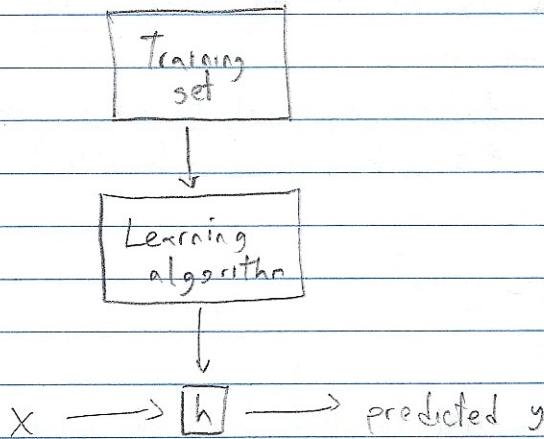


Lecture notes 1: Supervised Learning and Discriminative Algorithms

- Let $x^{(i)}$ denote the input variables or features and $y^{(i)}$ denote the output or target variable that we are trying to predict
- A pair $(x^{(i)}, y^{(i)})$ is called a training example
- The dataset that we use to learn is a list of m training examples $\{(x^{(i)}, y^{(i)}): i=1, \dots, m\}$ and it is called a training set
- NOTE: the superscript "(i)" is simply an index into the training set; it has nothing to do with exponentiation!
- We use X_f to denote the space of all possible input values and Y_f to denote the space of all possible output values
- The supervised learning problem consists in, given a training set, learning function $h: X \rightarrow Y$ so that $h(x)$ is a "good" predictor for the corresponding value of y .
- h is called a hypothesis.



- When the target variable y is continuous, the learning problem is called a regression problem
- When the target variable y can take on only a small number of discrete values, we call the learning problem a classification problem

§1: Linear Regression

- When the target variable $X^{(i)}$ is multi-dimensional, $X_j^{(i)}$ denotes the j^{th} feature of the i^{th} training example
- To perform supervised learning, we must decide how we are going to represent functions/hypotheses h
- As an initial choice, suppose we approximate y as a linear function of x :

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

- The θ 's are the parameters (or weights) parametrizing the space of linear functions mapping from X to Y .
- To simplify our notation, we introduce the convention of letting $x_0 = 1$ (this is the intercept term). Then

$$h_{\theta}(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x$$

- Now, given a training set, how do we learn the parameters θ ?
- One reasonable method is to make $h(x)$ close to y for at least the training examples we have
- To formalize this, we define the cost function which measures, for each value of the θ 's, how close the $h(x^{(i)})$'s are to the corresponding $y^{(i)}$'s:

$$J(\theta) = \frac{1}{2} \sum_{i=0}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

- This is the least-squares cost function, which gives rise to the ordinary least squares regression model

LMS Algorithm

- We want to choose θ so as to minimize $J(\theta)$.
- To do so, we will use a search algorithm that starts with some "initial"

"guess" for θ , and that repeatedly changes θ to make $J(\theta)$ smaller, until hopefully we converge to a value of θ that minimizes $J(\theta)$

- Specifically, we consider the gradient descent algorithm

(1) Start with some initial θ

(2) Repeatedly perform the update

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

(This update is simultaneously performed for all $j = 1, \dots, m$)

- α is called the learning rate

= This is a very natural algorithm that repeatedly takes a step in the direction of steepest decrease of J (at θ)

- In order to implement the algorithm, we need to figure out what is the partial derivative

- We work it out for the case of a single training example (x, y) , i.e. $m=1$:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{\partial}{\partial \theta_j} \left[\frac{1}{2} (h_\theta(x) - y)^2 \right]$$

$$= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \frac{\partial}{\partial \theta_j} (h_\theta(x) - y)$$

$$= (h_\theta(x) - y) \frac{\partial}{\partial \theta_j} \left(\sum_{j=1}^m \theta_j x_j - y \right)$$

$$= (h_\theta(x) - y) x_j$$

- For a single training example, this gives the update rule

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)} \quad (i=1 \text{ in this case})$$

- This rule is called the LMS, least mean squares, update rule

- It is also known as the Widrow-Hoff learning rule.

- This rule has several natural and intuitive properties:

(i) The magnitude of the update is proportional to the error term $(y^{(i)} - h_\theta(x^{(i)}))$

- This, if our prediction is accurate, and so the error term is small, there is little need to change the parameters

- Similarly, if the error term is large.

