Week 3: Gradient Descent

Yunwen Lei

School of Computer Science, University of Birmingham

In this week, we will study linear regression and gradient descent. First, we will give a basic formulation of linear regression problems. Then we will show how to solve the linear regression problems by a closed-form solution. This is based on the fundamental concept of gradients. Therefore, we recall basic concepts of derivative and gradient. We also show how the simple linear regression method can be extended naturally to nonlinear regression methods such as polynomial regression.

1 Gradient Descent

1.1 Visualization of the Gradient

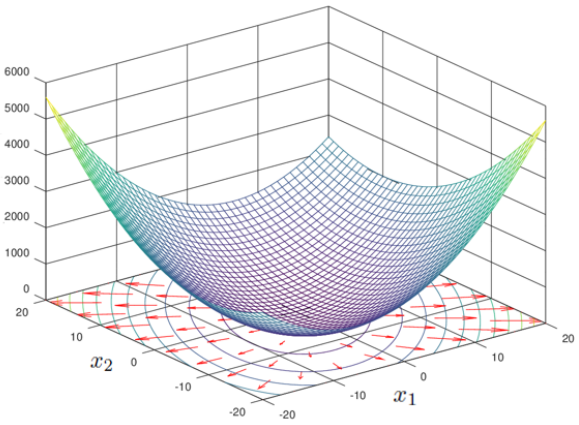
The gradient vector at a point $\mathbf{x}$ points in the direction of greatest increase of the function f: each element of the gradient shows how fast $f(\mathbf{x})$ is changing w.r.t. each of the coordinate axes.

Example: Consider $f(x_1, x_2) = 6x_1^2 + 4x_2^2 - 4x_1x_2$

The gradient vector is

$$\nabla f(x_1, x_2) = \begin{pmatrix} 12x_1 - 4x_2 \\ 8x_2 - 4x_1 \end{pmatrix}$$

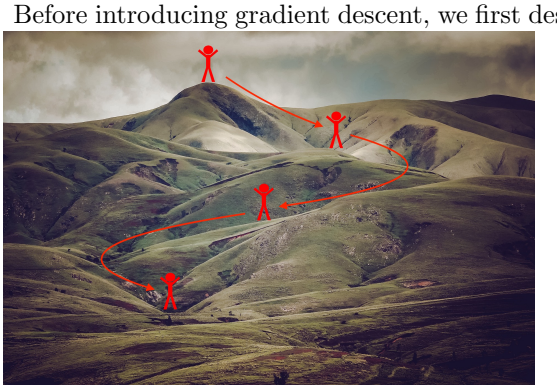
Right: the graph of the function and its gradient vector field



- The red arrow in the bottom indicates the gradient at each point. As you can see, all these red arrows point toward out. These are the directions where function value increase.
- The length of the red arrows indicates the magnitude of the gradients. In the middle, the gradients are small. As points move from inside to outside, the gradients are becoming larger and larger. We can interpret the gradient as the change rate of the function value, therefore the function value changes slowly in the middle and changes fast outside.

1.2 Optimization

Before introducing gradient descent, we first describe the general scheme of optimization as follows.



Minimizing the cost is like finding the lowest point in a hilly landscape

Template Optimization Algorithm

- Start with a point $\mathbf{w}$ (initial guess)
- Find a direction $\mathbf{d}$ to move on, and determine how far (η) to move along $\mathbf{d}$
- Gets updated $\mathbf{w} \leftarrow \mathbf{w} + \eta\mathbf{d}$

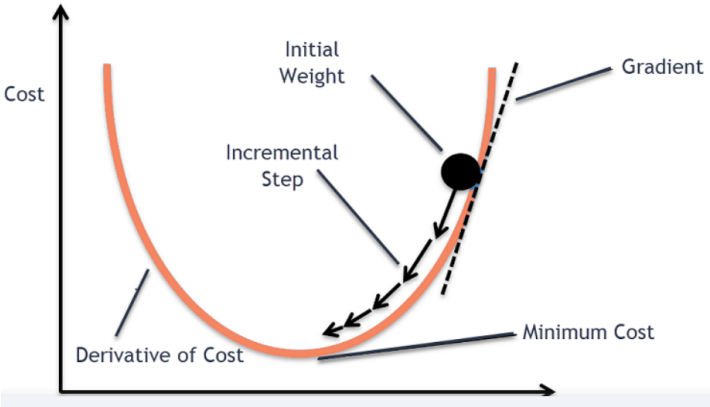
This process is very similar to downhill. In the beginning, we are at a high place. We can find a direction to move on according to the local landscape of the hill. We move along that direction with a distance and arrive at a lower location. At the new location, we find a new direction and move along that direction to another location. We repeat this process again and again until we successfully arrive at a very low point.

