STA 314: Statistical Methods for Machine Learning I

Lecture 7 - Logistic regression and Gradient Descent

Xin Bing

Department of Statistical Sciences
University of Toronto

Review

- In classification, $X \in \mathcal{X}$ and $Y \in C = \{0, 1, \dots, K-1\}$.

- The Bayes rule

$$\arg\max_{k \in C} \mathbb{P}\{Y = k \mid X = \mathbf{x}\}, \qquad \forall \mathbf{x} \in \mathcal{X}$$

  has the smallest expected error rate.

- For binary classification, our goal is to estimate

$$p(\mathbf{x}) = \mathbb{P}\{Y = 1 \mid X = \mathbf{x}\}, \qquad \forall \mathbf{x} \in \mathcal{X}.$$

Logistic Regression

Logistic Regression is a parametric approach that assumes parametric structure on

$$p(\mathbf{x}) = \mathbb{P}(Y = 1 \mid X = \mathbf{x}).$$

- It assumes

  $$p(\mathbf{x}) = \frac{e^{\beta_0+\beta_1 x_1+\cdots+\beta_p x_p}}{1 + e^{\beta_0+\beta_1 x_1+\cdots+\beta_p x_p}}.$$

  The function $f(t) = e^t/(1 + e^t)$ is called the logistic function. $\beta_0, \ldots, \beta_p$ are the parameters.

- It is easy to see that we always have $0 \le p(\mathbf{x}) \le 1$.

- Note that $p(\mathbf{x})$ is **NOT** a linear function either in $\mathbf{x}$ or in $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_p)$.

Logistic Regression

- A bit of rearrangement gives

$$\underbrace{\frac{p(\mathbf{x})}{1-p(\mathbf{x})}}_{\text{odds}} = e^{\beta_0+\beta_1 x_1+\cdots+\beta_p x_p},$$

$$\underbrace{\log\left[\frac{p(\mathbf{x})}{1-p(\mathbf{x})}\right]}_{\text{log-odds (a.k.a. logit)}} = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p.$$

  odds $\in [0, \infty)$ and log-odds $\in (-\infty, \infty)$.

- Similar interpretation as linear models.

- How to estimate $\boldsymbol{\beta}$?

Maximum Likelihood Estimator (MLE)

Given $\mathcal{D}^{train} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$ with $y_i \in \{0, 1\}$, we estimate the parameters by **maximizing the likelihood** of $\mathcal{D}^{train}$.

The maximum likelihood principle

We seek the estimates of parameters such that the fitted probability are the closest to the individual's observed outcome.

Cont'd: MLE under logistic regression

Recipe of computing the MLE:

1. Write down the likelihood, as always!
2. Solve the optimization (maximization) problem.

The MLE has many nice properties!

- Asymp. consistent
- Asymp. normal
- Asymp. efficient

Inference under logistic regression

Let $\hat{\boldsymbol{\beta}}$ be the MLE.

- Z-statistic is similar to t-statistic in regression, and is defined as

$$\frac{\hat{\beta}_j}{SE(\hat{\beta}_j)}, \qquad \forall j \in \{0, 1, \ldots, p\}.$$

- It produces p-value for testing the null hypothesis

$$H_0 : \beta_j = 0 \quad \text{v.s.} \quad H_1 : \beta_j \neq 0.$$

  A large (absolute) value of the z-statistic or small p-value indicates evidence against H_0.

Example: Default data

Consider the Default data using balance, income, and student status as predictors.

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{1+e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}$$

	Coefficient	Std. Error	Z-statistic	P-value
`Intercept`	-10.8690	0.4923	-22.08	< 0.0001
`balance`	0.0057	0.0002	24.74	< 0.0001
`income`	0.0030	0.0082	0.37	0.7115
`student[Yes]`	-0.6468	0.2362	-2.74	0.0062

Prediction at **different levels** under logistic regression

Let $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \ldots, \hat{\beta}_p)$ be the MLE.