- We have derived the LMS rule for when there is a single training example
- There are two ways to modify this method for a larger training set
- (1) We first derive what is $\frac{\partial J}{\partial \theta_j}$:

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta_j} &= \frac{\partial}{\partial \theta_j} \left[\frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \right] \\ &= \frac{1}{2} \sum_{i=1}^m 2(h_\theta(x^{(i)}) - y^{(i)}) X_j^{(i)} \\ &= \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) X_j^{(i)}\end{aligned}$$

- This gives us the following algorithm:

Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^m (y^{(i)} - h_\theta(x^{(i)})) X_j^{(i)} \quad (\text{for every } j)$$

- This method looks at every example in the entire training set on every step, and is called batch gradient descent
- Note that while gradient descent can be susceptible to local minima in general, the optimization problem we have posed here has only one global, and no other local, optima.
- Thus, gradient descent always converges (assuming α is not too large)
- Indeed, J is a convex quadratic function.

(2) Consider the following algorithm:

Loop {

for $i = 1$ to m {

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) X_j^{(i)} \quad (\text{for every } j)$$

}

}

- In this algo., we repeatedly run through the training set and for each training example, we update the parameters according to the gradient of the error w.r.t. to that single training example only

- This is known as stochastic, or incremental, gradient descent.

- Whereas batch gradient descent has to scan through the entire training set

before taking a single step (a costly operation if m is large), stochastic gradient descent starts making progress right away and continues to do with each example it looks at.

- Often SGD gets θ "closer" to the minimum much faster than BGD (even if θ is not exactly the minimum, but it is close, it is a reasonably good approx.)
- Thus, SGD is often preferred over BGD, especially when the training set is large.

2: The normal equations

- We now discuss another way of minimizing J
- This approach also minimizes J explicitly and doesn't require an iterative algo.

2.1. Matrix derivatives

- For a function $f: \mathbb{R}^{n \times n} \rightarrow \mathbb{R}$ mapping from m -by- n matrices to \mathbb{R} , we define the derivative of f w.r.t. to A by:

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{m1}} & \cdots & \frac{\partial f}{\partial A_{mn}} \end{bmatrix}$$

- Thus the gradient $\nabla f(A)$ is itself an $m \times n$ matrix

- The trace operator $\text{tr}: \mathbb{R}^{n \times n} \rightarrow \mathbb{R}$ is defined by

$$\text{tr}(A) = \sum_{i=1}^n A_{ii}$$

- If $a \in \mathbb{R}$, then $\text{tr}(a) = a$

Prop. For two matrices A and B , such that AB is a square, $\text{tr} AB = \text{tr} BA$

pf: Since AB is a $n \times m$ matrix, A must be of dimensions $n \times m$ and B of dimensions $m \times n$. Then

$$(AB)_{ii} = (\text{i^{th} row of } A) \cdot (\text{i^{th} column of } B) = [A_{i1} \ A_{i2} \ \dots \ A_{in}] \begin{bmatrix} B_{1i} \\ B_{2i} \\ \vdots \\ B_{ni} \end{bmatrix}$$

$$= \sum_{j=1}^n A_{ij} B_{ji}$$

Thus,

$$\begin{aligned} \text{tr}(AB) &= \sum_{i=1}^n \sum_{j=1}^n A_{ij} B_{ji} = \sum_{j=1}^n \sum_{i=1}^n B_{ji} A_{ij} \\ &= \sum_{j=1}^n (\text{j^{th} row of } B) \cdot (\text{j^{th} column of } A) \\ &= \sum_{j=1}^n (BA)_{jj} = \text{tr}(BA) \end{aligned}$$

- As corollaries of this, we have that for matrices A, B, C such that ABC is a square,

$$\overbrace{\text{tr}ABC}^{\text{tr}CAB} = \text{tr}BCA$$

$$\text{tr}ABCD = \text{tr}DABC = \text{tr}CDAB = \text{tr}BCDA$$

- The trace operator also has these properties :

$$(i) \text{tr } A = \text{tr } A^T$$

$$(ii) \text{tr}(A+B) = \text{tr } A + \text{tr } B$$

$$(iii) \text{tr}(aA) = a \text{tr } A$$

- We now state some facts of matrix derivatives

Prop 2 Fix some matrix $B \in \mathbb{R}^{n \times n}$ and define a function $f: \mathbb{R}^{n \times n} \rightarrow \mathbb{R}$ by $f(A) = \text{tr } AB$. Then

$$\nabla_A f(A) = B^T$$

Pf Recall that $f(A) = \text{tr } AB = \sum_{i=1}^n (AB)_{ii} = \sum_{i=1}^n \sum_{j=1}^n A_{ij} B_{ji}$. So