1.3 Gradient Descent

Gradient descent is one of the simplest, but very general algorithm for minimizing an objective function $C(\mathbf{w})$ (first proposed by Cauchy in 1847). It is an iterative algorithm, starting from $\mathbf{w}^{(0)}$ and producing a new $\mathbf{w}^{(t+1)}$ at each iteration as

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta_t \nabla C(\mathbf{w}^{(t)}), \quad t = 0, 1, \ldots,$$

where $\eta_t > 0$ is the learning rate or step size. Therefore, GD uses negative gradient as a search direction and move along this direction with step size η_t.



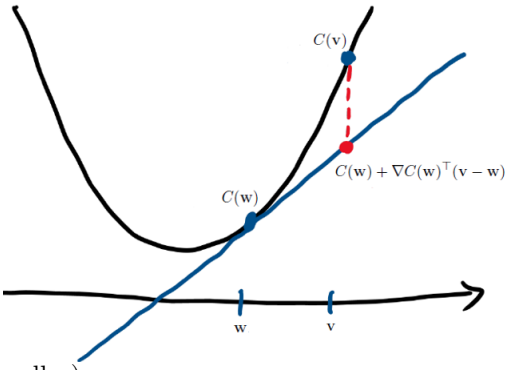
Here we see that the essential part in implementing gradient descent is to compute the gradient. Once we know how to compute the gradient, it is trivial to update the iterate along the negative gradient direction. Fortunately, the gradient computation is easy in some case. As we show in the previous discussions, the gradient computation has a closed-form solution for linear regression. For some other problems like neural networks, while we do not have a closed-form solution for gradient, the gradient can be computed by some algorithms (backpropagation).

Remark 1 (Why Gradient Descent Works?)**.**

Recall we can use gradient to get a tangent as approximation of C

$$C(\mathbf{v}) \approx \underbrace{C(\mathbf{w}) + \nabla C(\mathbf{w})^\top (\mathbf{v} - \mathbf{w})}_{\text{linear function of } \mathbf{v}} \tag{1.1}$$

The red dashed line shows the gap between C at $\mathbf{v}$ and the linear function of $\mathbf{v}$. This gap is small if $\mathbf{v}$ is close to $\mathbf{w}$, the point where the gradient information is used in the linear approximation.



We can take $\mathbf{v} = \mathbf{w} - \eta \nabla C(\mathbf{w})$ in Eq (1.1) and get (for small η)

$$\begin{aligned} C(\mathbf{v}) &\approx C(\mathbf{w}) + \nabla C(\mathbf{w})^\top (\underbrace{\mathbf{w} - \eta \nabla C(\mathbf{w})}_{\mathbf{v}} - \mathbf{w}) \\ &= C(\mathbf{w}) + \nabla C(\mathbf{w})^\top (-\eta \nabla C(\mathbf{w})) \\ &= C(\mathbf{w}) - \eta \underbrace{\nabla C(\mathbf{w})^\top \nabla C(\mathbf{w})}_{= \|\nabla C(\mathbf{w})\|_2^2 > 0} < C(\mathbf{w}). \end{aligned}$$

GD always decreases function values if choosing appropriate learning rates (under some regularity conditions)!

Remark 2 (Another Intuition of GD)**.**

- By Eq. (1.1), if $\mathbf{w}$ is close to $\mathbf{w}^{(t)}$ then

$$C(\mathbf{w}) \approx C(\mathbf{w}^{(t)}) + (\mathbf{w} - \mathbf{w}^{(t)})^\top \nabla C(\mathbf{w}^{(t)})$$

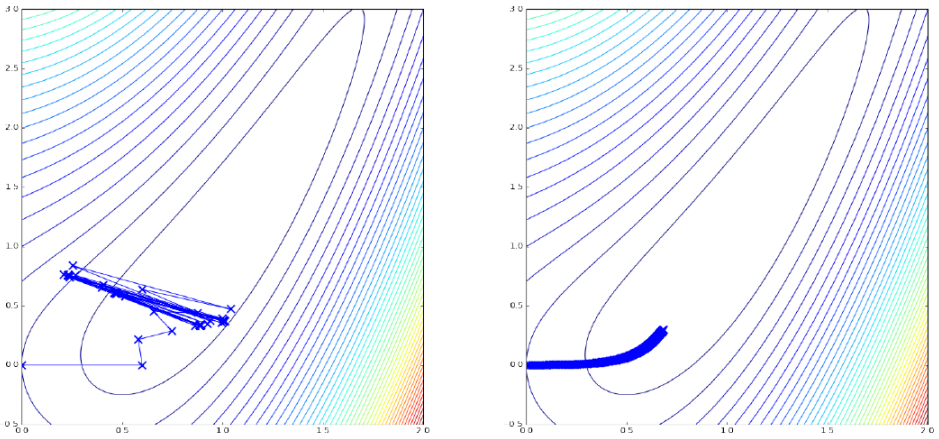
- Hence, we want to minimize the approximation while staying close to $\mathbf{w}^{(t)}$