- Prediction of **the logit** at $\mathbf{x} \in \mathcal{X}$:

$$\widehat{\text{logit}}(\mathbf{x}) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p.$$

- Prediction of **the conditional probability** $\mathbb{P}(Y = 1 \mid X = \mathbf{x})$:

$$\hat{\mathbb{P}}(Y = 1 \mid X = \mathbf{x}) = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p}}$$

- Prediction of **the label** Y (i.e. *classification*) at $X = \mathbf{x}$:

$$\hat{y} = \begin{cases} 1, & \text{if} \quad \hat{\mathbb{P}}(Y = 1 \mid X = \mathbf{x}) \geq 0.5; \\ 0, & \text{otherwise.} \end{cases}$$

Prediction of $\mathbb{P}(Y = 1 \mid X = \mathbf{x})$

Consider the Default data with student status as the only feature.

What is the probability of default for a student?

To fit the model, we encode student status as 1 for student and 0 otherwise.

	Coefficient	Std. Error	Z-statistic	P-value
Intercept	-3.5041	0.0707	-49.55	< 0.0001
student[Yes]	0.4049	0.1150	3.52	0.0004

$$\widehat{\Pr}(\texttt{default}=\texttt{Yes}|\texttt{student}=\texttt{Yes}) = \frac{e^{-3.5041+0.4049\times 1}}{1+e^{-3.5041+0.4049\times 1}} = 0.0431,$$

$$\widehat{\Pr}(\texttt{default}=\texttt{Yes}|\texttt{student}=\texttt{No}) = \frac{e^{-3.5041+0.4049\times 0}}{1+e^{-3.5041+0.4049\times 0}} = 0.0292.$$

Metrics used for evaluating classifiers

In classification, we have several metrics that can be used to evaluate a given classifier.

- The most commonly used metric is the overall classification accuracy.
- For binary classification, there are a few more out there.....

Logistic Regression on the Default Data

- Classify whether or not an individual will default on the basis of credit card balance and student status.

- **The confusion matrix** of fitted logistic regression

		True default status		
		No	Yes	Total
Predicted default status	No	$9,644$	252	$9,896$
	Yes	23	81	104
	Total	$9,667$	333	$10,000$

Type of Errors for binary classification

		True default status		
		No	Yes	Total
Predicted	No	9,644	252	9,896
default status	Yes	23	81	104
	Total	9,667	333	10,000

1. The training error rate is $(23 + 252)/10000 = 2.75\%$.
2. **False positive rate (FPR)**: The fraction of negative examples that are classified as positive: $23/9667 = 0.2\%$ in default data.
3. **False negative rate (FNR)**: The fraction of positive examples that are classified as negative: $252/333 = 75.7\%$ in default data.[1]

[1] For a credit card company that is trying to identify high-risk individuals, the error rate 75.7% among individuals who default is unacceptable.

Types of Errors for binary classification

Q: How to modify the logistic classifier to lower the FNR?

- The current classifier is based on the rule

$$\hat{\mathbb{P}}(\text{default} = \textit{yes} \mid X = \mathbf{x}) \geq 0.5.$$

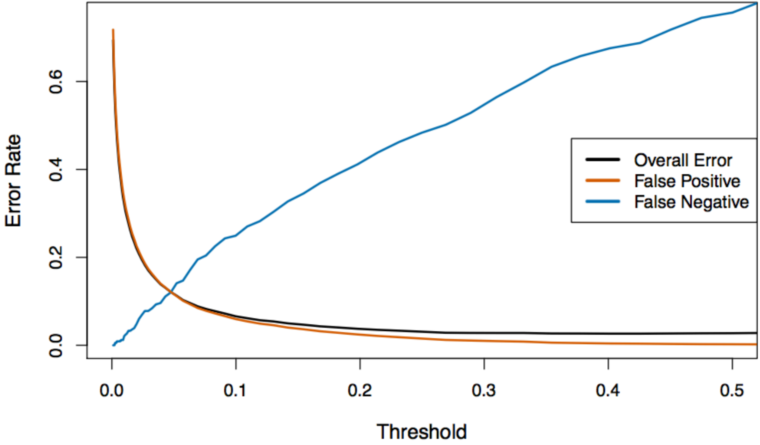
- To lower FNR, we reduce the number of negative predictions. Classify $X = \mathbf{x}$ to *yes* if

$$\hat{\mathbb{P}}(Y = \textit{yes} \mid X = \mathbf{x}) \geq t.$$

for some $t < 0.5$.

