Optimization and Machine Learning

Chih-Jen Lin
Department of Computer Science
National Taiwan University

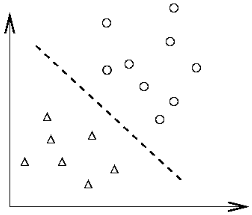
Talk at TWSIAM Annual Meeting, July 24, 2020

Outline

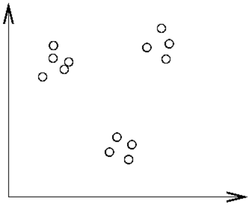
Outline

What is Machine Learning

- Extract knowledge from data
- Representative tasks: classification, clustering, and others



Classification



Clustering

- Today we will focus on classification

Data Classification

- Given training data in different classes (labels known)

  Predict test data (labels unknown)
- Classic example
  1. Find a patient's blood pressure, weight, etc.
  2. After several years, know if he/she recovers
  3. Build a machine learning model
  4. New patient: find blood pressure, weight, etc
  5. Prediction
- Two main stages: training and testing

Why Is Optimization Used?

- Usually the goal of classification is to

  minimize the number of errors
- Therefore, many classification methods solve optimization problems
- We will discuss a topic called empirical risk minimization that can connect many classification methods

Outline

Minimizing Training Errors

- Basically a classification method starts with minimizing the training errors

$$\min_{\text{model}} \quad \text{(training errors)}$$

- That is, all or most training data with labels should be correctly classified by our model
- A model can be a decision tree, a neural network, etc.

Minimizing Training Errors (Cont'd)

- For simplicity, let's consider the model to be a vector $\boldsymbol{w}$
- That is, the decision function is

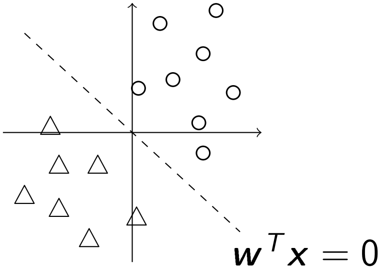
$$\text{sgn}(\boldsymbol{w}^T\boldsymbol{x})$$

- For any data, $\boldsymbol{x}$, the predicted label is

$$\begin{cases} 1 & \text{if } \boldsymbol{w}^T\boldsymbol{x} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Minimizing Training Errors (Cont'd)

- The two-dimensional situation



- This seems to be quite restricted, but practically $\boldsymbol{x}$ is in a much higher dimensional space

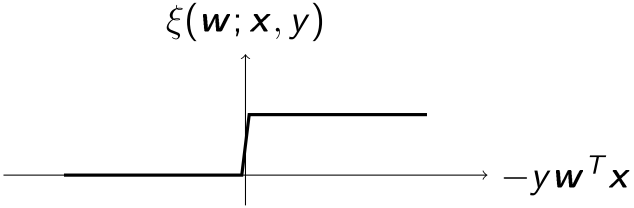
Minimizing Training Errors (Cont'd)

- To characterize the training error, we need a loss function $\xi(\boldsymbol{w}; \boldsymbol{x}, y)$ for each instance $(\boldsymbol{x}, y)$
- Ideally we should use 0–1 training loss:

$$\xi(\boldsymbol{w}; \boldsymbol{x}, y) = \begin{cases} 1 & \text{if } y\boldsymbol{w}^T\boldsymbol{x} < 0, \\ 0 & \text{otherwise} \end{cases}$$

Minimizing Training Errors (Cont'd)

- However, this function is discontinuous. The optimization problem becomes difficult



- We can do continuous approximations

Common Loss Functions

- Hinge loss (l1 loss)

