ECE 0402 - Pattern Recognition

Lecture 4 on 2/1/2024

Review:

- 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.

Starting today's lecture, and most of the remainder of this course, we will focus on "nonparametric" methods that avoid making such strong assumptions about the (unknown) process generating our data.

Non-parametric linear classifiers

Let $K = 2$ and $Y \in \{-1, +1\}$ and $\tilde{x} = [1, x(1), x(2), ..., x(d)]^T$, we can express linear classifier compactly by comparing $\theta^T \tilde{x}$ to a threshold for some θ.

$$y = \begin{cases} 1 & \text{if } \theta^T \tilde{x} > \text{threshold} \\ -1 & \text{otherwise} \end{cases}$$

$$y = \text{sign}(\theta^T \tilde{x})$$

How do we actually learn θ if we are not making any assumptions on probability distributions that are generating our data?

Perceptron Learning Algorithm – Frank Rosenblatt (1957)

Given

- training data $\{x_i, y_i\}_{i=1}^n$
- a guess for θ^0

Pick any observation (i.e., i), and update according to

$$\theta^{j+1} = \begin{cases} \theta^j + y_i \tilde{x}_i & \text{if } y_i \neq sign((\theta^j)^T \tilde{x}) \\ \theta^j & \text{otherwise} \end{cases}$$

And repeat this for every training points, and that's it!

Why might this work?

Suppose θ we learned is not correctly classifying, what happens after we do our update $(\theta^j + y_i \tilde{x}_i)$:

$$\begin{aligned} (\theta^{j+1})^T \tilde{x}_i &= (\theta^j + y_i \tilde{x}_i)^T \tilde{x}_i \\ &= (\theta^j)^T \tilde{x}_i + y_i \tilde{x}_i^T \tilde{x}_i \end{aligned}$$

You can intuitively see that the term $y_i \tilde{x}_i^T \tilde{x}_i$ pushing you in the right direction. To see this more clearly, we can run it thru the two possibilities:

- If $y_i = 1$ and $sign((\theta^j)^T \tilde{x}_i) = -1$, the update results in bigger $(\theta^{j+1})^T \tilde{x}_i$
- Similarly, if $y_i = -1$ and $sign((\theta^j)^T \tilde{x}_i) = 1$, the update makes $(\theta^{j+1})^T \tilde{x}_i$ smaller

There is another way of looking at this >> the core iteration of the PLA is:

$$\theta^{j+1} = \begin{cases} \theta^j + y_i \tilde{x}_i & \text{if } y_i \neq sign((\theta^j)^T \tilde{x}) \\ \theta^j & \text{otherwise} \end{cases}$$

Rather than these update rules, we can re-write this update rules as,

$$\theta^{j+1} = \theta^j + \frac{y_i - sign((\theta^j)^T \tilde{x})}{2} \tilde{x}_i$$

Does this look familiar?

If not, it will be with homework 2 (will be posted after lecture today). In the homework you will implement a "stochastic gradient descent" for logistic regression where each update is of the form

$$\theta^{j+1} = \theta^j + \alpha(y_i - g((\theta^j)^T \tilde{x}_i)) \cdot \tilde{x}_i$$

This is in contrast with the PLA which consists of updates of the form

$$\theta^{j+1} = \theta^j + \frac{1}{2}(y_i - sign((\theta^j)^T \tilde{x}_i)) \cdot \tilde{x}_i$$

We can view the PLA as simply stochastic gradient descent applied to slightly different likelihood function than we considered in the context of logistic regression.

We can also provide simple proofs that, under certain assumptions (nonparametric), the PLA will result in a good classifier. In particular, we will see that if the training data is **linearly separable**, then the PLA will find a **separating hyperplane** in a finite number of iterations. It always converges!

Linearly Separable

We say that a data set $\{(x_i, y_i)\}_{i=1}^n$ is linearly separable if there exists $\omega \in \mathbb{R}^d$ and $b \in \mathbb{R}$ such that

$$y_i = sign(\mathrm{w}^T x_i + b), \quad \text{for } i = 1, ..., n$$

We refer to $\{x : \mathrm{w}^T \mathrm{x} + b = 0\}$ as **a separating hyperplane** Set notation is intentional here.

How can we find a separating hyperplane?

One approach is to use the PLA. If a separating hyperplane exists for linearly separable data, perceptron learning algorithm can find it. You will prove this in the next homework, to see roughly how the proof works, we need to do a bit of basic geometry first.