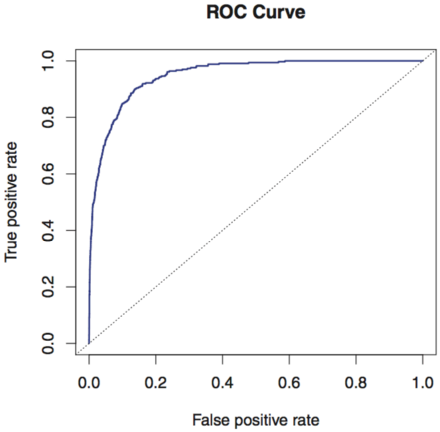
Trade-off between FPR and FNR

We can achieve better balance between FPR and FNR by varying the threshold:



ROC Curve

The **ROC curve** is a popular graphic for simultaneously displaying FPR and TPR = 1 - FNR for all possible thresholds.



The overall performance of a classifier, summarized over all thresholds, is given by the area under the curve (**AUC**). High AUC is good.

More metrics in the binary classification

		Predicted class		
		− or Null	+ or Non-null	Total
True	− or Null	True Neg. (TN)	False Pos. (FP)	N
class	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N*	P*	

Name	Definition	Synonyms
False Pos. rate	FP/N	Type I error, 1−Specificity
True Pos. rate	TP/P	1−Type II error, power, sensitivity, recall
Pos. Pred. value	TP/P*	Precision, 1−false discovery proportion
Neg. Pred. value	TN/N*	

The above also defines **sensitivity** and **specificity**.

Computation of the MLE under Logistic Regression

General steps of computing the MLE:

- Write down the likelihood, as always!
- Solve the optimization problem.

Likelihood under Logistic Regression

For simplicity, let us set $\beta_0 = 0$ such that

$$p(\mathbf{x}) = \frac{e^{\mathbf{x}^\top \boldsymbol{\beta}}}{1 + e^{\mathbf{x}^\top \boldsymbol{\beta}}}, \qquad 1 - p(\mathbf{x}) = \frac{1}{1 + e^{\mathbf{x}^\top \boldsymbol{\beta}}}.$$

The data consists of $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ with

$$y_i \sim \text{Bernoulli}(p(\mathbf{x}_i)), \qquad p(\mathbf{x}_i) = \frac{e^{\mathbf{x}_i^\top \boldsymbol{\beta}}}{1 + e^{\mathbf{x}_i^\top \boldsymbol{\beta}}}, \quad 1 \leq i \leq n.$$

- What is the likelihood of y_i?

Likelihood under Logistic Regression

The likelihood of each data point $(\mathbf{x}_i, y_i)$ at any $\boldsymbol{\beta}$ is

$$L(\boldsymbol{\beta}; \mathbf{x}_i, y_i) = [p(\mathbf{x}_i)]^{y_i} [1 - p(\mathbf{x}_i)]^{1-y_i}$$

with

$$p(\mathbf{x}_i) = \frac{e^{\mathbf{x}_i^\top \boldsymbol{\beta}}}{1 + e^{\mathbf{x}_i^\top \boldsymbol{\beta}}}.$$

The joint likelihood of all data points is

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} [p(\mathbf{x}_i)]^{y_i} [1 - p(\mathbf{x}_i)]^{1-y_i}.$$

Log-likelihood under Logistic Regression

The log-likelihood at any $\boldsymbol{\beta}$ is

$$\begin{aligned}
\ell(\boldsymbol{\beta}) &= \log\left\{\prod_{i=1}^{n}[p(\mathbf{x}_i)]^{y_i}[1-p(\mathbf{x}_i)]^{1-y_i}\right\} \\
&= \sum_{i=1}^{n}[y_i \log(p(\mathbf{x}_i)) + (1-y_i)\log(1-p(\mathbf{x}_i))] \\
&= \sum_{i=1}^{n}\left[y_i \log\left(\frac{p(\mathbf{x}_i)}{1-p(\mathbf{x}_i)}\right) + \log(1-p(\mathbf{x}_i))\right] \\
&= \sum_{i=1}^{n}\left[y_i \mathbf{x}_i^{\top}\boldsymbol{\beta} - \log\left(1+e^{\mathbf{x}_i^{\top}\boldsymbol{\beta}}\right)\right].
\end{aligned}$$

How to compute the MLE?