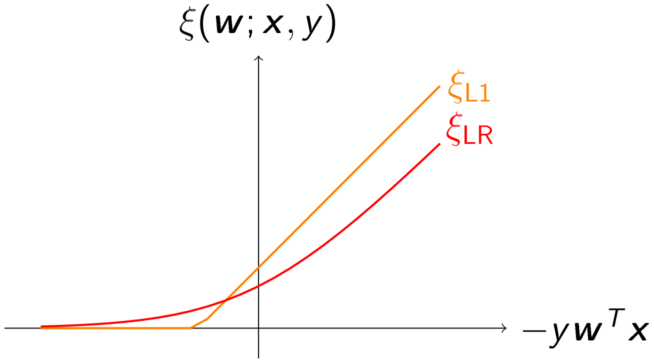
$$\xi_{\text{L1}}(\boldsymbol{w}; \boldsymbol{x}, y) \equiv \max(0, 1 - y\boldsymbol{w}^T\boldsymbol{x}) \qquad (1)$$

- Logistic loss

$$\xi_{\text{LR}}(\boldsymbol{w}; \boldsymbol{x}, y) \equiv \log(1 + e^{-y\boldsymbol{w}^T\boldsymbol{x}}) \qquad (2)$$

- Support vector machines (SVM): Eq. (1). Logistic regression (LR): (2)
- SVM and LR are two very fundamental classification methods

Common Loss Functions (Cont'd)



- Logistic regression is very related to SVM
- Their performance is usually similar

Common Loss Functions (Cont'd)

- However, minimizing training losses may not give a good model for future prediction
- Overfitting occurs

Overfitting

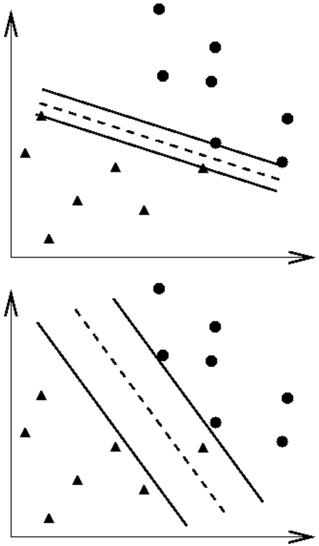
- See the illustration in the next slide
- For classification,

  You can easily achieve 100% training accuracy
- This is useless
- When training a data set, we should

  Avoid underfitting: small training error

  Avoid overfitting: small testing error

● and ▲: training; ○ and △: testing



Regularization

- To minimize the training error we manipulate the $\boldsymbol{w}$ vector so that it fits the data
- To avoid overfitting we need a way to make $\boldsymbol{w}$'s values less extreme.
- One idea is to make $\boldsymbol{w}$ values closer to zero
- We can add, for example,

$$\frac{\boldsymbol{w}^T\boldsymbol{w}}{2} \quad \text{or} \quad \|\boldsymbol{w}\|_1$$

to the function that is minimized

General Form of Linear Classification

- Training data $\{y_i, \boldsymbol{x}_i\}, \boldsymbol{x}_i \in R^n, i = 1, \ldots, l,\ y_i = \pm 1$
- l: # of data, n: # of features

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}), \quad f(\boldsymbol{w}) \equiv \frac{\boldsymbol{w}^T\boldsymbol{w}}{2} + C\sum_{i=1}^{l} \xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i)$$

- $\boldsymbol{w}^T\boldsymbol{w}/2$: regularization term
- $\xi(\boldsymbol{w}; \boldsymbol{x}, y)$: loss function
- C: regularization parameter (chosen by users)

