Introduction to Machine Learning
Lecture II

Oleg Filatov [1] Stepan Zakharov [2]

[1]DESY [2]Novosibirsk State University

Outline

- **Linear models: regression**
  - Analytic solution
  - (Stochastic) Gradient Descent
  - When to stop training?
  - (Under) Overtraining and its treatment
  - Loss functions & metrics
- **Linear models: classification**
  - Binary classifier
  - Logistic regression
  - Metrics
  - Multiclass case

Recap — SL fundamentals

Given:

- Set of objects that forms a 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 a that approximates g on the whole set X

Define:

- Loss function $\mathcal{L}(a(\boldsymbol{x}), y)$
- $Q(a, X^N) = \frac{1}{N} \sum\limits_{i=1}^{N} \mathcal{L}\big(a(\boldsymbol{x_i}), y_i\big)$

$$\textbf{Then: } \hat{a} = \arg\min_{a \in A} Q(a, X^N)$$

Recap — features and answers

Objects are described by a set of **features** $f_j(\boldsymbol{x}) \in D_j$:

- $D_j = \{0, 1\} \Rightarrow$ binary feature
- $|D_j| < \infty \Rightarrow$ categorical feature
- $|D_j| < \infty$ + total order $\Rightarrow$ ordinal feature
- $D_j = \mathbb{R} \Rightarrow$ numerical feature

Type of answer $y_i \in Y$ generally defines **problem type**:

- $Y = \{0, 1\} \Rightarrow$ binary classification
- $|Y| < \infty \Rightarrow$ multi-class classification
- $|Y| < \infty$ + total order $\Rightarrow$ ranking
- $Y = \mathbb{R} \Rightarrow$ regression

Linear models

regression

Linear regression

- $Y = \mathbb{R}$
- N objects with K real features: $D = \mathbb{R}^K$
- $a(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \sum\limits_{j=1}^{K} f_j(\boldsymbol{x}) \cdot \theta_j$
- Extend and reassign:
  $[1, f_1(\boldsymbol{x}), \ldots, f_K(\boldsymbol{x})] \equiv \boldsymbol{x}$
  $[\theta_0, \ldots, \theta_K] \equiv \boldsymbol{\theta}$
- Then $a(\boldsymbol{x}, \boldsymbol{\theta}) = \langle \boldsymbol{x}, \boldsymbol{\theta} \rangle$

- Minimization problem:
  - $\mathcal{L}(\boldsymbol{\theta}) = \big(\langle \boldsymbol{x}, \boldsymbol{\theta} \rangle - y\big)^2$
  - $Q(\boldsymbol{\theta}) = \frac{1}{N} \sum\limits_{i=1}^{N} \big(\langle \boldsymbol{x_i}, \boldsymbol{\theta} \rangle - y_i\big)^2$
- Then we need to minimize $Q(\boldsymbol{\theta})$ by varying $\boldsymbol{\theta}$:
  - $\hat{a} = \arg\min\limits_{\boldsymbol{\theta} \in \Theta} Q(\boldsymbol{\theta})$

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:

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

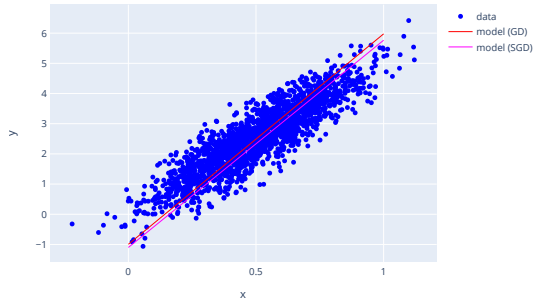
$$\hat{\boldsymbol{\theta}} = (F^T F)^{-1} F^T \boldsymbol{y}$$

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

Better approach? — example

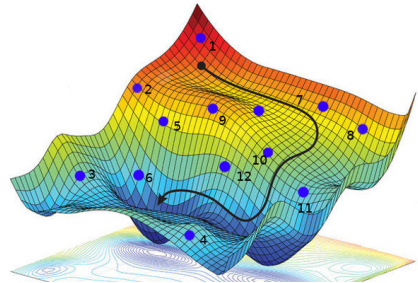
- Let's consider a problem where objects have only one feature: $\boldsymbol{x} = [1, f_1(\boldsymbol{x})]$
- Then $\boldsymbol{\theta} = [\theta_0, \theta_1]$
- How can we find minimum of $Q(\boldsymbol{\theta})$ without solving matrix equation?



