Introduction to Machine Learning

Oleg Filatov [1] Stepan Zakharov [2]

[1]DESY [2]Novosibirsk State University

Outline

- Briefly about data
- Supervised Learning
  - General picture: objects, features, models
  - Types of problems
  - Model training/testing
  - Examples
- Semi-supervised Learning
- Unsupervised Learning
- Reinforcement Learning

Briefly about data

Introduction

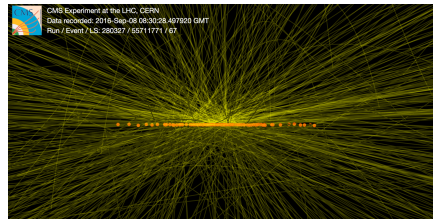
In the beginning was the Data:

- Information about goods and sales
- Data from subsystems of detector
- Subscribers database

Shall one question the Data:

- predict demand for this product?
- estimate energy of a particle?
- like or dislike the post?

How can we answer these questions?



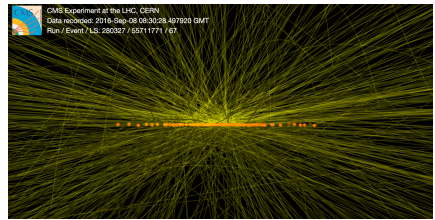
Introduction

In the beginning was the Data:

- Information about goods and sales
- Data from subsystems of detector
- Subscribers database

Shall one question the Data:

- predict demand for this product?
- estimate energy of a particle?
- like or dislike the post?



How can we answer these questions? → this lecture

Can we use data out of the box?

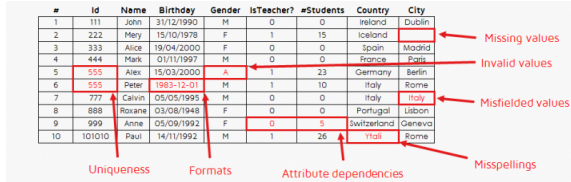
- Most likely, no: garbage in, garbage out!
- Main drawbacks of raw data:
  - Not in ML comprehensive format
  - Can have missing values or might be noisy
  - Useless/lack of useful features

! Analysing data improves its understanding

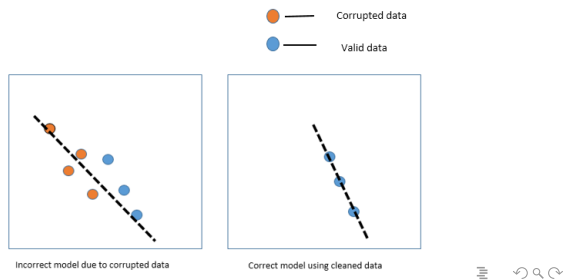
Data preprocessing

Main techniques:

- Imputation of missing data
- Data cleaning
- Handling outliers
- Shuffling
- Balancing classes
- Partitioning



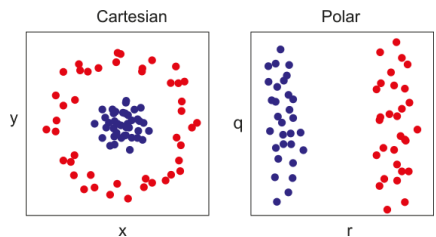
#	Id	Name	Birthday	Gender	IsTeacher?	#Students	Country	City
1	111	John	31/12/1990	M	0	0	Ireland	Dublin
2	222	Mery	15/10/1978	F	1	15	Iceland	
3	333	Alice	19/04/2000	F	0	0	Spain	Madrid
4	444	Mark	01/11/1997	M	0	0	France	Paris
5	555	Alex	15/03/2000	A	1	23	Germany	Berlin
6	555	Peter	1983-12-01	M	1	10	Italy	Rome
7	777	Calvin	05/05/1995	M	0	0	Italy	Italy
8	888	Roxane	03/08/1948	F	0	0	Portugal	Lisbon
9	999	Anne	05/09/1992	F	0	5	Switzerland	Geneva
10	101010	Paul	14/11/1992	M	1	26	Ytali	Rome



Feature engineering

- **Construction:** vectorization techniques + your domain knowledge
- **Transformation:** functions, scaling, binning, croping, resizing, blurring
- **Selection:** PCA, statistical tests, feature importance, hand-picking

Example: Using using Polar coordinates it's much easier to separate red dots from blue than using Cartesian coordinates:



Supervised Learning

More or less formal

- Let's introduce **object** (house) $\boldsymbol{x} \in X$, where X is a set of objects
- Objects are described by **features** (house area)
- We use objects to make predictions
- Output of the prediction is **answer** (house price) $y \in Y$, where Y is a set of answers
- There is an **unknown function** $g\colon X \to Y$ we are looking for

Supervised Learning is called "supervised" because there is a set of objects with **known answers** and we use this prior knowledge while approximating $g\colon X \to Y$ relation. Effectively, the **model is supervised by our prior knowledge.**

Problem statement

Given:

- Set of object that forms training sample $\{\boldsymbol{x_1} \dots \boldsymbol{x_N}\} \subset X$
- Set of answers $y_i = g(\boldsymbol{x_i}),\ \{y_1 \dots y_N\} \subset Y$

Find:

- a: $X \to Y$ – algorithm that approximates g on the whole set X

Questions:

- Why do we need a?
- What does ”approximates” mean?
- How to find this algorithm?

What is algorithm?

- Let $a(\boldsymbol{x}, \boldsymbol{\theta})$ be a function that depends on object $\boldsymbol{x}$ and vector of parameters $\boldsymbol{\theta}$.
- We know that g: $\begin{bmatrix} g(\boldsymbol{x_1}) \\ \cdots \\ g(\boldsymbol{x_N}) \end{bmatrix} = \begin{bmatrix} y_1 \\ \cdots \\ y_N \end{bmatrix} = \boldsymbol{y}$, the exact expression for g **is unknown**
- Let's try to approximate g with model a such as: $\begin{bmatrix} a(\boldsymbol{x_1}, \boldsymbol{\theta}) \\ \cdots \\ a(\boldsymbol{x_N}, \boldsymbol{\theta}) \end{bmatrix} = \begin{bmatrix} y'_1 \\ \cdots \\ y'_N \end{bmatrix} = \boldsymbol{y'}$
- If approximation is reliable, then $\boldsymbol{y'}$ is close to $\boldsymbol{y}$
- **Loss function** $\mathcal{L}(a(\boldsymbol{x_i}), y_i)$ gives the state of closeness between prediction $a(\boldsymbol{x_i})$ and real answer y_i (error estimation for the particular object)

Optimisation problem

Note: Loss function returns the error only for a single object.

- Define **empirical risk** for a whole set of objects: $Q(a, X, Y) = \sum_{i=1}^{N} \mathcal{L}\big(a(\boldsymbol{x_i}), y_i\big)$
- $Q(a, X, Y)$ is usually called "loss function" too because there is no need to use directly $\mathcal{L}\big(a(\boldsymbol{x}), y\big)$ for single object
- Optimal algorithm $\hat{a}$ minimizes $Q(a, X, Y)$ on the training set $[X, Y]$

$$\hat{a} = \arg\min_{a \in A} Q(a, X, Y)$$

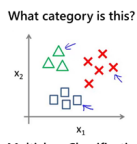
This procedure is called **training**

Supervised Learning types

Answer set Y can be different. Its domain mainly defines the problem to work with:

- $Y = \{0, 1\} \Rightarrow$ **binary classification**
  Ex: $Y = \{\text{signal, background}\}$
- $Y \subset \mathbb{Z} \Rightarrow$ **multiclass classification**
  Ex: $Y = \{\text{electron, muon, tau, } \ldots\}$
- $Y \subset \mathbb{Z}$ + total order $\Rightarrow$ **ranking**
  Ex: $Y = \{0, 1, \ldots, N\}$
- $Y = \mathbb{R} \Rightarrow$ **regression**
  Ex: $Y = p_T$ of jets