Let w, b define a hyperplane, and pick two points x^1, x^2

$$\begin{aligned} 0 &= (\mathrm{w}^T \mathbf{x^1} + b) - (\mathrm{w}^T \mathbf{x^2} + b) \\ &= \mathrm{w}^T (\mathrm{x}^1 - \mathrm{x}^2) \end{aligned}$$

Hence, w is orthogonal to all vectors that are parallel to the hyper plane. We call $\frac{\mathrm{w}}{\|\mathrm{w}\|}$ the normal vector to the hyperplane.This is a unique thing (up to its sign).

Let $z \in \mathbb{R}^d$ be an arbitrary vector. How far z from $\{x \in \mathbb{R}^d : \mathrm{w}^T \mathrm{x} + b = 0\}$? This is the key question, that is going to allow us make more concrete statements about separating hyperplanes, and when we find a good hyperplane that separates our dataset.

- Decompose z into two components: $z = z_0 + \delta \frac{\mathrm{w}}{\|\mathrm{w}\|}$ where $\mathrm{w}^T z_0 + b = 0$ and $\delta \in \mathbb{R}$
- we can find a formula for the distance by observing that

$$\begin{aligned} \mathrm{w}^T z + b &= \mathrm{w}^T (z_0 + \delta \frac{\mathrm{w}}{\|\mathrm{w}\|}) + b \\ &= \mathrm{w}^T z_0 + b + \delta \frac{\mathrm{w}^T \mathrm{w}}{\|\mathrm{w}\|} \\ &= \delta \|\mathrm{w}\| \end{aligned}$$

Hence,

$$| \delta | = \frac{| \mathrm{w}^T z + b |}{\|\mathrm{w}\|}$$

Why is this useful?

Theorem: If a separating hyperplane exists the the PLA will find one in finitely many iterations.

Let θ^* define a separating hyperplane normalized so that

$$\rho = min_i \mid (\theta^*)^T x_i \mid = min_i \mid (\mathrm{w}^*)^T x_i + b^* \mid$$

calculates the distance from the hyperplane to the closest x_i in the entire training data (ρ is usually called margin), and let

$$R = max_i \ \|x_i\| \tag{1}$$

(R is the norm of the largest x_i).

Under these assumptions, you will show that if at <u>iteration j</u> there exits an x_i such that $y_i \neq sign((\theta^j)^T \tilde{x}_i)$, then we necessarily have

$$j \leq \frac{(R^2+1)\,\|\theta^*\|^2}{\rho^2}$$

In other words, if

$$j > \frac{(R^2+1)\,\|\theta^*\|^2}{\rho^2}$$

then we must have found a separating hyperplane–or in other words, we can't find points that are misclassified.

Drawbacks:

- we don't know $\frac{\|\theta^*\|^2}{\rho^2}$
- if the data set is not separable it may jump around forever
- even your data set is linearly separable, all this is saying is that you will get to a point where you will find "a" separating hyperplane. You will eventually find some line that separates your data (in R^2). But a separating hyperplane may not be the **best** hyperplane.

The maximum margin hyperplane

The margin ρ of separating hyperplane is the distance form the hyperplane to closest x_i

$$\rho(\mathrm{w}, b) = min_i \ \frac{\mid \mathrm{w}^T x_i + b \mid}{\|\mathrm{w}\|_2}$$

The maximum margin or **optimal separating hyperplane** is the solution to

$$(\mathrm{w}^*, b^*) = \arg\max_{\mathrm{w},b} \rho(\mathrm{w}, b)$$

$$\text{Larger margin} \implies \text{better generalization to new data}$$

This is just an optimization problem but if you think about actually achieve this maximum margin hyperplane, the first problem you will get into will be it has a few to many degrees of freedom. And that's because whenever we talk about w and b together, there is lots of different w and b that refer to exact same hyperplane. More precisely our parameterization of a hyperplane as a normal vector w and an offset b is overdetermined

- e.g., in $\mathbb{R}^2$ we are using three parameters to describe a line which really only needs two

We are always going to have one extra parameter and the reason why is to realize that

$$\{x : (\alpha \mathrm{w})^T x + \alpha b = 0\}$$

describes the same hyperplane for all $\alpha \in \mathbb{R}$–didn't change the position of the hyperplane. It is often useful to remove this ambiguity by choosing a particular scaling by normalizing it out.

