ECE 0402 - Pattern Recognition

Lecture 3

Review:

Consider (X, Y) pair where

- X is a random vector in $\mathbb{R}^d$
- $Y \in \{0, 1, ..., K-1\}$ is a random variable that depends on X.

Let $f : \mathbb{R}^d \to \{0, 1, ..., K-1\}$ be a some classifier with **probability of error/risk** given by

$$R(f) \ := \ \mathbb{P}[f(X) \neq Y]$$

The **Bayes classifier**, f^*, is the optimal classifier – with smallest risk possible. We can calculate the Bayes classifier explicitly if we actually know the joint distribution of (X, Y). In general we can't do this but it is a useful place to start. Let's remember all the notation– since it was a little heavy.

- $g(x, k)$ denote the joint distribution of (X, Y). (This is a little ugly the talk about when X is continuous)
- Instead we can write this as $g(x, k) = \pi_k \ g_k(x)$
  - $\pi_k \ = \ \mathbb{P}[Y = k]$ is the **a priori probability** of class k.
  - $g_k(x)$ is the conditional pdf of $X|Y = k$, i.e., class conditional pdf.
- In turn, from Bayes' rule we have

$$\begin{aligned} \pi_k g_k(x) = \ & \pi_k(\frac{\mathbb{P}[Y = k|X = x] \ (\sum_{l=0}^{K-1} \pi_l g_l(x))}{\pi_k}) \\ & = \eta_k(x) \ (\sum_{l=0}^{K-1} \pi_l g_l(x)) \end{aligned}$$

- $\eta_k(x)$ is the **a a posteriori probability of class** k
- $\sum_{l=0}^{K-1} \pi_l g_l(x)$ is just the marginal pdf of x.

We showed last time that this Bayes classifier $f^*(x) := \arg\max_k \eta_k(x)$ is the optimum thing to do. In other words, $R(f^*) = min \ R(f)$ where the minimum is over all possible classifiers.

To calculate the Bayes classifier.Bayes risk, we need to know $\eta_k(x)$, or equivalently $\pi_k g_k(x)$.

Proof: Consider an arbitrary classifier f and denote the decision regions

$$\Gamma_k(f) := \{x : f(x) = k\}$$

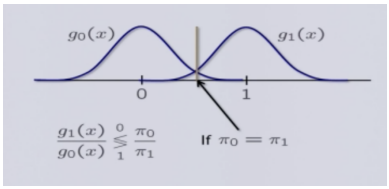
In English, for every x the classifier gives us a label. This $\gamma_k(f)$ sets are one way to define this classifier. f is breaking up the space of $\mathbb{R}^d$ into these subsets. The reason why this is useful is that this is exactly what you would kind of need to know if you calculate the probability of error. And we said rather than calculating probability of error we calculate:

$$\begin{aligned} 1 - R(f) &= \mathbb{P}[f(X) = Y] \\ &= \sum_{k=0}^{K-1} \pi_k \ \mathbb{P}[f(X) = k | Y = k] \\ &= \sum_{k=0}^{K-1} \pi_k \int_{\Gamma_k(f)} g_k(x) dx \\ &= \int_{\mathbb{R}^d} \sum_{k=0}^{K-1} \pi_k g_k(x) 1_{\Gamma_k(f)}(x) dx \end{aligned}$$

To maximize this, we should select f such that

$$x \in \Gamma_k(f) \ \leftrightarrow \ \pi_k g_k(x) \text{ is maximized}$$

Therefore, the optimal f: $f^*(x) := \arg\max_k \pi_k g_k(x)$, or equivalently $f^*(x) := \arg\max_k \eta_k(x)$.



When could the Bayes risk be zero?

Generative models

All we have covered so far requires the knowledge of the joint distribution of (X, Y). But in learning set up, all we know is the training data $(x_1, y_1), ..., (x_n, y_n)$. A generative model is an assumption about the unknown distribution.