How do we maximize the log-likelihood

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[y_i \mathbf{x}_i^\top \boldsymbol{\beta} - \log\left(1 + e^{\mathbf{x}_i^\top \boldsymbol{\beta}}\right)\right]$$

for logistic regression?

- It is equivalent to minimize $-\ell(\boldsymbol{\beta})$ over $\boldsymbol{\beta}$.
- No direct solution: taking derivatives of $\ell(\boldsymbol{\beta})$ w.r.t. $\boldsymbol{\beta}$ and setting them to 0 doesn't have an explicit solution.
- Need to use iterative procedure.

A general problem of solving a minimization problem

Suppose we want to solve the following problem

$$\hat{\mathbf{w}} = \underset{\mathbf{w}\in\Theta}{\operatorname{argmin}}\, \mathcal{J}(\mathbf{w}; \mathcal{D}^{train}) := \underset{\mathbf{w}\in\Theta}{\operatorname{argmin}}\, \mathcal{J}(\mathbf{w})$$

where $\mathcal{J}(\mathbf{w}; \mathcal{D}^{train})$ is a differentiable function in $\mathbf{w}$, and depends on $\mathcal{D}^{train}$ as well, and Θ is a subspace of $\mathbb{R}^p$.

- The optimal solution (if exists) must be a **critical point**, i.e. point to which the derivative is zero (partial derivatives to zero for multi-dimensional parameter).

Finding the optimal solution requires to solve the equations

- **Partial derivatives**: derivatives of a multivariate function with respect to one of its arguments.

$$\frac{\partial}{\partial x_1} f(x_1, x_2) = \lim_{h \to 0} \frac{f(x_1 + h, x_2) - f(x_1, x_2)}{h}$$

- The minimum must occur at a point where the partial derivatives are zero.

$$\begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \vdots \\ \frac{\partial g}{\partial w_p} \end{bmatrix} = 0$$

- This turns out to give a system of linear equations, which we can solve analytically in some scenarios.
- We may also use optimization techniques that iteratively get us closer to the solution.

Direct solution

- OLS:

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^p} \mathcal{J}(\mathbf{w}) = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^p} ||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2.$$

The partial derivatives w.r.t. $\mathbf{w}$ are

$$\frac{\partial g}{\partial \mathbf{w}} = -2\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}).$$

(If not familiar with multi-dimensional derivatives, calculate $\frac{\partial g}{\partial w_j}$ and stack them together).
Setting the above equal to zero results

$$\mathbf{X}^\top\mathbf{X}\hat{\mathbf{w}} = \mathbf{X}^\top\mathbf{y}, \qquad \Rightarrow \qquad \hat{\mathbf{w}} = \left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\mathbf{y}.$$

Direct solution

- Ridge:

$$\hat{\mathbf{w}}_\lambda^R = \underset{\mathbf{w}\in\mathbb{R}^p}{\operatorname{argmin}}\, \mathcal{J}(\mathbf{w}) = \underset{\mathbf{w}\in\mathbb{R}^p}{\operatorname{argmin}}\, ||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 + \lambda||\mathbf{w}||_2^2.$$

The partial derivatives w.r.t. $\mathbf{w}$ are

$$\frac{\partial g}{\partial \mathbf{w}} = -2\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda\mathbf{w}.$$

Setting the above equal to zero results

$$(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_p)\hat{\mathbf{w}}_\lambda^R = \mathbf{X}^\top\mathbf{y}, \qquad \Rightarrow \qquad \hat{\mathbf{w}}_\lambda^R = \left(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_p\right)^{-1}\mathbf{X}^\top\mathbf{y}.$$

Gradient Descent

- Now let's see a second way to solve

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}}\, \mathcal{J}(\mathbf{w})$$

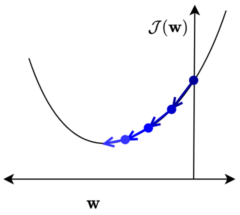
which is more broadly applicable: **gradient descent**.