$$\mathbf{w}^{(t+1)} = \arg\min_{\mathbf{w}} \frac{1}{2\eta_t} \|\mathbf{w} - \mathbf{w}^{(t)}\|^2 + \Big(C(\mathbf{w}^{(t)}) + (\mathbf{w} - \mathbf{w}^{(t)})^\top \nabla C(\mathbf{w}^{(t)}) \Big).$$

  As an exercise, show that the above update is exactly GD! This equivalence shows that by doing GD, we are actually minimizing a quadratic function at each step. Notice this quadratic function can be considered as a simplification (approximation) of the original objective function C, which may be complex. The quadratic function has two part, where the first part is a first-order approximation of C by using the gradient information $\nabla C(\mathbf{w}^{(t)})$. Since this first-order approximation is accurately locally, we add the quadratic stabilizer to make sure that $\mathbf{w}^{(t+1)}$ not far away from $\mathbf{w}^{(t)}$.

Remark 3 (Choosing a Step Size)**.** Choosing a good step-size is very important to the performance of gradient descent.

- It the step size is too large, algorithm may never converge. The underlying reason is that the gradient is a local information. This local information would not be precise if we move with a large step size.
- If step size is too small, convergence may be very slow. The underlying reason is that at each iteration we make little update in this case.



Example 1 (Gradient Descent for Least Square Regression)**.** To illustrate how to implement gradient descent, we give an example to linear regression (least square regression).

- For least square regression, recall

$$C(\mathbf{w}) = \frac{1}{2n}\Big(\underbrace{\mathbf{w}^\top X^\top X\mathbf{w}}_{\text{quadratic}} - 2\underbrace{\mathbf{w}^\top X^\top \mathbf{y}}_{\text{linear}} + \underbrace{\mathbf{y}^\top \mathbf{y}}_{\text{constant}}\Big)$$

- The gradient is computed in the previous lecture by

$$\nabla C(\mathbf{w}) = \frac{1}{n}\Big(X^\top X\mathbf{w} - X^\top \mathbf{y}\Big).$$

- GD updates $\mathbf{w}^{(t)}$ by

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta\nabla C(\mathbf{w}^{(t)}) = \mathbf{w}^{(t)} - \frac{\eta}{n}\Big(X^\top X\mathbf{w}^{(t)} - X^\top \mathbf{y}\Big).$$

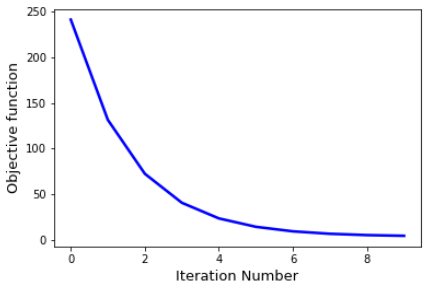
It is clear that the gradient descent is very simple to implement in python. We just need some matrix operation.

Example 2 (Back to Toy Example (Commute Time))**.** We now apply the above gradient descent for least square regression to the problem of commute time. For this problem, we have

$$\mathbf{y} = \begin{pmatrix} 25 \\ 33 \\ 15 \\ 45 \\ 22 \end{pmatrix}, \quad X = \begin{pmatrix} 1 & 2.7 & 1 \\ 1 & 4.1 & 1 \\ 1 & 1.0 & 0 \\ 1 & 5.2 & 1 \\ 1 & 2.8 & 0 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$$