Binary Classification



Multiclass Classification

Regression

Linear models

Classification

- $Y = \{0, 1\}$
- N objects with K numeric features: $D = \mathbb{R}^K$
- $a(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{j=1}^{K} f_j(\boldsymbol{x}) \cdot \boldsymbol{\theta_j} = \langle \boldsymbol{x}, \boldsymbol{\theta} \rangle$
- $\mathcal{L}\big(a(\boldsymbol{x_i}), y_i\big) = \big[a(\boldsymbol{x}) \neq y_i\big]$
- $Q(a, X, Y) = \sum_{i=1}^{N} \mathcal{L}\big(a(\boldsymbol{x_i}), y_i\big)$

Regression

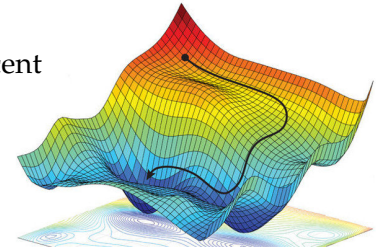
- $Y = \mathbb{R}$
- N objects with K numeric features: $D = \mathbb{R}^K$
- $a(\boldsymbol{x}, \boldsymbol{\theta}) = \langle \boldsymbol{x}, \boldsymbol{\theta} \rangle$
- $\mathcal{L}\big(a(\boldsymbol{x_i}), y_i\big) = \big(a(\boldsymbol{x}) - y_i\big)^2$
- $Q(a, X, Y) = \sum_{i=1}^{N} \mathcal{L}\big(a(\boldsymbol{x_i}), y_i\big)$

At the final step both problems transfer into the procedure of empirical risk minimization.

Regression — example

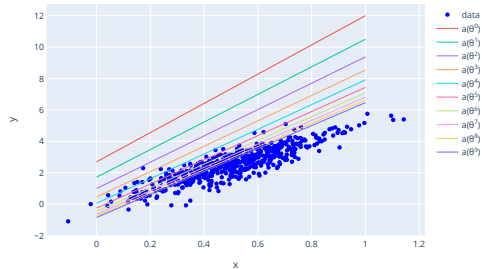
- Let's consider a problem where objects have only one feature: $\boldsymbol{x} = [1, f_1(\boldsymbol{x})]$
- Define parameters vector $\boldsymbol{\theta} = [\theta_0, \theta_1]$
- Well there is analytical solution for this problem (look at the backup)
- How can we find minimum of $Q(\boldsymbol{\theta})$ without solving matrix equation?

Better approach? — gradient descent

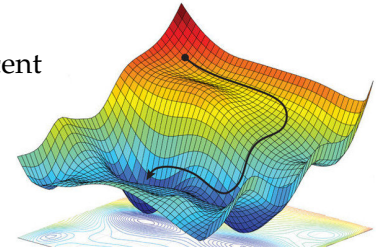


Idea: bring up the information about the direction to the minimum of Q using its **antigradient**

- Start with some vector $\boldsymbol{\theta^0}$
- $\hat{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta^0}$
- ↓ Calculate direction to the minimum:
$$-\nabla Q(\hat{\boldsymbol{\theta}}) = -\left[\frac{\partial Q}{\partial \theta_0}, \frac{\partial Q}{\partial \theta_1}\right]_{\hat{\boldsymbol{\theta}}}$$
- ↓ Move towards the minimum:
$\hat{\boldsymbol{\theta}} \leftarrow \hat{\boldsymbol{\theta}} - \eta \nabla Q(\hat{\boldsymbol{\theta}})$
- ↓ Repeat until convergence

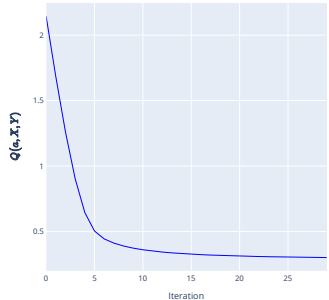


Better approach? — gradient descent



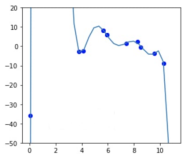
Idea: bring up the information about the direction to the minimum of Q using its **antigradient**

- Start with some vector $\boldsymbol{\theta^0}$
- $\hat{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta^0}$
- ↓ Calculate direction to the minimum:
$$-\nabla Q(\hat{\boldsymbol{\theta}}) = -\left[\frac{\partial Q}{\partial \theta_0}, \frac{\partial Q}{\partial \theta_1}\right]_{\hat{\boldsymbol{\theta}}}$$
- ↓ Move towards the minimum:
$\hat{\boldsymbol{\theta}} \leftarrow \hat{\boldsymbol{\theta}} - \eta \nabla Q(\hat{\boldsymbol{\theta}})$
- ↓ Repeat until convergence



Measuring model performance

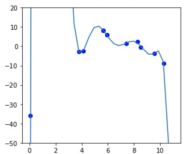
- After training we want to understand the quality of the model
- Therefore we need something like:



$\Rightarrow$ Number

Measuring model performance

- After training we want to understand the quality of the model
- Therefore we need something like:

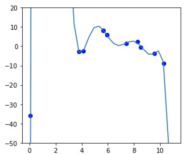


$\Rightarrow$ Number

- Since $\mathcal{L}(a(\boldsymbol{x}), y)$ is an error estimation

Measuring model performance

- After training we want to understand the quality of the model
- Therefore we need something like:

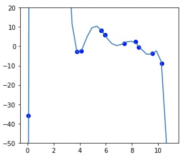


$\overset{\text{metric}}{\Rightarrow}$ Number

- Since $\mathcal{L}(a(\boldsymbol{x}), y)$ is an error estimation
- Then it can be used as **metric**

Measuring model performance

- After training we want to understand the quality of the model
- Therefore we need something like:



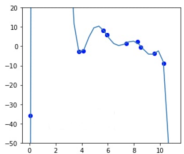
$\overset{\text{metric}}{\Rightarrow}$ Number

- Since $\mathcal{L}(a(\boldsymbol{x}), y)$ is an error estimation
- Then it can be used as **metric**

- Which data to use for measurements?

Measuring model performance

- After training we want to understand the quality of the model
- Therefore we need something like:

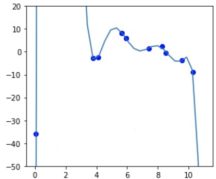


$\overset{\text{metric}}{\Rightarrow}$ Number

- Since $\mathcal{L}(a(\boldsymbol{x}), y)$ is an error estimation
- Then it can be used as **metric**

- Which data to use for measurements?

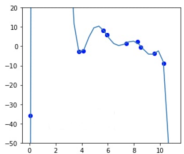
- Well, we can use the training dataset:



- Then it might happen that $\mathcal{L}_{\text{train}} \to 0$

Measuring model performance

- After training we want to understand the quality of the model
- Therefore we need something like:

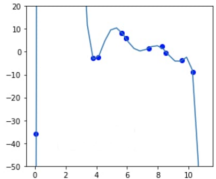


$\overset{\text{metric}}{\Rightarrow}$ Number

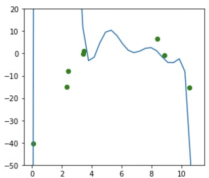
- Since $\mathcal{L}(a(\boldsymbol{x}), y)$ is an error estimation
- Then it can be used as **metric**

- Which data to use for measurements?

- Well, we can use the training dataset:



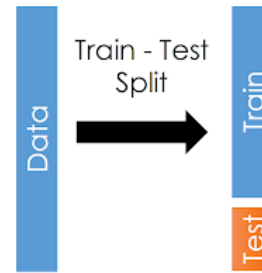
- Then it might happen that $\mathcal{L}_{\text{train}} \to 0$
- But on unseen data $\mathcal{L}_{\text{test}}$ will be obscenely large:



Where to get new data?

train/test split

- Divide initial dataset into two parts: train and test
- Train on a train set, then measure performance on a test set
- ! Test dataset should not be used in training
- ! Split between train and test should be uniform



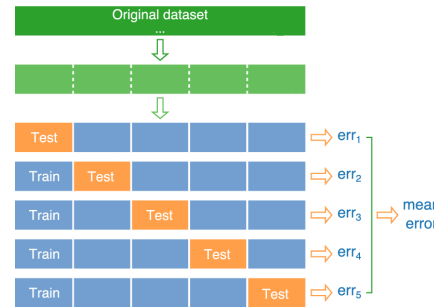
Problem: results depend on the choice of splitting

→ how to check model performance on the whole dataset w/o split dependance?