- typically **parametric**

- build classifier by estimating the parameters via training data
- plug the result into formula for Bayes classifier.

1. Linear Discriminant Analysis

We assume $X|Y = k \ \sim \mathcal{N}(\mu_k, \Sigma)$ for $k = 0, ..., K-1$ Notice in this assumption the only thing changes between each class is the mean. Each class has the same covariance matrix Σ.

Here $\mathcal{N}(\mu_k, \Sigma)$ is multivariate normal distribution with:

$$\phi(x : \mu, \Sigma) = \frac{1}{(2\pi)^{d/2} \ |\Sigma|^{1/2}} \ exp\{-\frac{1}{2}(x - \mu)^T \ \Sigma^{-1} \ (x - \mu)\}$$

In LDA we assume that the prior probabilities π_k, the mean vectors μ_k, and the covariance matrix Σ are all known. That means we can just plug this into our previous formula for the Bayes Classifier (this is why LDA is one of the plug-in methods).

To estimate these from the data, we use

$$\begin{aligned} \hat{\pi}_k &= \frac{|\{i : y_i = k\}|}{n} \\ \hat{\mu}_k &= \frac{\sum_{i:y_i=k} x_i}{|\{i : y_i = k\}|} \\ \hat{\Sigma} &= \frac{1}{n} \sum_{k=0}^{K-1} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T \end{aligned}$$

$\hat{\Sigma}$ is pooled covariance estimate. The LDA classifier is then

$$\begin{aligned} f(x) &= \arg\max_k \hat{\pi}_k \ \phi(x : \hat{\mu}_k, \hat{\Sigma}) \\ &= \arg\max_k \ log \ \hat{\pi}_k \ + \ log \ \phi(x : \hat{\mu}_k, \hat{\Sigma}) \\ &= \arg\max_k \ log \ \hat{\pi}_k \ - \ \frac{1}{2}(x - \hat{\mu}_k)^T \hat{\Sigma}^{-1}(x - \hat{\mu}_k) \\ &= \arg\min_k \ (x - \hat{\mu}_k)^T \hat{\Sigma}^{-1}(x - \hat{\mu}_k) - 2 \ log \ \hat{\pi}_k \end{aligned}$$

In literature, Mahalanobis distance (between x and μ) is defined as:

$$d_M(x : \mu, \Sigma) = \sqrt{(x - \hat{\mu}_k)^T \hat{\Sigma}^{-1}(x - \hat{\mu}_k)}$$

Suppose $K = 2$, then

$$d_M(x : \hat{\mu}_0, \hat{\Sigma}) - 2 \ log \ \hat{\pi}_0 \ \underset{1}{\overset{0}{\lessgtr}} \ d_M(x : \hat{\mu}_1, \hat{\Sigma}) - 2 \ log \ \hat{\pi}_1$$

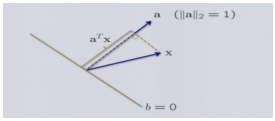
It turns out that by setting

$$\begin{aligned} a &= \hat{\Sigma}^{-1}(\hat{\mu}_0 - \hat{\mu}_1) \\ b &= -\frac{1}{2} \ \hat{\mu}_0^T \hat{\Sigma}^{-1} \hat{\mu}_0 + \frac{1}{2} \ \hat{\mu}_1^T \hat{\Sigma}^{-1} \hat{\mu}_1 + log \ \frac{\hat{\pi}_0}{\hat{\pi}_1} \end{aligned}$$

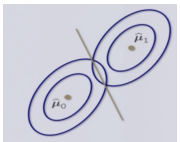
we can re-write the test as:

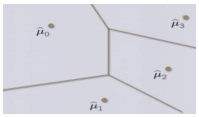
$$a^T x + b \ \underset{1}{\overset{0}{\lessgtr}} \ 0$$

which in general describes a linear classifier.



