Mathematics for Machine Learning

Lecture 9
(20.06.2024)

Optimization and Regression

Mahammad Namazov

Table of contents

- Loss Functions
- Linear Regression
- Optimization
- Conclusion

Loss Functions

Idea

MSE

Cross Entropy

Idea

- How to measure the performance of the model?
- We predict a value, but it is different than the **ground truth** value:
$$\hat{\boldsymbol{y}} \neq \boldsymbol{y}$$
- Is Euclidean distance okay for distance computation?
- There are several ways to compute this differences;
- Functions to measure such distances are called Loss (or Objective) Functions;
- What will we see?
  - Mean Squared Error
  - Cross Entropy Loss

Mean Squared Error

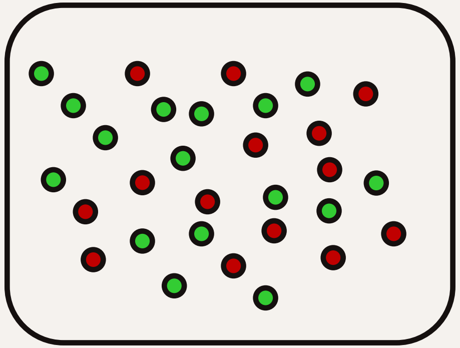
- What does Euclidean Distance do?
- Now we do not compute this distance only for 2 points;
- Since the model tries to generalize, it must see all data;
- Suppose you have n data, and you develop a model to predict something based on your data ($\hat{\boldsymbol{y}}$);
- But these data are not same with the original values ($\boldsymbol{y}$);

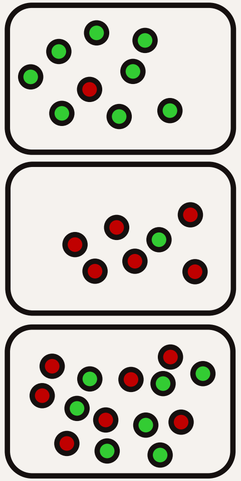
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{y_i} - \hat{\boldsymbol{y}}_{\boldsymbol{i}})^2$$

Average of distances between target and predicted values

Surprise

- Assume you have n data from 2 classes:





$$P(green) = \frac{8}{9}, P(red) = \frac{1}{9}$$

$$P(green) = \frac{1}{7}, P(red) = \frac{6}{7}$$

$$P(green) = \frac{1}{2}, P(red) = \frac{1}{2}$$

Surprise

- Surprise is high, when probability is low.
- Then is it okay to use following equation?

$$Surprise(x) = \frac{1}{p(x)}$$

  - Answer is No!
- Better to use log:

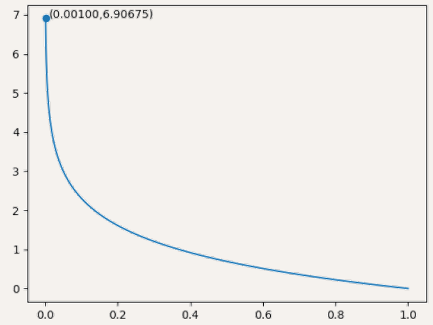
$$Surprise(x) = \log(\frac{1}{p(x)})$$

- Expected Surprise is Entropy:

$$E(Surprise) = \sum p(x) \log(\frac{1}{p(x)}) \;=\; Entropy$$

- Paraphrase:

$$Entropy = \; -\sum p(x) \log(p(x))$$



Cross Entropy Loss

- For one data point where you have C classes:

$$CELoss = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

- Expected Surprise of each class, where probability of occurrence is target and surprise is prediction;
- However, in Machine Learning Applications, you don’t use single data;
- In case you have n data in dataset, **average** loss will be:

$$CELoss = -\frac{1}{n}\sum_{k=1}^{n}\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

Regression

Idea

Linear

Nonlinearity

Idea

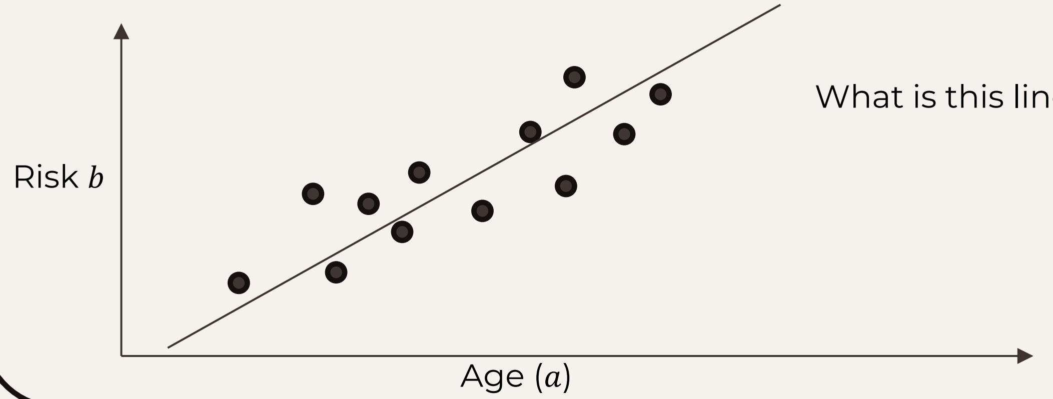
- Suppose you have a dependent variable, and several independent variables;
- These independent variables are utilized to manipulate the behavior of dependent variable;
- In other words: **You have an output which varies with respect to several variables;**
- To see the relation of each variable to generalize the outcome, we use regression:
  - Prediction in Stock markets;
  - Using risk factors as a data to predict disease;