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \frac{\partial f}{\partial A_{1n}} \\ \vdots & \vdots \\ \frac{\partial f}{\partial A_{m1}} & \frac{\partial f}{\partial A_{mn}} \end{bmatrix} = \begin{bmatrix} B_{11} & B_{21} & \dots & B_{n1} \\ B_{12} & B_{22} & \dots & B_{n2} \\ \vdots & \vdots & & \vdots \\ B_{1m} & B_{2m} & \dots & B_{nm} \end{bmatrix} = B^T$$

$A = m \times n$ $A^T = n \times m$

$B = m \times n$ $C = n \times m$

Prop 3 $\nabla_{A^T} f(A) = (\nabla_A f(A))^T$

Pf: If

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mn} \end{bmatrix}_{m \times n}, \text{ then } A^T = \begin{bmatrix} A_{11} & \cdots & A_{m1} \\ A_{21} & \cdots & A_{m2} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{mn} \end{bmatrix}_{n \times m}$$

Thus,

$$\nabla_{A^T} f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{n1}} & \cdots & \frac{\partial f}{\partial A_{nn}} \end{bmatrix}^T = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{m1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{n1}} & \cdots & \frac{\partial f}{\partial A_{mn}} \end{bmatrix}^T = (\nabla_A f(A))^T$$

Prop 4 $\nabla_A (ABC) = CAB + C^T AB^T$

Pf: If A has dimensions $m \times n$ then A^T has dimensions $n \times m$. Hence B must have dimensions $n \times n$ and C , $m \times n$. Thus

Prop 5, $\nabla_A |A| = |A| (A^{-1})^T$ where $|A|$ is the determinant of A .

2.2. Least Squares Revisited

- We now proceed to find the closed form of θ that minimizes $J(\theta)$

- We first rewrite J in matrix-vector form notation

- Given a training set, we define the design matrix X to be the $m \times n$ matrix (or $m \times (n+1)$ if we include the intercept term) that contains the training examples' inputs in a row

$$X = \begin{bmatrix} -(x^{(1)})^T \\ \vdots \\ -(x^{(m)})^T \end{bmatrix}$$

- Let \vec{y} be the m -dimensional vector containing all target values from the training set

$$\vec{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

- Then

$$h_{\theta}(x^{(i)}) = \sum_{j=0}^n \theta_j x_j^{(i)} = [x_0^{(i)} \dots x_n^{(i)}] \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix} = (x^{(i)})^T \theta$$

and so

$$X\theta - \vec{y} = \begin{bmatrix} -(x^{(1)})^T \\ \vdots \\ -(x^{(m)})^T \end{bmatrix}_{m \times (n+1)} \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix}_{(n+1) \times 1} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}_{m \times 1}$$

$$\begin{aligned} &= \begin{bmatrix} (x^{(1)})^T \theta - y^{(1)} \\ \vdots \\ (x^{(m)})^T \theta - y^{(m)} \end{bmatrix} = \begin{bmatrix} h_{\theta}(x^{(1)}) - y^{(1)} \\ \vdots \\ h_{\theta}(x^{(m)}) - y^{(m)} \end{bmatrix} \end{aligned}$$

- Then using the fact that for a vector z , $z^T z = \sum_i z_i^2$,

$$\frac{1}{2} (X\theta - \vec{y})^T (X\theta - \vec{y}) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$= J(\theta)$$

- We now find the derivative w.r.t. θ . From Prop 3 and Prop 4,

$$\nabla_{A^T} t_c(ABA^T C) = (\nabla_A t_c(ABA^T C))^T = (CAB + C^T A B^T)^T = B^T A^T C^T + B A^T C \quad (*)$$

and so

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \frac{1}{2} (X\theta - \vec{y})^T (X\theta - \vec{y}) = \nabla_{\theta} \frac{1}{2} (\theta^T X^T - \vec{y}^T)(X\theta - \vec{y})$$

$$= \frac{1}{2} \nabla_{\theta} (\theta^T X^T X \theta - \theta^T X^T \vec{y} - \vec{y}^T X \theta + \vec{y}^T \vec{y})$$

$$= \frac{1}{2} \nabla_{\theta} \text{tr}(\downarrow) \quad \text{b/c the trace of a } R \text{ is just a}$$