Better approach? — random search

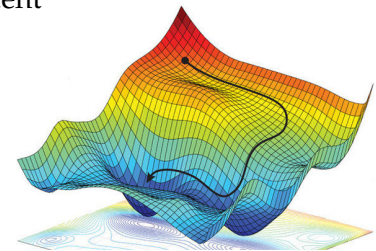
- Start with some vector $\boldsymbol{\theta^0}$
- Initialize $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta^0}$
- ↓ Take randomly another vector $\boldsymbol{\theta'}$
- ↓ Compare $Q(\boldsymbol{\theta'})$ and $Q(\hat{\boldsymbol{\theta}})$
- ↓ $\hat{\boldsymbol{\theta}} = \arg\min_{\hat{\boldsymbol{\theta}}, \boldsymbol{\theta'}} (Q(\hat{\boldsymbol{\theta}}), Q(\boldsymbol{\theta'}))$
- ↓ Repeat until convergence

Problem: random search of $\hat{\boldsymbol{\theta}}$ is not effective for practical use in high dimensional Θ space



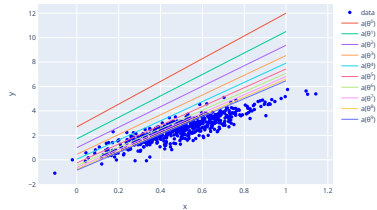
Q as a function of θ_0 and θ_1.
Blue dots represent Q values for different $\boldsymbol{\theta}$

Better approach? — gradient descent



Idea: bring up additional 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



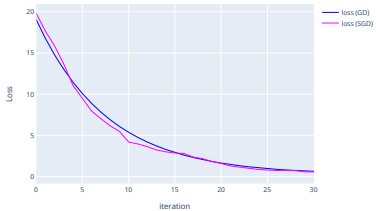
Better approach? — stochastic gradient descent

Gradient descent

Calculating $-\nabla Q$ over all objects

✓ Precise gradient

✗ Can be time-consuming for large dataset

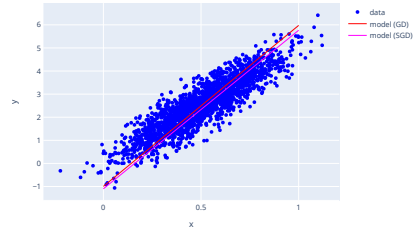


Stochastic gradient descent

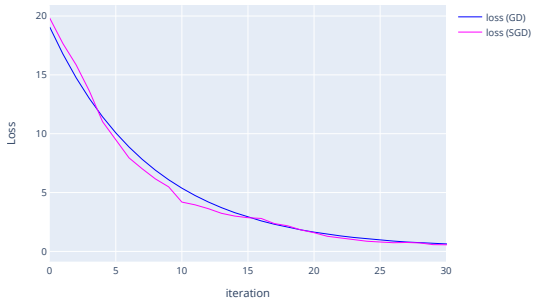
Calculating $-\nabla Q$ over minibatch

✓ Big data friendly

✗ Noisy gradient



When to stop training?

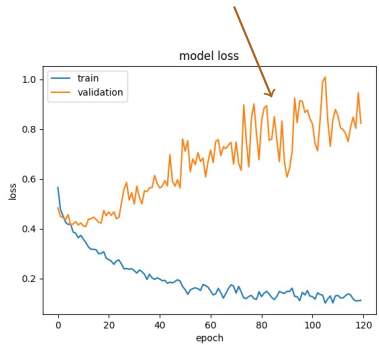


When to stop training?

vanilla approaches

1. Stop once reached predefined number of iteration
   - **?** what if too early? ➔ undertraining
   - ? what if too late? ➔ overtraining
2. Stop once training loss is not improving
   - can use $\mathcal{L}(a(\boldsymbol{x}), y)$ as an error estimator
   - measure it on training data at each iteration
   - stop once $\mathcal{L}_{\text{train}}$ is not improving by predefined ε
   - ➔ monitor the progress online throughout the training
   - ➔ can overfit easily

badly overfitted but $\mathcal{L}_{\text{train}}$ is decreasing

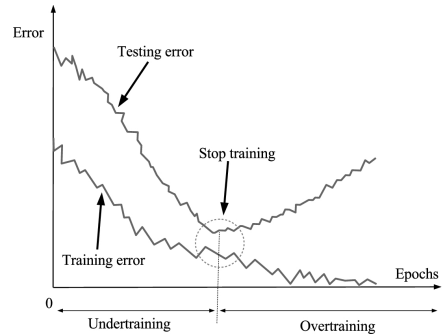


Note: you can use **any metric** instead of $\mathcal{L}$

When to stop training? — using a test set

- To check model's capability to generalise we apply it to **unseen test data**
- Can do this not only after training is done but **throughout**
- So check train and test error for each iteration
- And stop once test metric/loss is not improving by ε → **early stopping**

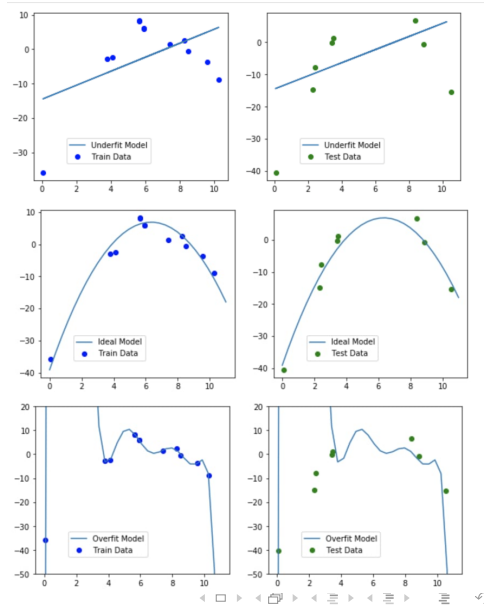
→ detect and prevent overfitting before it happens