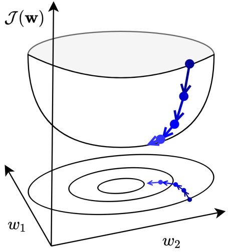
- Many times, we do not have a direct solution to

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = 0.$$

- Gradient descent is an **iterative algorithm**, which means we apply an update repeatedly until some criterion is met.

We **initialize** $\mathbf{w}$ to something reasonable (e.g. all zeros) and repeatedly adjust them in the **direction of steepest descent**.





What is the direction of the steepest descent of $\mathcal{J}(\mathbf{w})$ at $\mathbf{w}$?

Gradient Descent

- By definition, the direction of the greatest increase in $\mathcal{J}(\mathbf{w})$ at $\mathbf{w}$ is its gradient $\partial\mathcal{J}/\partial\mathbf{w}$. So, we should update $\mathbf{w}$ in the **opposite** direction of the gradient descent.
- The following update always decreases the cost function for small enough α (unless $\partial\mathcal{J}/\partial w_j = 0$): at the $(k+1)$th iteration,

$$w_j^{(k+1)} \leftarrow w_j^{(k)} - \alpha \cdot \frac{\partial \mathcal{J}}{\partial w_j}\bigg|_{\mathbf{w}=\mathbf{w}^{(k)}}$$

- $\alpha > 0$ is a **learning rate** (or step size). The larger it is, the faster $\mathbf{w}^{(k+1)}$ changes relative to $\mathbf{w}^{(k)}$
  - We'll see later how to tune the learning rate, but values are typically small, e.g. 0.01 or 0.0001.

Gradient descent for OLS

Example

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^p} \mathcal{J}(\mathbf{w}), \qquad \mathcal{J}(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2.$$

Update rule in vector form at the $k+1$th iteration:

$$\begin{aligned}\mathbf{w}^{(k+1)} &\leftarrow \mathbf{w}^{(k)} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}}\Big|_{\mathbf{w}=\mathbf{w}^{(k)}} \\ &= \mathbf{w}^{(k)} + 2\alpha \mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}^{(k)}).\end{aligned}$$

Initialization: $\mathbf{w}^{(0)} = 0$.

Stopping criteria

When do we stop?

- The objective value stops changing:

$$|\mathcal{J}(\mathbf{w}^{(k+1)}) - \mathcal{J}(\mathbf{w}^{(k)})| \text{ is small, i.e. } \leq 10^{-6}.$$

- The parameter stops changing: $||\mathbf{w}^{(k+1)} - \mathbf{w}^{(k)}||_2$ is small or $||\mathbf{w}^{(k+1)} - \mathbf{w}^{(k)}||_2 / ||\mathbf{w}^{(k)}||_2$ is small.

- When we reach the maximum number (M) of iterations, e.g. $M = 1000$.

Gradient descent for solving the MLE under logistic regression

Recall we would like to solve

$$\min_{\mathbf{w}\in\mathbb{R}^p} \mathcal{J}(\mathbf{w})$$

where

$$\mathcal{J}(\mathbf{w}) = -\ell(\mathbf{w}) = \sum_{i=1}^{n}\left[-y_i\mathbf{x}_i^\top\mathbf{w} + \log\left(1 + e^{\mathbf{x}_i^\top\mathbf{w}}\right)\right].$$

The gradient at any $\mathbf{w}$ is that, for any $j \in \{1, \ldots, p\}$,

$$-\frac{\partial\ell(\mathbf{w})}{\partial w_j} = \sum_{i=1}^{n}\left[-y_i + \frac{e^{\mathbf{x}_i^\top\mathbf{w}}}{1 + e^{\mathbf{x}_i^\top\mathbf{w}}}\right]x_{ij} \qquad \text{(verify this!)}$$

Updates and stopping criteria

Therefore, at the $(k+1)$th iteration, with the learning rate α,

$$\hat{\mathbf{w}}^{(k+1)} = \hat{\mathbf{w}}^{(k)} - \alpha \sum_{i=1}^{n} \left[-y_i + \frac{e^{\mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)}}}{1 + e^{\mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)}}} \right] \mathbf{x}_i.$$

Initialization $\mathbf{w}^{(0)} = 0$.