$$= \frac{1}{2} \nabla_{\theta} (f_{\theta}(\theta^T X^T \theta) - f_{\theta}((\tilde{y}^T X \theta)^T) - f_{\theta}(\tilde{y}^T X \theta)) + \frac{1}{2} \nabla_{\theta} f(\tilde{y}^T \tilde{y})$$

$$= \frac{1}{2} \nabla_{\theta} (f_{\theta}(\theta^T X^T X \theta) - f_{\theta}(\tilde{y}^T X \theta) - f_{\theta}(\tilde{y}^T X \theta)) + 0 \quad b/c \quad \begin{matrix} f(A) = f(A^T) \\ \nabla_{\theta} f(A^T g^T g) = 0 \end{matrix}$$

$$= \frac{1}{2} \nabla_{\theta} (f_{\theta}(\theta^T X^T X \theta) - 2 f_{\theta}(\tilde{y}^T X \theta))$$

$$= \frac{1}{2} (X^T X \theta + X^T X \theta) - \nabla_{\theta} (2 f_{\theta}(\tilde{y}^T X \theta)) \quad \text{by } (*) \text{ with } \begin{matrix} A^T = \theta, & B = B^T = X^T \\ C = I \end{matrix}$$

$$= \frac{1}{2} (X^T X \theta + X^T X \theta - \nabla_{\theta} (2 f_{\theta}(\tilde{y}^T X \theta)))$$

$$= \frac{1}{2} (2 X^T X \theta - 2 X^T \tilde{y}) \quad b/c \quad \nabla_{\theta} f(AB) = B^T$$

$$= X^T X \theta - X^T \tilde{y}$$

- To minimize J , we set $\nabla_{\theta} J(\theta) = 0$ and so

$$\boxed{X^T X \theta = X^T \tilde{y}}$$

which are called the normal equations

- Hence the value of θ that minimizes $J(\theta)$ is given by

$$\theta = (X^T X)^{-1} X^T \tilde{y}$$

§3: Probabilistic interpretation

- When faced by a regression problem, why might linear regression, and specifically the least-squares cost function, be a reasonable choice?

- In this section, we will give a set of probabilistic assumptions under which LSR is derived very naturally

- Let us assume that the target and input variables are related via the equation

$$y^{(i)} = \theta^T X^{(i)} + \varepsilon^{(i)}$$

where $\varepsilon^{(i)}$ is an error term that captures either unmodeled effects (e.g. missing features) or random noise

- Let us further assume that the $\varepsilon^{(i)}$ are distributed IID according to a Gaussian distribution with $\mu=0$ and some variance σ^2

- This can be written as $\varepsilon^{(i)} \sim N(0, \sigma^2)$

- The density of $\varepsilon^{(i)}$ is given by

$$p(\varepsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\varepsilon^{(i)})^2}{2\sigma^2}\right)$$

Since $y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)} \Rightarrow y^{(i)} - \theta^T x^{(i)} = \varepsilon^{(i)}$ and so,

$$p(y^{(i)} | x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

- Note: the dist. is not conditional on θ (this could be written as $p(y^{(i)} | x^{(i)}, \theta)$)
- θ is not a random variable.
- This can also be written as $y^{(i)} | x^{(i)} \sim N(\theta^T x^{(i)}, \sigma^2)$

- Given X and θ , what is the distribution of the $y^{(i)}$'s?

- The probability of the data is given by $p(\vec{y} | X; \theta)$

- This quantity is typically viewed as a function of \vec{y} for some fixed θ

- When we wish to view it as a function of θ , we call it the likelihood function:

$$L(\theta) = L(\theta; X, \vec{y}) = p(\vec{y} | X; \theta)$$

- By the independence assumption on the $\varepsilon^{(i)}$'s (and hence also the $y^{(i)}$'s given the $x^{(i)}$'s),

$$L(\theta) = \prod_{i=1}^m p(y^{(i)} | x^{(i)}; \theta) = \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

- Given this probabilistic model relating the $y^{(i)}$'s and $x^{(i)}$'s, what is a reasonable way of best guessing θ ?

- The principle of maximum likelihood says that we should choose θ so as to make the observed data as high probability as possible.

