

Introduction to Machine Learning

Lecture I

Oleg Filatov ¹ Stepan Zakharov ²

¹DESY ²Novosibirsk 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

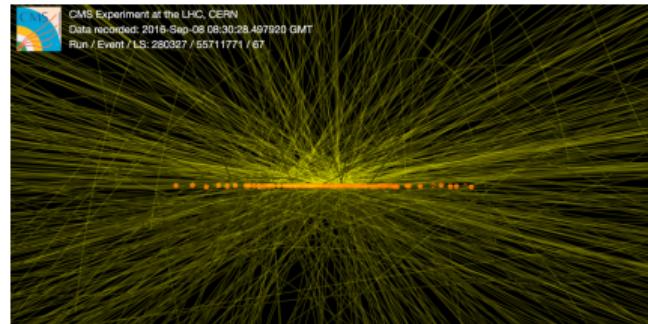
In the beginning was the Data:

- 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

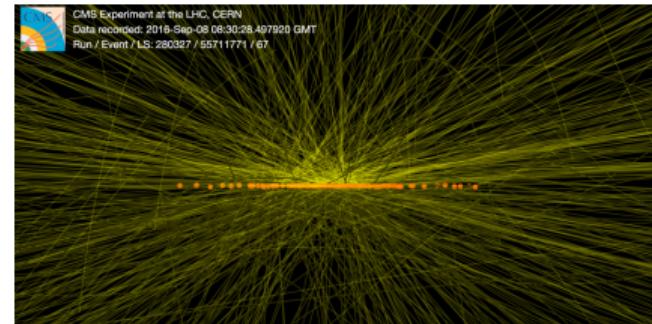
In the beginning was the Data:

- 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	Italy	Rome

Uniqueness

Formats

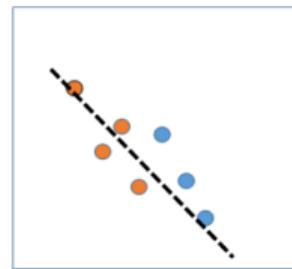
Attribute dependencies

Missing values

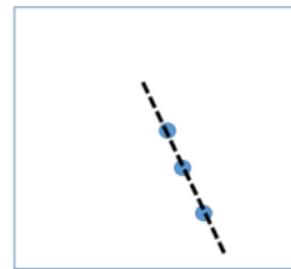
Invalid values

Misfielded values

Misspellings



Incorrect model due to corrupted data

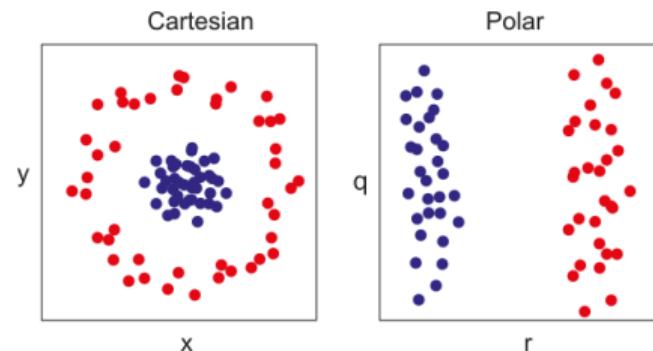


Correct model using cleaned data

Feature engineering

- **Construction:** vectorization techniques + your domain knowledge
- **Transformation:** functions, scaling, binning, cropping, 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) $x \in X$, 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** $g: X \rightarrow Y$ 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 $g: X \rightarrow Y$ relation.
Effectively, the **model is supervised by our prior knowledge**.



Problem statement

Given:

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

Find:

- $a: X \rightarrow Y$ – 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 $a(x, \theta)$ be a function that depends on object x and vector of parameters θ .
- We know that $g: \begin{bmatrix} g(x_1) \\ \vdots \\ g(x_N) \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = y$, the exact expression for g is **unknown**
- Let's try to approximate g with model a such as: $\begin{bmatrix} a(x_1, \theta) \\ \vdots \\ a(x_N, \theta) \end{bmatrix} = \begin{bmatrix} y'_1 \\ \vdots \\ y'_N \end{bmatrix} = y'$
- If approximation is reliable, then y' is close to y
- **Loss function** $\mathcal{L}(a(x_i), y_i)$ gives the state of closeness between prediction $a(x_i)$ 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: $Q(a, X, Y) = \sum_{i=1}^N \mathcal{L}(a(x_i), y_i)$
- $Q(a, X, Y)$ is usually called "loss function" too because there is no need to use directly $\mathcal{L}(a(x), y)$ for single object
- Optimal algorithm \hat{a} minimizes $Q(a, X, Y)$ on the training set $[X, Y]$

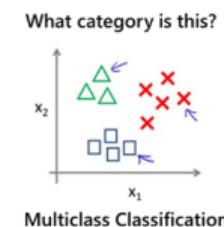
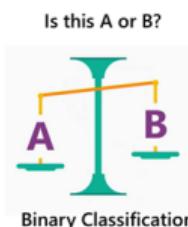
$$\hat{a} = \arg \min_{a \in A} Q(a, X, Y)$$

This procedure is called **training**

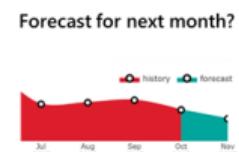
Supervised Learning types

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

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



What category is this?



Regression

Linear models

Classification

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

Regression

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

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:
 $x = [1, f_1(x)]$
- Define parameters vector $\theta = [\theta_0, \theta_1]$
- Well there is analytical solution for this problem (look at the backup)
- How can we find minimum of $Q(\theta)$ 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 θ^0

- $\hat{\theta} \leftarrow \theta^0$

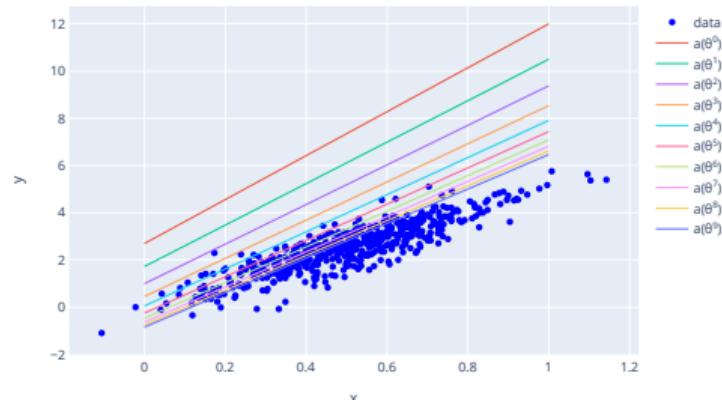
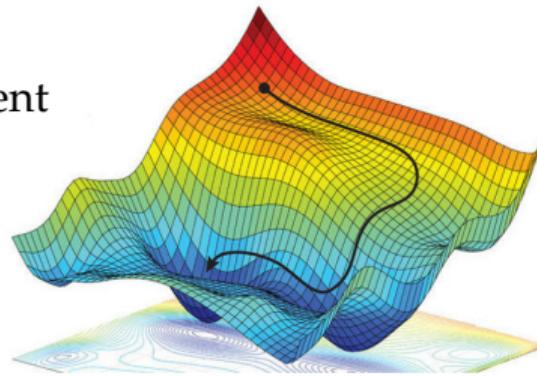
- Calculate direction to the minimum:

$$-\nabla Q(\hat{\theta}) = -\left[\frac{\partial Q}{\partial \theta_0}, \frac{\partial Q}{\partial \theta_1}\right]_{\hat{\theta}}$$

- Move towards the minimum:

$$\hat{\theta} \leftarrow \hat{\theta} - \eta \nabla Q(\hat{\theta})$$

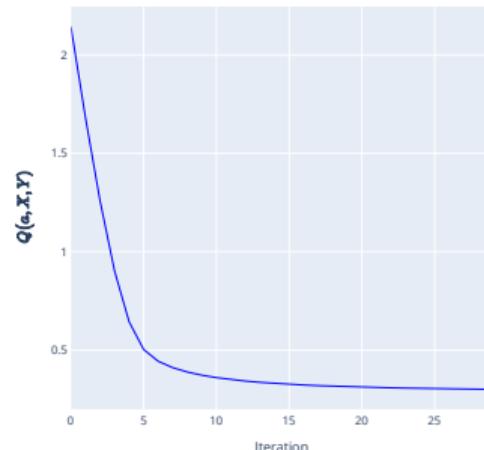
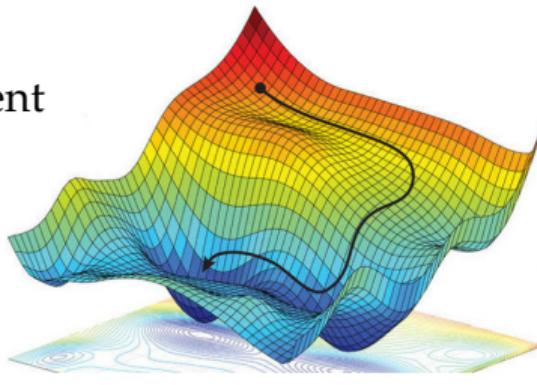
- Repeat until convergence