Neural Networks

- We all know that recently deep learning (i.e., deep neural networks) is very hot.
- We will explain neural networks using the the same empirical risk minimization framework
- Among various types of networks, we consider

  fully-connected feed-forward networks

  for

  multi-class classification

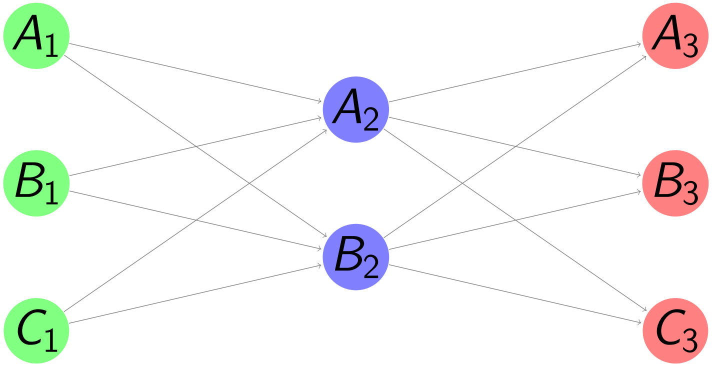
Neural Networks (Cont'd)

- Our training set includes $(\boldsymbol{y}_i, \boldsymbol{x}_i)$, $i = 1, \ldots, l$.
- $\boldsymbol{x}_i \in R^{n_1}$ is the feature vector.
- $\boldsymbol{y}_i \in R^K$ is the label vector.
- K: $\#$ of classes
- If $\boldsymbol{x}_i$ is in class k, then

$$\boldsymbol{y}_i = [\underbrace{0, \ldots, 0}_{k-1}, 1, 0, \ldots, 0]^T \in R^K$$

Neural Networks (Cont'd)

- A neural network maps each feature vector to one of the class labels by the connection of nodes
- Between two layers a weight matrix maps input to output



Neural Networks (Cont'd)

- The weight matrix W^m at the mth layer is

$$W^m = \begin{bmatrix} w_{11}^m & w_{12}^m & \cdots & w_{1n_m}^m \\ w_{21}^m & w_{22}^m & \cdots & w_{2n_m}^m \\ \vdots & \vdots & \vdots & \vdots \\ w_{n_{m+1}1}^m & w_{n_{m+1}2}^m & \cdots & w_{n_{m+1}n_m}^m \end{bmatrix}_{n_{m+1} \times n_m}$$

- n_m : # input features at layer m
- n_{m+1} : # output features at layer m, or # input features at layer $m+1$
- L: number of layers
- $n_1 =$ # of features, $n_{L+1} =$ # of classes

Neural Networks (Cont'd)

Let $\boldsymbol{z}^m$ be the input of the mth layer, $\boldsymbol{z}^1 = \boldsymbol{x}$ and $\boldsymbol{z}^{L+1}$ be the output

From mth layer to $(m+1)$th layer

$$\begin{aligned} \boldsymbol{s}^m &= W^m \boldsymbol{z}^m, \\ z_j^{m+1} &= \sigma(s_j^m),\ j = 1, \ldots, n_{m+1}, \end{aligned}$$

$\sigma(\cdot)$ is the activation function. We collect all variables:

$$\boldsymbol{\theta} = \begin{bmatrix} \text{vec}(W^1) \\ \vdots \\ \text{vec}(W^L) \end{bmatrix} \in R^n \quad \begin{aligned} n &: \text{total } \# \text{ variables} \\ &= (n_1+1)n_2 + \cdots + (n_L+1)n_{L+1} \end{aligned}$$

Neural Networks (Cont'd)

- We solve the following optimization problem,

  $$\min_{\boldsymbol{\theta}} \quad f(\boldsymbol{\theta}),$$

  where

  $$f(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\theta} + C\sum_{i=1}^{l} \xi(\boldsymbol{z}^{L+1,i}(\boldsymbol{\theta}); \boldsymbol{x}_i, \boldsymbol{y}_i).$$

  C: regularization parameter
- $\boldsymbol{z}^{L+1}(\boldsymbol{\theta}) \in R^{n_{L+1}}$: last-layer output vector of $\boldsymbol{x}$.
- $\xi(\boldsymbol{z}^{L+1}; \boldsymbol{x}, \boldsymbol{y})$: loss function. Example:

  $$\xi(\boldsymbol{z}^{L+1}; \boldsymbol{x}, \boldsymbol{y}) = ||\boldsymbol{z}^{L+1} - \boldsymbol{y}||^2$$

