris_x1)

# directions of data variations
pca.explained_variance_ratio_
array([ 0.92461621,  0.05301557,  0.01718514,  0.00518309])

# drop the last two features
pca = decomposition.PCA(n_components=2)

# repeat PCA
iris_X_prime = pca.fit_transform(iris_x1)
iris_X_prime.shape
(150, 2)
pca.explained_variance_ratio_.sum()
0.97763177502480336
```

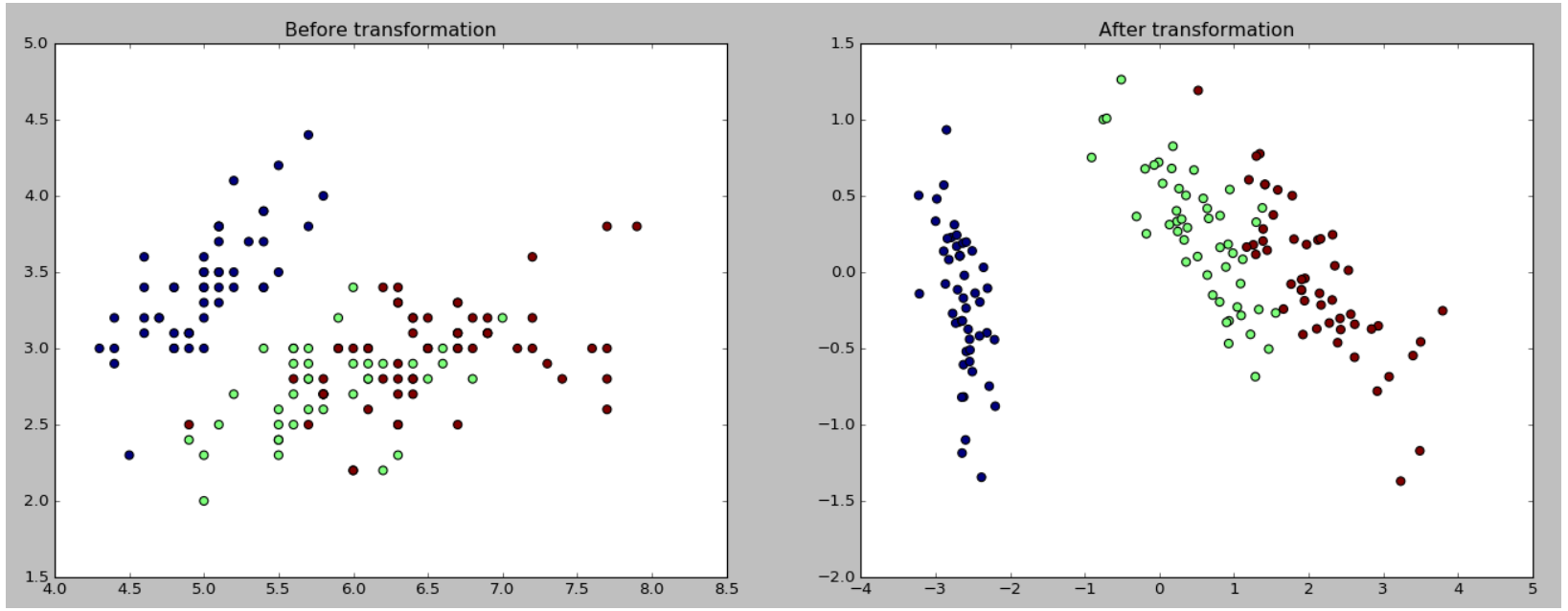
- This 97.7% is a compromise, could include the third variable to increase this – depends on the problem and possible consequences

Perceptron training – PCA

- Ok, let's have a look at what happend

```
# use matplotlib to visualise
fig = plt.figure(figsize=(20,7))
input_data = fig.add_subplot(121)

# before the transfomation...
input_data.scatter(iris_x1[:,0], iris_x1[:,1], c=x2, s=40)
<matplotlib.collections.PathCollection object at 0x7ff4b3a49390>
input_data.set_title('Before transformation')
<matplotlib.text.Text object at 0x7ff4b3a92290>

# ... and after
tr_data = fig.add_subplot(122)
tr_data.scatter(iris_X_prime[:,0], iris_X_prime[:,1], c=x2, s=40)
<matplotlib.collections.PathCollection object at 0x7ff4b3a37bd0>
tr_data.set_title('After transformation')
<matplotlib.text.Text object at 0x7ff4b3af2690>
plt.show()
```

Perceptron training – PCA



- We see some interesting changes
- I **strongly recommend you** to use the python code we developed earlier to try to check the performance with the reduced data set!