Better approach? ——○ gradient descent

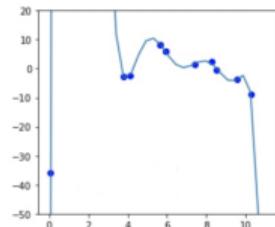
Idea: bring up the information about the direction to the minimum of Q using its **antigradient**

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



Measuring model performance

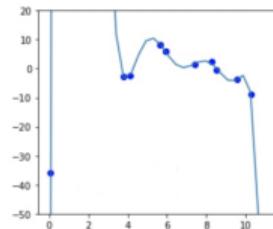
- After training we want to understand the quality of the model
- Therefore we need something like:



⇒ Number

Measuring model performance

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

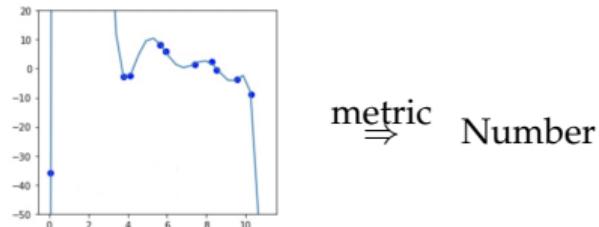


⇒ Number

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

Measuring model performance

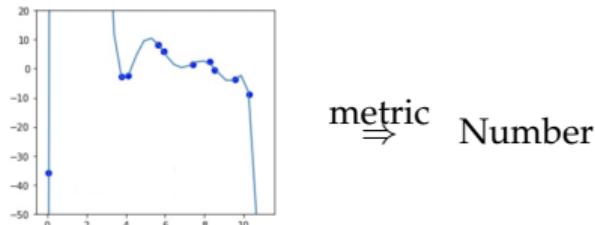
- After training we want to understand the quality of the model
- Therefore we need something like:



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

Measuring model performance

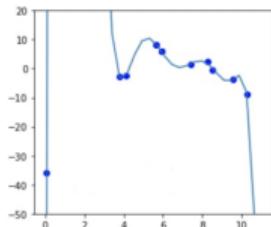
- After training we want to understand the quality of the model
- Therefore we need something like:



- Since $\mathcal{L}(a(x), y)$ is an error estimation
- Then it can be used as **metric**
- Which data to use for measurements?

Measuring model performance

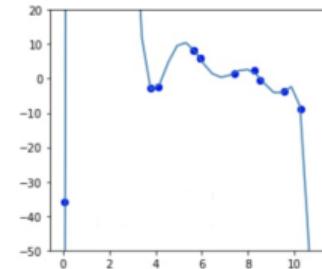
- After training we want to understand the quality of the model
- Therefore we need something like:



metric \Rightarrow Number

- Since $\mathcal{L}(a(x), y)$ is an error estimation
- Then it can be used as **metric**
- Which data to use for measurements?

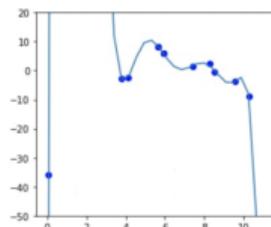
- Well, we can use the training dataset:



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

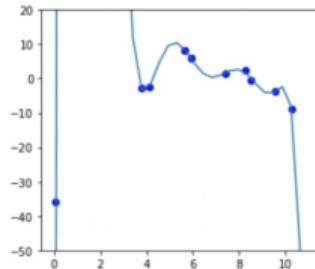
Measuring model performance

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

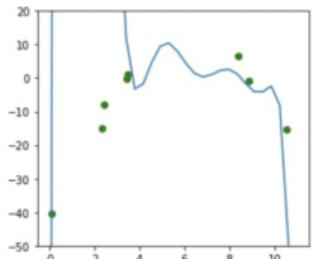


- Since $\mathcal{L}(a(x), y)$ is an error estimation
- Then it can be used as **metric**
- Which data to use for measurements?

- Well, we can use the training dataset:

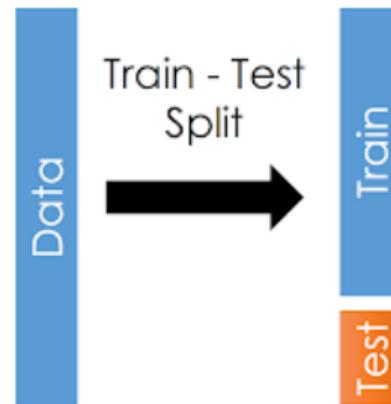


- Then it might happen that $\mathcal{L}_{\text{train}} \rightarrow 0$
- But on unseen data $\mathcal{L}_{\text{test}}$ will be obscenely large:



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 dependence?

Where to get new data?

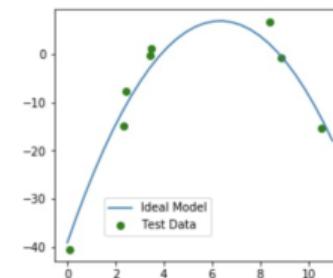
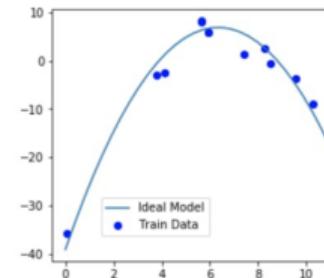
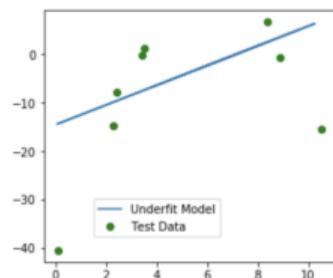
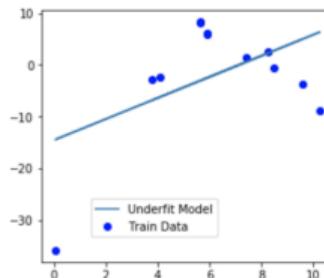
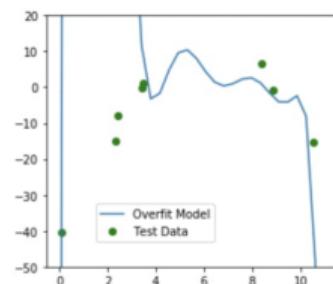
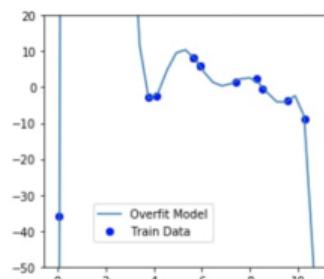
—○ cross-validation

- 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 \Rightarrow **overtraining**
- Center model performs bad both on train and test sets \Rightarrow **undertraining**
- Lower model shows acceptable results on train and test sets

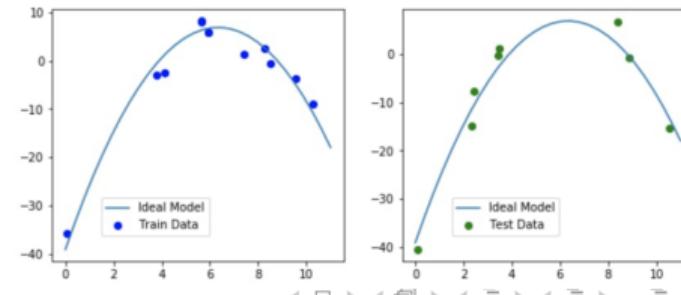
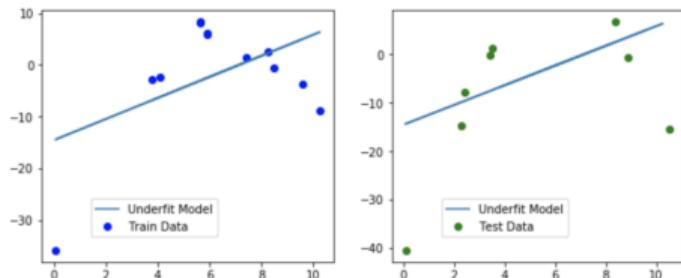
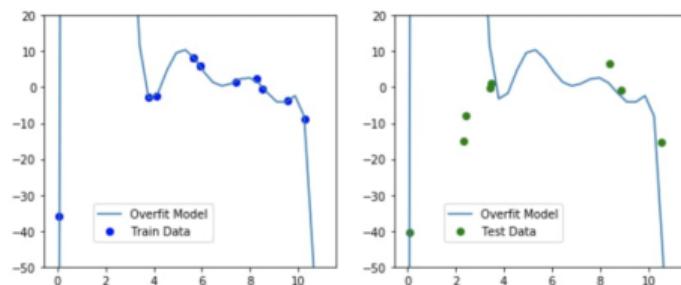


Example —○ Polynomial regression

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



Let's analyze these outcomes



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

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

$$\mathcal{L}_{\text{train}} = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(a(\mathbf{x}_i), y_i) \equiv \mathbb{E}_{P(x,y)}[\mathcal{L}(a(\mathbf{x}), y)] \leftarrow \mathbf{\text{expected value}}$$

