PERCEPTRON LEARNING ALGORITHM

Matthieu R Bloch

January 24, 2019

LOGISTICS

Lecture slides and notes

- Lecture 4 slides corrected for typos
- Partial Lecture 4 notes available (still missing Naive Bayes details)
- Detailed Lecture 3 notes available (clarified what's advanced material)
- Report typos and errors on Piazza (thank you!)

Self-assignment

- Make sure you define your *notation*! We can't guess for you
- Try to avoid "one can show that" without reference

Problem set #1

- Still being finalized
- Don't worry, you'll have plenty of time to complete it
- Theoretical problems + programming

RECAP: LDA

Consider $\mathbf{x} = [x_1, \cdots, x_d]^\intercal \in \mathbb{R}^d$ (random) feature vector and y the label

- *Assumption:* Given y, the feature vector have a Gaussian distribution $P_{\mathbf{x}|y} \sim \mathcal{N}(\boldsymbol{\mu}_k, \Sigma)$
- The mean is class dependent but the covariance matrix is class *independent*

$$\phi(\mathbf{x}; \boldsymbol{\mu}, \Sigma) \triangleq \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\intercal \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

Estimate parameters $\hat{\pi}_k, \hat{\boldsymbol{\mu}}_k, \hat{\Sigma}$ from data

Lemma.

The LDA classifier is

$$h^{\text{LDA}}(\mathbf{x}) = \underset{k}{\text{argmin}}\left(\frac{1}{2}(\mathbf{x}-\hat{\boldsymbol{\mu}}_k)^\intercal \hat{\Sigma}^{-1}(\mathbf{x}-\hat{\boldsymbol{\mu}}_k) - \log \hat{\pi}_k\right)$$

For $K = 2$, the LDA is a *linear* classifier

$$\eta(\mathbf{x}) \triangleq \eta_1(\mathbf{x}) = \frac{\pi_1 \phi(\mathbf{x}; \boldsymbol{\mu}_1, \Sigma)}{\pi_1 \phi(\mathbf{x}; \boldsymbol{\mu}_1, \Sigma) + \pi_0 \phi(\mathbf{x}; \boldsymbol{\mu}_0, \Sigma)} = \frac{1}{1 + \exp(-(\mathbf{w}^\intercal \mathbf{x} + b))}$$

RECAP: LOGISTIC REGRESSION

Assume that $\eta(\mathbf{x})$ is of the form $\frac{1}{1+\exp(-(\mathbf{w}^\intercal \mathbf{x}+b))}$

Estimate $\hat{\mathbf{w}}$ and $\hat{b}$ from the data directly

Plugin the result to obtain $\hat{\eta}(\mathbf{x}) = \frac{1}{1+\exp(-(\hat{\mathbf{w}}^\intercal \mathbf{x}+\hat{b}))}$

The function $x \mapsto \frac{1}{1+e^{-x}}$ is called the *logistic function*

The binary logistic classifier is $h^{\text{LC}}(\mathbf{x}) = \mathbf{1}\{\hat{\eta}(\mathbf{x}) \geq \frac{1}{2}\} = \mathbf{1}\{\hat{\mathbf{w}}^\intercal \mathbf{x} + \hat{b} \geq 0\}$ (*linear*)

Direct estimation of $(\hat{\mathbf{w}}, b)$ from maximum likelihood

- No closed form expression for MLE

GRADIENT DESCENT

Consider the canonical problem

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) \text{ with } f:\mathbb{R}^d \to \mathbb{R}$$

Find minimum by find iteratively by "rolling downhill"

- Start from point $\mathbf{x}^{(0)}$
- $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \eta\nabla f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^{(0)}}$; η is the *step size*
- $\mathbf{x}^{(2)} = \mathbf{x}^{(1)} - \eta\nabla f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^{(1)}}$
- $\cdots$
- Choice of step size η really matters: too small and convergence takes forever, too big and might never converge