GAN – new and powerful

- First appeared in 2014/2015 – amazing fact: we can build two competing models (usual we mean two deep ANNs) where on can fool the other
- An adversary can learn to generate fake data that can make any trained model to make bad decisions 100% of a time
- These fake data (synthetic data) may not resemble the real data at all! But in time we also can make this happen
- For some time it seemed that the ANN approach is doomed!

GAN mastery

- It is kind of scary, that the AI is able to produce such images…
- They can be used to fool a trained model for face recognition to make a bad decision
- But this weakness can be forge into success as well we just need a bit of imagination!

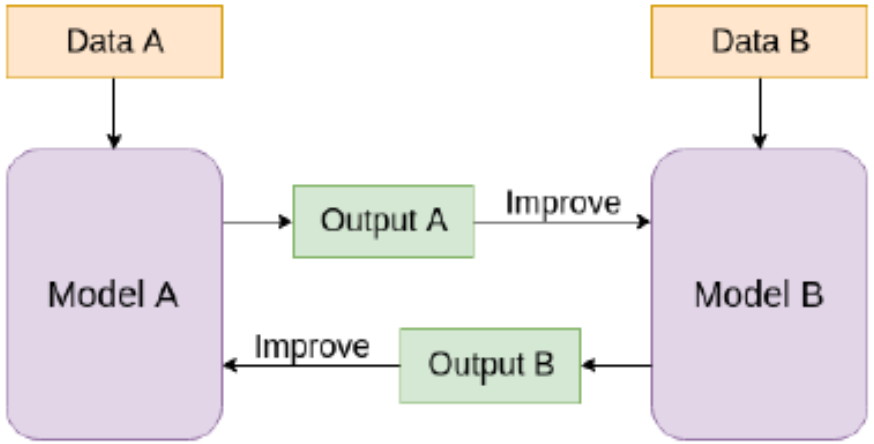
A tale of two kingdoms

- This story goes in different flavours, but the conclusions are always the same!
- Imagine you have two kingdoms, each have its own blacksmiths that can make armour and weapon. One king never allowed any conflicts and the other demanded constant cross-checks of armour and weapon
- You can guess which kingdom would be better in military technology!
- **The same goes for the GAN approach – constant challenge**!

Events generators

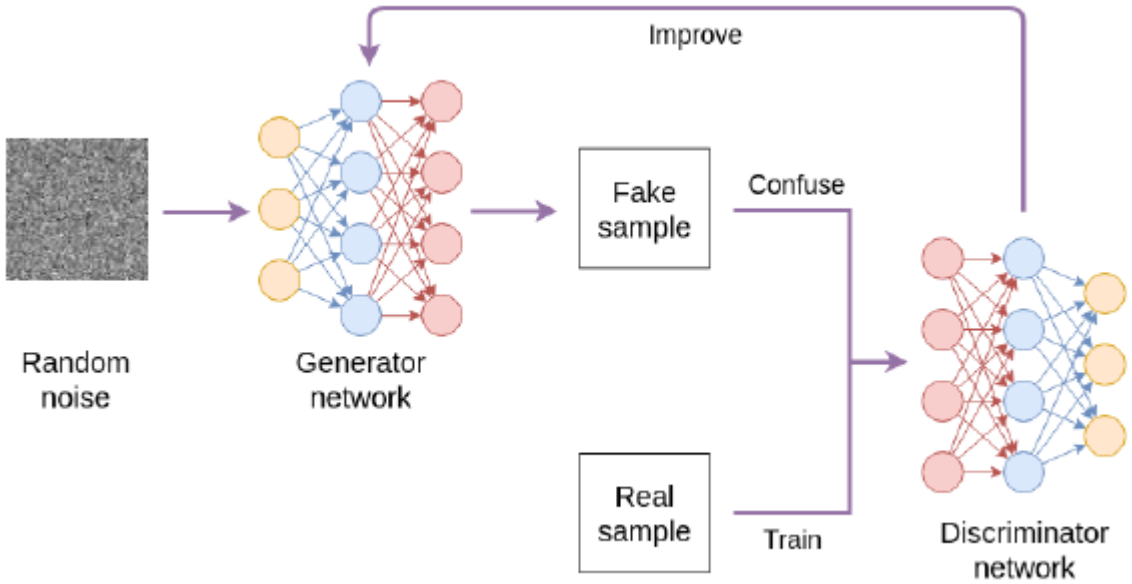
- The idea is actually quite old: physics generators that mimic Nature
- We can say that the **generators** tries to „**map low-dimension data to high-dimension data**”
- Classification models **do the opposite**!