Before we end with regression

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

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

- **MSE** = $\mathbb{E}_{p(x,y)}[(a(\mathbf{x}) - y)^2]$
- **RMSE** = $\sqrt{\text{MSE}}$
- **MSLE** = $\mathbb{E}_{p(x,y)}[(\log(a(\mathbf{x}) + 1) - \log(y + 1))^2]$
- **Coefficient of determination**
$$R^2 = 1 - \frac{\mathbb{E}_{p(x,y)}[(a(\mathbf{x}) - y)^2]}{\mathbb{E}_{p(x,y)}[(\bar{y} - y)^2]}$$
- **MAE** = $\mathbb{E}_{p(x,y)}[|a(\mathbf{x}) - y|]$
- **MAPE** = $100\% \cdot \mathbb{E}_{p(x,y)} \left[\left| \frac{y - a(\mathbf{x})}{y} \right| \right]$
- **SMAPE** = $100\% \cdot \mathbb{E}_{p(x,y)} \left[\frac{|y - a(\mathbf{x})|}{|y| + |a(\mathbf{x})|} \right]$

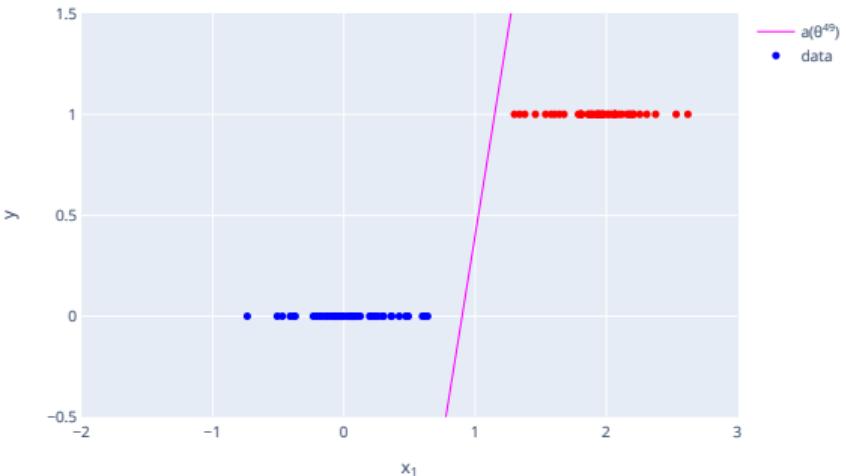
$$\text{Huber Loss} = \mathbb{E}_{p(x,y)} \left\{ \begin{array}{ll} \frac{1}{2}(a(\mathbf{x}) - y)^2, & |a(\mathbf{x}) - y| < \delta \\ \delta|a(\mathbf{x}) - y| - \frac{1}{2}\delta^2, & \text{otherwise} \end{array} \right\}$$

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:
 $x = [1, x_1]$
- Algorithm: $a(x, \theta) = \text{sign}\langle x, \theta \rangle$

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

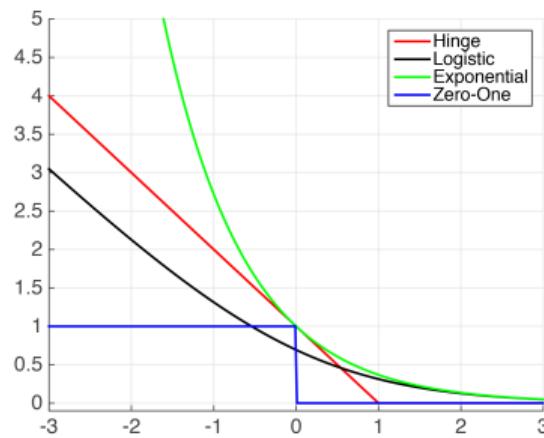


- $\mathcal{L}(\theta) = [\text{sign}\langle x, \theta \rangle \neq y] \leftarrow$ step function
- $Q(\theta) = \mathbb{E}_{p(x,y)}[\text{sign}\langle x, \theta \rangle \neq y] \leftarrow$ accuracy
- Then we need to minimize $Q(a(\theta))$ by varying θ

From binary classification to logistic regression

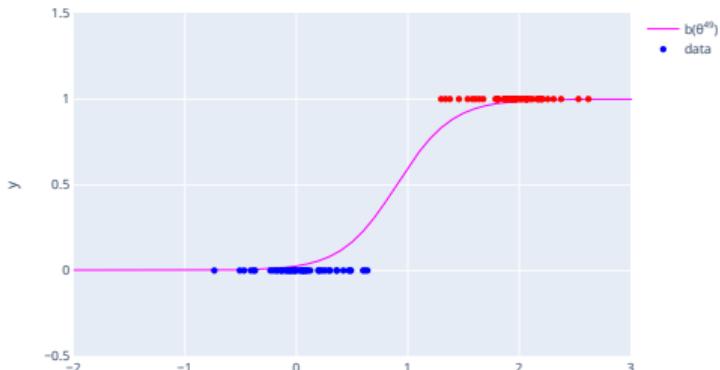
Problem 1: $\mathcal{L}(\theta)$ isn't a smooth function

Solution: use smooth approximations



Problem 2: discrete $\{0, 1\}$ output from the classifier

Solution: reframe the problem to probabilistic perspective
⇒ logistic regression



Understanding the concept of probability estimation

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

Understanding the concept of probability estimation

- Assume that algorithm $a(\mathbf{x}, \boldsymbol{\theta})$ returns probability that object \mathbf{x}_i is from positive class $y_i = 1$
- Probability that \mathbf{x}_i belongs to class y_i :

$$p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i) = a(\mathbf{x}_i, \boldsymbol{\theta})^{[y_i=+1]} (1 - a(\mathbf{x}_i, \boldsymbol{\theta}))^{[y_i=0]}$$

Understanding the concept of probability estimation

- Assume that algorithm $a(\mathbf{x}, \boldsymbol{\theta})$ returns probability that object \mathbf{x}_i is from positive class $y_i = 1$
- Probability that \mathbf{x}_i belongs to class y_i :

$$p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i) = a(\mathbf{x}_i, \boldsymbol{\theta})^{[y_i=+1]} (1 - a(\mathbf{x}_i, \boldsymbol{\theta}))^{[y_i=0]}$$

- Function to be optimised: $\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^N p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i)$ ← maximize probability to observe such data

Understanding the concept of probability estimation

- Assume that algorithm $a(\mathbf{x}, \boldsymbol{\theta})$ returns probability that object \mathbf{x}_i is from positive class $y_i = 1$
- Probability that \mathbf{x}_i belongs to class y_i :

$$p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i) = a(\mathbf{x}_i, \boldsymbol{\theta})^{[y_i=+1]} (1 - a(\mathbf{x}_i, \boldsymbol{\theta}))^{[y_i=0]}$$

- Function to be optimised: $\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^N p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i)$ ← maximize probability to observe such data
- Product is not useful, let's make sum:

$$\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^N [y_i = 1] \log a(\mathbf{x}_i, \boldsymbol{\theta}) + [y_i = 0] \log (1 - a(\mathbf{x}_i, \boldsymbol{\theta}))$$

Understanding the concept of probability estimation

- Assume that algorithm $a(\mathbf{x}, \boldsymbol{\theta})$ returns probability that object \mathbf{x}_i is from positive class $y_i = 1$
- Probability that \mathbf{x}_i belongs to class y_i :

$$p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i) = a(\mathbf{x}_i, \boldsymbol{\theta})^{[y_i=+1]} (1 - a(\mathbf{x}_i, \boldsymbol{\theta}))^{[y_i=0]}$$

- Function to be optimised: $\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^N p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i)$ ← maximize probability to observe such data
- Product is not useful, let's make sum:

$$\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^N [y_i = 1] \log a(\mathbf{x}_i, \boldsymbol{\theta}) + [y_i = 0] \log (1 - a(\mathbf{x}_i, \boldsymbol{\theta}))$$

- So that: $\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^N y_i \log a(\mathbf{x}_i, \boldsymbol{\theta}) + (1 - y_i) \log (1 - a(\mathbf{x}_i, \boldsymbol{\theta})) = \mathbb{E}_{P(\mathbf{x}, y)} [\log(p_{\boldsymbol{\theta}}(y|\mathbf{x}))]$

Understanding the concept of probability estimation

- Assume that algorithm $a(\mathbf{x}, \boldsymbol{\theta})$ returns probability that object \mathbf{x}_i is from positive class $y_i = 1$
- Probability that \mathbf{x}_i belongs to class y_i :

$$p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i) = a(\mathbf{x}_i, \boldsymbol{\theta})^{[y_i=+1]} (1 - a(\mathbf{x}_i, \boldsymbol{\theta}))^{[y_i=0]}$$

- Function to be optimised: $\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^N p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i)$ ← maximize probability to observe such data
- Product is not useful, let's make sum:

$$\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^N [y_i = 1] \log a(\mathbf{x}_i, \boldsymbol{\theta}) + [y_i = 0] \log (1 - a(\mathbf{x}_i, \boldsymbol{\theta}))$$

aka log-likelihood
↓

- So that: $\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^N y_i \log a(\mathbf{x}_i, \boldsymbol{\theta}) + (1 - y_i) \log (1 - a(\mathbf{x}_i, \boldsymbol{\theta})) = \mathbb{E}_{P(x,y)} [\log(p_{\boldsymbol{\theta}}(y|\mathbf{x}))]$

Understanding the concept of probability estimation

- Assume that algorithm $a(\mathbf{x}, \boldsymbol{\theta})$ returns probability that object \mathbf{x}_i is from positive class $y_i = 1$
- Probability that \mathbf{x}_i belongs to class y_i :

$$p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i) = a(\mathbf{x}_i, \boldsymbol{\theta})^{[y_i=+1]} (1 - a(\mathbf{x}_i, \boldsymbol{\theta}))^{[y_i=0]}$$

- Function to be optimised: $\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^N p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i)$ ← maximize probability to observe such data
- Product is not useful, let's make sum:

$$\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^N [y_i = 1] \log a(\mathbf{x}_i, \boldsymbol{\theta}) + [y_i = 0] \log (1 - a(\mathbf{x}_i, \boldsymbol{\theta}))$$

aka log-likelihood
↓

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

How to make classifier return probabilities?