If we run GD with $\mathbf{w}^{(1)} = (0, 0, 0)^\top$ and $\eta = 0.02$, then we get (python implementation as an exercise)

```
the grad of 1-th iteration is [ -28.   -102.68  -20.6 ]
the weight of 1-th iteration is [0.56   2.0536 0.412 ]
the grad of 2-th iteration is [-20.703424  -75.2866144 -15.08816  ]
the weight of 2-th iteration is [0.97406848 3.55933229 0.7137632 ]
the grad of 3-th iteration is [-15.35018357 -55.1911618  -11.0449035 ]
the weight of 3-th iteration is [1.28107215 4.66315552 0.93466127]
the grad of 4-th iteration is [-11.42255963 -40.44941129  -8.07898669]
the weight of 4-th iteration is [1.50952334 5.47214375 1.096241  ]
the grad of 5-th iteration is [ -8.5407578  -29.6350914   -5.90339639]
the weight of 5-th iteration is [1.6803385  6.06484558 1.21430893]
the grad of 6-th iteration is [ -6.42616412 -21.70190135  -4.30758215]
the weight of 6-th iteration is [1.80886178 6.4988836  1.30046057]
the grad of 7-th iteration is [ -4.87438968 -15.88228366  -3.13708593]
the weight of 7-th iteration is [1.90634958 6.81652928 1.36320229]
the grad of 8-th iteration is [ -3.73549653 -11.61316462  -2.27859861]
the weight of 8-th iteration is [1.98105951 7.04879257 1.40877427]
the grad of 9-th iteration is [-2.89949141 -8.48147805 -1.64899757]
the weight of 9-th iteration is [2.03904933 7.21842213 1.44175422]
the grad of 10-th iteration is [-2.2856842  -6.18420209 -1.18730475]
the weight of 10-th iteration is [2.08476302 7.34210617 1.46550031]
```



According to the above figure, we see that gradient descent solves this toy problem well!

2 Stochastic Gradient Descent

Gradient descent is easy to implement since at each iteration one only needs to compute the gradient. This gradient computation has a closed-form solution for some problems, e.g., linear regression. However, gradient computation can be expensive if n is large since it requires to go through all training examples to compute the gradient

$$\nabla C(\mathbf{w}^{(t)}) = \frac{1}{n}\sum_{i=1}^{n} \nabla C_i(\mathbf{w}^{(t)}),$$

where $C_i(\mathbf{w}^{(t)})$ is the cost by using the model $\mathbf{w}^{(t)}$ to do prediction on the i-th example. The complexity is therefore $O(nd)$ per iteration (d is the number of features). To speed up the computation, people have developed a variant of gradient descent called stochastic gradient descent (SGD). The basic idea is to replace the computationally expensive term $\nabla C(\mathbf{w}^{(t)})$ by a stochastic gradient computed on a random example, using the finite-sum structure of the objective function (by finite-sum structure we mean the objective function is an average of n terms). At the t-th iteration, we randomly choose an index i_t uniformly from $\{1, 2, \ldots, n\}$ (the likelihood of choosing each index is the same). After that we compute a stochastic gradient $\nabla C_{i_t}(\mathbf{w}^{(t)})$ and update the model as follows

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta_t \nabla C_{i_t}(\mathbf{w}^{(t)}).$$

The detailed implementation is given in Algorithm 1.

Algorithm 1 (Stochastic Gradient Descent (Robbins & Monro 1951))**.** Let $\{\eta_t\}$ be a sequence of step sizes

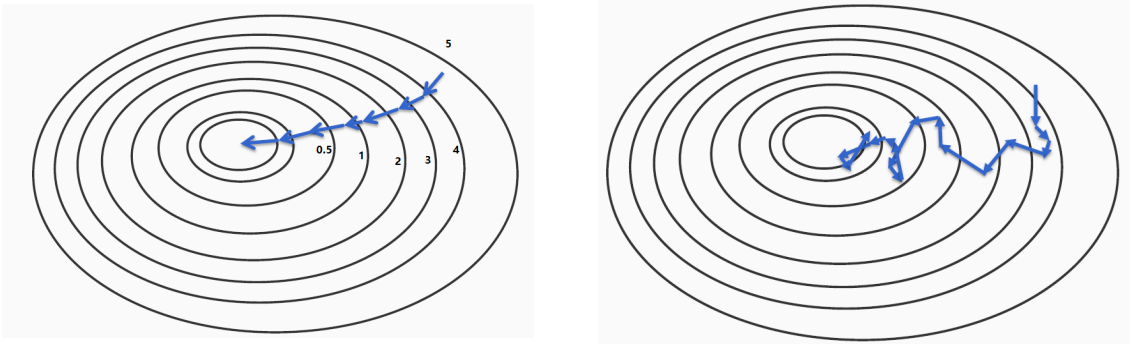
- Initialize the weights $\mathbf{w}^{(0)}$
- For $t = 0, 1, \ldots, T$
  - Draw i_t from $\{1, \ldots, n\}$ with equal probability
  - Compute stochastic gradient $\nabla C_{i_t}(\mathbf{w}^{(t)})$ and update

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta_t \nabla C_{i_t}(\mathbf{w}^{(t)})$$