The contour $\{x : d_M(x : \mu, \Sigma) = c\}$ is an ellipse





The decision will be a straight line as a consequence of the shared covariance assumption.

What happens if you have more than two classes? The **decision regions** are convex polytopes (intersections of linear half spaces).

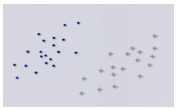
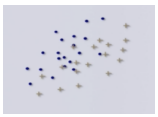
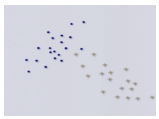




Figure 1: Which sets LDA appropriate?

Weakness of LDA:

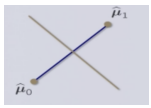
1. The generative model is rarely valid
2. A lot of methods rely on Gaussian distributions are susceptible to outliers.
3. How much data do we need to have idea that this will work? The number of parameters to be estimated is
   - class prior probabilities: $K-1$
   - means: Kd
   - covariance matrix: $\frac{1}{2}d(d+1)$

   If d is relatively small and n is large, then we can accurately estimate these parameters (using Hoeffding).

   But n is small an d is large, than we have more unknowns than observations, and will likely obtain poor estimate.

   - as a remedy to this, maybe we can apply dimensionality reduction technique like PCA to reduce d
   - or assume a more structured covariance matrix
     – An example of structured covariance matrix: assume $\Sigma = \sigma^2 I$ and estimate $\sigma^2 = \frac{1}{d}tr(\hat{\Sigma})$
     – If $K = 2$ and $\hat{\pi_0} = \hat{\pi_1}$, then LDA becomes "nearest centroid classifier":

$$\frac{1}{\hat{\sigma^2}}\|x-\hat{\mu}_0\|^2 \underset{1}{\overset{0}{\lessgtr}} \frac{1}{\hat{\sigma^2}}\|x-\hat{\mu}_1\|^2 \implies \|x-\hat{\mu}_0\|^2 \underset{1}{\overset{0}{\lessgtr}} \|x-\hat{\mu}_1\|^2$$



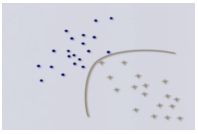
2. Quadratic Discriminant Analysis

We may expand the generative model to

$X|Y=k \ \sim \mathcal{N}(\mu_k, \Sigma_k)$ for $k = 0, ..., K-1$

Set $\hat{\Sigma}_k = \frac{1}{|\{i:\ y_i=k\}|} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T$

Proceed as before, but this case the decision boundaries will be quadratic.



Here in this figure, there are two data sets with Gaussian distribution but have different covariance matrices.

Are we asking too much with plugin method? Let's have another look:

Suppose $K = 2$

Define $\eta(x) = \eta_1(x)$ which is $(= 1 - \eta_0(x))$

In this case, another way to express Bayes classifier is as

$$f^*(x) = \begin{cases} 1 & if\ \eta(x) \geq 1/2 \\ 0 & if\ \eta(x) < 1/2 \end{cases}$$

Note that to express the classifier in this way, we don't actually need to know the full distribution of (X, Y). All we have to know is the distribution of $Y|X$.

Recap so far: A generative model is an assumption about the unknown distribution

- typically parametric
- build classifier by estimating the parameters via training data
- plug the result into formula for Bayes classifier
- often called "plug-in" methods

By implementing LDA we are using training data to estimate what the distribution is, and just plugging that into Bayes classifier rule. LDA basically says, let's assume that our data is drawn from a Gaussian distribution, estimate the parameters (every class has different mean, they all have the same covariance matrix:

$$\hat{\pi}_k = \frac{|\{i : y_i = k\}|}{n}$$
$$\hat{\mu}_k = \frac{1}{|\{i : y_i = k\}|} \sum_{i:y_i=k} x_i$$
$$\hat{\Sigma} = \frac{1}{n} \sum_{k=0}^{K-1} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T$$