Where to get new data? — cross-validation

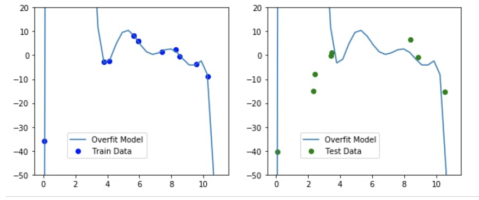
- Split dataset into K folds
- Train on K-1 folds, test on remaining one
- Repeat for different folds
- Aggregate metrics to get mean and variance

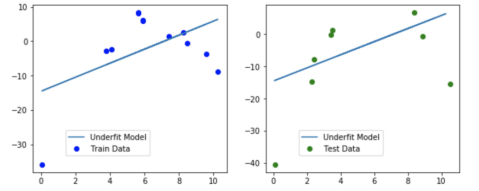
✓ Provides interval error estimation
✓ More robust to train–test split
× Requires computational resources
× Use only a part of dataset for training

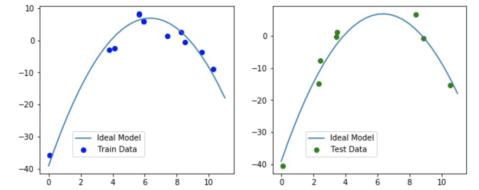


Example — Polynomial regression

- Upper model performs well on train and fails on test ⇒ **overtraining**
- Center model performs bad both on train and test sets ⇒ **undertraining**
- Lower model shows acceptable results on train an test sets





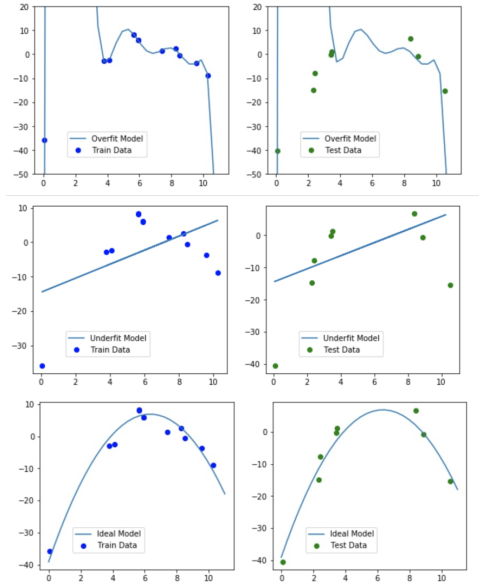


Example — Polynomial regression

- Upper model performs well on train and fails on test ⇒ **overtraining**
- Center model performs bad both on train and test sets ⇒ **undertraining**
- Lower model shows acceptable results on train an test sets

⇓

Let's analyze these outcomes



Possible outcomes

Train – bad
Test – bad
$\Downarrow$
Undertraining

- Model is too simple
- Model can't capture the underlying structure of data
- Low variance, high bias

Train – good
Test – bad
$\Downarrow$
Overtraining

- Model is too complicated
- Model captures noise instead of structure of data
- High variance, low bias

What to do?

- Increase model's capacity
- Train longer

- Catch it before it happens (e.g. early stopping)
- Reflect about noise level vs model complexity

Possible outcomes

Train – bad
Test – bad
⇓
Undertraining

- Model is too simple
- Model can't capture the underlying structure of data
- Low variance, high bias

Train – good
Test – bad
⇓
Overtraining

- Model is too complicated
- Model captures noise instead of structure of data
- High variance, low bias

What to do?

- Increase model's capacity
- Train longer

- Catch it before it happens (e.g. early stopping)
- Reflect about noise level vs model complexity
- Automatically limit algorithm space → **regularization** (backup)

Before we end with regression

There exist variety of loss functions. Choose one that suits to your problem

$$\mathcal{L}_{\text{train}} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}\big(a(\boldsymbol{x_i}), y_i\big) \equiv \mathbb{E}_{\text{p}(\boldsymbol{x},y)}[\mathcal{L}\big(a(\boldsymbol{x}), y\big)] \leftarrow \textbf{expected value}$$

Before we end with regression

There exist variety of loss functions. Choose one that suits to your problem

$$\mathcal{L}_{\text{train}} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}\big(a(\boldsymbol{x_i}), y_i\big) \equiv \mathbb{E}_{\text{p}(x,y)}[\mathcal{L}\big(a(\boldsymbol{x}), y\big)] \leftarrow \textbf{expected value}$$

- **MSE**$= \mathbb{E}_{\text{p}(x,y)}\left[\big(a(\boldsymbol{x}) - y\big)^2\right]$
- **RMSE** $= \sqrt{\text{MSE}}$
- **MSLE** $= \mathbb{E}_{\text{p}(x,y)}\left[\big(\log(a(\boldsymbol{x}) + 1) - \log(y + 1)\big)^2\right]$
- **Coefficient of determination**

$$R^2 = 1 - \frac{\mathbb{E}_{\text{p}(x,y)}\left[\big(a(\boldsymbol{x}) - y\big)^2\right]}{\mathbb{E}_{\text{p}(x,y)}\left[\big(\bar{y} - y\big)^2\right]}$$

- **MAE**$= \mathbb{E}_{\text{p}(x,y)}\left[\big|a(\boldsymbol{x}) - y\big|\right]$
- **MAPE** $= 100\% \cdot \mathbb{E}_{\text{p}(x,y)}\left[\left|\dfrac{y - a(\boldsymbol{x})}{y}\right|\right]$
- **SMAPE**$= 100\% \cdot \mathbb{E}_{\text{p}(x,y)}\left[\dfrac{|y - a(\boldsymbol{x})|}{|y| + |a(\boldsymbol{x})|}\right]$

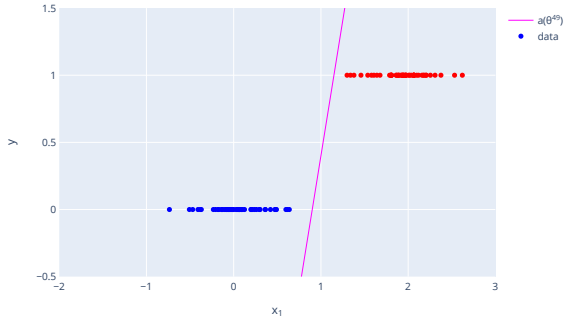
$$\textbf{Huber Loss} = \mathbb{E}_{\text{p}(x,y)}\begin{cases}\frac{1}{2}\big(a(\boldsymbol{x}) - \boldsymbol{y}\big)^2, & \big|a(\boldsymbol{x}) - \boldsymbol{y}\big| < \delta \\ \delta\big|a(\boldsymbol{x}) - \boldsymbol{y}\big| - \frac{1}{2}\delta^2, & \text{otherwise}\end{cases}$$

Classification

Example

- Let's consider the problem of two class (red, blue) classification: $Y = \{0, 1\}$
- We need to find a line* that provides the best separation of objects from different classes
- Consider objects have single feature: $\boldsymbol{x} = [1, x_1]$
- Algorithm: $a(\boldsymbol{x}, \boldsymbol{\theta}) = \text{sign}\langle \boldsymbol{x}, \boldsymbol{\theta} \rangle$

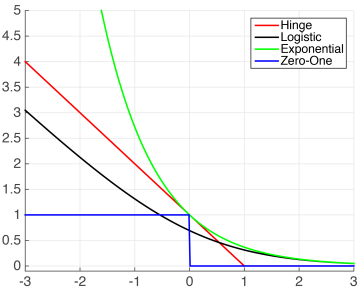
*for a model with multiple features we are looking for hyperplanes