Neural Networks (Cont'd)

- The formulation is as before, but loss function is more complicated
- Note that we discussed the simplest type of networks
- Nowadays people use much more complicated networks
- The optimization problem is non-convex

Discussion

- We have seen that many classification methods are under the empirical risk minimization framework
- We also see that optimization problems must be solved

Outline

Optimization Techniques for Machine Learning

- Standard optimization packages may be directly applied to machine learning applications
- However, efficiency and scalability are issues
- Many optimization researchers want to do machine learning
- Some are more successful, but some are not
- Very often properties from machine learning side must be considered
- I will illustrate this point by some examples

Differences between Optimization and Machine Learning

- The two topics may have different focuses. We give the following example
- Recall that the optimization problem for empirical risk minimization is

$$\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C(\text{sum of training losses})$$

- A large C means to fit training data
- The optimization problem becomes more difficult

- In contrast, if $C \to 0$,

$$\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$$

  is easy
- Optimization researchers may rush to solve difficult cases of large C
- It turns out that C should not be too large
- A large C causes severe overfitting and bad accuracy
- Thus knowing what is useful and what is not on the machine learning side is very important

Stochastic Gradient for Deep Learning

- In optimization, gradient descent is a basic method
- But it has slow convergence
- So in many application domains higher-order optimization methods (e.g., Newton, quasi Newton) were developed for faster convergence
- However, in deep learning people use an even lower-order method: stochastic gradient
- Why?

Estimation of the Gradient

- Let us rewrite the objective function as

$$f(\boldsymbol{\theta}) = \frac{1}{2C}\boldsymbol{\theta}^T\boldsymbol{\theta} + \frac{1}{l}\sum\nolimits_{i=1}^{l} \xi(\boldsymbol{z}^{L+1,i}(\boldsymbol{\theta}); \boldsymbol{x}_i, \boldsymbol{y}_i)$$

- The gradient is

$$\frac{\boldsymbol{\theta}}{C} + \frac{1}{l}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{l} \xi(\boldsymbol{z}^{L+1,i}(\boldsymbol{\theta}); \boldsymbol{x}_i, \boldsymbol{y}_i)$$

- Going over all data is time consuming

Estimation of the Gradient (Cont'd)

- We may use a subset S of data

$$\frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|}\nabla_{\boldsymbol{\theta}} \sum_{i:i\in S} \xi(\boldsymbol{z}^{L+1,i}(\boldsymbol{\theta}); \boldsymbol{x}_i, \boldsymbol{y}_i)$$

- This works if data points are under the same distribution

$$E_{\boldsymbol{y},\boldsymbol{x}}(\nabla_{\boldsymbol{\theta}}\xi(\boldsymbol{z}^{L+1,i}; \boldsymbol{x}, \boldsymbol{y})) = \frac{1}{l}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{l} \xi(\boldsymbol{z}^{L+1,i}(\boldsymbol{\theta}); \boldsymbol{x}_i, \boldsymbol{y}_i)$$

Stochastic Gradient Algorithm

```
1: Given an initial learning rate $\eta$.
2: while do
3:     Choose $S \subset \{1, \ldots, l\}$.
4:     Calculate
       $$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\left(\frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|}\nabla_{\boldsymbol{\theta}} \sum_{i:i\in S} \xi(z^{L+1,i}(\boldsymbol{\theta}); \boldsymbol{x}_i, \boldsymbol{y}_i)\right)$$
5:     May adjust the learning rate $\eta$
6: end while
```

Issues of Stochastic Gradient Algorithm

- People often use the name SGD (stochastic gradient descent) but it is not a descent algorithm

  Note that we didn't (and cannot) do things like line search to ensure the function-value decrease
- It's known that deciding a suitable learning rate is difficult
  - Too small learning rate: very slow convergence
  - Too large learning rate: the procedure may diverge
- Despite such drawbacks, SG is widely used in deep learning. Why?