- Instead of maximizing $L(\theta)$, we maximize $\log L(\theta)$ b/c it is a strictly increasing function

- The log likelihood $\ell(\theta)$ is defined by $\ell(\theta) = \log L(\theta)$

- In this case,

$$\ell(\theta) = \log L(\theta)$$

$$= n \log \frac{1}{\sqrt{2\pi}\sigma} = \frac{1}{\sigma^2} \frac{1}{2} \sum_{i=1}^n (y^{(i)} - \theta^T x^{(i)})^2$$

- Hence maximizing $\ell(\theta)$ gives the same answer as minimizing $\frac{1}{2} \sum_{i=1}^n (y^{(i)} - \theta^T x^{(i)})^2$ which is just $J(\theta)$

- Note, however, that these probabilistic assumptions are not necessary for LS to be a perfectly good and rational procedure; there are other natural assumptions that can be made to justify it.

- Note further that our choice of σ didn't depend on σ^2 and so we can arrive to the same result even if σ^2 were unknown.

84. Locally-weighted linear regression

- Informally, underfitting occurs when the data clearly shows structure not captured by the model, and overfitting occurs when the model captures the structure of the data too exactly.

- The choice of feature is important to ensuring good performance of a learning algorithm.

- We now discuss the locally weighted linear regression (LWLR) algorithm, which, assuming we have sufficient training data, makes the choice of features less critical.

- In contrast to the original linear regression which fits θ to minimize $\sum_i (y^{(i)} - \theta^T x^{(i)})^2$, LWLR instead minimizes $\sum_i w^{(i)} (y^{(i)} - \theta^T x^{(i)})^2$

- The $w^{(i)}$ are >0 valued weights
- If $w^{(i)}$ is large, we will have to try hard to make $(y^{(i)} - \theta^T x^{(i)})^2$ small
if $w^{(i)}$ is small, we can pretty much ignore $(y^{(i)} - \theta^T x^{(i)})^2$ in ℓ_2

- A standard choice for the weight is

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

and if x is vector-valued, this can be generalized to:

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^T (x^{(i)} - x)}{2\tau^2}\right)$$

- Note that the weights depend on the particular point x that we are trying to evaluate

- Also, if $|x^{(i)} - x|$ is small, then $w^{(i)}$ is close to 1; if $|x^{(i)} - x|$ is large, the $w^{(i)}$ is small

- Hence, θ is chosen giving a much higher weight to the (errors on) the training example is close to the query point.

- The bandwidth parameter τ controls how quickly the weight of a training example falls off with distance $|x^{(i)} - x|$

- LWR is an example of a non-parametric algo.

- The (unweighted) linear regression is known as a parametric algo. b/c it has a fixed, finite no. of parameters (the θ 's), which are fit to the data and once fit, they are stored away and we no longer need to keep the training data around to make future predictions

- In contrast, for LWR we need to keep the training data around

Part II: Classification and Logistic Regression

- We will now discuss the classification problem, i.e. the values of y take on only a small number of discrete values.
- For now, we will focus on the binary classification problem, where $y \in \{0, 1\}$
- 0 is called the negative class, and denoted by "-", and 1 is called the positive class, and denoted by "+".
- Given $x^{(i)}$, the corresponding $y^{(i)}$ is also called the label for the training example.

§5: Logistic Regression

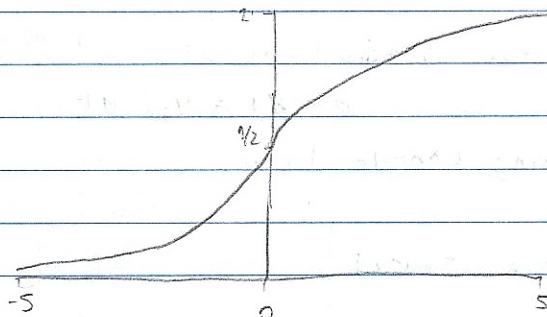
- We can approach this problem by ignoring the fact that y is discrete-valued and simply use linear regression
- For this to work properly, we must change the form for our hypothesis $h_\theta(x)$ by

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

where

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

is called the logistic or sigmoid function



- Notice that

- $g(z) \rightarrow 1$ as $z \rightarrow \infty$,
- $g(z) \rightarrow 0$ as $z \rightarrow -\infty$
- $0 < g(z) < 1 \quad \forall z$

- We will see later that the logistic function is a very natural choice

- Note the following useful property of the derivative of the sigmoid function:

$$\begin{aligned} g'(z) &= \frac{d}{dz} \frac{1}{1 + e^{-z}} = -\frac{1}{(1 + e^{-z})^2} (-e^{-z}) = \frac{1}{1 + e^{-z}} \left(1 - \frac{1}{1 + e^{-z}}\right) \\ &= g(z) (1 - g(z)) \end{aligned}$$

- So given the logistic regression model, how do we fit θ ?

