Theoretical Minimum for Machine Learning

Bernhard Huang

Correlaid Muenchen

23 Mar, 2022

Overview

High Level Overview

- What is the goal of this workshop?
  - Prepare you adequately with the lay of the land so that you can know what to Google
  - Discuss the bare minimum machine learning theory you need to know to get started

Great Non-Wikipedia Resources

- K. P. Murphy, Machine Learning: A Probabilistic Perspective
- C. M. Bishop, Pattern Recognition and Machine Learning
- S. Shalev-Shwartz, S. Ben-David, Understanding Machine Learning

Typical Problem

- Input: x
- Output: y
- Typical "supervised learning" problem: find an "optimal mapping" $f : x \rightarrow y$ given samples of x and y
  - Note "unsupervised learning" corresponds to the special case $y = x$
- Can you think of some examples for x and y and maybe f?

Regression/Classification

- There are two classes of problems depending on the values y can take on
  - $y \in R$: regression!
    - My favorite example: x is the number of years of quality education, y is annual earnings
  - $y \in N$: classification!
    - My favorite example: x is an image of a pet, y is the classification label (cat versus dog)

Recipe

- Regardless, the basic recipe for a typical problem is given by:
  1. Define the problem
  2. Collect the data (training data - samples with both x_i and y_i)
  3. Choose a representation of the data
  4. Model the problem by choosing a hypothesis class (i.e. set of possible mappings from x to y)
  5. Find the "best" model within the chosen hypothesis class
  6. If there are multiple hypothesis classes, then choose one based on some criteria
- For this workshop, we assume that we have the data in matrix form $X = [x_1, x_2, ..., x_N]$ and $Y = [y_1, y_2, ..., y_N]$ (e.g. x_i and y_i are the ith samples of x and y respectively) and implicitly assume that we compute relevant quantities from samples instead of the true population

Loss Function

- Even if you have a mapping f, how do you evaluate how "good" it is?
  - You need something called a loss function $l(\hat{y}, y)$, comparing the difference between the prediction and the target outputs, e.g. squared loss and cross entropy loss (the nuance here is that y and $\hat{y}$ are probabilities)

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

$$l(\hat{y}, y) = -y log \hat{y}$$

- Typically, we consider a parametric function $f(x; w)$
  - Linear function: $f(x; w) = w \cdot x$
  - Generalization: $f(x; w) = w \cdot \phi(x)$ (different choices of $\phi(\cdot)$ correspond to different feature spaces)

Optimization Problem

- Empirical loss:

$$L = \frac{1}{N} \sum_i l(f(x; w), y)$$

- To control overfitting, a penalty or regularization term is often added so the typical optimization problem

$$min_w L(w; X, Y) + R(w)$$

  - Examples of $R(w)$:
    - Lasso regression: $\lambda \sum_j |w_j|$
    - Ridge regression: $\lambda \sum_j w_j^2$

Optimization Problem

- Empirical loss:

$$L = \frac{1}{N} \sum_i l(f(x; w), y)$$

- To control overfitting, a penalty or regularization term is often added so the typical optimization problem looks like

$$min_w L(w; X, Y) + R(w)$$

  - Examples of $R(w)$:
    - Lasso: $\lambda \sum_j |w_j|$
    - Ridge regression: $\lambda \sum_j w_j^2$

Optimization Technique

- With the exception of certain "easy" classes of problems (e.g. convex optimization), there is often no guarantee for a global minimum or convergence to such a minimum
- Thus, the typical approach is called (stochastic) gradient descent (SGD)
- Basic SGD recipe:
  - Start with $t = 0$
  - Initialize $w^{(t)}$ (either 0 or some small random vector)
  - For $t = 1, ...,$ compute gradient and update model (η is known as the learning rate):

$$g^{(t)} = \nabla L(X, Y; w^{(t-1)})$$

$$w^{(t)} = w^{(t-1)} - \eta \cdot g^{(t)}$$

  - Stop when some convergence criterion is reached

Questions?

Exercises

As a simple example for classification we consider a logistic regression case, where we want to predict if a city belongs to bavaria or not. To make things easier we only consider their lat and lng coordinates. We can simplify the cross entropy loss to binary cross entropy loss which looks like the following:

$$l(\hat{y}_i, y_i) = -(y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i))$$

For our parametric function we choose the so-called sigmoid function:

$$\hat{y}_i = f(z_i) = \frac{1}{1 + e^{-z_i}}$$

where $z_i = w_0 + w_1 \cdot x_{i,lat} + w_2 \cdot x_{i,lng}$ (w_0 is a bias term).

Exercises

Take a look at the prepared jupyter notebook.

1. Compute the binary cross entropy loss for every sample using the parameter $w = [1, 0.5, 0.5]$.
2. Compute the gradient of the empirical loss with the binary cross entropy loss function with regards to the weight vector w.
3. Execute a few iterations of gradient descent with learning rate $\eta = 0.001$
4. How did the empirical loss change compared to the beginning and what is your achieved accuracy on your training data?