- So, the training were two models make an attempt to weaken each other and on the long run enhance each other is called **adversarial learning**
- So, when designing a GAN system we need two models!

Adversarial systems



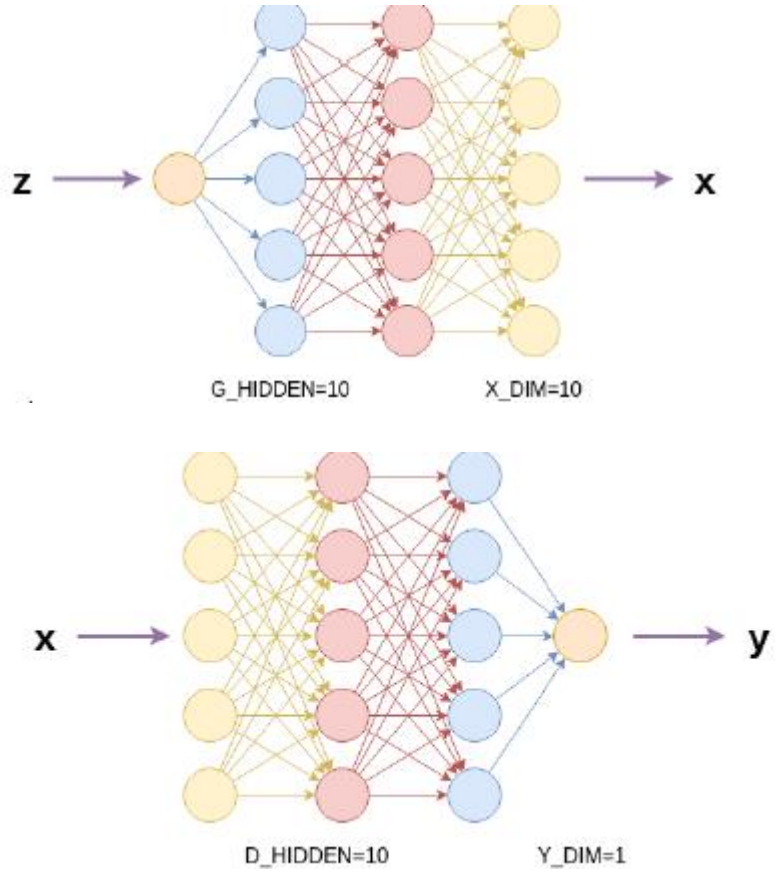
- Adversarial learning
  - Need two models A and B
  - The output of B is going to improve the A and the output of A do the same for B
  - One model will need „a real data” sample for training
  - Not all data are going to be fake

GAN architecture



- Here the generator model is using the noise to produce fake data that are fed to the second model
- The second one is a classification model that makes an attempt on detecting the fake data
- The differences between the generated and real data are used to improve the generator
- Real data are used to train the discriminator model

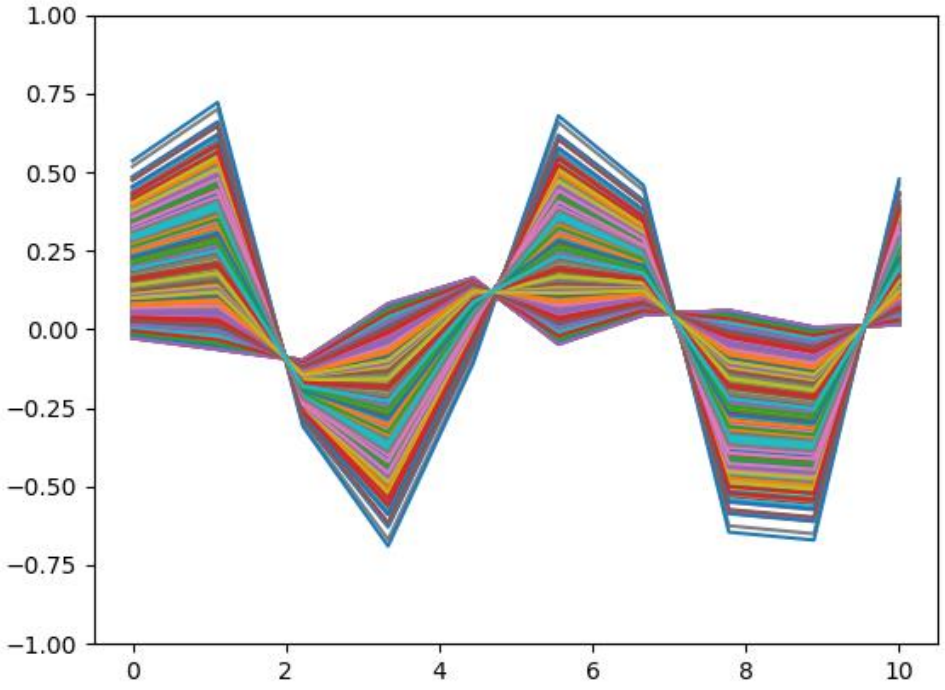
Simple GAN

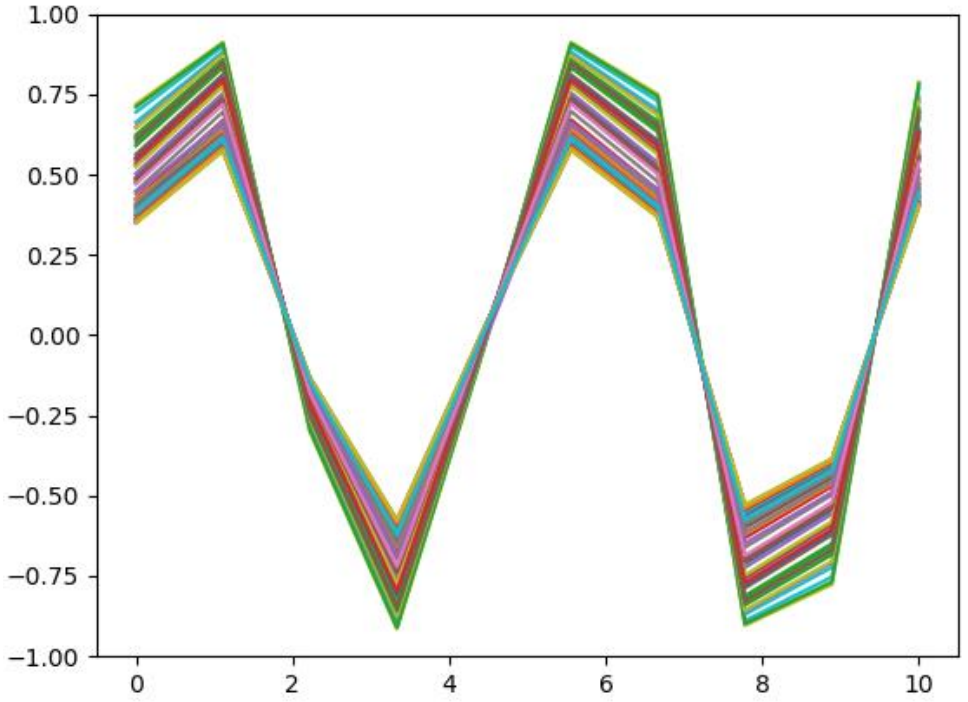


- Can implement a simple GAN system that generates sine waveform and feed it to the discriminator that aims at recognition fake/true
- Two MLP ANNs
- Code can be written by hand to understand the basic principles of such systems (just need numpy)

Performance of the Generator

- Example output of the generator for different settings





The END is The BEGINNING

GAN optimisation rules