Note: "test set" and "validation set" are often used interchangeably

Example — measuring model performance

- After training is done, we want to measure model's performance by applying it to test (unseen) data
- In this example, upper model performs bad both on train and test sets
- Center model shows acceptable results on train an test sets
- Lower model performs well on train and fails on test

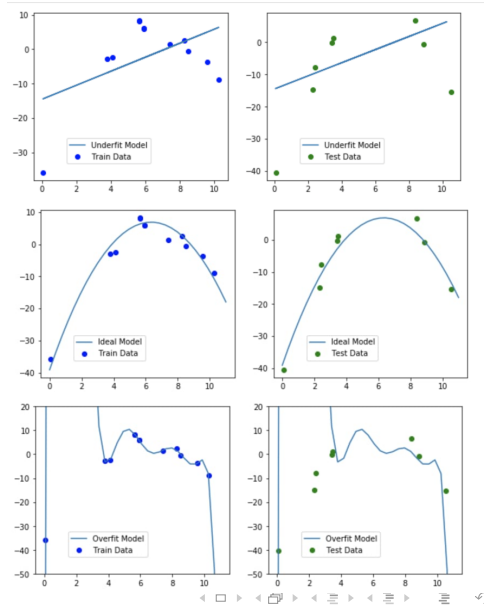


Example — measuring model performance

- After training is done, we want to measure model's performance by applying it to test (unseen) data
- In this example, upper model performs bad both on train and test sets
- Center model shows acceptable results on train an test sets
- Lower model performs well on train and fails on test

$\Downarrow$

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
$\Downarrow$
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

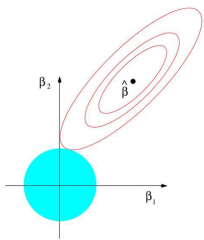
- Catch it before it happens (e.g. early stopping)
- Reflect about noise level vs model complexity
- Automatically limit algorithm space → **regularization**

Regularization

playgrounds click, click

L2 regularization (Tikhonov)

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

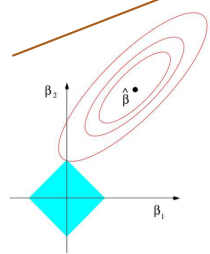


penalize model for too large weights

L1 regularization (LASSO)

least absolute shrinkage and selection operator

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



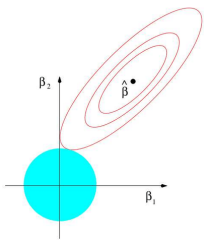
Regularization

playgrounds click, click

L2 regularization (Tikhonov)

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

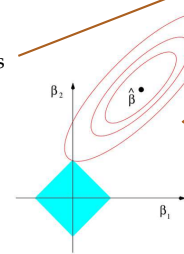
penalize model for too large weights



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!

↓

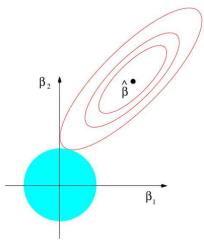
feature selection

Regularization

playgrounds click, click

L2 regularization (Tikhonov)

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

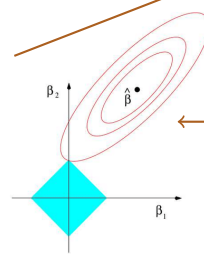


penalize model for too large weights

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!

↓

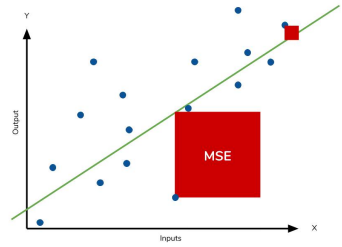
feature selection

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

Why did we choose $\mathcal{L}(a(\boldsymbol{x}), y) = \big(a(\boldsymbol{x}) - y\big)^2$?

Why did we choose $\mathcal{L}(a(\boldsymbol{x}), y) = \big(a(\boldsymbol{x}) - y\big)^2$?

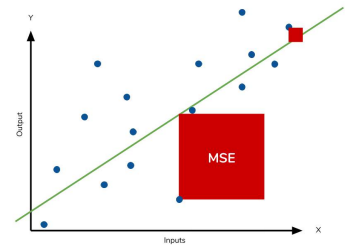
- we chose it because it represents model's error
- which we want to be as small as possible
  (keeping overfitting in mind)
- and therefore we **directly** minimize its **average over the dataset** in training



$$\mathcal{L}_{\text{train}} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\big(a(\boldsymbol{x}_i), y_i\big)$$

Why did we choose $\mathcal{L}(a(\boldsymbol{x}), y) = \big(a(\boldsymbol{x}) - y\big)^2$?

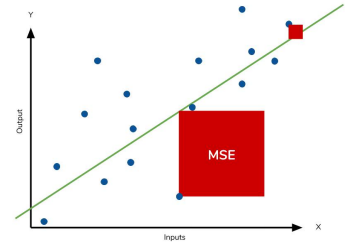
- we chose it because it represents model's error
- which we want to be as small as possible
  (keeping overfitting in mind)
- and therefore we **directly** minimize its **average over the dataset** in training



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

Why did we choose $\mathcal{L}(a(\boldsymbol{x}), y) = \big(a(\boldsymbol{x}) - y\big)^2$?