Notice ∇C_{i_t} is not a true gradient. One cannot guarantee that SGD would behave as smoothly as GD (it may not decrease the objective function per iteration). However, since i_t is equally likely to be any number in $\{1, 2, \ldots, n\}$, the on-average of stochastic gradient is $\frac{1}{n}\sum_{i=1}^{n} \nabla C_i(\mathbf{w}^{(t)}) = \nabla C(\mathbf{w}^{(t)})$. Then, in the long run we expect that SGD would solve the optimization problem. Here is a comparison between GD and SGD:

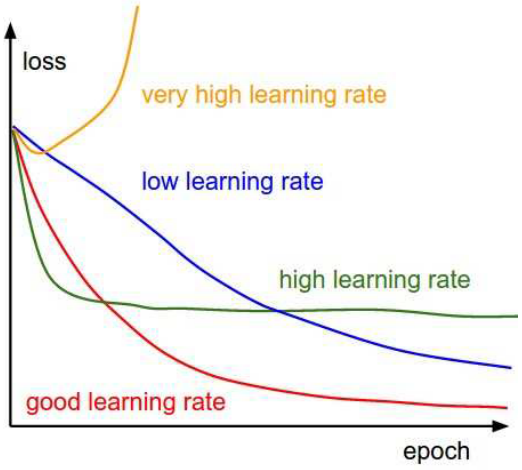
- GD requires more computation per iteration. However, GD at each iteration can make a good progress. Therefore, it needs few iterations to arrive at a good solution with a high accuracy.
- SGD requires less computation per iteration. However, SGD makes less update per iteration. Therefore, it needs more iterations to arrive at a good solution.
- Therefore, GD and SGD cannot always dominate the other. If we want a solution with a high accuracy and the sample size is small, then GD is a better choice. If we want a solution with a moderate accuracy and the problem size is large, then SGD is a better choice.

The computation cost of SGD is independent of the sample size and therefore SGD is especially efficient for large-scale problems. The following figure shows the behavior of GD and SGD: GD is smooth since negative gradient is a direction where function values decrease. SGD is not stable since stochastic gradient is a noisy estimate of the true gradient.



Property 1 (Learning rate)**.** The learning rate has a large effect on the behavior of SGD:

- If we choose a low learning rate, then SGD would converge very slowly
- If we choose a large learning rate, then SGD would converge quickly at the beginning, and would not go further as we run more and more iterations
- If we choose a huge learning rate, then SGD would become unstable
- We should choose an appropriate learning rate sequence. Since the stochastic gradient is a noisy estimate of the true gradient, we would decrease the learning rate in the training process. A typical choice is $\eta_t = c/\sqrt{t}$, where c is a parameter needed to tune according to the problem. That is, we use smaller and smaller learning rates as we run more and more iterations



3 Minibatch SGD

SGD uses a single example to build a stochastic gradient. A natural extension of SGD is the minibatch SGD, where instead of using only one example to compute a stochastic gradient, we use several examples to compute a stochastic gradient. At each iteration, we first randomly select a batch of indices: $B_t \subseteq \{1, 2, \ldots, n\}$ and update the model by

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \frac{\eta_t}{b} \sum_{i \in B_t} \nabla C_i(\mathbf{w}^{(t)}),$$

where b is the batch size. For example, if $b = 2$ and $B_t = \{2, 6\}$ then

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \frac{\eta_t}{2} \Big(\nabla C_2(\mathbf{w}^{(t)}) + \nabla C_6(\mathbf{w}^{(t)}) \Big).$$

If $b = 1$, it is clear that minibatch SGD is SGD. The implementation of the minibatch SGD is in Algorithm 2

Algorithm 2 (Minibatch SGD)**.** Let $\{\eta_t\}$ be a sequence of step sizes

- Initialize the weights $\mathbf{w}^{(0)}$
- For $t = 0, 1, \ldots, T$
  - Randomly selecting a batch $B_t \subseteq \{1, 2, \ldots, n\}$ of size b
  - Update the model by

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \frac{\eta_t}{b} \sum_{i \in B_t} \nabla C_i(\mathbf{w}^{(t)}).$$

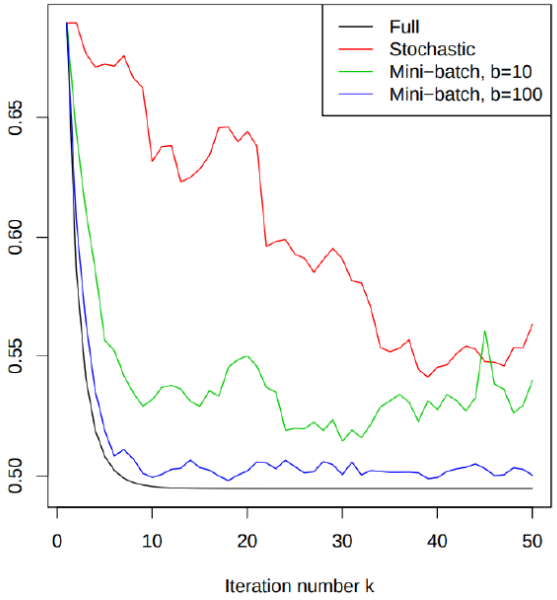
There are two choices of random sampling

