Machine learning from scratch

Lecture 1: Mathematical background

Alexis Zubiolo
alexis.zubiolo@gmail.com

Data Science Team Lead @ Adcash

January 26, 2017

Before we start

IT STEP will be organizing a Tech night on **February 16th** (Thursday) from 7pm. I will (probably) be giving a talk. The course will most likely be postponed.

Motivation, vocabulary and notations

In **supervised learning** tasks, we are given a *data set* of the form:

$$D = \left\{ \left(\boldsymbol{x}^{(i)}, y^{(i)} \right), \boldsymbol{x}^{(i)} \in \mathcal{X}, y^{(i)} \in \mathcal{Y}, i \in \{1, \ldots, n\} \right\}$$

Motivation, vocabulary and notations

In **supervised learning** tasks, we are given a *data set* of the form:

$$D = \left\{ \left(\boldsymbol{x}^{(i)}, y^{(i)} \right), \boldsymbol{x}^{(i)} \in \mathcal{X}, y^{(i)} \in \mathcal{Y}, i \in \{1, \ldots, n\} \right\}$$

- n is the size of the data set (number of *instances*/*samples*)
- In most applications:
  - $\mathcal{X} = \mathbb{R}^d$ (d is the *dimensionality*)
  - $\mathcal{Y} = \mathbb{R}$ (*regression*) or $\mathcal{Y} \subset \mathbb{N}$ (*classification*)
- $\boldsymbol{x} \in \mathcal{X}$ is the *feature vector* and $y \in \mathcal{Y}$ is the *label*

Motivation, vocabulary and notations

Solving a **supervised learning problem** is finding (or *learning*) a function (or *hypothesis*) $h : \mathcal{X} \mapsto \mathcal{Y}$ such that for $(\boldsymbol{x}, y) \in D$, $h(\boldsymbol{x})$ is a *good* estimation (or approximation) of y.

Motivation, vocabulary and notations

Solving a **supervised learning problem** is finding (or *learning*) a function (or *hypothesis*) $h : \mathcal{X} \mapsto \mathcal{Y}$ such that for $(\boldsymbol{x}, y) \in D$, $h(\boldsymbol{x})$ is a *good* estimation (or approximation) of y.

We often write $h(\boldsymbol{x}) = \hat{y}$ ($\hat{y}$ is the *prediction* of $\boldsymbol{x}$ by h).

Motivation, vocabulary and notations

Solving a **supervised learning problem** is finding (or *learning*) a function (or *hypothesis*) $h : \mathcal{X} \mapsto \mathcal{Y}$ such that for $(\boldsymbol{x}, y) \in D$, $h(\boldsymbol{x})$ is a *good* estimation (or approximation) of y.

We often write $h(\boldsymbol{x}) = \hat{y}$ ($\hat{y}$ is the *prediction* of $\boldsymbol{x}$ by h).

This raises **2 questions**:

- How to define h?
- How to assess whether $\hat{y}$ is a good approximation of y?

Hypothesis parametrization

h is often defined by a parameter vector θ and can be noted h_θ.

Hypothesis parametrization

h is often defined by a parameter vector θ and can be noted h_θ.
Several ways to parametrize h exist:

- **Linear model**: $h(\boldsymbol{x}) = \theta^T \boldsymbol{x}$
- **Polynomial kernel** (degree k): $h(\boldsymbol{x}) = \left(1 + \theta^T \boldsymbol{x}\right)^k$
- Other kernels exist, more on this when we talk about duality
- With a **neural net**, more on this later as well
- ...

Hypothesis parametrization

h is often defined by a parameter vector θ and can be noted h_θ.
Several ways to parametrize h exist:

- **Linear model**: $h(\boldsymbol{x}) = \theta^T \boldsymbol{x}$
- **Polynomial kernel** (degree k): $h(\boldsymbol{x}) = \left(1 + \theta^T \boldsymbol{x}\right)^k$
- Other kernels exist, more on this when we talk about duality
- With a **neural net**, more on this later as well
- ...

How you define h highly depends on the application, for example:

- Sometimes a lot of data preprocessing has been made and a simple model (e.g. linear) would work well
- You might have **time/hardware constraints**: In this case going for a too complex model might be crippling
- For neural net, the architecture depends a lot on the type of data you have

Linear Algebra concepts (1)

Recall the regression example from the previous lecture:

Linear Algebra concepts (1)

Recall the regression example from the previous lecture:

living area (m^2)	**# bedrooms**	price (1000's BGN)
50	**1**	30
76	**2**	48
26	**1**	12
102	**3**	90

Linear Algebra concepts (1)

Recall the regression example from the previous lecture:

living area (m^2)	**# bedrooms**	price (1000's BGN)
50	**1**	30
76	**2**	48
26	**1**	12
102	**3**	90
61	**2**	?