Problem Definition

- You have a person whose age is 68, and heart disease risk rate is 0.6;
- You have another person with age of 72, and with 0.8 of same risk;
- Can we say heart disease risk increases while you get older?
- Assume you have done this experiment on n people;
- This information, will provide generic knowledge for prediction, but not "specific";
- To increase "being specific" rate, we need more parameters!
  - Is person smoker?
  - Did person have heart surgery?
  - ...

Linear Regression

- With one parameter, you have such distribution (**assuming**):



What is this line?

Linear Regression

- Remember: System of Linear equations but in different way!
- For non-square matrices, we have utilized different methodology, but now we will dive deeper;
- What is pseudo-inverse?
  - Pseudo can be understood as “somehow”;
  - To have an inverse of a matrix, matrix **must be** square (or must it be?);
- Remember Singular Value Decomposition?
  - Using SVD, we can take pseudo-inverse of non-square matrices:

$$A^{\dagger} = V\bar{\Sigma}U^{T}$$

Linear Regression

- We have n people in our dataset. Thus,
  - n ages in our dataset
  - n risks in our dataset
- Then mathematically:

$$\boldsymbol{a}x = \boldsymbol{b}$$

- What if you have several parameters, let's say k parameters to guess the risk:

$$A\boldsymbol{x} = \begin{bmatrix} \cdots & \boldsymbol{a_1} & \cdots \\ \vdots & \ddots & \vdots \\ \cdots & \boldsymbol{a_n} & \cdots \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_k \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = \boldsymbol{b}$$

Aim: Find such x that will generalize these changes, so that you will be able to predict risk with same data of unknown person;

Closed Form Solution

- Note: From now on, notation will change (same letters stand for different parameters);
- Assume we have such a problem that, only 2 parameters are enough to solve it (i.e., simple linear equation problem with random parameters a and b):

$$a\boldsymbol{x} + b = \hat{\boldsymbol{y}}$$

- Can we present it with matrix multiplication?

$$\begin{bmatrix} x_1 & 1 \\ \vdots & 1 \\ x_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = X\boldsymbol{\theta} = \hat{\boldsymbol{y}}$$

- Because of the randomness, they don't provide exact line that will fit to our data, where each point has a ground truth value y_i. The difference will be:

$$L_{MSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \|\boldsymbol{y} - \boldsymbol{X\theta}\|^2 = (\boldsymbol{y} - \boldsymbol{X\theta})^T (\boldsymbol{y} - \boldsymbol{X\theta})$$

Gradient

- Following equation computes MSE loss for all possible points:

$$L_{MSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \|\boldsymbol{y} - \boldsymbol{X\theta}\|^2 = (\boldsymbol{y} - \boldsymbol{X\theta})^T (\boldsymbol{y} - \boldsymbol{X\theta})$$

- When we perform multiplication to expand the equation:

$$L_{MSE} = \boldsymbol{y^T y - 2y^T X\theta + \theta^T X^T X\theta}$$

- We want to minimize this loss, or find such parameters that makes this function minimal:

$$\nabla_{\boldsymbol{\theta}} L_{MSE} = \boldsymbol{-2X^T y + 2X^T X\theta = 0}$$

- Closed Form Solution:

$$\boldsymbol{\theta} = \left(\boldsymbol{X^T X}\right)^{\boldsymbol{-1}} \boldsymbol{X^T y} = \boldsymbol{X}^{\dagger} \boldsymbol{y}$$

Regression as Likelihood

- Consider regression as a likelihood problem:

$$p(y|\boldsymbol{x}) = \mathcal{N}(y|f(\boldsymbol{x}), \sigma^2)$$

- $\boldsymbol{x} \in \mathbb{R}^D$ are inputs and $y \in \mathbb{R}$ are noisy function values (targets):

$$y = f(\boldsymbol{x}) + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

- Now think the problem as a parametric model:
  - It takes input ($\boldsymbol{x}$), use it together with parameters ($\boldsymbol{\theta}$) and produces output (y)

$$p(y|\boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{N}(y|\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{\theta}, \sigma^2)$$