- sampling with replacement: we sample b indices one-by-one. After sampling an index j, we put it back to the bag so that the index j is possible to be selected again in the future (it is possible $B_t = \{2, 3, 2\}$)
- sampling without replacement: we sample b indices one-by-one. After sampling an index j, we do not put it back to the bag so that we cannot select the index j again (it is not possible $B_t = \{2, 3, 2\}$).

It is clear that the behavior of minibatch SGD is between SGD and GD:

- It requires more computation cost than SGD per iteration. However since it uses more examples to build a stochastic gradient, the stochastic gradient is a more accurate estimate of the gradient $\nabla C(\mathbf{w}^{(t)})$. Therefore it makes better progress than SGD per iteration.
- A large batch size means more computation per iteration and more accurate stochastic gradient. A small b means less computation per iteration and less accurate stochastic gradient.
- We need to balance the accuracy and computation by choosing an appropriate b. A typical choice of batch size is $b = 32, 64, 128$.

The following figure compares the behavior of minibatch SGD versus GD/SGD



4 Linear Models for Classification

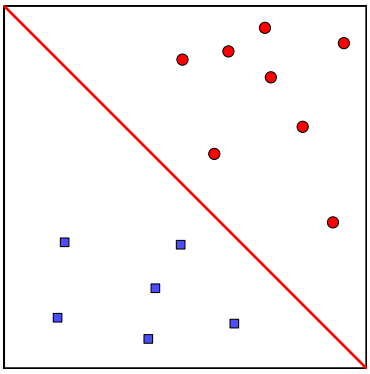
4.1 Linear Models for Classification

In the previous section, we consider linear models for regression. In this section, we consider linear models for binary classification.

Suppose we have

$$S = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}, y_i \in \{-1, +1\}.$$

We want to build a linear model to separate positive examples from negative examples: positive examples are on one-hand of the linear function, while negative examples are on the other hand.



A linear model $\mathbf{x} \mapsto \mathbf{w}^\top \mathbf{x}$ outputs a real number. To use it for prediction we can use its sign, that is $\hat{y} = \text{sgn}(\mathbf{w}^\top \mathbf{x})$, where $\text{sgn}(a) = 1$ if $a > 0$; and $\text{sgn}(a) = -1$ otherwise (we do not consider the case $a = 0$ which is not important). The performance of the model $\mathbf{w}$ on $(\mathbf{x}, y)$ can be measured by the 0-1 loss

$$L(\hat{y}, y) = \mathbb{I}[\hat{y} \neq y] = \begin{cases} 1, & \text{if } \hat{y} \neq y \\ 0, & \text{otherwise.} \end{cases}$$

Then the behavior of a model on S can be measured by

$$C(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}[\text{sgn}(\mathbf{w}^\top \mathbf{x}^{(i)}) \neq y^{(i)}].$$

One may want to minimize $C(\mathbf{w})$ to get a model. However, this is very challenging since it involves the indicator function, which is not continuous. Therefore, the objective function C is not differentiable and we can not use gradient information to build an optimization algorithm. According to the prediction function, we know

$$\hat{y} = \begin{cases} 1, & \text{if } \mathbf{w}^\top \mathbf{x} > 0 \\ -1, & \text{otherwise.} \end{cases} \Longrightarrow \begin{cases} y = \hat{y} = 1, & \text{if } y = 1, y(\mathbf{w}^\top \mathbf{x}) > 0 \\ y = \hat{y} = -1, & \text{if } y = -1, y(\mathbf{w}^\top \mathbf{x}) \geq 0 \\ y = 1, \hat{y} = -1, & \text{if } y = 1, y(\mathbf{w}^\top \mathbf{x}) \leq 0 \\ y = -1, \hat{y} = 1, & \text{if } y = -1, y(\mathbf{w}^\top \mathbf{x}) < 0 \end{cases}$$

It is clear that $\hat{y} \neq y$ amounts to saying $y\mathbf{w}^\top \mathbf{x} \leq 0$ (the case of being exactly 0 is not important); that is we make a mistake if $y\mathbf{w}^\top \mathbf{x} \leq 0$. This motivates an important concept called margin.

Definition 1 (Margin)**.** The margin of a model $\mathbf{w}$ on an example $(\mathbf{x}, y)$ is defined as $y\mathbf{w}^\top x$.