Suppose $K = 2$, then

$$d_M^2(x : \hat{\mu}_0, \hat{\Sigma}) - 2\ log\ \hat{\pi}_0 \underset{1}{\overset{0}{\lessgtr}} d_M^2(x : \hat{\mu}_1, \hat{\Sigma}) - 2\ log\ \hat{\pi}_1$$

It turns out that by setting

$$w = \hat{\Sigma}^{-1}(\hat{\mu}_1 - \hat{\mu}_0)$$
$$b = \frac{1}{2}\ \hat{\mu}_0^T \hat{\Sigma}^{-1} \hat{\mu}_0 - \frac{1}{2}\ \hat{\mu}_1^T \hat{\Sigma}^{-1} \hat{\mu}_1 + log\ \frac{\hat{\pi}_0}{\hat{\pi}_1}$$

we can re-write the test as:

$$w^T x + b \underset{1}{\overset{0}{\lessgtr}} 0$$

which in general describes a **linear classifier**. We talked a little bit of the challenges in the context of LDA:

- The generative model is rarely valid
- It is susceptible to outliers–as like the other methods that rely on Gaussian distributions a. If you have paid close attention to the homework, you may have noticed that the estimate of the covariance can be very sensitive to outliers.
- The number of parameters to be estimated can get quite big (maybe you observed this with our first homework problem)
  – class prior probabilities: $K-1$
  – means: Kd
  – covariance matrix: $\frac{1}{2}d(d+1)$

  If d is relatively small and number of observations n is large, then we can accurately estimate these parameters (using Hoeffding).

  But n is small an d is large, than we have more unknowns than observations, and will likely obtain poor estimate.

  There are ways to deal with these challenges.

- as a remedy to this, maybe we can apply dimensionality reduction technique like PCA to reduce d
- or assume a more structured covariance matrix
  * An example of structured covariance matrix: assume $\Sigma = \sigma^2 I$ and estimate $\sigma^2 = \frac{1}{d}tr(\hat{\Sigma})$
  * If $K = 2$ and $\hat{\pi_0} = \hat{\pi_1}$, then LDA becomes "nearest centroid classifier":

$$\frac{1}{\hat{\sigma^2}}\|x-\hat{\mu}_0\|^2 \underset{1}{\overset{0}{\lessgtr}} \frac{1}{\hat{\sigma^2}}\|x-\hat{\mu}_1\|^2 \implies \|x-\hat{\mu}_0\|^2 \underset{1}{\overset{0}{\lessgtr}} \|x-\hat{\mu}_1\|^2$$

But even with that, we probably should avoid ever trying to estimate this covariance matrix to begin with. Maybe we are simply asking too much with this plugin method (?) Let's have another look:

Suppose $K = 2$

Define $\eta(x) = \eta_1(x)$ which is $(= 1 - \eta_0(x))$

In this case, Bayes classifier is

$$f^*(x) = \begin{cases} 1 & if\ \eta(x) \geq 1/2 \\ 0 & if\ \eta(x) < 1/2 \end{cases}$$

What we are trying to do with LDA is to estimate the full distribution of (X, Y). We try to estimate all these prior probabilities, conditional densities, but this is actually asking a lot more. Note that to express the classifier as we did above ($\eta(x)$ is just one function), we don't actually need to know whole distribution of the data to express the Bayes classifier. All we really need to know is the distribution of $Y|X$. The intermediate steps should not be harder than the actual solution.