- Minimize the negative log-likelihood to get optimal parameters for our model:

$$-\log[\boldsymbol{p}(\boldsymbol{y}|X, \boldsymbol{\theta})] = ?$$

Polynomial Regression

- Now the regression problem that we want to analyze is in the following form:

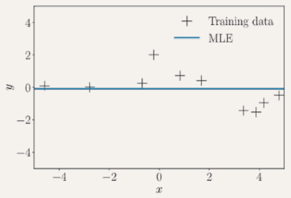
$$y = \phi^T(x)\boldsymbol{\theta} + \epsilon, x \in \mathbb{R}, \boldsymbol{\theta} \in \mathbb{R}^k$$

- Original 1-dimensional input space is lifted into a K-dimensional feature space consisting of all monomials x^k for $k = 0, 1, \dots, K-1$. We can make use of it in linear regression:

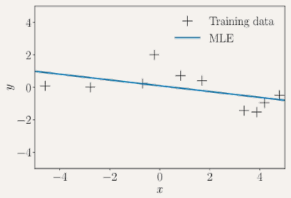
$$f(x) = \sum_{k=0}^{K-1} \theta_k x^k = \phi^T(x)\boldsymbol{\theta} = \phi^T(x)[\theta_1 \quad \theta_2 \quad \cdots \quad \theta_{K-1}]^T$$

- Now let's create feature matrix ($\boldsymbol{\Phi}$) out of this knowledge we have:
  - Training inputs: $x_i \in \mathbb{R}^D, i = 1, 2, \dots, N$
  - Targets: $y_i \in \mathbb{R}, i = 1, 2, \dots, N$
  - $\Phi \coloneqq [\phi(\boldsymbol{x}_1) \quad \phi(\boldsymbol{x}_2) \quad \cdots \quad \phi(\boldsymbol{x}_N)]^T$

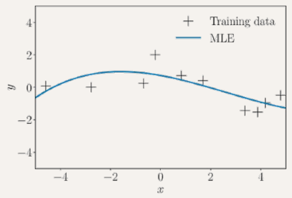
Polynomial Regression



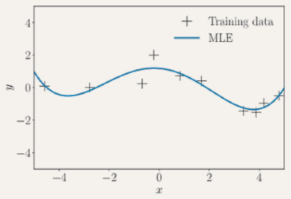
(a) $M = 0$



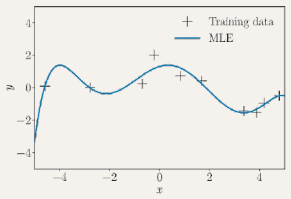
(b) $M = 1$



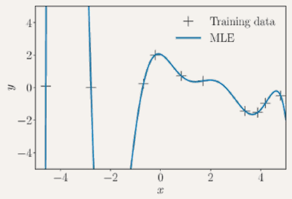
(c) $M = 3$



(d) $M = 4$



(e) $M = 6$



(f) $M = 9$

Optimization

Idea

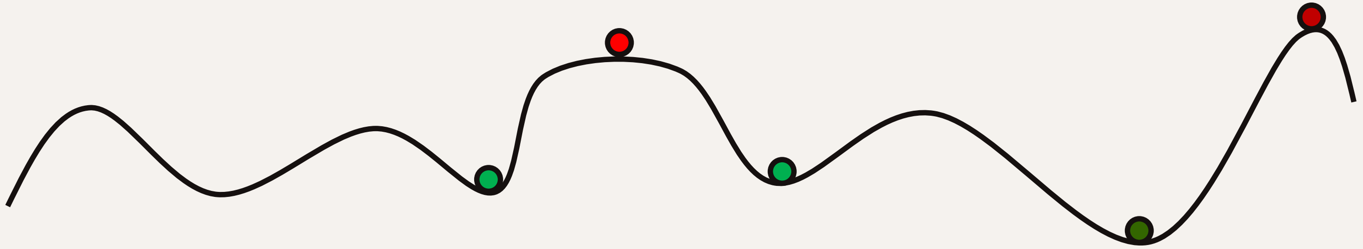
Gradient Descent

SGD

Adam

Idea

- In scenarios, where we cannot find closed form solution, optimizing the function with respect to its parameters;
- Optimization sometimes is maximizing the function’s value, but sometimes to minimize it;
- In most scenarios, we will use it to optimize objective function (i.e., loss function), thus we will need to find specific parameters that will provide us with minimum value;
- We will start with Gradient Descent, continue with Momentum and finish with Stochastic GD;



Gradient Descent

- Assume we have a function f which varies with respect to its arguments $\boldsymbol{x}$:
$$f: \mathbb{R}^n \to \mathbb{R}$$

