ECE 0402 - Pattern Recognition

Lecture 5 on 1/31/2022

Review: 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

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

It turns out that by setting

Typo in previous lecture

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

we can re-write the test as:

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

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":

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

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

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

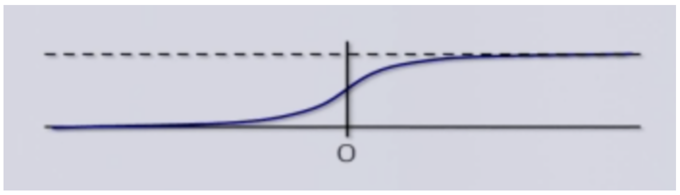
What we are trying to do with LDA is to estimate the full distribution of (X, Y). We try to estimate all these prior probabilities, conditional densities, but this is actually asking a lot more. Note that to express the classifier as we did above ($\eta(x)$ is just one function), we don't actually need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

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

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

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

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

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

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

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

Note that $\eta(x)$ is really function of both x and θ, so we will use the notation $\eta(x;\theta)$ to highlight this dependence.

The "a posteriori" probability of our data

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 this probability– because of conditional independence, we also have that

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

Turning the table: We don't actually know θ, but we actually 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.

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

> **maximum likelihood** aims to estimate θ by finding the θ that **maximizes** the likelihood $L(\theta)$.

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.

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

Since the *log* is a monotonic transformation, maximizing $l(\theta)$ is equivalent to maximizing $L(\theta)$.

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

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 $\boldsymbol{\theta}$?

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. 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.

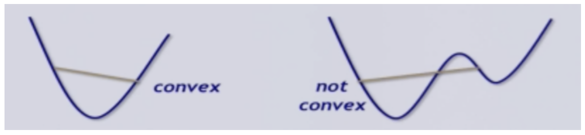
$$\begin{aligned} x^1 &= x^0 - \alpha_0 \ \nabla f(x)|_{x=x^0}, \qquad \alpha_0 : \text{"step size"} \\ x^2 &= x^1 - \alpha_1 \ \nabla f(x)|_{x=x^1} \\ &\ \ . \\ &\ \ . \\ &\ \ . \end{aligned}$$

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.

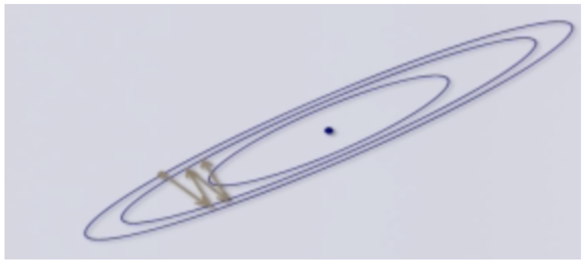


Figure 1: Not so-lucky step-size

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

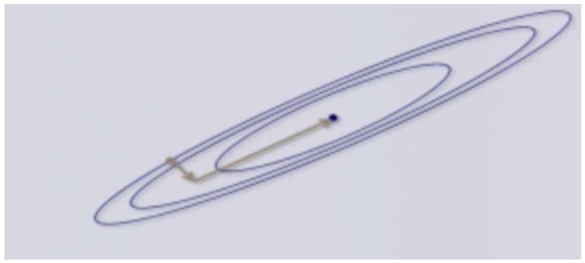


Figure 2: Lucky step-size

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.

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

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. **Why?**

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? (–maybe more on this next week's homework)

Compare:

- LDA and LR are two different methods that result in linear classifier.
- LDA is best if Gaussianity assumptions are valid.
- LR models only the distribution of $X|Y$, not (X, Y)
  - valid of a larger class of distributions
  - fewer parameters to estimate
- There is another plugin method called Naive-Bayes (this will be covered later in the class)
  - based on density estimation
  - scales well to high-dimensions and naturally handles mixture of discrete and continuous features

Beyond plugin methods: Plugin methods are useful and worth knowing about, but ultimately they are very limited.

- There are always distributions where our assumptions are violated
- If our assumptions are wrong, the output is totally unpredictable
- It is also hard to verify whether our assumptions are right
- Require solving a more difficult problem as an intermediate step (more true with Naive-Bayes).

  For example, we don't need to know whole distribution of our data, all we need is $\eta(x)$, even that's not true. We actually don't need to know $\eta(x)$ except for where it is equal to half, just need to know one boundary. All we need to do is estimate one boundary.

For most of the remainder of this course will focus on **nonparametric** methods that avoid such strong assumptions about the (unknown) process generating the data.