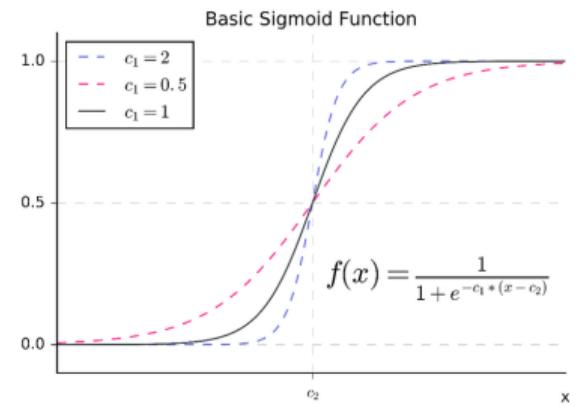
- $a(x, \theta)$ has a range of \mathbb{R} , but it should be $[0, 1]$

How to make classifier return probabilities?

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

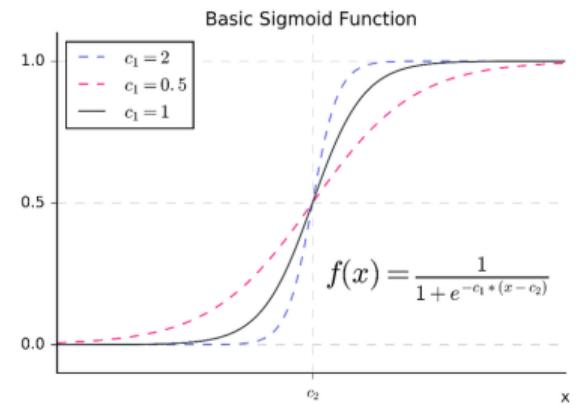
How to make classifier return probabilities?

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



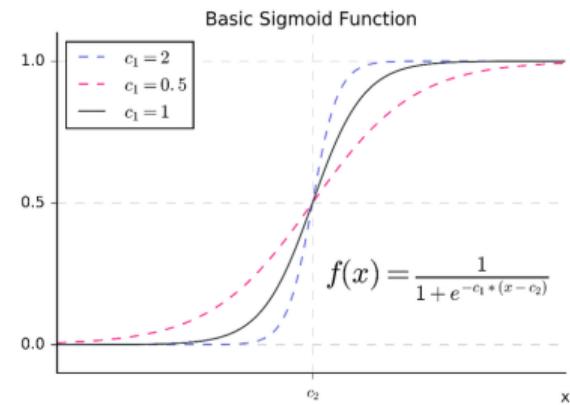
How to make classifier return probabilities?

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



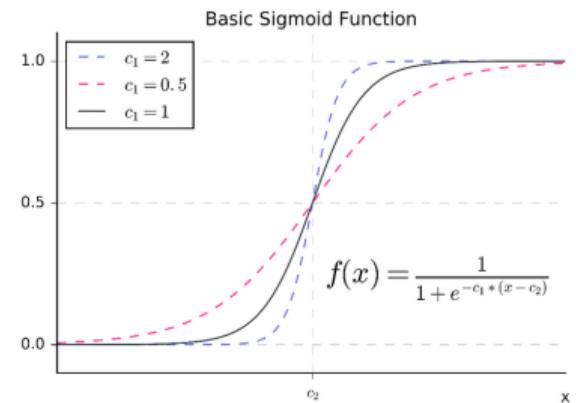
How to make classifier return probabilities?

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



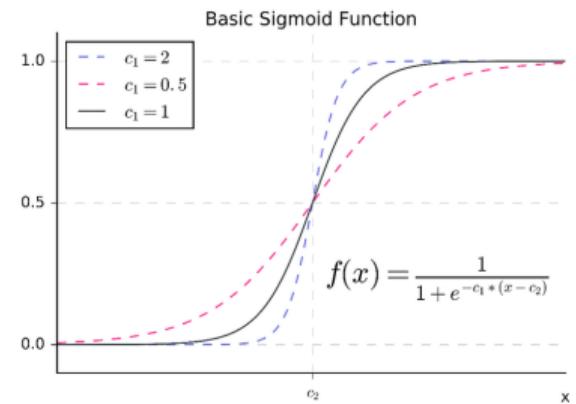
How to make classifier return probabilities?

- $a(x, \theta)$ has a range of \mathbb{R} , but it should be $[0, 1]$
- Let's think of a function that converts $\mathbb{R} \mapsto [0, 1]$
- **Sigmoid function** looks nice: $\sigma(z) = \frac{1}{1 + e^{-z}}$
- Wrap it around: $a_\sigma(x, \theta) = \sigma(a(x, \theta)) = \frac{1}{1 + e^{-\langle x, \theta \rangle}}$
- Now it is possible to plug a_σ into log-likelihood and train:
$$\hat{\theta} = \arg \max_{\theta \in \Theta} \log \mathcal{L}(\theta) =$$
$$= \arg \max_{\theta \in \Theta} \sum_{i=1}^N y_i \log \sigma(\langle x_i, \theta \rangle) + (1 - y_i) \log (1 - \sigma(\langle x_i, \theta \rangle))$$



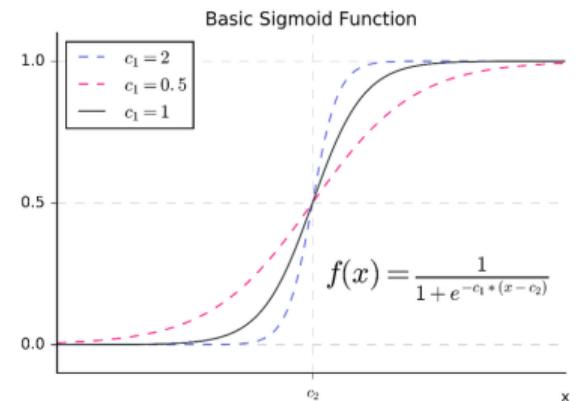
How to make classifier return probabilities?

- $a(x, \theta)$ has a range of \mathbb{R} , but it should be $[0, 1]$
- Let's think of a function that converts $\mathbb{R} \mapsto [0, 1]$
- **Sigmoid function** looks nice: $\sigma(z) = \frac{1}{1 + e^{-z}}$
- Wrap it around: $a_\sigma(x, \theta) = \sigma(a(x, \theta)) = \frac{1}{1 + e^{-\langle x, \theta \rangle}}$
- Now it is possible to plug a_σ into log-likelihood and train:
$$\hat{\theta} = \arg \max_{\theta \in \Theta} \log \mathcal{L}(\theta) =$$
$$= \arg \max_{\theta \in \Theta} \sum_{i=1}^N y_i \log \sigma(\langle x_i, \theta \rangle) + (1 - y_i) \log (1 - \sigma(\langle x_i, \theta \rangle))$$
- This approach is called **logistic regression**



How to make classifier return probabilities?