- Let's endow our classification model with a set of probabilistic assumptions and then fit the parameters via maximum likelihood.

- Assume that

$$P(y=1|X;\theta) = h_{\theta}(x)$$

$$P(y=0|X;\theta) = 1 - h_{\theta}(x)$$

or more compactly:

$$P(y|X;\theta) = (h_{\theta}(x))^{y} (1-h_{\theta}(x))^{1-y}$$

Basically, $P(y|X;\theta)$ has to follow a Bernoulli distribution with prob. of success $h_{\theta}(x)$

- Assuming the m training examples were generated independently,

$$L(\theta) = P(\vec{y}|X;\theta) = \prod_{i=1}^m P(y^{(i)}|X^{(i)};\theta)$$

$$= \prod_{i=1}^m (h_{\theta}(x^{(i)}))^{y^{(i)}} (1-h_{\theta}(x^{(i)}))^{1-y^{(i)}}$$

and so

$$\ell(\theta) = \log L(\theta) = \sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1-y^{(i)}) \log(1-h_{\theta}(x^{(i)}))$$

- How do we maximize $\ell(\theta)$? Let's use gradient ascent \rightarrow b/c we want to maximize!

- In vectorial notation, our updates are given by $\theta = \theta + \alpha \nabla_{\theta} \ell(\theta)$

- We first consider a single training example (x, y) :

$$\frac{\partial}{\partial \theta_j} \ell(\theta) = \frac{\partial}{\partial \theta_j} \left[y \log h(x) + (1-y) \log(1-h(x)) \right]$$

$$= \left[\frac{y}{g(\theta^T x)} \frac{\partial g(\theta^T x)}{\partial \theta_j} + \frac{(1-y)}{1-g(\theta^T x)} \frac{\partial}{\partial \theta_j} (-g(\theta^T x)) \right] \text{ b/c } h(x) = g(\theta^T x)$$

$$= \left[\frac{y}{g(\theta^T x)} - \frac{(1-y)}{1-g(\theta^T x)} \right] \frac{\partial g(\theta^T x)}{\partial \theta_j}$$

$$= \left[\frac{y}{g(\theta^T x)} - \frac{(1-y)}{1-g(\theta^T x)} \right] g(\theta^T x) (1-g(\theta^T x)) \frac{\partial \theta^T x}{\partial \theta_j} \text{ b/c } g'(x) = g(x)(1-g(x))$$

$$= [y(1 - g(\theta^T x)) - (1-y)g(\theta^T x)] x_i$$

$$= (y - g(\theta^T x)) x_i = (y - h_\theta(x)) x_i$$

- This gives us the stochastic gradient ascent rule

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$

- This seems to be exactly like the LMS update rule, but recall that $h_\theta(x^{(i)})$ is now a non-linear function of $\theta^T x$

- But this similarity, is it coincidental or is there a deeper reason behind this? We'll answer this when we discuss GLM models.

§6: Digression: The perceptron learning algorithm

- Consider modifying the logistic regression method to "force" it to output either 0 or 1

- The natural way to do this:

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

- If we let $h_\theta(x) = g(\theta^T x)$ and use the same update rule for gradient ascent, we have the perceptron learning algorithm

- In the 1960s, this "perceptron" was argued to be a rough model for how individual neurons in the brain work.

§7: Another algorithm for maximizing $\ell(\theta)$

- We consider a different algo. for maximizing $\ell(\theta)$ in the logistic regression model.

- We first consider Newton's method for finding a zero of a function

- Let $f: \mathbb{R} \rightarrow \mathbb{R}$ be a function. We want to find a $\theta \in \mathbb{R}$ s.t. $f(\theta) = 0$.

- Newton's method performs the update:

$$\theta := \theta - \frac{F(\theta)}{f'(\theta)}$$

- To maximize λ , we find the zeros of $\lambda'(\theta)$. Letting $f(\theta) = \lambda'(\theta)$, we obtain the update rule

$$\theta := \theta - \frac{\lambda'(\theta)}{\lambda''(\theta)}$$

- In our logistic regression setting, θ is vector-valued, so we need to generalize Newton's method; this generalization to a multidimensional setting is called the Newton-Raphson method and is given by:

$$\theta := \theta - H^{-1} \nabla \lambda(\theta)$$

where H is an $n \times n$ matrix called the Hessian and is given by

$$H_{ij} = \frac{\partial^2 \lambda(\theta)}{\partial \theta_i \partial \theta_j}$$

- Newton's method typically enjoys faster convergence than (batch) gradient descent, and requires many fewer iterations to get very close to the minimum
- One iteration of Newton's, however, may be more expensive b/c you have to invert an $n \times n$ matrix
- But so long as n is not too large, it is usually faster overall.
- When Newton's method is applied to maximize the logistic regression log likelihood function, the resulting method is also called Fisher scoring.