- we chose it because it represents model's error
- which we want to be as small as possible
  (keeping overfitting in mind)
- and therefore we **directly** minimize its **average over the dataset** in training



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

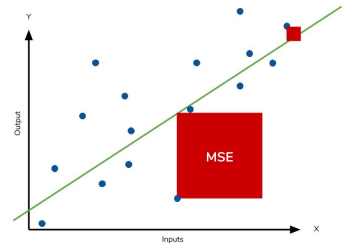
There is something more behind this...

Art of choosing loss function

Mean Squared Error (L2)

$$\text{MSE}(a, X, Y) = \mathbb{E}_{p(x,y)}\left[\left(a(\boldsymbol{x}) - y\right)^2\right]$$

- Grows quadratically with the error ⇒ punish for large mistakes ⇒ **sensitive to outliers** (″mean-unbiased″)
- **Symmetrical** for under/overestimates
- **Smooth** ⇒ no problem with finding optimum
- **Doesn't preserve units** of measurement ⇒ poorly interpretable
- **Good statistical properties** ⇒ might give you BLUE (see Gauss-Markov theorem)



Art of choosing loss function

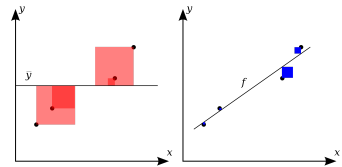
MSE variations

$$\text{MSE} = \mathbb{E}_{p(x,y)}\left[\left(a(x) - y\right)^2\right]$$

- **RMSE** = $\sqrt{\text{MSE}}$
  - ➔ preserve units of measurement ⇒ interpretable
- **MSLE** = $\mathbb{E}_{p(x,y)}\left[\left(\log(a(x)+1) - \log(y+1)\right)^2\right]$
  - ➔ penalize on the "order of magnitude" scale
  - ➔ not symmetric for over/underestimation
- **Coeff. of determination**

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

  - ➔ fraction of "explained" variance: $1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$
  - ➔ "normalised" MSE



Art of choosing loss function

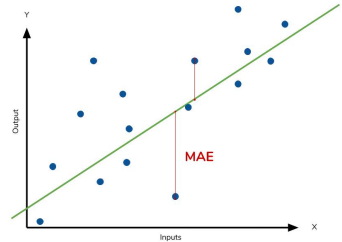
Mean Absolute Error (L1)

$$\text{MAE}(a, X, Y) = \mathbb{E}_{p(x,y)}\big[|a(x) - y|\big]$$

- But for example, we could also take absolute value

* btw why not $\mathcal{L} = \mathbb{E}\big[a(x) - y\big]$?

- ➔ **Simple to interpret**
- ➔ **Symmetrical** for under/overestimates
- ➔ **Robust** to outliers (”median-unbiased”)
- ➔ **Not smooth**
- ➔ **Not backed** much with statistics



Art of choosing loss function

MAE variations

- $\text{MAPE} = \mathbb{E}_{p(x,y)}\left[\left|\frac{y - a(x)}{y}\right|\right]$
  - ➔ "normalised" MAE
- $\text{SMAPE} = 100\% \cdot \mathbb{E}_{p(x,y)}\left[\frac{|y - a(x)|}{|y| + |a(x)|}\right]$
  - ➔ symmetrized MAPE
  - ➔ not really symmetrized for over/underestimation

$$\text{MAE} = \mathbb{E}_{p(x,y)}\left[|a(x) - y|\right]$$

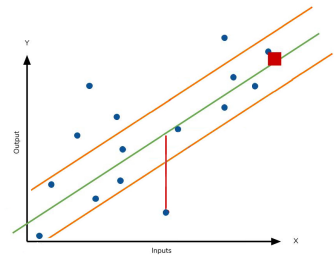
Art of choosing loss function

Huber Loss

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

- Or we could also combine them altogether

→ **Robust** to outliers in tails
→ **Better performance** near minimum (smaller variance)
→ δ is a **hyperparameter** $\Rightarrow$ should be tuned



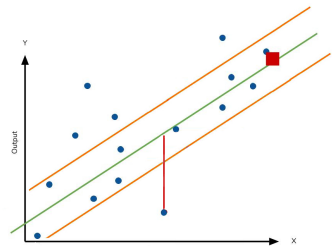
Art of choosing loss function

Huber Loss

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

- Or we could also combine them altogether

→ **Robust** to outliers in tails
→ **Better performance** near minimum (smaller variance)
→ δ is a **hyperparameter** $\Rightarrow$ should be tuned



All of the above losses can (and should) be used as metrics!