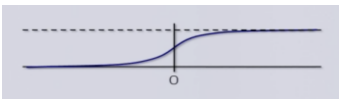
Example: Suppose again that $K = 2$ and that $X|Y = k \sim \mathcal{N}(\mu_k, \Sigma)$

$$\begin{aligned}
\eta(x) &= \frac{\pi_1\ \phi(x : \mu_1, \Sigma)}{\pi_1\ \phi(x : \mu_1, \Sigma) + \pi_0\ \phi(x : \mu_0, \Sigma)} \\
&= \frac{\pi_1\ e^{-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)}}{\pi_1\ e^{-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)} + \pi_0\ e^{-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)}} \\
&= \frac{1}{1\ +\ \frac{\pi_0}{\pi_1}\ e^{\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1) - \frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)}} \\
&= \frac{1}{1 + e^{-(w^T x + b)}}
\end{aligned}$$

If our data actually satisfies this LDA model, we know that $\eta(x)$ will have this parametric form. This observation gives rise to another class of plugin methods, the most important

which is logistic regression. These ones are far more commonly applied and far more robust than LDA.

Logistic Regression The function $\frac{1}{1+e^{-t}}$ is called logistic function (or a sigmoid function).



The basic strategy:

1. Assume $\eta(x) = \frac{1}{1+e^{-(w^T x+b)}}$ $(w \in \mathbb{R}^d,\ b \in \mathbb{R})$

   (This is all we need to know to implement Bayes classifier and we assume it is of this form. This is true for the case where our classes are both multivariate Gaussian densities with same covariance matrix. But it is actually true for a lot of different kind of densities. In other words, you can show this arises in other cases besides this specific LDA model).

2. Estimate w, b somehow from the data

3. Plug the estimate
$$\hat{\eta}(x) = \frac{1}{1+e^{-(\hat{w}^T x+\hat{b})}}$$
into the formula for Bayes classifier. Basically pretending that this is the true a posteriori.

What happens if we actually do this? Let's denote the logistic regression classifier by

$$\hat{f}(x) = 1_{\{\hat{\eta}(x)\geq 1/2\}}\ (x)$$

Again, this is a indicator function notation here.

Note that

$$\begin{aligned}\hat{f}(x) = 1 &\iff \hat{\eta}(x) \geq 1/2\\ &\iff \frac{1}{1+e^{-(\hat{w}^T x+\hat{b})}} \geq \frac{1}{2}\\ &\iff e^{-(\hat{w}^T x+\hat{b})} \leq 1\\ &\iff (\hat{w}^T x+\hat{b}) \geq 0\end{aligned}$$

So, once again we got,

$$\hat{f}(x) = \begin{cases}1 & \text{if } \hat{w}^T x+\hat{b} \geq 0\\ 0 & \text{otherwise}\end{cases}$$

To do logistic regression, obviously we have to estimate the parameters for w, b.

$$\hat{\eta}(x) = \frac{1}{1+e^{-(w^T x+b)}}$$

When we were talking about LDA we tried to estimate the giant covariance matrix (d^2 entries), but then we really didn't cared what the covariance matrix was. We only cared about what happens to w– here we have far fewer parameters then that. That's why I kind of claim logistic regression is more useful in practice. Because you only need to estimate $d+1$ parameters–we don't have to estimate d^2 things... fewer things to estimate but still don't know how we would do that? The downside will be that, they are going to be harder (no free lunch).

So far:

1. Assume $\eta(x) = \frac{1}{1+e^{-(w^T x+b)}}$ $(w \in \mathbb{R}^d,\ b \in \mathbb{R})$
2. Estimate w, b somehow from the data
3. Plug the estimate
$$\hat{\eta}(x) = \frac{1}{1+e^{-(\hat{w}^T x+\hat{b})}}$$
into the formula for Bayes classifier. Basically pretending that this is the true a posteriori.

Here we need to estimate the parameters for w, b, i.e. $d+1$ parameters.

In order to talk about this estimation, we have to take a **detour** to **maximum likelihood estimation**.