Part II: Generalized Linear Models

- So far we have seen

(i) Regression example, in which $y|x; \theta \sim N(\mu, \sigma^2)$

(ii) Classification example, in which $y|x; \theta \sim \text{Bernoulli}(\phi)$

where μ and ϕ are appropriately defined functions of x and θ .

- In this section, we will show that both of these methods are special cases of a broader family of models, called Generalized Linear Models (GLM).
- We will also show other models in the GLM family can be derived and applied to other classification and regression problems.

§8: The Exponential Family

- We say that a class of distributions is in the exponential family if it can be written

$$p(y|\eta) = b(y) \exp(\eta^T T(y) - a(\eta))$$

- Here η is called the natural parameter (or canonical parameter) of the distribution; $T(y)$ is the sufficient statistic (for the distributions we consider, it will often be the case that $T(y)=y$); and $a(\eta)$ is the log partition function.
• The quantity $e^{-a(\eta)}$ essentially plays the role of a normalization constant, which ensures that the dist. $p(y|\eta)$ sums/integrates to 1.

- A fixed choice of T, a and b defines a family of distributions that is parametrized by η .

- We now show that the Bernoulli and Gaussian distributions are examples of exponential family distributions.

(i) Bernoulli: $\text{Bernoulli}(\phi)$, i.e. w.r.t. ϕ , specifies a distribution over $y \in \{0, 1\}$, so that $p(y=1|\phi) = \phi$; $p(y=0|\phi) = 1-\phi$.

• Thus,

$$\begin{aligned} p(y|\phi) &= \phi^y (1-\phi)^{1-y} \\ &= \exp(y \log \phi + (1-y) \log(1-\phi)) \\ &= \exp\left(\left[\log\left(\frac{\phi}{1-\phi}\right)\right]y + \log(1-\phi)\right) \end{aligned}$$

- We can see that

$$\eta = \log \left(\frac{\phi}{1-\phi} \right)$$

$$T(y) = y$$

$$a(\eta) = -\log(1-\phi) = \log(1+e^\eta)$$

$$b(y) = 1$$

(ii) Gaussian: Since σ^2 has no effect on our final choice for θ and $h(x)$ when deriving linear regression, we can choose any arbitrary value for it

• To simplify, set $\sigma^2 = 1$

∴ So,

$$p(y|\mu) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(y-\mu)^2\right)$$

$$= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) \exp\left(\mu y - \frac{1}{2}\mu^2\right)$$

- We can see that

$$\eta = \mu$$

$$T(y) = y$$

$$a(\eta) = \mu^2/2 = \eta^2/2$$

$$b(y) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right)$$

§9: Constructing GLMs

- Consider a classification or regression problem where we would like to predict the value of some random variable y as a function of x

- To derive a GLM for this problem, we will make the following three assumptions:

(i) $y|x;\theta \sim \text{Exponential Family}(\eta)$

(ii) Given x , our goal is to predict the expected value of $T(y)$ given x .

• In most of our examples, $T(y) = y$, so this means we would like the prediction $h(x)$ output by our learned hypothesis h to satisfy $h(x) = E[y|x]$

(iii) The natural parameter η and the inputs x are related linearly: $\eta = \theta^T x$, or if η is vector-valued, $\eta_i = \theta_i^T x$

- These assumptions will allow us to derive a very elegant class of learning algorithms, namely GLMs, that have many desirable properties

9.1 Ordinary Least Squares

- To show that ordinary least squares is a special case of the GLM family of models, consider the setting where the target variable y (also called the response variable in GLM terminology) is continuous
- Further, we have that $y|x; \theta \sim \text{Gaussian}(\mu, \sigma^2)$ (where μ may depend on x)
- We now that $\mu = \eta$. So,

$$h_\theta(x) \stackrel{\text{Assumption (ii)}}{=} E[y|x;\theta] = \mu$$

$$\text{Assumption (iii)}: \mu = \eta = \theta^T x$$

~ Assumption (iii)