Linear models

classification

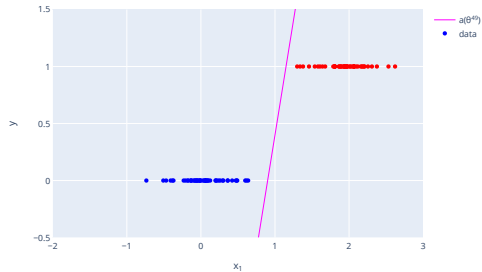
Classification

- $Y = \{-1, +1\}$
- N objects with K real features: $D = \mathbb{R}^K$
- $a(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \sum\limits_{j=1}^{K} f_j(\boldsymbol{x}) \cdot \theta_j$
- Extend and reassign:
  $[1, f_1(\boldsymbol{x}), \ldots, f_K(\boldsymbol{x})] \equiv [1, x_1, \ldots, x_K] \equiv \boldsymbol{x}$
  $[\theta_0, \ldots, \theta_K] \equiv \boldsymbol{\theta}$
- Then $a(\boldsymbol{x}, \boldsymbol{\theta}) = \langle \boldsymbol{x}, \boldsymbol{\theta} \rangle$

- Minimization problem:
  - $\mathcal{L}(\boldsymbol{\theta}) = \big[\text{sign}\langle \boldsymbol{x}, \boldsymbol{\theta} \rangle \neq y\big]$
  - $Q(\boldsymbol{\theta}) = \frac{1}{N} \sum\limits_{i=1}^{N} \big[\text{sign}\langle \boldsymbol{x_i}, \boldsymbol{\theta} \rangle \neq y_i\big]$
- Then we need to minimize $Q(\boldsymbol{\theta})$ by varying $\boldsymbol{\theta}$:
  - $\hat{a} = \arg\min\limits_{\boldsymbol{\theta} \in \Theta} Q(\boldsymbol{\theta})$

Example

- Let's consider the problem of two class (red, blue) classification
- We need to find a line* that provides the best separation of objects from different classes
- Objects have single feature: $\boldsymbol{x} = [1, x_1]$
- Then $\boldsymbol{\theta} = [\theta_0, \theta_1]$



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

Minimization problem for classification

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

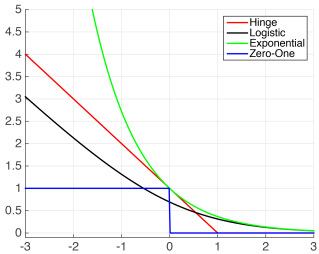
Problem 1: $\mathcal{L}(\boldsymbol{\theta})$ isn't a smooth function $\Rightarrow$ gradient methods are not applicable

Problem 2: Discrete $\{-1, +1\}$ output from the classifier

From binary classification to logistic regression

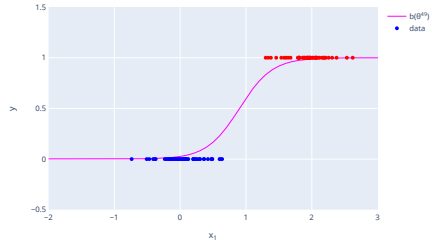
Problem 1:

Solution: use smooth approximations



Problem 2:

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

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:
  $p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x}_i) = a\big(\boldsymbol{x_i}, \boldsymbol{\theta}\big)^{[y_i=+1]}\big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)^{[y_i=-1]}$
- 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:
  $p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x}_i) = a\big(\boldsymbol{x_i}, \boldsymbol{\theta}\big)^{[y_i=+1]}\big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)^{[y_i=-1]}$
- 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:
  $\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} [y_i = 1] \log a(\boldsymbol{x_i}, \boldsymbol{\theta}) + [y_i = -1] \log \big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)$

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:
  $p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x}_i) = a\big(\boldsymbol{x_i}, \boldsymbol{\theta}\big)^{[y_i=+1]}\big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)^{[y_i=-1]}$
- 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:
  $\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} [y_i = 1] \log a(\boldsymbol{x_i}, \boldsymbol{\theta}) + [y_i = -1] \log \big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)$
- Substitute targets: $-1 \to 0$

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:
  $p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x_i}) = a\big(\boldsymbol{x_i}, \boldsymbol{\theta}\big)^{[y_i=+1]}\big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)^{[y_i=-1]}$
- 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:
  $\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} [y_i = 1] \log a(\boldsymbol{x_i}, \boldsymbol{\theta}) + [y_i = -1] \log \big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)$
- Substitute targets: $-1 \to 0$
- 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:
  $p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x}_i) = a\big(\boldsymbol{x_i}, \boldsymbol{\theta}\big)^{[y_i=+1]}\big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)^{[y_i=-1]}$
- 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:
  $\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} [y_i = 1] \log a(\boldsymbol{x_i}, \boldsymbol{\theta}) + [y_i = -1] \log \big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)$
- Substitute targets: $-1 \to 0$
- 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:
  $p_{\boldsymbol{\theta}}(y_i|\boldsymbol{x}_i) = a\big(\boldsymbol{x_i}, \boldsymbol{\theta}\big)^{[y_i=+1]}\big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)^{[y_i=-1]}$
- 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:
  $\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} [y_i = 1] \log a(\boldsymbol{x_i}, \boldsymbol{\theta}) + [y_i = -1] \log \big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big)$
- Substitute targets: $-1 \to 0$
- So that: $\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} y_i \log a(\boldsymbol{x_i}, \boldsymbol{\theta}) + (1 - y_i) \log \big(1 - a(\boldsymbol{x_i}, \boldsymbol{\theta})\big) = \mathbb{E}_{p(x,y)}\big[\log\big(p_{\boldsymbol{\theta}}(y|\boldsymbol{x})\big)\big]$

  aka log-likelihood ↓
- Finally: $\hat{a} = \arg\max_{\boldsymbol{\theta} \in \Theta} \log \mathcal{L}(\boldsymbol{\theta})$

How to make classifier return probabilities?