- The objective value stops changing: $|\ell(\hat{\mathbf{w}}^{(k+1)}) - \ell(\hat{\mathbf{w}}^{(k)})|$ is small, say, $\leq 10^{-6}$.
- The parameter stops changing: $||\hat{\mathbf{w}}^{(k+1)} - \hat{\mathbf{w}}^{(k)}||_2$ is small or $||\hat{\mathbf{w}}^{(k+1)} - \hat{\mathbf{w}}^{(k)}||_2 / ||\hat{\mathbf{w}}^{(k)}||_2$ is small.
- Stop after M iterations for some specified M, e.g. $M = 1000$.

When should we expect Gradient Descent to work?

Recall we try to solve

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}\in\Theta} \mathcal{J}(\mathbf{w}).$$

- Obviously, $\mathcal{J}$ needs to be differentiable.
- If $\mathcal{J}$ is also a convex function and Θ is a convex set, then Gradient Descent finds the optimal solution.
- In many cases, $\Theta = \mathbb{R}^p$ which is convex.

Convex Sets

A set $\mathcal{S}$ is convex if for any $\mathbf{x}_0, \mathbf{x}_1 \in \mathcal{S}$,

$$(1 - \lambda)\mathbf{x}_0 + \lambda\mathbf{x}_1 \in \mathcal{S} \quad \text{for all } 0 \leq \lambda \leq 1.$$

The Euclidean space $\mathbb{R}^p$ is a convex set.

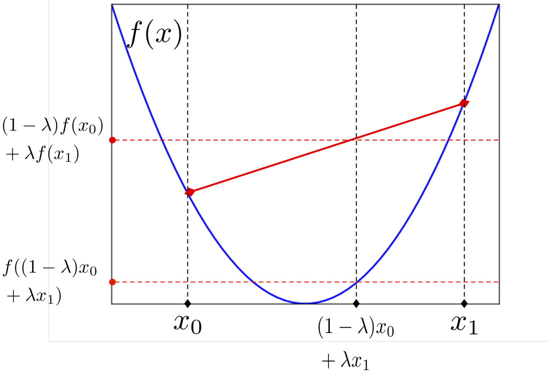
Convex Sets and Functions

- A function f is **convex** if for any $\mathbf{x}_0, \mathbf{x}_1$ in the domain of f,

$$f((1-\lambda)\mathbf{x}_0 + \lambda\mathbf{x}_1) \leq (1-\lambda)f(\mathbf{x}_0) + \lambda f(\mathbf{x}_1), \quad \forall \lambda \in [0,1].$$

- Equivalently, the set of points lying above the graph of f is convex.

- Intuitively: the function is bowl-shaped.



How to tell a loss is convex?

1. Verify the definition.

2. If f is twice differentiable and $f''(x) \geq 0$ for all x, then f is convex.
   - the least-squares loss function $(y - t)^2$ is convex as a function of t
   - the function
     $$-yt + \log\left(1 + e^t\right)$$
     is convex in t.

3. There are other sufficient conditions for convex, but non-differentiable, functions!

4. A composition rule: linear functions preserve convexity.
   - If f is a convex function and g is a linear function, then both $f \circ g$ and $g \circ f$ are convex.
     - the least-square loss $(y - \mathbf{x}^\top \mathbf{w})^2$ is convex in $\mathbf{w}$
     - the negative log-likelihood under logistic regression

$$-y\mathbf{x}^\top \mathbf{w} + \log\left(1 + e^{\mathbf{x}^\top \mathbf{w}}\right)$$

     is convex in $\mathbf{w}$.
   - Both $\sum_i (y_i - \mathbf{x}_i^\top \mathbf{w})^2$ and $\sum_i \left[-y_i \mathbf{x}_i^\top \mathbf{w} + \log\left(1 + e^{\mathbf{x}_i^\top \mathbf{w}}\right)\right]$ are convex in $\mathbf{w}$.