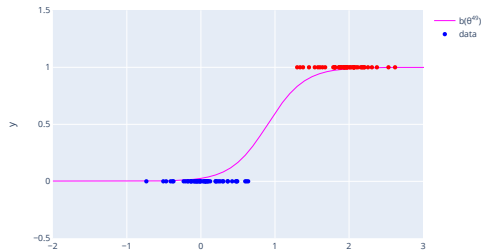
- $\mathcal{L}(\boldsymbol{\theta}) = \big[\text{sign}\langle \boldsymbol{x}, \boldsymbol{\theta} \rangle \neq y\big]$ ← step function
- $Q(\boldsymbol{\theta}) = \mathbb{E}_{\text{p}(\boldsymbol{x},y)}\big[\text{sign}\langle \boldsymbol{x}, \boldsymbol{\theta} \rangle \neq y\big]$ ← accuracy
- Then we need to minimize $Q(a(\boldsymbol{\theta}))$ by varying $\boldsymbol{\theta}$

From binary classification to logistic regression

Problem 1: $\mathcal{L}(\boldsymbol{\theta})$ isn't a smooth function
Solution: use smooth approximations



Problem 2: discrete $\{0, 1\}$ output from the classifier
Solution: reframe the problem to probabilistic perspective
$\Rightarrow$ logistic regression



Understanding the concept of probability estimation

- Assume that algorithm $a(\boldsymbol{x}, \boldsymbol{\theta})$ returns probability that object $\boldsymbol{x_i}$ is from positive class $y_i = 1$

Understanding the concept of probability estimation

- Assume that algorithm $a(\boldsymbol{x}, \boldsymbol{\theta})$ returns probability that object $\boldsymbol{x_i}$ is from positive class $y_i = 1$
- Probability that $\boldsymbol{x_i}$ belongs to class y_i:
  $p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x}_i) = a\big(\boldsymbol{x_i}, \boldsymbol{\theta}\big)^{[y_i=+1]}\big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)^{[y_i=0]}$

Understanding the concept of probability estimation

- Assume that algorithm $a(\boldsymbol{x}, \boldsymbol{\theta})$ returns probability that object $\boldsymbol{x_i}$ is from positive class $y_i = 1$
- Probability that $\boldsymbol{x_i}$ belongs to class y_i:
  $p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x}_i) = a\big(\boldsymbol{x_i}, \boldsymbol{\theta}\big)^{[y_i=+1]}\big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)^{[y_i=0]}$
- Function to be optimised: $\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x}_i)$ ← maximize probability to observe such data

Understanding the concept of probability estimation

- Assume that algorithm $a(\boldsymbol{x}, \boldsymbol{\theta})$ returns probability that object $\boldsymbol{x_i}$ is from positive class $y_i = 1$
- Probability that $\boldsymbol{x_i}$ belongs to class y_i:
  $p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x}_i) = a\big(\boldsymbol{x_i}, \boldsymbol{\theta}\big)^{[y_i=+1]}\big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)^{[y_i=0]}$
- Function to be optimised: $\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x}_i)$ ← maximize probability to observe such data
- Product is not useful, let's make sum:
  $\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} [y_i = 1] \log a(\boldsymbol{x_i}, \boldsymbol{\theta}) + [y_i = 0] \log \big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)$

Understanding the concept of probability estimation

- Assume that algorithm $a(\boldsymbol{x}, \boldsymbol{\theta})$ returns probability that object $\boldsymbol{x_i}$ is from positive class $y_i = 1$
- Probability that $\boldsymbol{x_i}$ belongs to class y_i:
  $p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x_i}) = a\big(\boldsymbol{x_i}, \boldsymbol{\theta}\big)^{[y_i=+1]}\big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)^{[y_i=0]}$
- Function to be optimised: $\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x}_i)$ ← maximize probability to observe such data
- Product is not useful, let's make sum:
  $\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} [y_i = 1] \log a(\boldsymbol{x_i}, \boldsymbol{\theta}) + [y_i = 0] \log \big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)$
- So that: $\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} y_i \log a(\boldsymbol{x_i}, \boldsymbol{\theta}) + (1 - y_i) \log \big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big) = \mathbb{E}_{\mathrm{P}(x,y)} \big[\log\big(p_{\boldsymbol{\theta}}(y|\boldsymbol{x})\big)\big]$

Understanding the concept of probability estimation

- Assume that algorithm $a(\boldsymbol{x}, \boldsymbol{\theta})$ returns probability that object $\boldsymbol{x_i}$ is from positive class $y_i = 1$
- Probability that $\boldsymbol{x_i}$ belongs to class y_i:
  $p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x_i}) = a\big(\boldsymbol{x_i}, \boldsymbol{\theta}\big)^{[y_i=+1]}\big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)^{[y_i=0]}$
- Function to be optimised: $\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x}_i)$ ← maximize probability to observe such data
- Product is not useful, let's make sum:
  $\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} [y_i = 1] \log a(\boldsymbol{x_i}, \boldsymbol{\theta}) + [y_i = 0] \log \big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)$
- So that: $\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} y_i \log a(\boldsymbol{x_i}, \boldsymbol{\theta}) + (1 - y_i) \log \big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big) = \mathbb{E}_{\mathrm{P}(x,y)} \big[\log\big(p_{\boldsymbol{\theta}}(y|\boldsymbol{x})\big)\big]$

aka log-likelihood ↓

Understanding the concept of probability estimation

- Assume that algorithm $a(\boldsymbol{x}, \boldsymbol{\theta})$ returns probability that object $\boldsymbol{x_i}$ is from positive class $y_i = 1$
- Probability that $\boldsymbol{x_i}$ belongs to class y_i:
  $p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x_i}) = a\big(\boldsymbol{x_i}, \boldsymbol{\theta}\big)^{[y_i=+1]}\big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)^{[y_i=0]}$
- Function to be optimised: $\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x}_i)$ ← maximize probability to observe such data
- Product is not useful, let's make sum:
  $\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} [y_i = 1] \log a(\boldsymbol{x_i}, \boldsymbol{\theta}) + [y_i = 0] \log \big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)$

  aka log-likelihood ↓
- So that: $\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} y_i \log a(\boldsymbol{x_i}, \boldsymbol{\theta}) + (1 - y_i) \log \big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big) = \mathbb{E}_{\mathrm{P}(x,y)} \big[\log\big(p_{\boldsymbol{\theta}}(y|\boldsymbol{x})\big)\big]$
- Finally: $\hat{a} = \arg\max_{\boldsymbol{\theta} \in \Theta} \log \mathcal{L}(\boldsymbol{\theta})$

How to make classifier return probabilities?

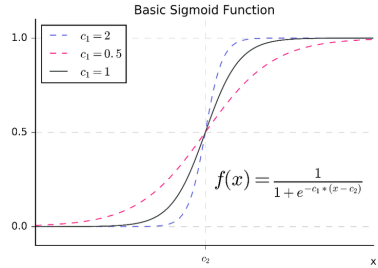
- $a(\boldsymbol{x}, \boldsymbol{\theta})$ has a range of $\mathbb{R}$, but it should be $[0, 1]$

How to make classifier return probabilities?

- $a(\boldsymbol{x}, \boldsymbol{\theta})$ has a range of $\mathbb{R}$, but it should be $[0, 1]$
- Let's think of a function that converts $\mathbb{R} \longmapsto [0, 1]$

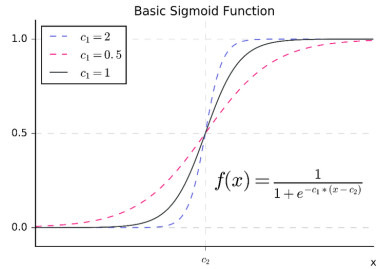
How to make classifier return probabilities?

- $a(\boldsymbol{x}, \boldsymbol{\theta})$ has a range of $\mathbb{R}$, but it should be $[0, 1]$
- Let's think of a function that converts $\mathbb{R} \longmapsto [0, 1]$
- **Sigmoid function** looks nice: $\sigma(z) = \dfrac{1}{1 + e^{-z}}$



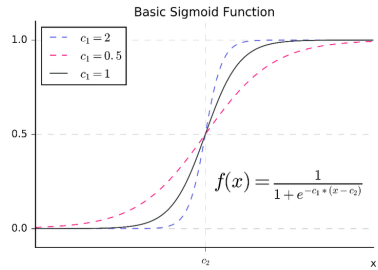
How to make classifier return probabilities?