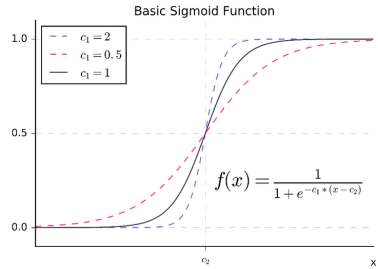
- $a(\boldsymbol{x}, \boldsymbol{\theta})$ has a range of $\mathbb{R}$, but it should be $[0, 1]$

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 $\mathbb{R} \longmapsto [0, 1]$

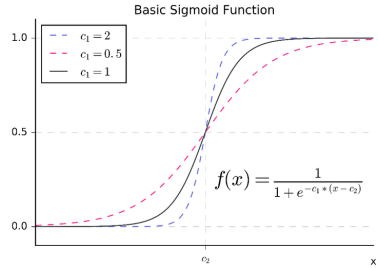
How to make classifier return probabilities?

- $a(\boldsymbol{x}, \boldsymbol{\theta})$ 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: $\sigma(z) = \dfrac{1}{1 + e^{-z}}$



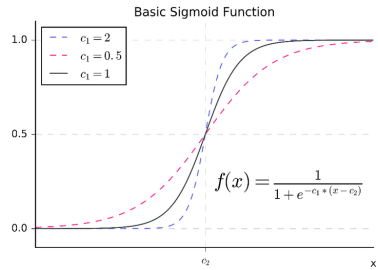
How to make classifier return probabilities?

- $a(\boldsymbol{x}, \boldsymbol{\theta})$ 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:

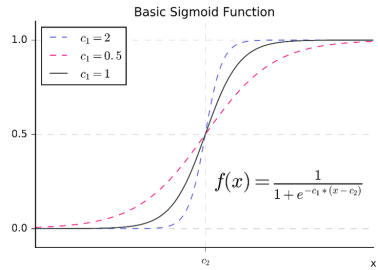


How to make classifier return probabilities?

- $a(\boldsymbol{x}, \boldsymbol{\theta})$ 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:

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

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



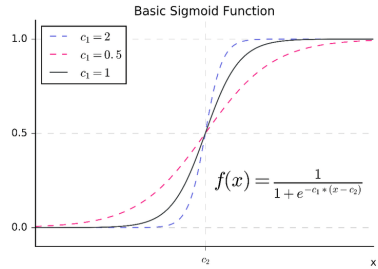
How to make classifier return probabilities?

- $a(\boldsymbol{x}, \boldsymbol{\theta})$ 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**



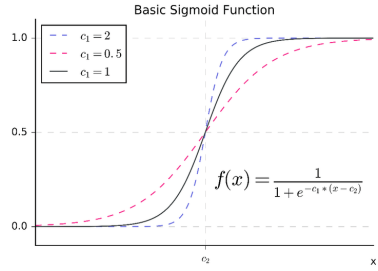
How to make classifier return probabilities?

- $a(\boldsymbol{x}, \boldsymbol{\theta})$ 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

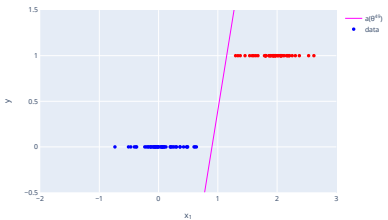
Linear classifier

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

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

✓ Easy to interpret

× Gives only yes-no answer



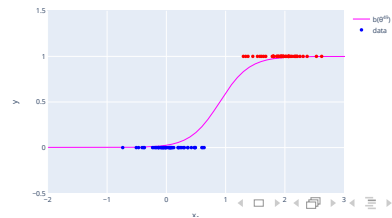
Logistic regression

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

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

✓ More profound interpretation

✓ Returns probability of positive class $y = 1$

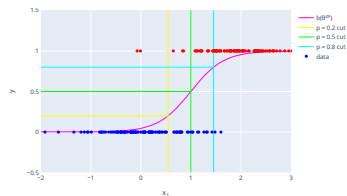


Output of binary classifier

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



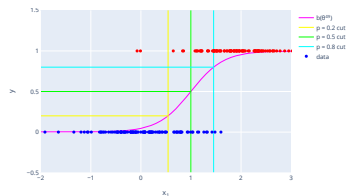
Output of binary classifier

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

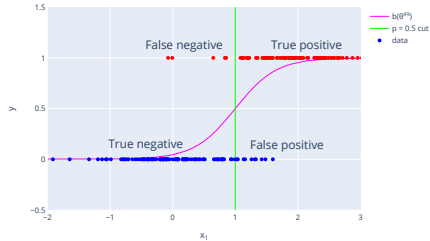
➔ Then how do we measure model's performance?



Metrics

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

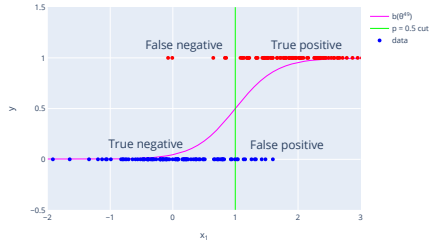
- **accuracy** = $\dfrac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$
  - ➔ fraction of correct answers
- **precision** = $\dfrac{\text{TP}}{\text{TP} + \text{FP}}$
  - ➔ fraction of correct positive answers in all positive answers
- **recall** = $\dfrac{\text{TP}}{\text{TP} + \text{FN}}$
  - ➔ 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!