- $a(x, \theta)$ has a range of \mathbb{R} , but it should be $[0, 1]$
- Let's think of a function that converts $\mathbb{R} \mapsto [0, 1]$
- **Sigmoid function** looks nice: $\sigma(z) = \frac{1}{1 + e^{-z}}$
- Wrap it around: $a_\sigma(x, \theta) = \sigma(a(x, \theta)) = \frac{1}{1 + e^{-\langle x, \theta \rangle}}$
- Now it is possible to plug a_σ into log-likelihood and train:
$$\hat{\theta} = \arg \max_{\theta \in \Theta} \log \mathcal{L}(\theta) =$$
$$= \arg \max_{\theta \in \Theta} \sum_{i=1}^N y_i \log \sigma(\langle x_i, \theta \rangle) + (1 - y_i) \log (1 - \sigma(\langle x_i, \theta \rangle))$$
- This approach is called **logistic regression**



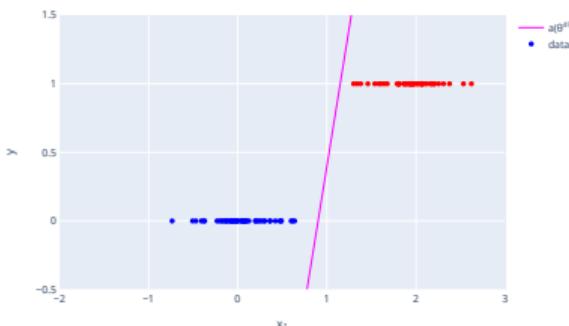
NB: usage of σ and log-likelihood in general does not guarantee probabilistic interpretation

Comparison

Linear classifier

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

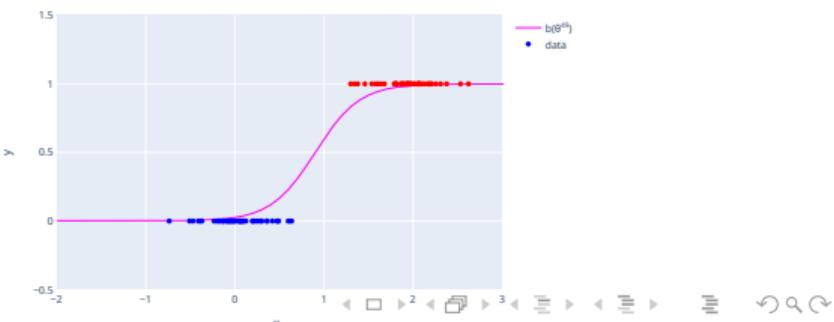
- $a(x, \boldsymbol{\theta}) = \langle x, \boldsymbol{\theta} \rangle$
- $a(x, \boldsymbol{\theta}) : \mathbb{R}^K \times \mathbb{R}^K \mapsto \mathbb{R}$
- ✓ Easy to interpret
- ✗ Gives only yes-no answer



Logistic regression

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

- $a_\sigma(x, \boldsymbol{\theta}) = \sigma[a(x, \boldsymbol{\theta})] = \sigma[\langle x, \boldsymbol{\theta} \rangle]$
- $a_\sigma(x, \boldsymbol{\theta}) : \mathbb{R}^K \times \mathbb{R}^K \mapsto [0, 1]$
- ✓ More profound interpretation
- ✓ Returns probability of positive class $y = 1$

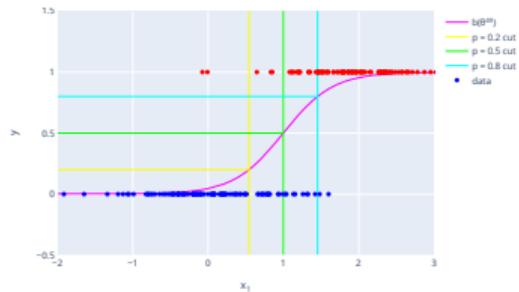


Output of binary classifier

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



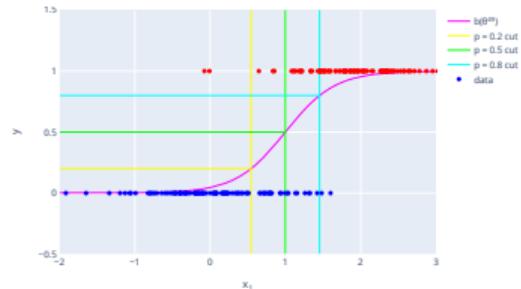
Output of binary classifier

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:

$$\bullet \text{ accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

→ fraction of correct answers

$$\bullet \text{ precision} = \frac{TP}{TP + FP}$$

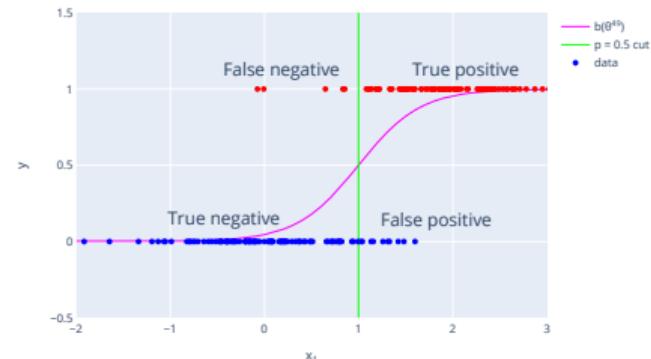
→ fraction of correct positive answers in all positive answers

$$\bullet \text{ recall} = \frac{TP}{TP + FN}$$

→ fraction of correctly classified positive objects

$$\bullet \text{ F1-score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

→ harmonic mean of precision and recall



Metrics

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

- **accuracy** = $\frac{TP + TN}{TP + TN + FP + FN}$

→ fraction of correct answers

- **precision** = $\frac{TP}{TP + FP}$

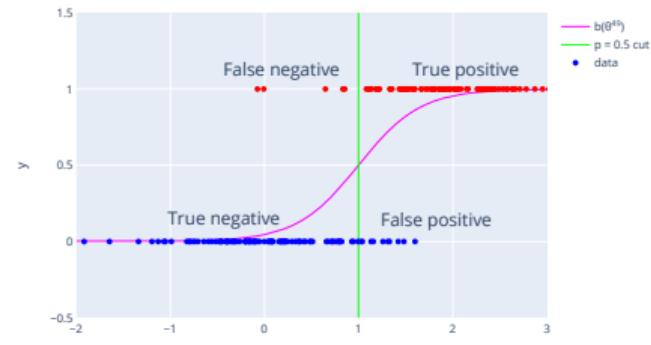
→ fraction of correct positive answers in all positive answers

- **recall** = $\frac{TP}{TP + FN}$

→ fraction of correctly classified positive objects

- **F1-score** = $\frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$

→ 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?

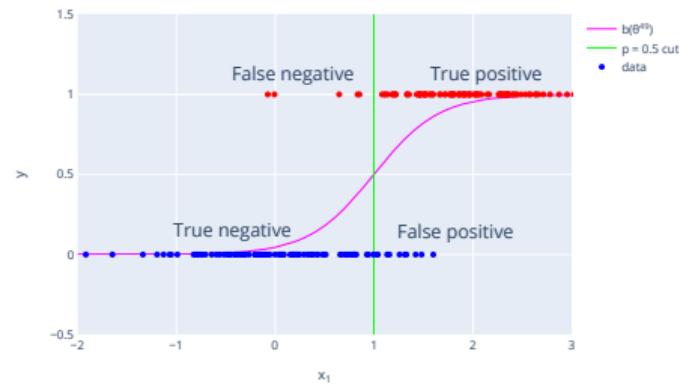
- define **true positive rate**: $\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$

- and **false positive rate**: $\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$

- move the **green** line on the plot

- moving to the left results in $\text{TP} \uparrow$ and $\text{FN} \downarrow$
(sensitive, but not specific)

- moving to the right results in $\text{TP} \downarrow$ and $\text{FN} \uparrow$
(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?

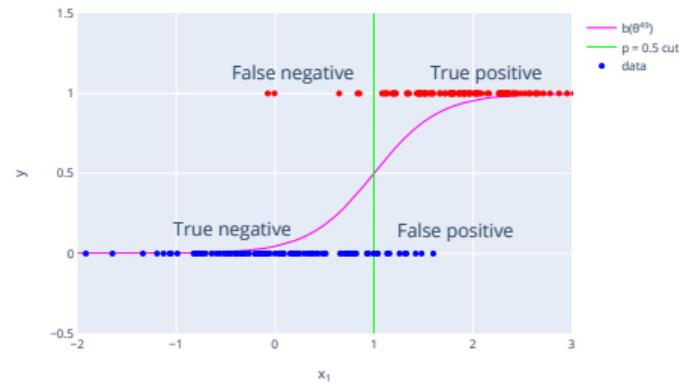
- define **true positive rate**: $TPR = \frac{TP}{TP + FN}$

- and **false positive rate**: $FPR = \frac{FP}{FP + TN}$

- move the **green** line on the plot

- moving to the left results in $TP \uparrow$ and $FN \downarrow$
(sensitive, but not specific)

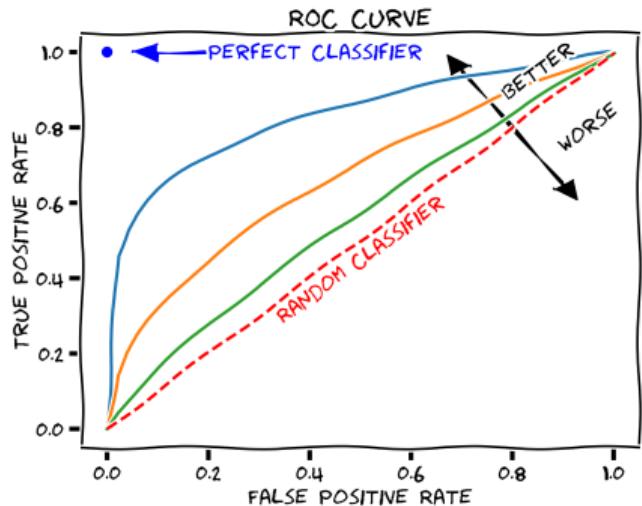
- moving to the right results in $TP \downarrow$ and $FN \uparrow$
(specific, but not sensitive)