- $a(\boldsymbol{x}, \boldsymbol{\theta})$ has a range of $\mathbb{R}$, but it should be $[0, 1]$
- Let's think of a function that converts $\mathbb{R} \longmapsto [0, 1]$
- **Sigmoid function** looks nice: $\sigma(z) = \dfrac{1}{1 + e^{-z}}$
- Wrap it around: $a_\sigma(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\big(a(\boldsymbol{x}, \boldsymbol{\theta})\big) = \dfrac{1}{1 + e^{-\langle \boldsymbol{x}, \boldsymbol{\theta} \rangle}}$



How to make classifier return probabilities?

- $a(\boldsymbol{x}, \boldsymbol{\theta})$ has a range of $\mathbb{R}$, but it should be $[0, 1]$
- Let's think of a function that converts $\mathbb{R} \longmapsto [0, 1]$
- **Sigmoid function** looks nice: $\sigma(z) = \dfrac{1}{1 + e^{-z}}$
- Wrap it around: $a_\sigma(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\big(a(\boldsymbol{x}, \boldsymbol{\theta})\big) = \dfrac{1}{1 + e^{-\langle \boldsymbol{x}, \boldsymbol{\theta} \rangle}}$
- Now it is possible to plug a_σ into log-likelihood and train:

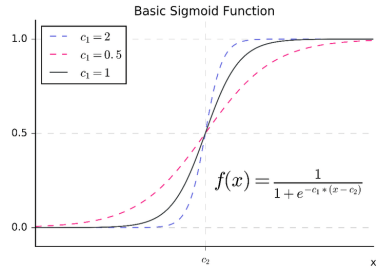


How to make classifier return probabilities?

- $a(\boldsymbol{x}, \boldsymbol{\theta})$ has a range of $\mathbb{R}$, but it should be $[0, 1]$
- Let's think of a function that converts $\mathbb{R} \longmapsto [0, 1]$
- **Sigmoid function** looks nice: $\sigma(z) = \dfrac{1}{1 + e^{-z}}$
- Wrap it around: $a_\sigma(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\big(a(\boldsymbol{x}, \boldsymbol{\theta})\big) = \dfrac{1}{1 + e^{-\langle \boldsymbol{x}, \boldsymbol{\theta} \rangle}}$
- Now it is possible to plug a_σ into log-likelihood and train:

$$\hat{a} = \arg\max_{\boldsymbol{\theta} \in \Theta} \log \mathcal{L}(\boldsymbol{\theta}) =$$

$$= \arg\max_{\boldsymbol{\theta} \in \Theta} \sum_{i=1}^{N} y_i \log \sigma(\langle \boldsymbol{x_i}, \boldsymbol{\theta} \rangle) + (1 - y_i) \log \big(1 - \sigma(\langle \boldsymbol{x_i}, \boldsymbol{\theta} \rangle)\big)$$



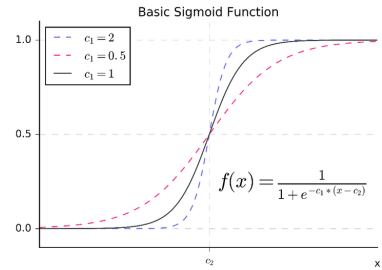
How to make classifier return probabilities?

- $a(\boldsymbol{x}, \boldsymbol{\theta})$ has a range of $\mathbb{R}$, but it should be $[0, 1]$
- Let's think of a function that converts $\mathbb{R} \longmapsto [0, 1]$
- **Sigmoid function** looks nice: $\sigma(z) = \dfrac{1}{1 + e^{-z}}$
- Wrap it around: $a_\sigma(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\big(a(\boldsymbol{x}, \boldsymbol{\theta})\big) = \dfrac{1}{1 + e^{-\langle \boldsymbol{x}, \boldsymbol{\theta} \rangle}}$
- Now it is possible to plug a_σ into log-likelihood and train:

$$\hat{a} = \arg\max_{\boldsymbol{\theta} \in \Theta} \log \mathcal{L}(\boldsymbol{\theta}) =$$

$$= \arg\max_{\boldsymbol{\theta} \in \Theta} \sum_{i=1}^{N} y_i \log \sigma(\langle \boldsymbol{x_i}, \boldsymbol{\theta} \rangle) + (1 - y_i) \log \big(1 - \sigma(\langle \boldsymbol{x_i}, \boldsymbol{\theta} \rangle)\big)$$

- This approach is called **logistic regression**



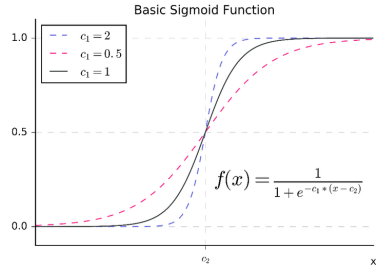
How to make classifier return probabilities?

- $a(\boldsymbol{x}, \boldsymbol{\theta})$ has a range of $\mathbb{R}$, but it should be $[0, 1]$
- Let's think of a function that converts $\mathbb{R} \longmapsto [0, 1]$
- **Sigmoid function** looks nice: $\sigma(z) = \dfrac{1}{1 + e^{-z}}$
- Wrap it around: $a_\sigma(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\big(a(\boldsymbol{x}, \boldsymbol{\theta})\big) = \dfrac{1}{1 + e^{-\langle \boldsymbol{x}, \boldsymbol{\theta} \rangle}}$
- Now it is possible to plug a_σ into log-likelihood and train:

$$\hat{a} = \arg\max_{\boldsymbol{\theta} \in \Theta} \log \mathcal{L}(\boldsymbol{\theta}) =$$

$$= \arg\max_{\boldsymbol{\theta} \in \Theta} \sum_{i=1}^{N} y_i \log \sigma(\langle \boldsymbol{x_i}, \boldsymbol{\theta} \rangle) + (1 - y_i) \log \big(1 - \sigma(\langle \boldsymbol{x_i}, \boldsymbol{\theta} \rangle)\big)$$

- This approach is called **logistic regression**



NB: usage of σ and log-likelihood in general does not guarantee probabilistic interpretation

Comparison

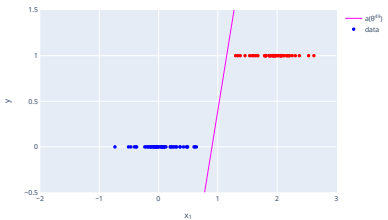
Linear classifier

$$Q(\boldsymbol{\theta}) = \sum_{i=1}^{N} \big[\text{sign}\big(a(\boldsymbol{x}_i, \boldsymbol{\theta})\big) \neq y_i\big]$$

- $a(\boldsymbol{x}, \boldsymbol{\theta}) = \langle \boldsymbol{x}, \boldsymbol{\theta} \rangle$
- $a(\boldsymbol{x}, \boldsymbol{\theta}) : \mathbb{R}^K \times \mathbb{R}^K \mapsto \mathbb{R}$

✓ Easy to interpret

× Gives only yes-no answer



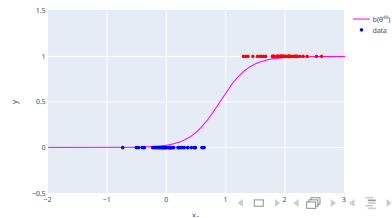
Logistic regression

$$\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} y_i \log a_\sigma(\boldsymbol{x}_i, \boldsymbol{\theta}) + (1 - y_i) \log \big(1 - a_\sigma(\boldsymbol{x}_i, \boldsymbol{\theta})\big)$$

- $a_\sigma(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\big[a(\boldsymbol{x}, \boldsymbol{\theta})\big] = \sigma\big[\langle \boldsymbol{x}, \boldsymbol{\theta} \rangle\big]$
- $a_\sigma(\boldsymbol{x}, \boldsymbol{\theta}) : \mathbb{R}^K \times \mathbb{R}^K \mapsto [0, 1]$

✓ More profound interpretation

✓ Returns probability of positive class $y = 1$

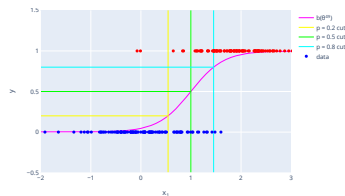


Output of binary classifier

Let red correspond to a positive class (1)
and blue – to a negative (0)