For convenience, let's let $\theta = (b, w)$

The "a posteriori" probability of our data

First let's suppose we knew θ, then we could compute

$$\begin{aligned}\mathbb{P}[y_i|x_i;\theta] &= \mathbb{P}[Y_i = y_i|X_i = x_i;\theta] \quad \text{this is just a shorthand}\\ &= \begin{cases}\eta(x_i;\theta) & \text{if } y_i = 1\\ 1-\eta(x_i;\theta) & \text{if } y_i = 0\end{cases}\\ &= \eta(x_i;\theta)^{y_i}(1-\eta(x_i;\theta))^{1-y_i}\end{aligned}$$

This is convenient way to express probability– because of conditional independence, we also have that

$$\begin{aligned}\mathbb{P}[y_1,...,y_n|x_1,...,x_n;\theta] &= \prod_{i=1}^{n}\mathbb{P}[y_i|x_i;\theta]\\ &= \prod_{i=1}^{n}\eta(x_i;\theta)^{y_i}(1-\eta(x_i;\theta))^{1-y_i}\end{aligned}$$

Flipping the mirrors: We don't actually know θ, but we do know $y_1, ..., y_n$

Suppose we view $y_1, ..., y_n$ to be fixed, and view $\mathbb{P}[y_1, ..., y_n|x_1, ..., x_n;\theta]$ as just the function of θ. In other words, if we change the parameters we are going to get a different a posteriori probability.

We actually observed the data $y_1, ..., y_n$, so you can view this function $\mathbb{P}[y_1, ..., y_n|x_1, ..., x_n;\theta]$ as telling us something about how much sense do those parameters make. So if you give me parameters θ and I plug them in and this turns out to be 0, then I know that those parameters makes no sense (again, because you actually did observe that data). On the other hand, if you have given me a θ and I plug it in, and the probability gets to be equal to be 1, yay! If those parameters are true, then this would make a lot of sense. Because if we have these parameters, given these inputs $x_1, ..., x_n$ we would expect to see this outputs $y_1, ..., y_n$ and we did see these outputs. This is the idea in maximum likelihood estimation!

When we do this, $L(\theta) = \mathbb{P}[y_1, ..., y_n|x_1, ..., x_n;\theta]$ is called the **likelihood** function.

We've got to observe some inputs and outputs, and there is some unknown parameter θ that we like to estimate. What we can do is, we can calculate the probability of these outputs given our inputs, and we are going to pick θ to maximize this likelihood. So, the method of **maximum likelihood** aims to estimate θ by finding the θ that **maximizes** the likelihood $L(\theta)$. This is not particularly surprising.

(Optional: Note that the likelihood function here is not a PDF, not a density. It is not actually computing any kind of probability , you view $L(\theta)$ as a function of θ because *theta* is not really a random quantity. θ is just some deterministic parameter we are trying to estimate. And in fact, if you vary for all the different possible θ values, in general $\mathbb{P}[y_1, ..., y_n|x_1, ..., x_n;\theta]$ will not integrate to 1. When you think about MLE, you shouldn't be thinking that we are calculating the probability of θ, we are thinking of "likelihood". This may confuse some of you since we have been talking about lots of probabilities (a priori, a posteriori, etc.), and now we are talking about the likelihood. And in English "probability" and "likelihood" pretty much mean the same thing, but in this context they are referring to very specific kind of functions. Probability is associated with random variables, in here we don't necessary think of θ being random. Likelihood of some deterministic variable conditioned upon random events occurring.)

In practice, it is often more convenient to maximize the **log-likelihood**, i.e., $log\ L(\theta)$.

Remember below is the likelihood function for our particular problem.

$$L(\theta) = \prod_{i=1}^{n}\eta(x_i;\theta)^{y_i}(1-\eta(x_i;\theta))^{1-y_i}$$

$$\begin{aligned}l(\theta) =& log\ L(\theta)\\ =& \sum_{i=1}^{n} y_i\ log\ \eta(x_i;\theta) + (1-y_i)\ log\ (1-\eta(x_i;\theta))\end{aligned}$$

In Logistic regression, we are assuming $\eta(x;\theta)$ has a particular form, and that form depends on these parameters θ. So we are going to choose θ to maximize the log-likehood. This is actually all there is to it. Let's simplify the log likelihood a bit so we can implement easier.