- Let set $\mathcal{G}$ and $\mathcal{D}$ to represent the generator and discriminator models respectively, the performance function is $\mathcal{V}$. The optimisation objective can be written as follow:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \mathcal{V}(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{\vec{x}}[log\mathcal{D}(\vec{x})] + \mathbb{E}_{\vec{x}^*}\left[log\left(1 - \mathcal{D}(\vec{x}^*)\right)\right]$$

- Here: $\vec{x}$ - real samples, $\vec{x}^* = \mathcal{G}(z)$ - generated samples (z represents noise), $\mathbb{E}_{\vec{x}}[f]$ is the average value of any function over the sample space

- Model $\mathcal{D}$ should maximise the „good" prediction for the real sample - we are looking for the max – gradient ascent update rule

$$\vec{\theta}_{\mathcal{D}} \leftarrow \vec{\theta}_{\mathcal{D}} + r \cdot \frac{1}{m} \nabla_{\vec{\theta}_{\mathcal{D}}} \sum_{i/1}^{i/m} \left[log\mathcal{D}(\vec{x}) + log\left(1 - \mathcal{D}(\vec{x}^*)\right)\right]$$

- Model $\mathcal{G}$ must trick the discriminator, thus, it minimise the $1 - \mathcal{D}(\vec{x}^*) = 1 - \mathcal{D}\left(\mathcal{G}(z)\right)$

$$\vec{\theta}_{\mathcal{G}} \leftarrow \vec{\theta}_{\mathcal{G}} - r \cdot \frac{1}{m} \nabla_{\vec{\theta}_{\mathcal{G}}} \sum_{i/1}^{i/m} \left[log\left(1 - \mathcal{D}(\vec{x}^*)\right)\right]$$