There are more composition rules!

A great book:

Convex Optimization, Stephen Boyd and Lieven Vandenberghe.

Gradient Descent for Linear Regression

- The squared error loss
$$\sum_{i=1}(y_i - \mathbf{x}_i^\top \mathbf{w})^2$$
of linear regression is a convex function. So there is a unique solution.
- Even in this case, we sometimes need to use GD.

- Why gradient descent, if we can find the optimum directly?
  - When p is large, GD is more efficient than direct solution
    - Linear regression solution: $(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$
    - Matrix inversion is an $\mathcal{O}(p^3)$ algorithm
    - Each GD update costs $\mathcal{O}(np)$
    - Or less with stochastic GD (Stochastic GD, later)
    - Huge difference if $p \gg \sqrt{n}$

Gradient descent for solving the MLE under logistic regression

- The negative log-likelihood

$$-\ell(\mathbf{w}) = \sum_{i=1}^{n} \left[-y_i \mathbf{x}_i^\top \mathbf{w} + \log\left(1 + e^{\mathbf{x}_i^\top \mathbf{w}}\right) \right]$$

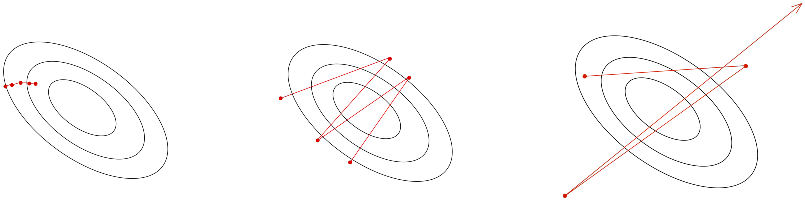
  is convex in $\mathbf{w}$.

- So we can use gradient descent to find the minima of the logistic loss!

- GD can be applied to more general settings!

Effect of the Learning Rate (Step Size)

- In gradient descent, the learning rate α is a hyperparameter we need to tune. Here are some things that can go wrong:



α too small: slow progress

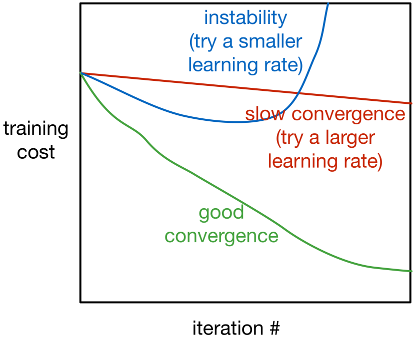
α too large: oscillations

α much too large: instability

- Good values are typically between 0.001 and 0.1. You should do a grid search if you want good performance (i.e. try $0.1, 0.03, 0.01, \ldots$).

Training Curves

- To diagnose optimization problems, it's useful to look at the **training cost**: plot the training cost as a function of iteration.



- **Warning**: the training cost could be used to check whether the optimization problem reaches certain convergence. But
  - It does not tell whether we reach the global minimum or not
  - It does not tell anything on the performance of the fitted model

Gradient descent

Visualization:

http://www.cs.toronto.edu/~guerzhoy/321/lec/W01/linear_regression.pdf#page=21

Batch Gradient Descent

- Recall that
  - OLS:
$$\hat{\mathbf{w}}^{(k+1)} = \hat{\mathbf{w}}^{(k)} + \alpha \sum_{i=1}^{n} \left[y_i - \mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)} \right] \mathbf{x}_i.$$
  - Logistic regression:
$$\hat{\mathbf{w}}^{(k+1)} = \hat{\mathbf{w}}^{(k)} + \alpha \sum_{i=1}^{n} \left[y_i - \frac{e^{\mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)}}}{1 + e^{\mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)}}} \right] \mathbf{x}_i.$$
- Computing the gradient requires summing over **all** of the training examples, which can be done via matrix / vector operations.
  The fact that it uses all training samples is known as **batch training**.

Stochastic Gradient Descent

- Batch training is impractical if you have a large dataset (e.g. millions of training examples, $n \approx 10$ millions)!