Notation:

- $\tilde{\mathrm{x}} = [1, x(1), ..., x(d)]^T$
- $\boldsymbol{\theta} = [b, w(1), ..., w(d)]^T$

This means that $w^T x + b = \boldsymbol{\theta}^T\tilde{\mathrm{x}}$, which lets us write

$$\eta(x_i;\boldsymbol{\theta}) = \frac{1}{1+e^{-\boldsymbol{\theta}\tilde{\mathrm{x}}}}$$

Thus, if we let $g(t) = \frac{1}{1+e^{-t}}$, the we can write

$$l(\theta) = \sum_{i=1}^{n} y_i\ log\ g(\boldsymbol{\theta}^T\tilde{\mathrm{x}})\ +\ (1-y_i)\ log\ (1-g(\boldsymbol{\theta}^T\tilde{\mathrm{x}}))$$

What is the log and $1-log$ of $g(t)$?

Facts:

$$log\ g(t) = log(\frac{1}{1+e^{-t}}) = -log\ (1+e^{-t})$$

$$\begin{aligned}log\ (1-g(t)) &= log(1-\frac{1}{1+e^{-t}})\\ &= log\ (\frac{e^{-t}}{1+e^{-t}}) = -t - log\ (1+e^{-t})\\ &= log(\frac{1}{1+e^{t}}) = -log\ (1+e^{t})\end{aligned}$$

Thus,

$$\begin{aligned} l(\theta) = & \sum_{i=1}^{n} y_i \; log \; g(\boldsymbol{\theta}^T \tilde{\mathrm{x}}_i) \; + \; (1-y_i) \; log \; (1-g(\boldsymbol{\theta}^T \tilde{\mathrm{x}})) \\ = & \sum_{i=1}^{n} -y_i \; log \; (1+e^{-\boldsymbol{\theta}^T \tilde{\mathrm{x}}_i}) - \; log \; (1+e^{\boldsymbol{\theta}^T \tilde{\mathrm{x}}_i}) - y_i(-\boldsymbol{\theta}^T \tilde{\mathrm{x}}_i - log(1+e^{-\boldsymbol{\theta}^T \tilde{\mathrm{x}}_i})) \\ = & \sum_{i=1}^{n} y_i \boldsymbol{\theta}^T \tilde{\mathrm{x}}_i - log(1+e^{\boldsymbol{\theta}^T \tilde{\mathrm{x}}_i}) \end{aligned}$$

How can we maximize

$$l(\theta) = \sum_{i=1}^{n} y_i \boldsymbol{\theta}^T \tilde{\mathrm{x}}_i - log(1+e^{\boldsymbol{\theta}^T \tilde{\mathrm{x}}_i})$$

with respect to $\boldsymbol{\theta}$? I could take the derivative and set it to 0, only here we have a bunch of things we want to solve for at once, so we need the partial derivatives.

Find a $\boldsymbol{\theta}$ such that $\nabla l(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial l(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_1} \\ .. \\ .. \\ \frac{\partial l(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_{d+1}} \end{bmatrix} = 0$ It is not hard to show that

$$\begin{aligned} l(\theta) = & \sum_{i=1}^{n} \nabla(y_i \boldsymbol{\theta}^T \tilde{\mathrm{x}}_i - log(1+e^{\boldsymbol{\theta}^T \tilde{\mathrm{x}}_i})) \\ = & \sum_{i=1}^{n} y_i \tilde{\mathrm{x}}_i - \tilde{\mathrm{x}}_i e^{\boldsymbol{\theta}^T \tilde{\mathrm{x}}_i} (1+e^{\boldsymbol{\theta}^T \tilde{\mathrm{x}}_i})^{-1} \\ = & \sum_{i=1}^{n} \tilde{\mathrm{x}}_i (y_i - g(\boldsymbol{\theta}^T \tilde{\mathrm{x}}_i)) \end{aligned}$$

Remember the dimension of $\tilde{\mathrm{x}}_i$ is $d+1$. this gave us $d+1$ equations, but they are **nonlinear** and have no-closed form solutions. We need to somehow solve an optimization problem.