In the case of a separating hyperplane, we will say w, b are in canonical form if they satisfy the following properties:

- $y_i(\mathrm{w}^T x_i + b) \geq 1$ for all i //[we are going to scale it]
- $y_i(\mathrm{w}^T x_i + b) = 1$ for some i //[but we are not going to scale it more than we have to!]

If we restrict ourselves to hyperplanes in canonical form, we can re-write ρ as,

$$\rho(\mathrm{w}, b) = min_i \ \frac{\mid \mathrm{w}^T x_i + b \mid}{\|\mathrm{w}\|_2} = \frac{1}{\|\mathrm{w}\|_2}$$

Thus we can express $(\mathrm{w}^*, b^*) = \arg\max_{\mathrm{w},b} \ \rho(\mathrm{w}, b)$ as

$$\begin{aligned}(\mathrm{w}^*, b^*) = &\arg\min_{\mathrm{w},b} \frac{1}{2}\|\mathrm{w}\|^2 \\ &\text{s.t.} \ \ y_i(\mathrm{w}^T x_i + b) \geq 1 \ \ \text{for all } i = 1, 2, ..., n\end{aligned}$$

This is now a **constrained** optimization problem, much easier one to solve! We will talk all about this when we start Kernel methods.

- At the solution, there will always be at least some x such that $y_i(\mathrm{w}^T + b) = 1$. These are called **support vectors**.

Not Linearly separable

The plugin methods can naturally accommodate data that isn't perfectly linearly separable

Can we extend the notion of an optimal separating hyperplane to the case where the data is not linearly separable? This is important because even if your data is linearly separable, you may not want to focus all your attention on a few outliers.

The constraint that $y_i(\mathrm{w}^T x_i + b) \geq 1$ for every training sample can only be satisfied for separable data. (This is the constraint that forcing us to come up with a hyperplane that perfectly separates our data, and it is constraint with every single outlier)

Fairly intuitive **idea** here is to introduce "slack variables" $\xi_1, ..., \xi_n \geq 0$ that allows us to violate some of the constraints by changing them to

$$y_i(\mathrm{w}^T x_i + b) \geq 1 - \xi_i$$

We will have one ξ_i for every pair of training data. We are allowing ourselves some freedom to weaken these constraints.

Now we have another issue: how should we set $\xi_1, ..., \xi_n$?

Optimal soft-margin hyperplane

Ideally we would like the most of the ξ_i to be zero. That means we are really separating two classes in our data. Maybe some of them are not going to be zero, how should we penalize that? Note that the only way if x_i is misclassified is when $\xi_i > 1$.

Hence our **training error** $\leq \frac{1}{n}\sum_{i=1}^n \xi_i$

This suggests that we want to be robust to some outliers–some of the outliers are allowed to be nonzero– but we want the overall training error still be small. So we don't want this sum to get really big. That what new optimization problem will let us to do.

The optimal **soft-margin** hyperplane is given by

$$\begin{aligned}(\mathrm{w}^*, b^*) = &\arg\min_{\mathrm{w},b} \frac{1}{2}\|\mathrm{w}\|^2 + \frac{C}{n}\sum_{i=1}^n \xi_i \\ &\text{s.t.} \ \ y_i(\mathrm{w}^T + b) \geq 1 - \xi_i \quad i = 1, ..., n \\ &\qquad\quad \xi_i \geq 0 \qquad\quad i = 1, ..., n\end{aligned}$$

You get to set C, this basically lets you handle the tradeoff: how sensitive you want to be to outliers? how much emphasis you wanna put on maximizing the margin versus being a little bit looser about letting some of the constraints getting violated. If you make C really big, this reduces to the previous problem we looked at (some calls it hard margin hyperplane).

And notice that if $C = 0$ this problem does not make sense–you have all this freedom to violate any of the constraints and if you have no penalty associated with that.

In other words, C allows us to trade off between fitting the data and having a large "margin". It also control the influence of outliers.

There are a lot of linear classifiers (variant of the perceptron/single layer neural nets) out there that I am not planning on necessarily talking about, at least for a little while:

- least square approaches
- linear programming approaches
- Bayesian approaches
- Naive Bayes
- ...

Sometimes linear classifiers are terrible! One way to create nonlinear estimators or classifiers is to first transform the data via nonlinear feature map

$$\Phi : \mathbb{R}^d \to \mathbb{R}^d$$

Basically, you can always think about taking your data sets and mapping it up into a higher dimensional space where you compute other features–and suddenly your data is well separated by a hyperplane.

After applying Φ we can then try applying a linear method to the transformed data $\Phi(x_1), ..., \Phi(x_n)$