Linear Algebra concepts (1)

Recall the regression example from the previous lecture:

living area (m^2)	**# bedrooms**	price (1000's BGN)
50	**1**	30
76	**2**	48
26	**1**	12
102	**3**	90
61	**2**	?

Illustration of the introduced notations:

- $\mathcal{X} = \mathbb{R}^2$ ($d = 2$ dimensions: living area and # bedrooms)
- $\mathcal{Y} = \mathbb{R}$ (regression task)
- $\boldsymbol{x}^{(1)} = [50, 1]^T$ and $y^{(1)} = 30000$

Linear algebra concepts (2)

living area (m^2)	**# bedrooms**	price (1000's BGN)
50	**1**	30
76	**2**	48
26	**1**	12
102	**3**	90

Linear algebra concepts (2)

living area (m^2)	**# bedrooms**	price (1000's BGN)
50	**1**	30
76	**2**	48
26	**1**	12
102	**3**	90

Using a **linear regression model** gives

$$h_\theta(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Generalization to any dimensionality d:

$$h(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j x_j = \theta^T \boldsymbol{x}$$

Here, we set $\boldsymbol{x}_0 = 1$ so that θ_0 is included in θ. $\theta^T \boldsymbol{x}$ is called the dot product (or inner product) between *theta* and $\boldsymbol{x}$ and is sometimes noted $\langle \theta, \boldsymbol{x} \rangle$.

Ordinary least squares

living area (m^2)	**# bedrooms**	price (1000's BGN)
50	**1**	30
76	**2**	48
26	**1**	12
102	**3**	90

$$h(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j x_j = \theta^T \boldsymbol{x}$$

Ordinary least squares

living area (m^2)	**# bedrooms**	price (1000's BGN)
50	**1**	30
76	**2**	48
26	**1**	12
102	**3**	90

$$h(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j x_j = \theta^T \boldsymbol{x}$$

Suppose we chose the following loss function:

$$\ell(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$$

This leads to the following least squares *cost function*:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} \left(h\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right)^2$$

This problem the **ordinary least squares** (OLS) regression model.

Least Mean Squares (LMS) update rule

We want to minimize the following cost function:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} \left(h\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right)^2$$

One way to do it is by using the **gradient descent** algorithm:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

for all $j \in \{0, \ldots, d\}$. α is called the **step size** or the **learning rate**. This update rule can be rewritten in a more compact way:

$$\theta := \theta - \alpha \nabla J(\theta)$$

where $\nabla J(\theta)$ is the **gradient** of J in θ. We have, by definition:

$$\nabla J(\theta) = \left[\frac{\partial}{\partial \theta_1} J(\theta), \ldots, \frac{\partial}{\partial \theta_d} J(\theta) \right]^T$$

Least Mean Squares (LMS) update rule

To apply the LMS update rule, we need to compute the gradient of J. Let's compute it for a single $(\boldsymbol{x}, y)$ sample:

$$\begin{aligned}\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \left(h\left(\boldsymbol{x}\right) - y\right)^2 \\ &= ?\end{aligned}$$

Exercise: Compute the gradient and find the update rule.

Least Mean Squares (LMS) update rule

Solution:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \left(h(\boldsymbol{x}) - y \right)^2 \\
&= 2 \frac{1}{2} \left(h(\boldsymbol{x}) - y \right) \frac{\partial}{\partial \theta_j} \left(h(\boldsymbol{x}) - y \right) \\
&= \left(h(\boldsymbol{x}) - y \right) \frac{\partial}{\partial \theta_j} \left(\sum_{k=1}^{d} \theta_k x_k - y \right) \\
&= \left(h(\boldsymbol{x}) - y \right) x_j
\end{aligned}$$

Least Mean Squares (LMS) update rule

Solution:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \left(h(\boldsymbol{x}) - y \right)^2 \\
&= 2 \frac{1}{2} \left(h(\boldsymbol{x}) - y \right) \frac{\partial}{\partial \theta_j} \left(h(\boldsymbol{x}) - y \right) \\
&= \left(h(\boldsymbol{x}) - y \right) \frac{\partial}{\partial \theta_j} \left(\sum_{k=1}^{d} \theta_k x_k - y \right) \\
&= \left(h(\boldsymbol{x}) - y \right) x_j
\end{aligned}$$

So the gradient descent update becomes

$$\begin{aligned}
\theta_j &:= \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\
&:= \theta_j + \alpha \left(y - h(\boldsymbol{x}) \right) x_j
\end{aligned}$$

LMS update rule interpretation

LMS update rule for one sample:

$$\theta_j := \theta_j + \alpha \left(y - h\left(\boldsymbol{x}\right)\right) x_j$$