- 1 classified as 1 $\Rightarrow$ True positive (TP)
- 0 classified as 0 $\Rightarrow$ True negative (TN)
- 1 classified as 0 $\Rightarrow$ False negative (FN)
- 0 classified as 1 $\Rightarrow$ False positive (FP)

NB: in case of logistic regression you need to specify a cut on the model's output to get deterministic prediction



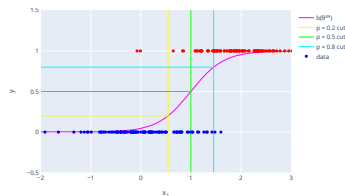
Output of binary classifier

Let red correspond to a positive class (1)
and blue – to a negative (0)

- 1 classified as 1 $\Rightarrow$ True positive (TP)
- 0 classified as 0 $\Rightarrow$ True negative (TN)
- 1 classified as 0 $\Rightarrow$ False negative (FN)
- 0 classified as 1 $\Rightarrow$ False positive (FP)

NB: in case of logistic regression you need to specify a cut on the model's output to get deterministic prediction

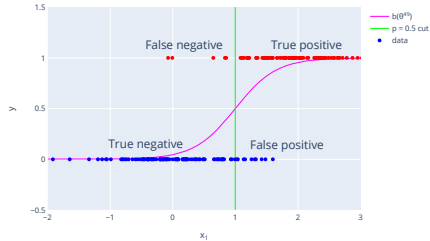
➔ Then how do we measure model's performance?



Metrics

Using these output for binary classifier we can construct several quality metrics:

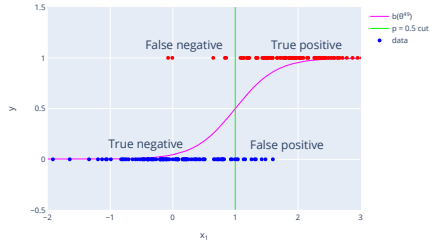
- **accuracy** = $\dfrac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$
  - ➔ fraction of correct answers
- **precision** = $\dfrac{\text{TP}}{\text{TP} + \text{FP}}$
  - ➔ fraction of correct positive answers in all positive answers
- **recall** = $\dfrac{\text{TP}}{\text{TP} + \text{FN}}$
  - ➔ fraction of correctly classified positive objects
- **F1-score** = $\dfrac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$
  - ➔ harmonic mean of precision and recall



Metrics

Using these output for binary classifier we can construct several quality metrics:

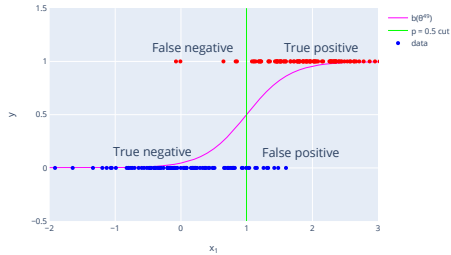
- **accuracy** = $\dfrac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$
  ➔ fraction of correct answers
- **precision** = $\dfrac{\text{TP}}{\text{TP} + \text{FP}}$
  ➔ fraction of correct positive answers in all positive answers
- **recall** = $\dfrac{\text{TP}}{\text{TP} + \text{FN}}$
  ➔ fraction of correctly classified positive objects
- **F1-score** = $\dfrac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$
  ➔ harmonic mean of precision and recall



choose metrics wisely and beware of class imbalanced data!

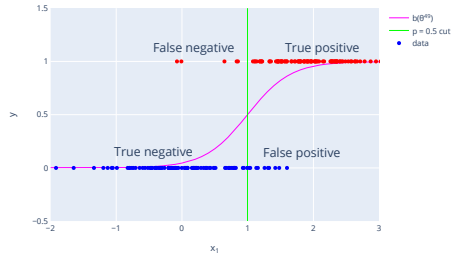
Choosing the right threshold

- As was mentioned before, classifier decision (thus performance) **depends on the threshold**
- How can we estimate its performance *for all cuts* in general?
  - define **true positive rate**: $\text{TPR} = \dfrac{\text{TP}}{\text{TP} + \text{FN}}$
  - and **false positive rate**: $\text{FPR} = \dfrac{\text{FP}}{\text{FP} + \text{TN}}$
  - move the green line on the plot
  - moving to the left results in TP↑ and FN↓ (sensitive, but not specific)
  - moving to the right results in TP↓ and FN↑ (specific, but not sensitive)



Choosing the right threshold

- As was mentioned before, classifier decision (thus performance) **depends on the threshold**
- How can we estimate its performance *for all cuts* in general?
  - define **true positive rate**: $\text{TPR} = \dfrac{\text{TP}}{\text{TP} + \text{FN}}$
  - and **false positive rate**: $\text{FPR} = \dfrac{\text{FP}}{\text{FP} + \text{TN}}$
  - move the green line on the plot
  - moving to the left results in TP↑ and FN↓ (sensitive, but not specific)
  - moving to the right results in TP↓ and FN↑ (specific, but not sensitive)



Idea: plot jointly values of TPR and FPR for different values of threshold

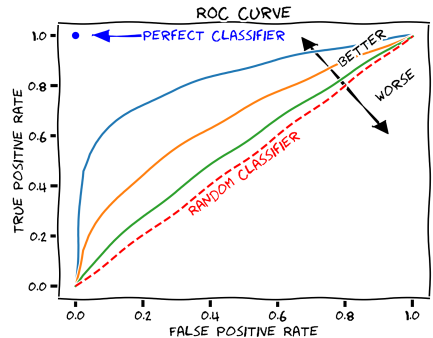
Receiver Operating Characteristic — curve

- Scanning through thresholds we can calculate for each of them TPR and FPR and plot them
- This will give a **ROC curve**

➔ Random classifier – diagonal line at ROC

➔ Perfect classifier – step function

➔ Something-went-wrong classifier – curve below diagonal line



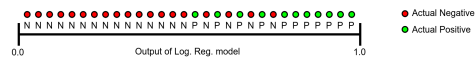
Receiver Operating Characteristic

area under curve

The power of a classifier can be represented by the **area under ROC curve**:

$$\text{ROC AUC} = \int_0^1 \text{TPR}\big(\text{FPR}\big) d\big(\text{FPR}\big)$$

- aggregated measure of performance across all possible thresholds $\Rightarrow$ **threshold-invariant**
- 1 $\Rightarrow$ predictions are 100% correct
- 0.5 $\Rightarrow$ random classifier
- 0 $\Rightarrow$ predictions are 100% wrong
- probability that a random positive (green) example is positioned to the right of a random negative (red) example $\Rightarrow$ **scale-invariant**



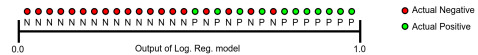
Receiver Operating Characteristic

area under curve

The power of a classifier can be represented by the **area under ROC curve**:

$$\text{ROC AUC} = \int_0^1 \text{TPR}\big(\text{FPR}\big) d\big(\text{FPR}\big)$$

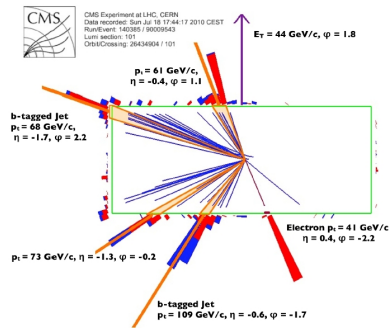
- aggregated measure of performance across all possible thresholds $\Rightarrow$ **threshold-invariant**
- 1 $\Rightarrow$ predictions are 100% correct
- 0.5 $\Rightarrow$ random classifier
- 0 $\Rightarrow$ predictions are 100% wrong
- probability that a random positive (green) example is positioned to the right of a random negative (red) example $\Rightarrow$ **scale-invariant**



Nice moment to show this animation

SL examples

- **Area:** HEP
- **Objects:** events in the detector
- **Features:**
  - Binary: jet from b-quark, signal presence in subsystems
  - Categorical: number of jets, number of tracks, lepton type
  - Numerical: pT, η, φ, m, E, $\Delta R(\text{jet}, \gamma)$ of particles
- **Problems:**
  - Classification: event is signal or background? particle type?
  - Regression: jet energy, invariant mass of lepton pair
  - Ranking: list of candidates for primary vertices