- Starting from the initial point $\boldsymbol{x_0}$ and move in the negative direction of gradient of the function will lead us to the minimum point:
$$\boldsymbol{x_1} = \boldsymbol{x_0} - \gamma\left[\nabla_{\boldsymbol{x_0}} f\right]^T$$

- For suitable step size (γ) function will converge to its local minima:
$$f(\boldsymbol{x_0}) \geq f(\boldsymbol{x_1}) \geq f(\boldsymbol{x_2}) \geq \cdots \geq f(\boldsymbol{x_n}) \geq \cdots$$

- Notice that, while we get closer to the minimum, gradient descent will be slower;

Step Size

- Or Learning Rate in Deep Learning methods;
- Choice is significant, because:
  - Too large → overshoot → divergence
  - Too small → too slow → ages to converge
- Adaptive Gradient methods to rescale the step size at each iteration depending on local properties of function:
  - Function value increases → the step size is too large → get back → decrease -> repeat;
  - Function value decreases, but the step could have been bigger → increase it a little bit → let it go;

Momentum

- Curvature might impact the convergence speed;
- Possible solution: Let's give the Gradient Descent a memory, so that it will remember what happened at the previous step;
- HOW?
  - Gradient Descent with Momentum, where momentum will be new parameter that will play a memory role for GD:

$$\boldsymbol{x_{i+1}} = \boldsymbol{x_i} - \gamma_i \left(\nabla_{\boldsymbol{x_i}} f\right)^T + \alpha \Delta \boldsymbol{x_i}$$
$$\Delta \boldsymbol{x_i} = \boldsymbol{x_i} - \boldsymbol{x_{i-1}}$$

  - Notice that α must be in the range of $[0, 1]$;
  - Memory as a moving average, which lets the gradient remember the update $\Delta \boldsymbol{x_i}$ at each iteration and determines the next update as a linear combination of steps;

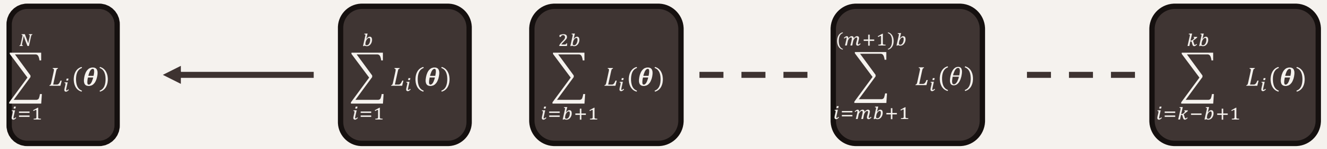
Stochastic Gradient Descent

- Once number of data increases, computation of gradient descent gets more complex;
- Let's analyze in a bit granular way:
  - When there are N data points, then loss function (average or not) will be computed as below (Note: Which loss?):

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{N} L_i(\boldsymbol{\theta})$$

- Now assume we separate the data into several batches with equal number of data points. Let's say we have k batches where each batch has b data points:

$$N = k \times b$$


$\sum_{i=1}^{N} L_i(\boldsymbol{\theta})$
$\sum_{i=1}^{b} L_i(\boldsymbol{\theta})$
$\sum_{i=b+1}^{2b} L_i(\boldsymbol{\theta})$
$\sum_{i=mb+1}^{(m+1)b} L_i(\theta)$
$\sum_{i=k-b+1}^{kb} L_i(\boldsymbol{\theta})$

Conclusion

Summary

Takeaways

References

Summary

- Objective function is used to compute the difference (i.e., distance) between prediction and target values;
- There are several Objective functions that are used in ML / DL applications (e.g., MSE, LSE, Cross-Entropy Loss, KL);
- If there is closed form solution, you can find it with several steps, such as loss definition, derivative and solve the problem;
- If there is not closed form solution, you need to apply parametric optimization, to find global minima of objective function that is defined for the problem;
- Gradient Descent is an optimization function that makes use of the gradient of the function and moves along the negative direction of this gradient;
- Curvature of the function impacts the convergence, thus memory might solve this issue;
- Stochasticity will be solution for time consumption;

Takeaways

- Linear Regression is one of the fundamental machine learning algorithms, to see how ”core” algorithm works;
- Gradient Descent is ”somehow” initial point of optimization algorithms in ML applications;
- Stochasticity brings generalization along with noise; however it is helpful for various reasons;
- There are several optimization functions can be used in the field, and you need to determine the one you will need;
- Learning rate or Step size is a hyperparameter that can be tuned by us, that will determine the model’s performance;

References

- Further information to read:
  - Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2020). *Mathematics for machine learning*. Cambridge University Press.
    - Chapter 7, Section 1: Optimization
    - Chapter 9, Section 1, 2: Linear Regression

The End

Thanks for your attention!

Mahammad Namazov