- **Stochastic gradient descent (SGD)**: update the parameters based on the gradient for a single training example.

  For each iteration $k \in \{1, 2, \ldots\}$,

  1. Choose $i \in \{1, \ldots, n\}$ uniformly at random
  2. Update the parameters by ONLY using this ith sample,

$$\hat{\mathbf{w}}^{(k+1)} = \hat{\mathbf{w}}^{(k)} + \alpha \left[y_i - \mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)} \right] \mathbf{x}_i$$

$$\hat{\mathbf{w}}^{(k+1)} = \hat{\mathbf{w}}^{(k)} + \alpha \left[y_i - \frac{e^{\mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)}}}{1 + e^{\mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)}}} \right] \mathbf{x}_i .$$

Stochastic Gradient Descent

$$\hat{\mathbf{w}}^{(k+1)} = \hat{\mathbf{w}}^{(k)} + \alpha \left[y_i - \mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)} \right] \mathbf{x}_i$$

$$\hat{\mathbf{w}}^{(k+1)} = \hat{\mathbf{w}}^{(k)} + \alpha \left[y_i - \frac{e^{\mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)}}}{1 + e^{\mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)}}} \right] \mathbf{x}_i.$$

Pros:

- Computational cost of each SGD update is independent of n!
- SGD can make significant progress before even seeing all the data!
- Mathematical justification: the gradients between SGD and GD have the same expectation for i.i.d. data.

Stochastic Gradient Descent

Cons: using single training example to estimate gradient:

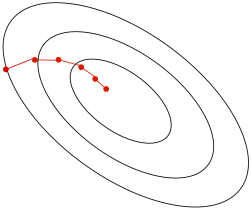
- Variance in the estimate may be high

Compromise approach:

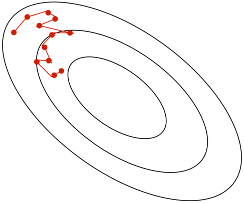
- compute the gradients on a randomly chosen medium-sized set of training examples $\mathcal{M} \subset \{1, \ldots, n\}$, called a **mini-batch**.
- Stochastic gradients computed on larger mini-batches have smaller variance.
- The mini-batch size $|\mathcal{M}|$ is a hyperparameter that needs to be set.

Stochastic Gradient Descent

- Batch gradient descent moves directly downhill. SGD takes steps in a noisy direction, but moves downhill on average.



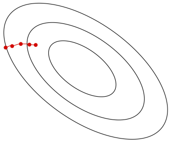
batch gradient descent



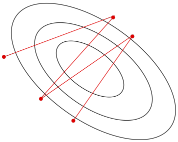
stochastic gradient descent

Learning Rate

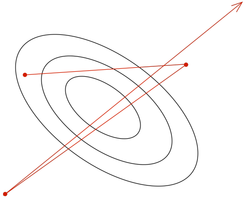
- In gradient descent, the learning rate α is a hyperparameter we need to tune. Here are some things that can go wrong:



α too small:
slow progress



α too large:
oscillations

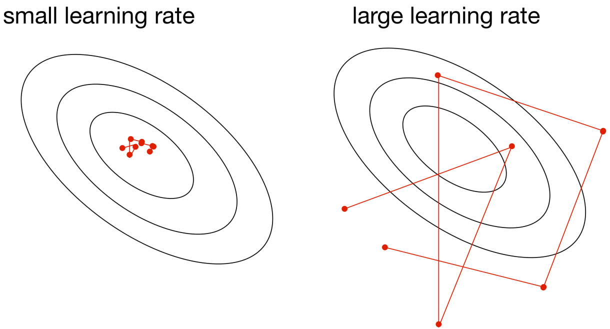


α much too large:
instability

- Good values are typically small. You should do a grid search if you want good performance (i.e. try $0.1, 0.03, 0.01, \ldots$).

SGD Learning Rate

- In stochastic training, the learning rate also influences the **fluctuations** due to the stochasticity of the gradients.



- Typical strategy:
  - Use a large learning rate early in training so you can get close to the optimum
  - Gradually decay the learning rate to reduce the fluctuations