Example — what if dataset is imbalanced?

- Consider a dataset with 10000 negative and 10 positive objects

Example — what if dataset is imbalanced?

- Consider a dataset with 10000 negative and 10 positive objects
- After training you might end up with a model which returns negative answers all the time

Example — what if dataset is imbalanced?

- Consider a dataset with 10000 negative and 10 positive objects
- After training you might end up with a model which returns negative answers all the time
- Because from loss perspective it might be optimal – contribution of misclassified positives to the loss function will be marginal:

$$\log \mathcal{L}(\boldsymbol{\theta}) \sim \sum_{i=1}^{10} \log a_\sigma(\boldsymbol{x}_i, \boldsymbol{\theta}) + \sum_{i=1}^{10000} \log \big(1 - a_\sigma(\boldsymbol{x}_i, \boldsymbol{\theta})\big)$$

Example — what if dataset is imbalanced?

- Consider a dataset with 10000 negative and 10 positive objects
- After training you might end up with a model which returns negative answers all the time
- Because from loss perspective it might be optimal – contribution of misclassified positives to the loss function will be marginal:

$$\log \mathcal{L}(\boldsymbol{\theta}) \sim \sum_{i=1}^{10} \log a_\sigma(\boldsymbol{x}_i, \boldsymbol{\theta}) + \sum_{i=1}^{10000} \log \left(1 - a_\sigma(\boldsymbol{x}_i, \boldsymbol{\theta})\right)$$

- Furthermore, if you used accuracy as a metric to be optimised, you would get:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{0 + 10000}{0 + 10000 + 0 + 10} \approx \mathbf{99.9\%!}$$

Example — what if dataset is imbalanced?

- Consider a dataset with 10000 negative and 10 positive objects
- After training you might end up with a model which returns negative answers all the time
- Because from loss perspective it might be optimal – contribution of misclassified positives to the loss function will be marginal:

$$\log \mathcal{L}(\boldsymbol{\theta}) \sim \sum_{i=1}^{10} \log a_\sigma(\boldsymbol{x}_i, \boldsymbol{\theta}) + \sum_{i=1}^{10000} \log \big(1 - a_\sigma(\boldsymbol{x}_i, \boldsymbol{\theta})\big)$$

- Furthermore, if you used accuracy as a metric to be optimised, you would get:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{0 + 10000}{0 + 10000 + 0 + 10} \approx \mathbf{99.9\%!}$$

- Hence, you could've assumed that the model is really good (but it's not)

Example — what if dataset is imbalanced?

more in homework!

- Consider a dataset with 10000 negative and 10 positive objects
- After training you might end up with a model which returns negative answers all the time
- Because from loss perspective it might be optimal – contribution of misclassified positives to the loss function will be marginal:

$$\log \mathcal{L}(\boldsymbol{\theta}) \sim \sum_{i=1}^{10} \log a_\sigma(\boldsymbol{x}_i, \boldsymbol{\theta}) + \sum_{i=1}^{10000} \log \left(1 - a_\sigma(\boldsymbol{x}_i, \boldsymbol{\theta})\right)$$

- Furthermore, if you used accuracy as a metric to be optimised, you would get:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{0 + 10000}{0 + 10000 + 0 + 10} \approx \mathbf{99.9\%!}$$

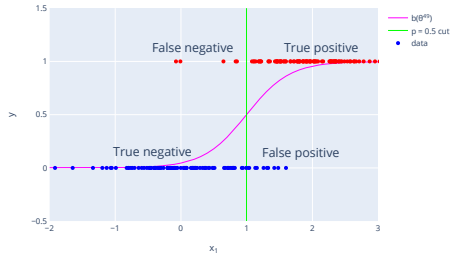
- Hence, you could've assumed that the model is really good (but it's not)

→ Always take imbalance into account! (e.g. up-/downsampling, reweighting)

→ Be careful when interpreting your metrics!

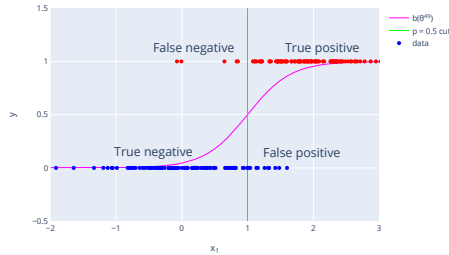
Choosing the right threshold

- 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

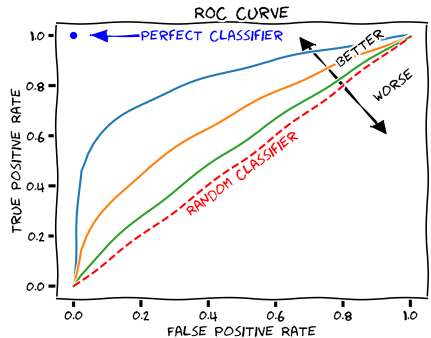
Receiver Operating Characteristic — curve