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



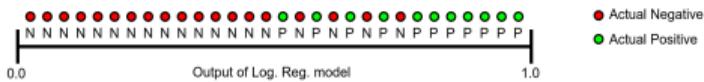
Receiver Operating Characteristic

—○ area under curve

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

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

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



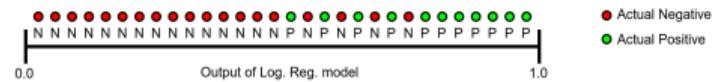
Receiver Operating Characteristic

—○ area under curve

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

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

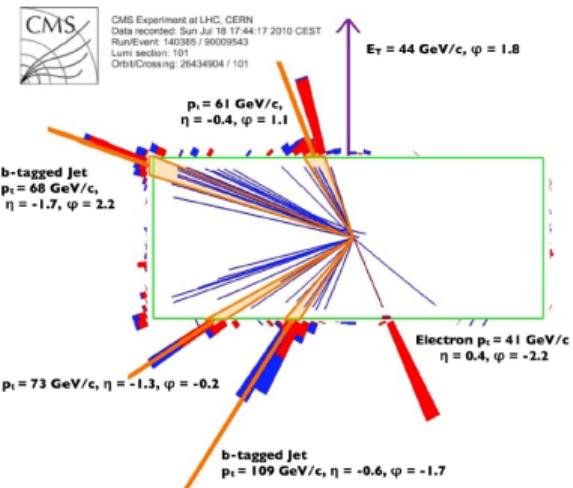
- aggregated measure of performance across all possible thresholds ⇒ **threshold-invariant**
- 1 ⇒ predictions are 100% correct
- 0.5 ⇒ random classifier
- 0 ⇒ predictions are 100% wrong
- probability that a random positive (green) example is positioned to the right of a random negative (red) example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



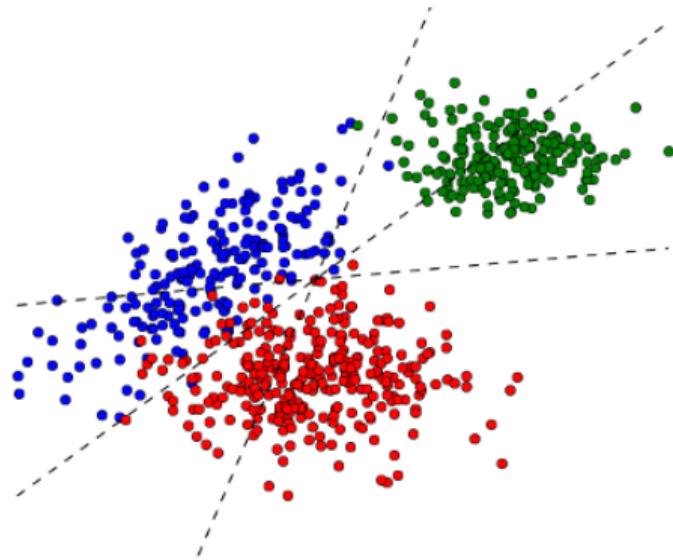
Intermediate Summary

- 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
- $|Y| < \infty = \{1, \dots, L\} \Rightarrow$ impossible to use single model



From multiclass problem to a set of binary problems

"One vs all"

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

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

"All vs all"

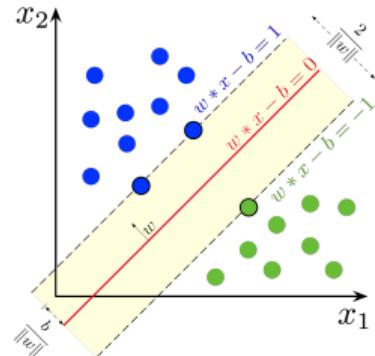
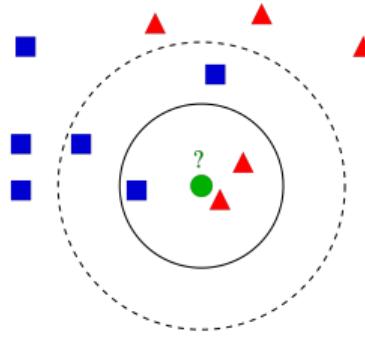
- Train C_L^2 binary classifiers to separate every two classes:
 $b_{i,j}(\mathbf{x}, \boldsymbol{\theta}), i, j \in 1, \dots, L, i \neq j$
- In this case each classifier decides only between two classes
- $\hat{y} = \arg \max_{\ell \in \{1, \dots, L\}} \sum_{i=1}^L \sum_{j \neq i} [b_{i,j}(\mathbf{x},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

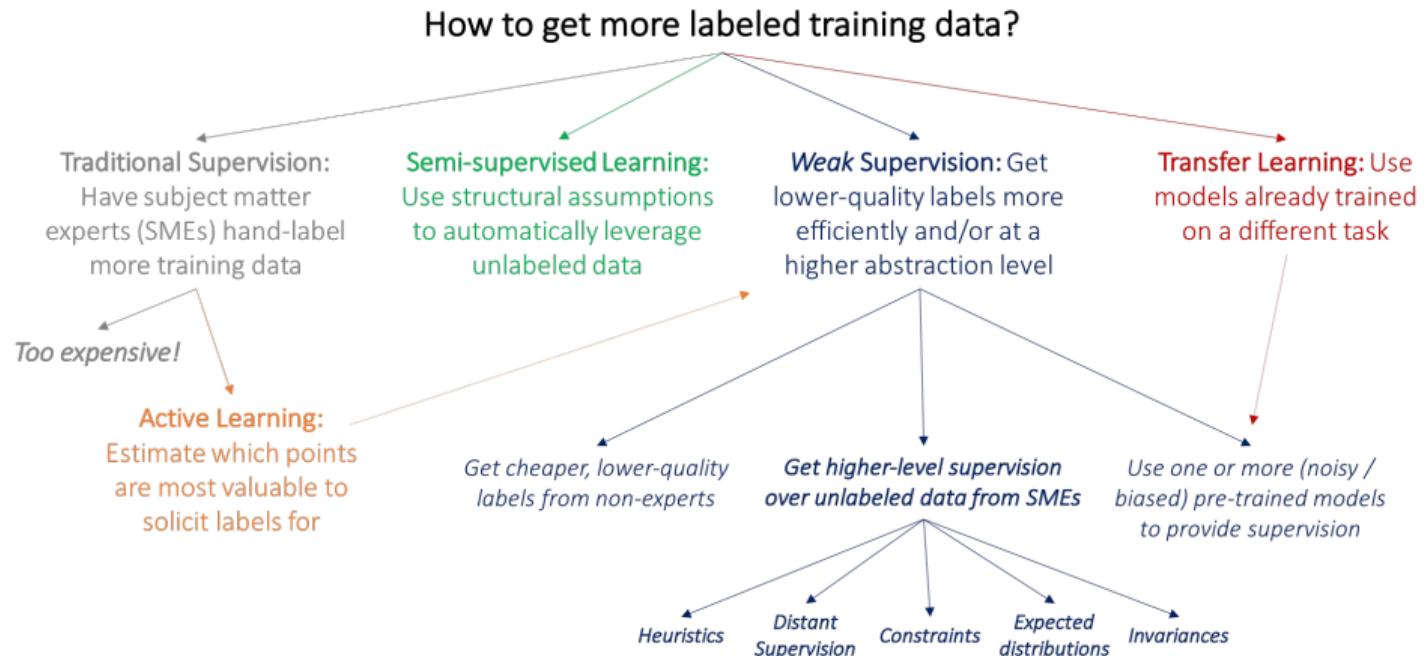
$$\hat{y} = \arg \max_y P(y) \prod_{i=1}^n P(x_i | y)$$



Semi-supervised Learning

What if we don't have labels?

weak-supervision overview

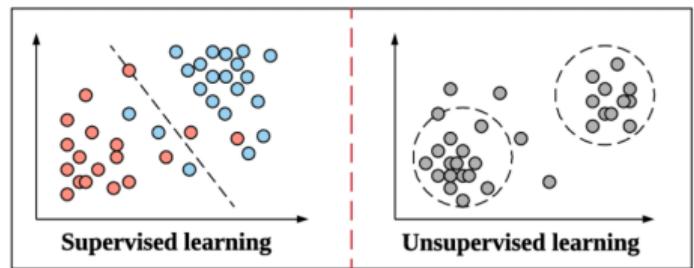


Unsupervised Learning

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

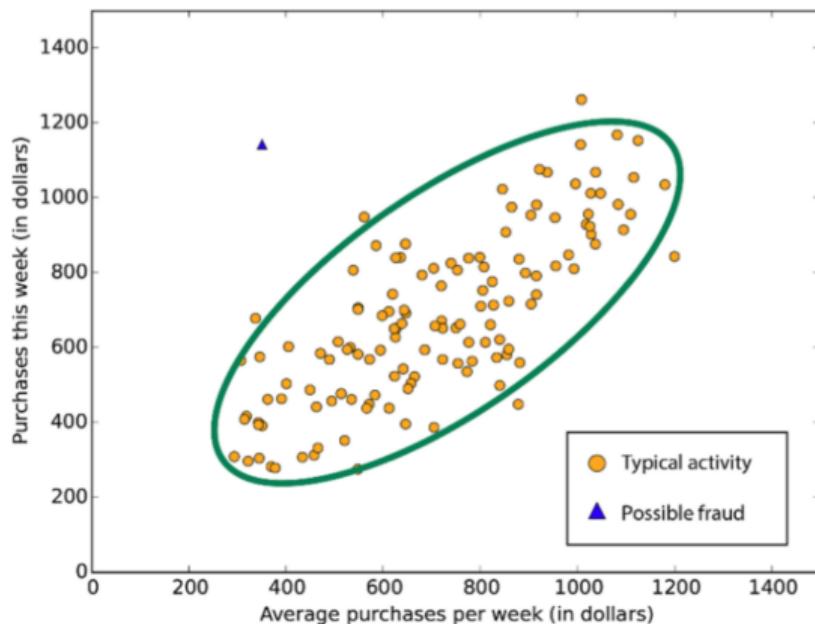
We just have *some* data and we want to:

- Study its **structure**
 - identify novelties/outliers
 - cluster objects
 - estimate densities
 - reduce dimensionality
 - Generate it
- no prior knowledge ⇒ learning is **unsupervised**



Unsupervised Learning

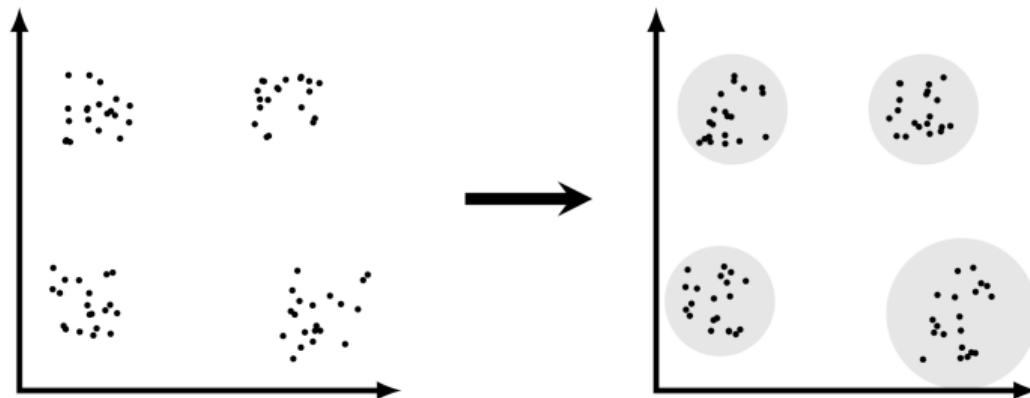
—○ identify anomalies



[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)

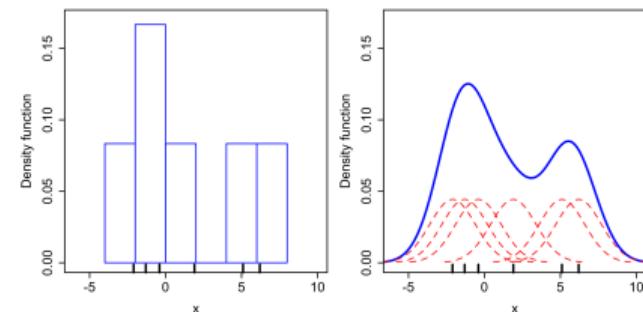
- Which would be nice to know for:

- 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

→ Kernel Density Estimation (KDE)

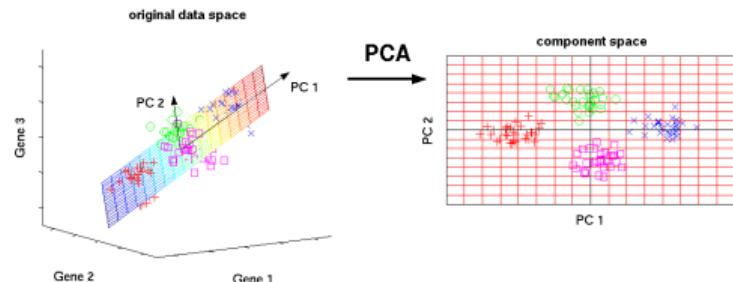
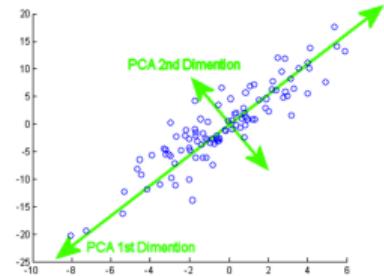


amazing visuals

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

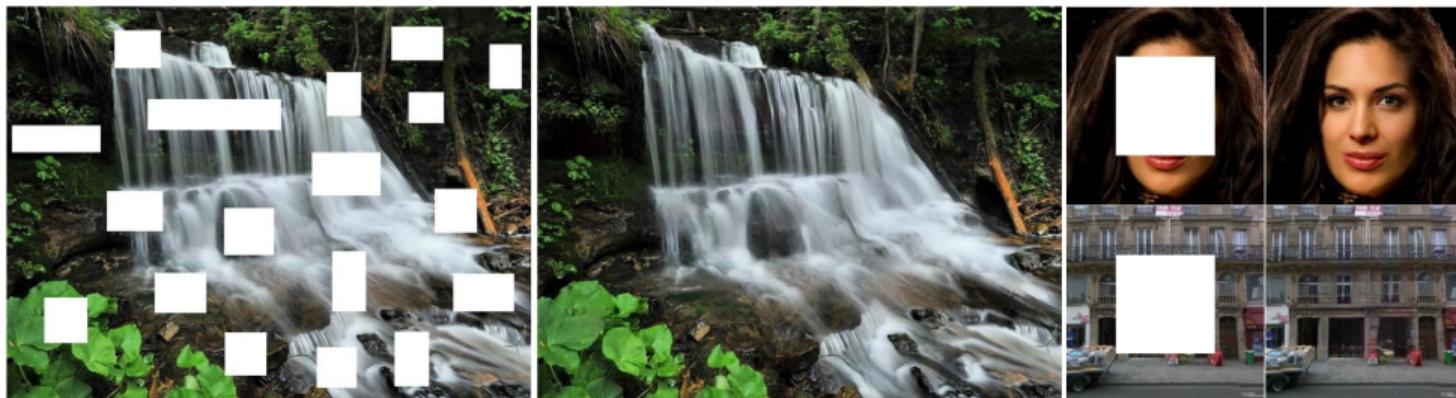


awesome playground

Unsupervised Learning

—○ generate data

1810.08771



playground

Unsupervised Learning

—○ generate data in HEP

Getting High

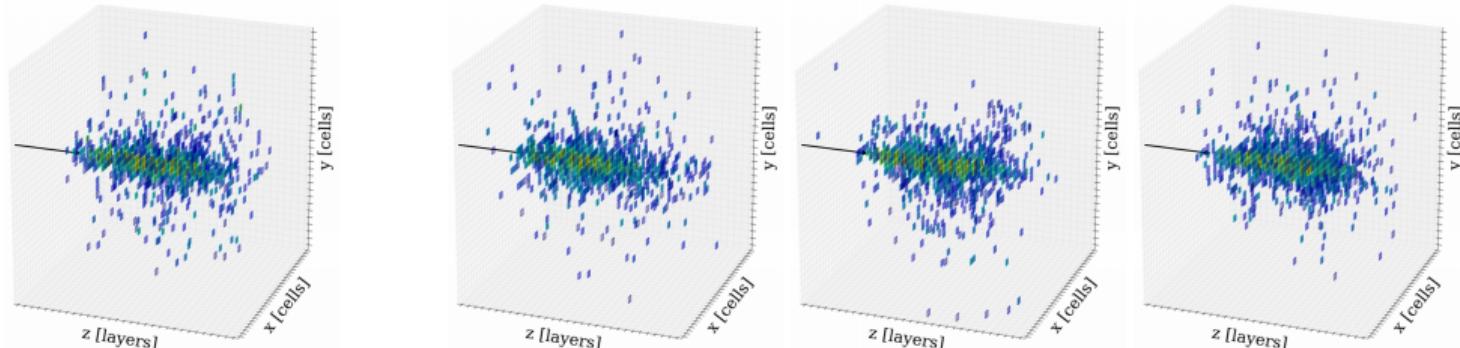
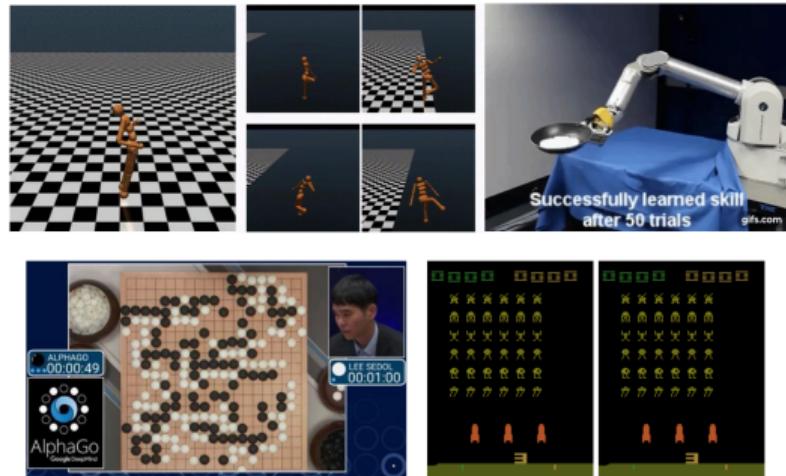


Fig. 5 Examples of individual 50 GeV photon showers generated by GEANT4 (left), the GAN (center left), WGAN (center right), and BIB-AE (right) architectures. Colors encode the deposited energy per cell.