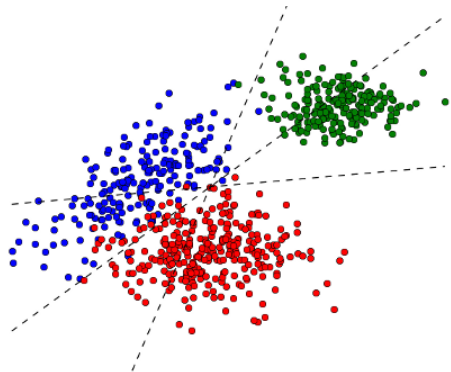
Intermediate Summary

- All ML problems start from:
  - Data preprocessing
  - Feature engineering
- Supervised Learning seek for a **predictive model**: for a given object based on its **features** returns the **answer**
  - classification
  - regression
  - ranking
- Model is constructed through the **training process** involving **loss function optimization**
- To evaluate **model performance** and check for **overfitting** we compute **metrics**
- For that we apply model to **unseen data** using **train/test and cross-validation** strategies
- Once we are sure that the model is reasonable we can **deploy it in production**

Supervised Learning: Additional Material

Multiclass classification

- If the number of classes > 2, then we need to modify classification approach
- $|Y| < \infty = \{1, \ldots, L\} \Rightarrow$ impossible to use single model



From multiclass problem to a set of binary problems

"One vs all"

- Train L binary classifiers: $b_\ell(\boldsymbol{x}, \boldsymbol{\theta}), \ell \in \{1, \ldots, L\}$
- Each $b_\ell(\boldsymbol{x}, \boldsymbol{\theta})$ is trained to separate between ℓ–th class and all the rest
- The most confident classifier gives the final separation:

$$a(\boldsymbol{x}, \boldsymbol{\theta}) = \arg\max_{\ell \in \{1,\ldots,L\}} b_\ell(\boldsymbol{x}, \boldsymbol{\theta})$$

"All vs all"

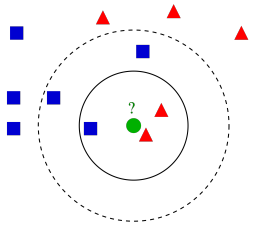
- Train C_L^2 binary classifiers to separate every two classes: $b_{i,j}(\boldsymbol{x}, \boldsymbol{\theta}), i, j \in 1, \ldots, L, i \neq j$
- In this case each classifier decides only between two classes
- $\hat{y} = \arg\max_{\ell \in \{1,\ldots,L\}} \sum_{i=1}^{L} \sum_{j \neq i} [b_{i,j}(\boldsymbol{x}, \boldsymbol{\theta}) = \ell]$
- Class with the highest vote over C_L^2 classifiers wins!

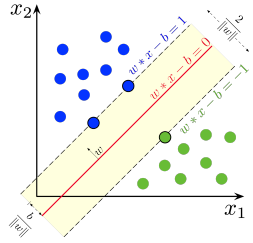
Classical algorithms

We didn't have time to describe them, but these guys are cool enough to be studied and tried out:

- **kNN** – k-nearest neighbors algorithm
- **SVM** – Support Vector Machine
- **Naive Bayes** – Naive Bayes classifier

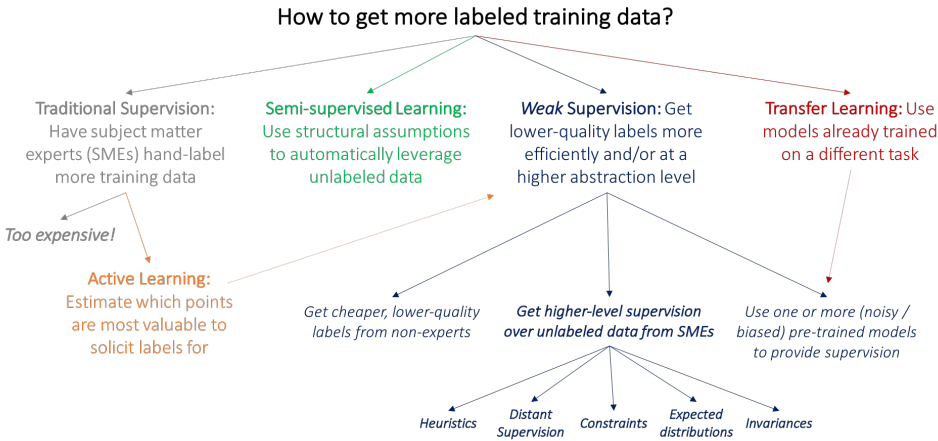
$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y)$$





Semi-supervised Learning

What if we don't have labels?

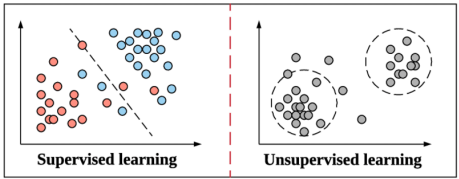


Unsupervised Learning

What if there is **no labels at all**?

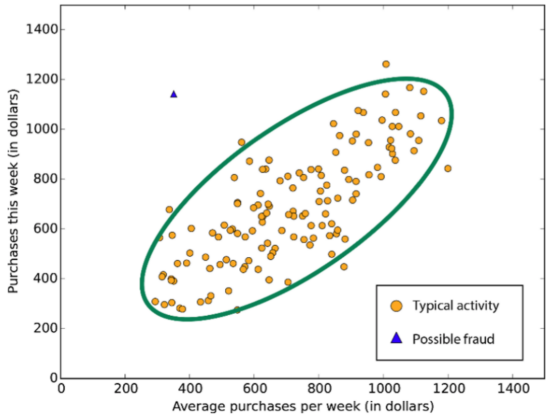
We just have *some* data and we want to:

- Study its **structure**
  - identify novelties/outliers
  - cluster objects
  - estimate densities
  - reduce dimensionality
- Generate it



→ no prior knowledge ⇒ learning is **unsupervised**

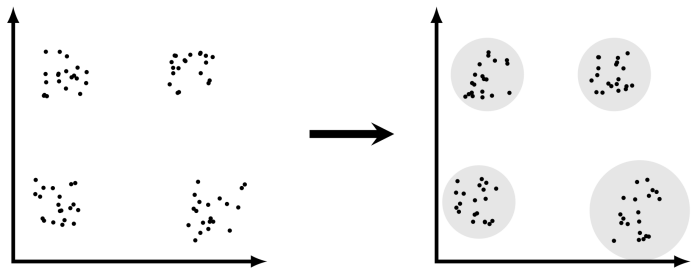
Unsupervised Learning —○ identify anomalies



link

Unsupervised Learning

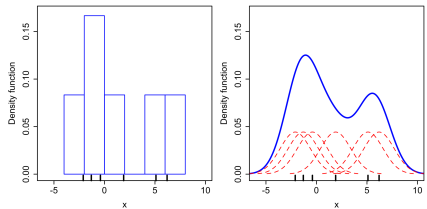
cluster objects



link

Unsupervised Learning — estimate densities

- Sometimes we have some data and don't know the underlying analytical **probability density function** (pdf)
- Which would be nice to know for:
  - visualizing data
  - understanding its shape
  - performing statistical inference
  - doing analytical calculations
  - sampling more data from it
- In HEP **histograms** have been used since the beginning of times
- But they induce information loss withing the bin and aren't nicely differentiable

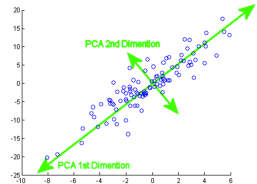


amazing visuals

→ Kernel Density Estimation (KDE)

Unsupervised Learning

reduce dimensionality



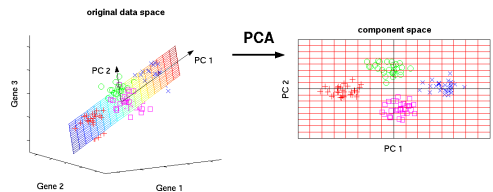
- might need to reduce number of features
- simply removing them might lead to information loss
- is there a clever way?