LMS update rule interpretation

LMS update rule for one sample:

$$\theta_j := \theta_j + \alpha \left(y - h\left(\boldsymbol{x}\right)\right) x_j$$

A few intuitive **interpretations**:

- The magnitude of the step we make is proportional to α
- The magnitude of the step we make is proportional to $\left(y - h\left(\boldsymbol{x}\right)\right)$, which is **the error** we make:
  - If $h\left(\boldsymbol{x}\right) \approx y$, then we won't move much
  - Else, we will make a bigger step

LMS update rule interpretation

LMS update rule for one sample:

$$\theta_j := \theta_j + \alpha \left(y - h\left(\boldsymbol{x}\right)\right) x_j$$

A few intuitive **interpretations**:

- The magnitude of the step we make is proportional to α
- The magnitude of the step we make is proportional to $\left(y - h\left(\boldsymbol{x}\right)\right)$, which is **the error** we make:
  - If $h\left(\boldsymbol{x}\right) \approx y$, then we won't move much
  - Else, we will make a bigger step

Given this update rule, we can derive two different algorithms:

- Batch gradient descent
- Stochastic gradient descent

LMS update rule interpretation: batch updates

LMS update rule for one sample:

$$\theta_j := \theta_j + \alpha \left(y - h\left(\boldsymbol{x} \right) \right) x_j$$

LMS update rule interpretation: batch updates

LMS update rule for one sample:

$$\theta_j := \theta_j + \alpha \left(y - h\left(\boldsymbol{x} \right) \right) x_j$$

Exercise: What would be the batch gradient update rule?

LMS update rule interpretation: batch updates

LMS update rule for one sample:

$$\theta_j := \theta_j + \alpha \left(y - h\left(\boldsymbol{x} \right) \right) x_j$$

Exercise: What would be the batch gradient update rule?

Solution: Repeat, until *convergence*: For every $j \in \{1, \ldots, d\}$:

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{n} \left(y^{(i)} - h\left(x_j^{(i)} \right) \right) x_j^{(i)}$$

LMS update rule interpretation: Stochastic updates

LMS update rule for one sample:

$$\theta_j := \theta_j + \alpha \left(y - h\left(\boldsymbol{x} \right) \right) x_j$$

LMS update rule interpretation: Stochastic updates

LMS update rule for one sample:

$$\theta_j := \theta_j + \alpha \left(y - h\left(\boldsymbol{x} \right) \right) x_j$$

Exercise: What would be the batch stochastic update rule?

LMS update rule interpretation: Stochastic updates

LMS update rule for one sample:

$$\theta_j := \theta_j + \alpha \left(y - h\left(\boldsymbol{x}\right)\right) x_j$$

Exercise: What would be the batch stochastic update rule?

Solution: Repeat, until *convergence*: For $i \in \{1, \ldots, n\}$, for every $j \in \{1, \ldots, d\}$:

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{n} \left(y^{(i)} - h\left(x_j^{(i)}\right)\right) x_j^{(i)}$$

Here, θ is updated everytime we read a sample!

Optimization convergence

The batch and stochastic gradient descents we discussed iterate until convergence. How do we know whether the algorithm has converged or now?

Optimization convergence

The batch and stochastic gradient descents we discussed iterate until convergence. How do we know whether the algorithm has converged or now?

We usually check the difference between two successive values of $J(\theta)$.

Standard loss functions

0-1 loss:

$$\ell(\hat{y}, y) = \mathbb{1}[\hat{y} = y]$$

Standard loss functions

0-1 loss:

$$\ell(\hat{y}, y) = \mathbb{1}[\hat{y} = y]$$

Squared loss:

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$

Standard loss functions

0-1 loss:

$$\ell(\hat{y}, y) = \mathbb{1}[\hat{y} = y]$$

Squared loss:

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$

Hinge loss:

$$\ell(\hat{y}, y) = \max(0, 1 - \hat{y}y)$$

Standard loss functions

0-1 loss:

$$\ell(\hat{y}, y) = \mathbb{1}[\hat{y} = y]$$

Squared loss:

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$

Hinge loss:

$$\ell(\hat{y}, y) = \max(0, 1 - \hat{y}y)$$

Squared hinge loss:

$$\ell(\hat{y}, y) = \max(0, 1 - \hat{y}y)^2$$

Standard loss functions

0-1 loss:

$$\ell(\hat{y}, y) = \mathbb{1}[\hat{y} = y]$$

Squared loss:

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$

Hinge loss:

$$\ell(\hat{y}, y) = \max(0, 1 - \hat{y}y)$$

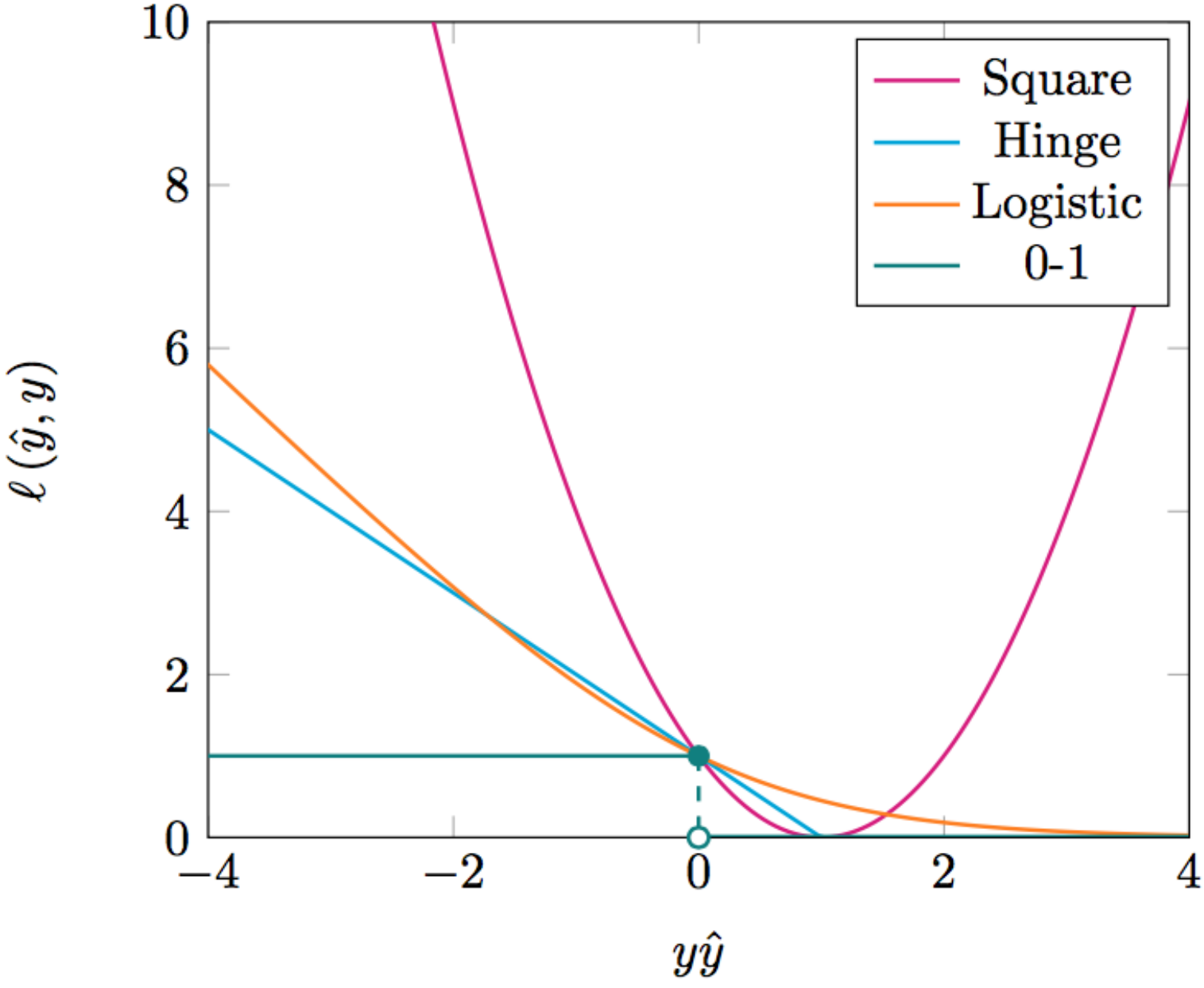
Squared hinge loss:

$$\ell(\hat{y}, y) = \max(0, 1 - \hat{y}y)^2$$

Log-loss:

$$\ell(\hat{y}, y) = \log(1 + \exp(-\hat{y}y))$$

Loss functions



Conclusion

We have seen the motivation and the formalization of a supervised learning problem with a basic example.

Conclusion

We have seen the motivation and the formalization of a supervised learning problem with a basic example.

Next week, there will be some implementation tasks to do. You can work on the machines at IT STEP, but feel free to bring your laptop if you prefer.

Conclusion

We have seen the motivation and the formalization of a supervised learning problem with a basic example.

Next week, there will be some implementation tasks to do. You can work on the machines at IT STEP, but feel free to bring your laptop if you prefer.

Also, we still have to see more general concepts about optimization, statistical interpretations, regularization.

Thank you! Questions?

alexis.zubiolo@gmail.com

https://github.com/azubiolo/itstep