According to this definition, a model with a positive margin means a correct prediction and a model with a negative margin means an incorrect prediction. We wish to build a model with a large margin This further motivates the building of a model with large margin: a large margin means the model is robust in making a correct prediction. Based on this observation, we can define margin-based loss functions.

Definition 2 (Margin-based loss)**.** We consider the loss function of the form

$$L(\hat{y}, y) = g(y\hat{y}),$$

where g is decreasing.

Notice that $y\hat{y}$ is the margin since $\hat{y} = \mathbf{w}^\top \mathbf{x}$ is the predicted output. The reason to choose a decreasing function g is as follows.

- We want to minimize the loss $L(\hat{y}, y) = g(\hat{y}y)$. Since g is decreasing, minimizing L is equivalent to maximizing the margin.
- Maximizing the margin means getting a model with good performance.

Remark 4. Notice we replace the non-continuous indicator function by a decreasing function g. If g has good property such as differentiable, then we can get an easy optimization problem to solve.

There are several choices of g. In this lecture, we consider

$$g(t) = \frac{1}{2}\big(\max\{0, 1-t\}\big)^2 = \begin{cases} 0, & \text{if } t \geq 1 \\ (1-t)^2, & \text{otherwise.} \end{cases}.$$

This leads to the following loss function

$$L(\hat{y}, y) = \frac{1}{2}\big(\max\{0, 1-\hat{y}y\}\big)^2 = \frac{1}{2}\big(\max\{0, 1-y\mathbf{w}^\top \mathbf{x}\}\big)^2.$$

This leads to the following performance measure of $\mathbf{w}$ on S

$$C(\mathbf{w}) = \frac{1}{2n}\sum_{i=1}^{n}\big(\max\{0, 1-y^{(i)}\mathbf{w}^\top \mathbf{x}^{(i)}\}\big)^2.$$

We want to minimize $C(\mathbf{w})$ to get a good model. The first-order necessary condition shows that the optimal $\mathbf{w}^*$ satisfies $\nabla C(\mathbf{w}^*) = 0$. To use this condition, we need to compute the gradient of the following function

$$C_i(\mathbf{w}) = \frac{1}{2}\big(\max\{0, 1-y^{(i)}\mathbf{w}^\top \mathbf{x}^{(i)}\}\big)^2 = \begin{cases} 0, & \text{if } y^{(i)}\mathbf{w}^\top \mathbf{x}^{(i)} \geq 1 \\ \frac{1}{2}(1-y^{(i)}\mathbf{w}^\top \mathbf{x}^{(i)})^2, & \text{otherwise.} \end{cases}$$

One can check the gradient of C_i is as follows

$$\nabla C_i(\mathbf{w}) = \begin{cases} 0, & \text{if } y^{(i)}\mathbf{w}^\top \mathbf{x}^{(i)} \geq 1 \\ y^{(i)}(y^{(i)}\mathbf{w}^\top \mathbf{x}^{(i)} - 1)\mathbf{x}^{(i)}, & \text{otherwise.} \end{cases}$$

Since $(y^{(i)})^2 = 1$, we further get

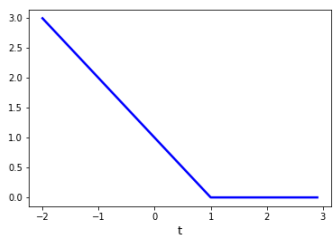
$$\nabla C_i(\mathbf{w}) = \begin{cases} 0, & \text{if } y^{(i)}\mathbf{w}^\top \mathbf{x}^{(i)} \geq 1 \\ (\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})\mathbf{x}^{(i)}, & \text{otherwise.} \end{cases} \tag{4.1}$$

The first-order optimality condition implies

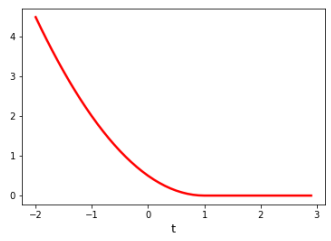
$$\frac{1}{n}\sum_{i=1}^{n}\nabla C_i(\mathbf{w}^*) = 0.$$

However, the above nonlinear equation is not easy to solve and one cannot get a closed-form solution for the optimal $\mathbf{w}^*$. In the next section, we will show how to solve this optimization problem by gradient-based optimization methods.