Optimization

$$\min_{x \in \mathbb{R}^d} \quad f(x)$$

(or $\min_{\boldsymbol{\theta} \in \mathbb{R}^{d+1}} -log \; (\boldsymbol{\theta})$ for today)

In most cases, we cannot compute the solution simply by setting $\nabla f(x) = 0$ and solving for x on paper–who would wanna do that anyway. However, there are many powerful algorithms for finding x. And the simplest possible algorithm for this is, yes you guessed it, **Gradient Descent**.

A simple way to try to find the minimum our objective function is to iteratively roll downhill.

From initial guess x^0, take a step in the direction of the negative gradient.

Notice we know how to calculate gradient for our problem, what we didn't know is how to solve when we set it to zero.

$$\begin{aligned} x^1 &= x^0 - \alpha_0 \; \nabla f(x)|_{x=x^0}, \quad \alpha_0 : \text{“step size”} \\ x^2 &= x^1 - \alpha_1 \; \nabla f(x)|_{x=x^1} \\ &. \\ &. \\ &. \end{aligned}$$

Convergence of gradient descent The core iteration of gradient descent is to compute

$$x^{j+1} = x^j - \alpha_j \; \nabla f(x)|_{x=x^j}$$

Note that if $\nabla f(x)|_{x=x^j} = 0$, then we have found the minimum and $x^{j+1} = x^j$, so the algorithm will terminate.

It turns out that if f is convex, then the gradient descent (with a fixed step size α) is guaranteed to converge to the global minimum of f.

For convex functions with fixed step size GD will converge, eventually. But this may be a long time.

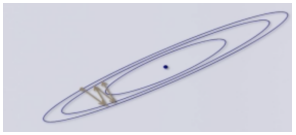


Figure 2: Not so-lucky step-size

Perhaps fixed step size perhaps not a good idea. What if I get lucky with my stepsize

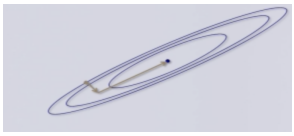


Figure 3: Lucky step-size

Step size matters. As long as you don't set the step size bigger than the norm of the gradient, you can show that it will still converge. So step seize depends on the properties of the function, like smoothness, how nicely convex it is...If you don't know anything about your function, you don't actually know for sure how to set this step size. What a lot of people do is, they kind of start α being 1, and slowly decrease it as the number of iterations go on to prevent this kind of oscillatory behavior.

There is a vast literature on intelligent methods for computing what the right step size is at the each step size of the algorithm. The most obvious thing to do is line search, of course there is a computational downside! There is also lots of tricks in literature to how to this optimally. There is also another approach most likely you have seen more than once in calculus, Newton's method–finding the roots of a function.

Also known as the Newton-Raphson method, this approach can be viewed as simply using the second derivative (Hessian matrix) to automatically select an appropriate step size.

$$x^{j+1} = x^j - (\nabla^2 f(x))^{-1} \; \nabla f(x)|_{x=x^j}$$

Newton's method, if you can do it, tends to be very powerful–if we can invert the Hessian matrix $(\nabla^2 f(x))^{-1}$. In most cases this could not be feasible however for logistic regression it does work. We can compute this.

The negative log-likelihood in logistic regression is a convex function. That means gradient descent guaranteed to converge, if we have the right step size. But also Newton's method can work. So both gradient descent and Newton's method are common strategies for setting the parameters in logistic regression.

Newton's method, typically, is much faster when the dimension d is small, but is impractical when d is large, order d^3 computation and there is also memory concerns. Can you even form the Hessian matrix?