Reinforcement Learning

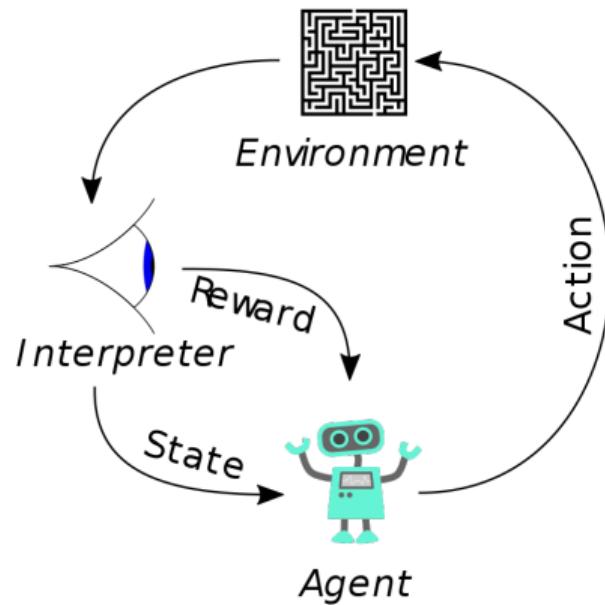
What if there's need for action?

- 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



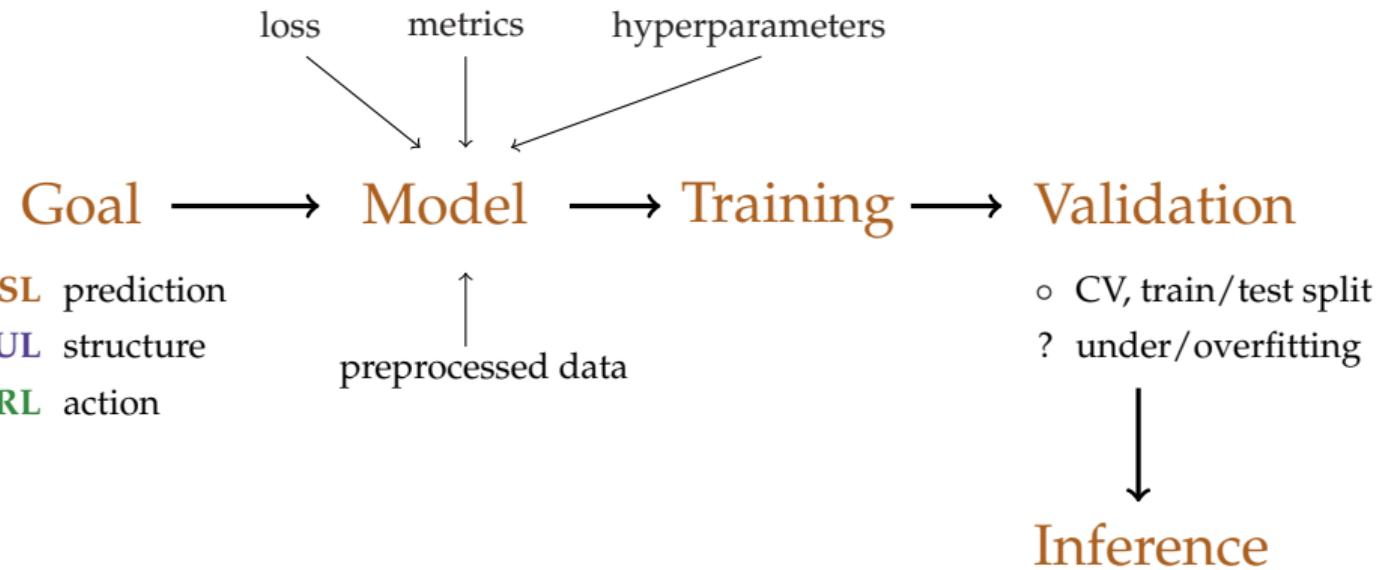
Reinforcement Learning

—○ in a nutshell



more in [cs234 course](#)

Summary



Backup slides

Solution can be found analytically

After defining feature matrix as:

$$F = \begin{bmatrix} f_1(\mathbf{x}_1) & \cdots & f_K(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ f_1(\mathbf{x}_N) & \cdots & f_K(\mathbf{x}_N) \end{bmatrix}$$

Minimization problem for linear regression:

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

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

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

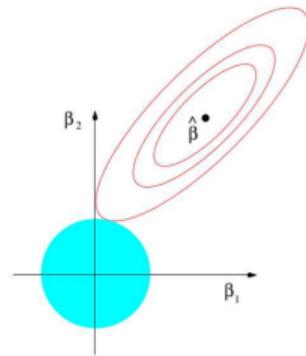
Problem 2: F is often **ill-conditioned** ⇒ computational errors causes instabilities and extremely large solutions

Regularization

playgrounds [click](#), [click](#)

L2 regularization (Tikhonov)

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

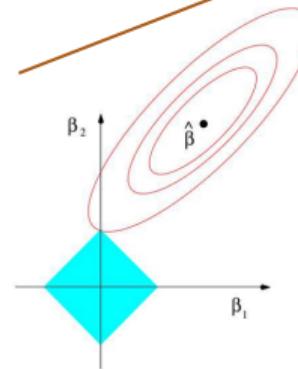


↑
penalize model for too large weights

L1 regularization (LASSO)

least absolute shrinkage and selection operator

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

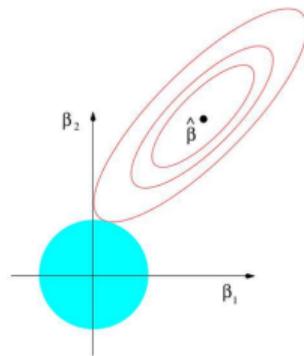


Regularization

playgrounds [click](#), [click](#)

L2 regularization (Tikhonov)

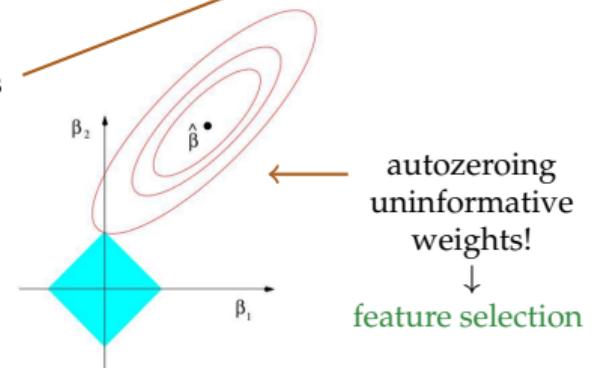
$$Q(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N (\langle \mathbf{x}, \boldsymbol{\theta} \rangle - y_i)^2 + \lambda \sum_{j=1}^K \boldsymbol{\theta}_j^2$$



↑
penalize model for too large weights

L1 regularization (LASSO)

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

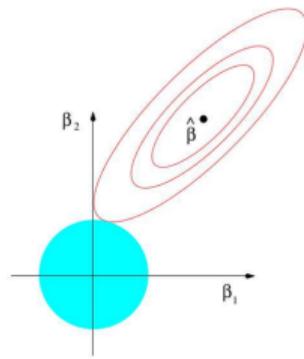


Regularization

playgrounds [click](#), [click](#)

L2 regularization (Tikhonov)

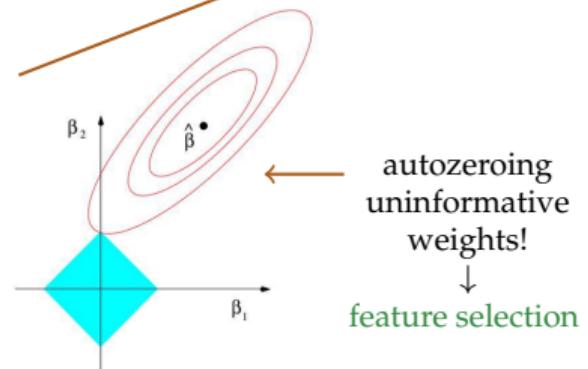
$$Q(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N (\langle \mathbf{x}, \boldsymbol{\theta} \rangle - y_i)^2 + \lambda \sum_{j=1}^K \theta_j^2$$



penalize model for too large weights

L1 regularization (LASSO)

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



↑

↑

λ is a **hyperparameter** → more in homework