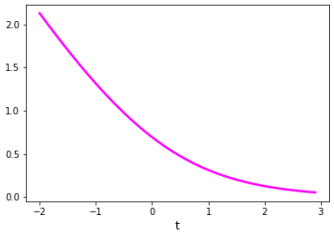
Remark 5. Here we consider the choice $g(t) = (\max\{0, 1-t\})^2/2$. There are some other choices, including $g(t) = \max\{0, 1-t\}$ and $g(t) = \log(1+\exp(-t))$. It is clear that these functions are decreasing functions of t. Below we show the picture of these functions.



$g(t) = \max\{0, 1-t\}$



$g(t) = \frac{1}{2}\big(\max\{0, 1-t\}\big)^2$



$g(t) = \log(1+\exp(-t))$

- If we consider $g(t) = \max\{0, 1-t\}$, then this leads to the following optimization problem

$$\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n}\max\{0, 1-y^{(i)}\mathbf{w}^\top \mathbf{x}^{(i)}\},$$

  which corresponds to the support vector machine.

- If one considers $g(t) = \log(1 + \exp(-t))$, then this leads to

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + \exp(-y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)})\right),$$

  which corresponds to another popular method called logistic regression. Note while we call it logistic regression, it is indeed a classification method.

This shows how we can recover popular machine learning methods by considering different margin-based loss functions.

4.2 SGD for Linear Classification Models

We now apply SGD to solve the optimization problem in linear classification. Suppose we consider the following loss function for linear classification

$$C_i(\mathbf{w}) = \begin{cases} 0, & \text{if } y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)} \geq 1 \\ \frac{1}{2}(1 - y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)})^2, & \text{otherwise.} \end{cases}$$

Suppose at the t-th iteration, the index we sample is i_t. According to Eq. (4.1), the update should be

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta_t \nabla C_{i_t}(\mathbf{w}^{(t)}) = \begin{cases} \mathbf{w}^{(t)}, & \text{if } y^{(i_t)} (\mathbf{w}^{(t)})^\top \mathbf{x}^{(i_t)} \geq 1 \\ \mathbf{w}^{(t)} - \eta_t y^{(i_t)} (y^{(i_t)} \mathbf{w}^\top \mathbf{x}^{(i_t)} - 1) \mathbf{x}^{(i_t)}, & \text{otherwise.} \end{cases}$$

The implementation is given in Algorithm 1

Algorithm 1: SGD for Linear Classification

```
for t = 0, 1, ... to T do
    i_t ← random index from {1, 2, ..., n}
    if y^(i_t) (w^(t))^T x^(i_t) ≥ 1 then
        w^(t+1) = w^(t)
    else
        w^(t+1) = w^(t) + η_t (1 − y^(i_t) w^T x^(i_t)) y^(i_t) x^(i_t)
    end
end
```

Intuition

- According to the implementation, we do not make any update if $y^{(i_t)} (\mathbf{w}^{(t)})^\top x^{(i_t)} \geq 1$, which is a case with margin larger than 1 (we are doing well with this example and there is no need to update).
- We need to update if the margin is smaller than 1 (we need to update when we are not doing quite well on the example). Note $1 > y^{(i_t)} \mathbf{w}^\top \mathbf{x}^{(i_t)}$. Then the update becomes

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \alpha y^{(i_t)} \mathbf{x}^{(i_t)}$$

  for $\alpha = \eta_t (1 - y^{(i_t)} \mathbf{w}^\top \mathbf{x}^{(i_t)}) > 0$. That is

$$\begin{aligned} y^{(i_t)} (\mathbf{w}^{(t+1)})^\top \mathbf{x}^{(i_t)} &= y^{(i_t)} (\mathbf{w}^{(t)})^\top \mathbf{x}^{(i_t)} + \alpha y^{(i_t)} y^{(i_t)} \|\mathbf{x}^{(i_t)}\|^2 \\ &> y^{(i_t)} (\mathbf{w}^{(t)})^\top \mathbf{x}^{(i_t)}. \end{aligned}$$

  That is, the margin of the new model $\mathbf{w}^{(t+1)}$ at $(\mathbf{x}^{(i_t)}, y^{(i_t)})$ is larger than the margin of the old model $\mathbf{w}^{(t)}$ at $(\mathbf{x}^{(i_t)}, y^{(i_t)})$.

Intuitively, we are increasing the margin by SGD!

Question 1. If we consider linear classification with the hinge loss

$$C_i(\mathbf{w}) = \max\{0, 1 - y^{(i)}\mathbf{w}^\top \mathbf{x}^{(i)}\},$$

what is the formula for SGD update?

Question 2. If we consider a regularization problem with

$$C_i(\mathbf{w}) = \left(\max\{0, 1 - y^{(i)}\mathbf{w}^\top \mathbf{x}^{(i)}\}\right)^2/2 + \lambda\|\mathbf{w}\|_2^2/2,$$

what is the formula for SGD update?