Why Stochastic Gradient Widely Used? I

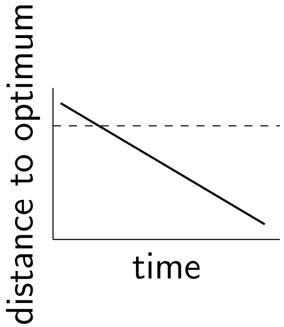
- In machine learning fast final convergence may not be important
  - An optimal solution $\boldsymbol{\theta}^*$ may not lead to the best model
  - Further, we don't need a point close to $\boldsymbol{\theta}^*$. In prediction we find

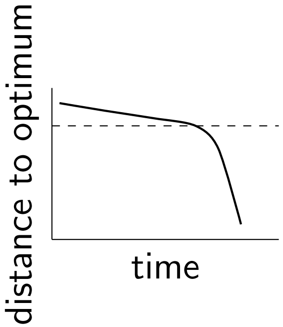
$$\arg\max_k z_k^{L+1}(\boldsymbol{\theta})$$

A not-so-accurate $\boldsymbol{\theta}$ may be good enough

An illustration

Why Stochastic Gradient Widely Used? II





Slow final convergence   Fast final convergence

Why Stochastic Gradient Widely Used? III

- The special property of data classification is essential

$$E(\nabla_{\boldsymbol{\theta}}\xi(\boldsymbol{z}^{L+1};\boldsymbol{x},\boldsymbol{y})) = \frac{1}{l}\nabla_{\boldsymbol{\theta}}\sum_{i=1}^{l}\xi(\boldsymbol{z}^{L+1,i}(\boldsymbol{\theta});\boldsymbol{x}_i,\boldsymbol{y}_i)$$

  We can cheaply get a good approximation of the gradient

  Indeed stochastic gradient is less used outside machine learning

Why Stochastic Gradient Widely Used? IV

- Easy implementation. It's simpler than methods using, for example, second derivative

  Now for complicated networks, (subsampled) gradient is calculated by automatic differentiation
- Non-convexity plays a role
  - For convex, other methods may possess advantages to more efficiently find the global minimum
  - But for non-convex, efficiency to reach a stationary point is less useful

Why Stochastic Gradient Widely Used? V

  - A global minimum usually gives a good model (as loss is minimized), but for a stationary point we are less sure
- Some variants of SG have been proposed to improve the robustness or the convergence
- All these explain why SG is popular for deep learning

Subsampled 2nd-order Method

- Recall for stochastic gradient method, we use

$$E(\nabla_{\boldsymbol{\theta}} \xi(\boldsymbol{z}^{L+1}; \boldsymbol{x}, \boldsymbol{y})) = \frac{1}{l} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{l} \xi(\boldsymbol{z}^{L+1,i}(\boldsymbol{\theta}); \boldsymbol{x}_i, \boldsymbol{y}_i)$$

- Can we extend this idea to 2nd derivative? Yes, Byrd et al. (2011); Martens (2010)

$$E(\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \xi(\boldsymbol{z}^{L+1}; \boldsymbol{y}, \boldsymbol{x})) = \frac{1}{l} \nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \sum_{i=1}^{l} \xi(\boldsymbol{z}^{L+1}; \boldsymbol{y}_i, \boldsymbol{x}_i).$$

Subsampled 2nd-order Method (Cont'd)

- We can consider

$$\frac{1}{|S|}\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}}\sum_{i\in S}\xi(\boldsymbol{z}^{L+1};\boldsymbol{y}_i,\boldsymbol{x}_i).$$

  in designing subsampled Newton or quasi Newton methods

Outline

Conclusions

- Many machine learning methods involve optimization problems
- However, designing useful optimization techniques for these applications may not be easy
- Incorporating machine learning knowledge is very essential