Many variants of gradient descent

- Momentum: $v_t = \gamma v_{t-1} + \eta\nabla f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^{(t)}}$ and $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - v_t$
- Accelerated: $v_t = \gamma v_{t-1} + \eta\nabla f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^{(t)}-\gamma v_{t-1}}$ and $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - v_t$

In practice, gradient has to be evaluated from *data*

NEWTON'S METHOD

Newton-Raphson method use the second derivative to automatically adapt step size

$$\mathbf{x}^{(j+1)} = \mathbf{x}^{(j)} - [\nabla^2 f(\mathbf{x})]^{-1}\nabla f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_j}$$

Hessian matrix

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix}$$

Newton's method is much faster when the dimension d is small but impractical when d is large

STOCHASTIC GRADIENT DESCENT

Often have a loss function of the form $\ell(\theta) = \sum_{i=1}^N \ell_i(\theta)$ where $\ell_i(\theta) = f(\mathbf{x}_i, y_i, \theta)$

The gradient is $\nabla_\theta \ell(\theta) = \sum_{i=1}^N \nabla\ell_i(\theta)$ and gradient descent update is

$$\theta^{(j+1)} = \theta^{(j)} - \eta\sum_{i=1}^N \nabla\ell_i(\theta)$$

- Problematic if dataset if huge of if not all data is availabel
- Use *iterative* technique instead

$$\theta^{(j+1)} = \theta^{(j)} - \eta\nabla\ell_i(\theta)$$

Tons of variations of principle

- Batch, minibatch, Adagrad, RMSprop, Adam, etc.

LINEARLY SEPARABLE DATASETS

Definition (Linearly separable)

Dataset $\{\mathbf{x}_i, y_i\}_{i=1}^N$ is *linearly separable* if there exists $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ such that

$$\forall i \in [1;N] \quad y_i = \operatorname{sgn}(\mathbf{w}^\intercal\mathbf{x} + b) \qquad y_i \in \{\pm 1\}$$

By definition $\operatorname{sgn}(x) = +1$ if $x > 0$ and -1 else. The affine set $\{\mathbf{x} : \mathbf{w}^\intercal\mathbf{x} + b = 0\}$ is called a *separating hyperplane*.

A bit of geometry…

Lemma.

Consider the hyperplane $\mathcal{H} \triangleq \{\mathbf{x} : \mathbf{w}^\intercal\mathbf{x} + b = 0\}$. The vector $\mathbf{w}$ is orthogonal to all vectors parallel to the hyperplane. For $\mathbf{z} \in \mathbb{R}^d$, the distance of $\mathbf{z}$ to the hyperplane is

$$d(\mathbf{z}, \mathcal{H}) = \frac{|\mathbf{w}^\intercal\mathbf{z} + b|}{\|\mathbf{w}\|_2}$$

How can we find a separating hyperplane? (assuming it exists…)

PERCEPTRON LEARNING ALGORITHM

Consider $\mathbf{x} = [1\, x_1, \cdots, x_d]^\intercal \in \mathbb{R}^{d+1}$ and hyperplane of the form $\theta^\intercal \mathbf{x} = 0$.

Data $\mathcal{D} \triangleq \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ such that $y_i \in \{\pm 1\}$

Perceptron Learning Algorithm (PLA) invented by Rosenblatt in 1958 to find separating hyperplanes

- Start from guess for $\theta^{(0)}$ and go over data points in sequence to update

$$\theta^{(j+1)} = \begin{cases} \theta^{(j)} + y_i \mathbf{x}_i \text{ if } y_i \neq \operatorname{sgn}\left(\theta^{(j)\intercal} \mathbf{x}_i\right) \\ \theta^{(j)} \text{ else} \end{cases}$$

- Geometric intuition behind operation
- Stochastic gradient descent view

PLA finds a *non parametric* linear classifier

- Can be viewed as single layer NN

Theorem.

PLA finds a separating hyperplane if the data is linearly separable