→ Principal Component Analysis (PCA)

→ Matrix Factorization

→ t-SNE

→ Autoencoder



awesome playground

Unsupervised Learning

generate data

1810.08771

playground

Unsupervised Learning

generate data in HEP

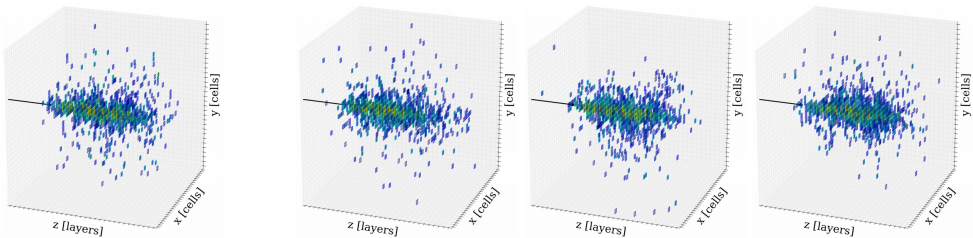


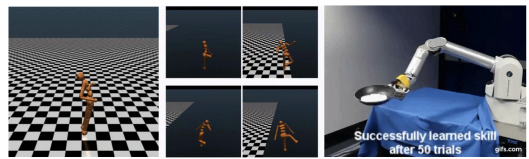
Fig. 5 Examples of individual 50 GeV photon showers generated by GEANT4 (left), the GAN (center left), WGAN (center right), and BIB-AE (right) architectures. Colors encode the deposited energy per cell.

Reinforcement Learning

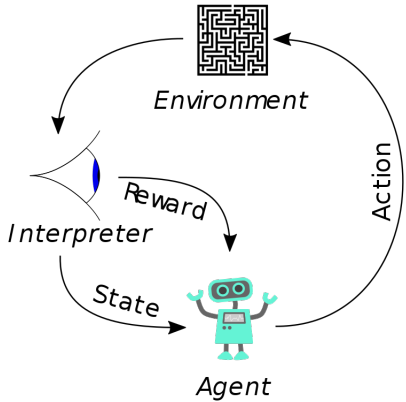
What if there's need for action?

- Sometimes we might want to model **behaviour**
  Ex: consumer modeling
- That is, to teach a machine to **act in a given world**
  Ex: robotics, trading, game playing

→ Reinforcement Learning makes this AI dream come true



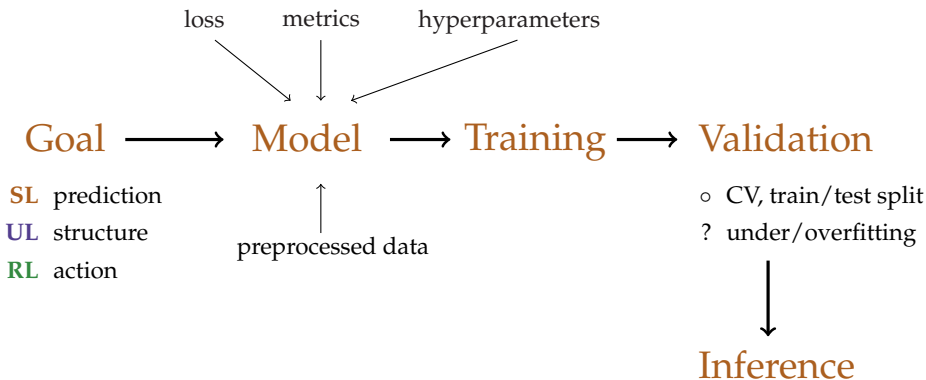
Reinforcement Learning — in a nutshell



more in cs234 course

Summary

- General pipeline



Backup slides

Solution can be found analytically

After defining feature matrix as:

$$F = \begin{bmatrix} f_1(\boldsymbol{x_1}) & \cdots & f_K(\boldsymbol{x_1}) \\ \vdots & \ddots & \vdots \\ f_1(\boldsymbol{x_N}) & \cdots & f_K(\boldsymbol{x_N}) \end{bmatrix}$$

Minimization problem for linear regression:

- $Q(\boldsymbol{\theta}) = \frac{1}{N}(F\boldsymbol{\theta} - \boldsymbol{y})^T(F\boldsymbol{\theta} - \boldsymbol{y}) = \\ = \boldsymbol{\theta}^T F^T F\boldsymbol{\theta} - 2\boldsymbol{y}^T F\boldsymbol{\theta} + \boldsymbol{y}^T\boldsymbol{y}$
- $\frac{\partial Q}{\partial \boldsymbol{\theta}^T} = 2F^T F\boldsymbol{\theta} - 2F^T\boldsymbol{y} = 0$

$$\hat{\boldsymbol{\theta}} = (F^T F)^{-1} F^T \boldsymbol{y}$$

Problem 1: calculation of $(F^T F)^{-1}$ has **complexity of $O(K^5 N)$** $\Rightarrow$ computationally-intensive for large number of features

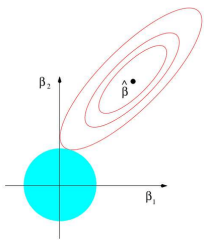
Problem 2: F is often **ill-conditioned** $\Rightarrow$ computational errors causes instabilities and extremely large solutions

Regularization

playgrounds click, click

L2 regularization (Tikhonov)

$$Q(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left(\langle \boldsymbol{x}, \boldsymbol{\theta}\rangle - y_i\right)^2 + \lambda \sum_{j=1}^{K} \boldsymbol{\theta}_j^2$$

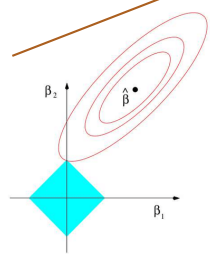


penalize model for too large weights

L1 regularization (LASSO)

least absolute shrinkage and selection operator

$$Q(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left(\langle \boldsymbol{x}, \boldsymbol{\theta}\rangle - y_i\right)^2 + \lambda \sum_{j=1}^{K} |\boldsymbol{\theta}_j|$$



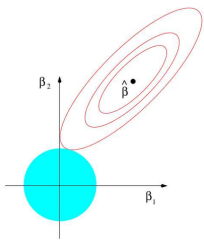
Regularization

playgrounds click, click

L2 regularization (Tikhonov)

$$Q(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left(\langle \boldsymbol{x}, \boldsymbol{\theta}\rangle - y_i\right)^2 + \lambda \sum_{j=1}^{K} \boldsymbol{\theta}_j^2$$

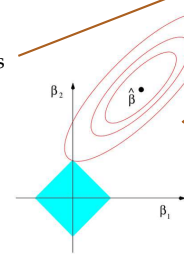
penalize model for too large weights



L1 regularization (LASSO)

least absolute shrinkage and selection operator

$$Q(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left(\langle \boldsymbol{x}, \boldsymbol{\theta}\rangle - y_i\right)^2 + \lambda \sum_{j=1}^{K} |\boldsymbol{\theta}_j|$$



autozeroing uninformative weights!

↓

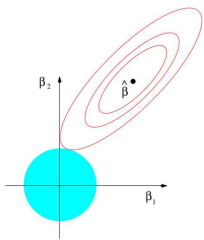
feature selection

Regularization

playgrounds click, click

L2 regularization (Tikhonov)

$$Q(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left(\langle \boldsymbol{x}, \boldsymbol{\theta}\rangle - y_i\right)^2 + \boldsymbol{\lambda}\sum_{j=1}^{K}\boldsymbol{\theta}_j^2$$

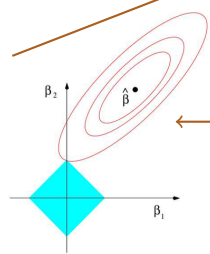


penalize model for too large weights

L1 regularization (LASSO)

least absolute shrinkage and selection operator

$$Q(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left(\langle \boldsymbol{x}, \boldsymbol{\theta}\rangle - y_i\right)^2 + \boldsymbol{\lambda}\sum_{j=1}^{K}|\boldsymbol{\theta}_j|$$



autozeroing uninformative weights!

↓

feature selection

$\boldsymbol{\lambda}$ is a **hyperparameter** → more in homework