- 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



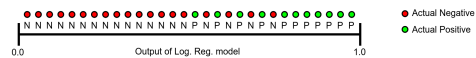
Receiver Operating Characteristic

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**



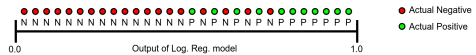
Receiver Operating Characteristic

area under curve

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

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

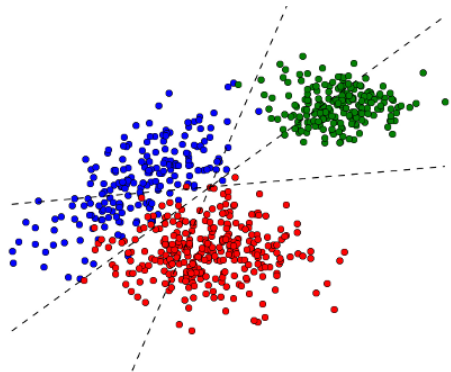
- aggregated measure of performance across all possible thresholds $\Rightarrow$ **threshold-invariant**
- 1 $\Rightarrow$ predictions are 100% correct
- 0.5 $\Rightarrow$ random classifier
- 0 $\Rightarrow$ predictions are 100% wrong
- probability that a random positive (green) example is positioned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



Nice moment to show this animation

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 $a_\ell(\boldsymbol{x}, \boldsymbol{\theta})$ is trained to separate between ℓ–th class and all the rest
- The most confident classifier gives the final separation:

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

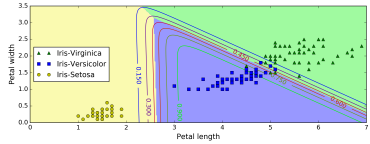
"All vs all"

- Train C_L^2 binary classifiers to separate every two classes: $a_{i,j}(\boldsymbol{x}, \boldsymbol{\theta}), i, j \in 1, \ldots, L, i \neq j$
- 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!

Multiclass logistic regression

- Suppose we trained $a_\ell(\boldsymbol{x}, \boldsymbol{\theta}) = \langle \boldsymbol{x}, \boldsymbol{\theta} \rangle, \ell = 1, \ldots, L$ each separating its individual class
- Now we need to turn their output into probabilities
- Reminder: $a_\ell(\boldsymbol{x}, \boldsymbol{\theta})$ has a range $\mathbb{R}$
- We need to convert vector $\big[a_1(\boldsymbol{x}, \boldsymbol{\theta}), \ldots, a_L(\boldsymbol{x}, \boldsymbol{\theta})\big]$ into a vector of probabilities
- We can put **softmax** on top:

$$\big[a_1(\boldsymbol{x}, \boldsymbol{\theta}), \ldots, a_L(\boldsymbol{x}, \boldsymbol{\theta})\big] \mapsto \left[\frac{\exp[a_1(\boldsymbol{x}, \boldsymbol{\theta})]}{\sum\limits_{\ell=1}^{L} \exp[a_\ell(\boldsymbol{x}, \boldsymbol{\theta})]}, \ldots, \frac{\exp[a_L(\boldsymbol{x}, \boldsymbol{\theta})]}{\sum\limits_{\ell=1}^{L} \exp[a_\ell(\boldsymbol{x}, \boldsymbol{\theta})]}\right]$$



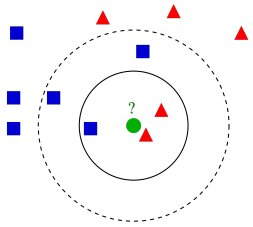
- Then components sum up to 1 and each component represents* probability that object $\boldsymbol{x}$ belongs to a correspondent class

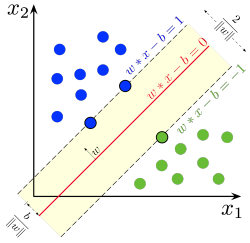
Final note

We didn't have time to describe them, but in the family of "classical algorithms" there are also these guys – and they are cool enough to be studied and tried out:

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

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





Summary

- **Linear models** are the simplest example of models used for SL
- Solution of optimization problem can be found analytically, but **(stochastic) gradient descent** is a more general approach to training
- During the process of training model's performance is checked **throughout** to control **under-** and **overtraining**
- Model **regularization** prevents overtraining thanks to additional term in the loss function
- Several variations to MSE and MAE loss functions exist, each with its unique properties – all can be used both **as losses and as metrics**

Summary

- **Linear models** are the simplest example of models used for SL
- Solution of optimization problem can be found analytically, but **(stochastic) gradient descent** is a more general approach to training
- During the process of training model's performance is checked **throughout** to control **under-** and **overtraining**
- Model **regularization** prevents overtraining thanks to additional term in the loss function
- Several variations to MSE and MAE loss functions exist, each with its unique properties – all can be used both **as losses and as metrics**
- Classification problem has pretty the same pipeline as the regression. Furthermore, it can be converted into **logistic regression** that estimates **probability** of belonging to a class
- **Dedicated classification metrics** exist, all being specific/sensitive to its own thing, both threshold dependent (e.g. accuracy) and not (e.g. ROC AUC)
- **Multiclass clasification** can be also performed using "one-vs-all" and "all-vs-all" methods, or in case of log.regression with applying **softmax function**