9.2 Logistic Regression

- In our formulation of the Bernoulli distribution as an exponential family dist., we had that $\phi = \frac{1}{1+e^{-\eta}}$
- Furthermore, if $y|x; \theta \sim \text{Bernoulli}(\phi)$, then $E[y|x;\theta] = \phi$. So

$$\begin{aligned} h_\theta(x) &= E[y|x;\theta] = \phi \\ &= \frac{1}{1+e^{-\eta}} \\ &= \frac{1}{1+e^{-\theta^T x}} \end{aligned}$$

- Introducing terminology, the function g defined by $g(\eta) = E[T(y); \eta]$ is called the canonical response function and g^{-1} is the canonical link function
- Basically, g gives the distribution's mean as a function of the natural parameter η
- For the Gaussian family, g is just the identity function s/c $g(\eta) = \eta = \mu$
- For the Bernoulli family, g is the sigmoid function s/c $g(\eta) = \frac{1}{1+e^{-\eta}} = \phi$

9.3: Softmax Regression

- We consider one more example of a GLM
 - Consider a classification in which the response variable $y \in \{1, 2, \dots, K\}$
 - Since the response variable, although still discrete, can take more than two values, we model it as a multinomial distribution
 - We derive a GLM for modeling this type of multinomial data
 - We first show that the multinomial dist. is an exponential family
- To parametrize a multinomial over K possible outcomes, we only need $K-1$ parameters $\phi_1, \dots, \phi_{K-1}$ that specify the probability of the outcomes $1, 2, \dots, K-1$ resp.
- For ϕ_K , we notice that $\phi_K = 1 - \sum_{i=1}^{K-1} \phi_i$ b/c we must have that $\sum_i \phi_i = 1$
 - Hence ϕ_K is determined by the other parameters
- Define $T(y) \in \mathbb{R}^{n-1}$ by

$$T(1) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \dots, T(K-1) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, T(K) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

- An indicator function $1\{\cdot\}$ takes only a value of 1 if the argument is true, and 0 otherwise
- Hence we have that

$$(T(y))_i = 1\{y=i\}$$

- Further,

$$E[(T(y))_i] = \sum_{k=1}^K (T(k))_i p(y=k) = (T(i))_i p(y=i) = p(y=i) = \phi_i$$

- We now have that

$$\begin{aligned}
 p(y; \phi) &= \phi_1^{y_1} \phi_2^{y_2} \cdots \phi_k^{y_k} \\
 &= \phi_1^{(T(y))_1} \phi_2^{(T(y))_2} \cdots \phi_k^{1 - \sum_{j=1}^{k-1} y_j} \\
 &= \exp \left((T(y))_1 \log \phi_1 + (T(y))_2 \log \phi_2 + \cdots + \left(1 - \sum_{j=1}^{k-1} (T(y))_j\right) \log \phi_k \right) \\
 &= \exp \left((T(y))_1 \log \frac{\phi_1}{\phi_k} + (T(y))_2 \log \frac{\phi_2}{\phi_k} + \cdots + (T(y))_{k-1} \log \frac{\phi_{k-1}}{\phi_k} + \log \phi_k \right) \\
 &= b(y) \exp(n^T T(y) - a(n))
 \end{aligned}$$

where

$$n = \begin{bmatrix} \log \frac{\phi_1}{\phi_k} \\ \vdots \\ \log \frac{\phi_{k-1}}{\phi_k} \end{bmatrix}, \quad a(n) = -\log \phi_k, \quad b(y) = 1$$

- Hence the multinomial is an exponential family distribution.

- The link function is given by

$$n_i = \log \frac{\phi_i}{\phi_k}$$

- For convenience, define $n_k = \log \frac{\phi_k}{\phi_k} = 0$

- To derive the response function, we invert the link function:

$$e^{n_i} = \frac{\phi_i}{\phi_k} \Rightarrow \phi_k e^{n_i} = \phi_i$$

$$\Rightarrow \phi_k \sum_{i=1}^k e^{n_i} = \sum_{i=1}^k \phi_i = 1$$

$$\text{Thus, } \phi_k = \frac{1}{\sum_{i=1}^k e^{n_i}}$$

- Plugging this back into

$$\boxed{\phi_i = \frac{e^{n_i}}{\sum_{j=1}^k e^{n_j}}}$$

- This function mapping the m 's to ϕ 's is called the softmax function.

- By our Assumption (iii), $m_i = \theta^T x$, where $\theta_0, \dots, \theta_{n-2} \in \mathbb{R}^{n+1}$

- For convenience, define $\theta_K = 0$, so that $m_K = \theta_K^T x = 0$, as given previously.

- Hence our model assumes that the conditional dist. of y given x :

$$\begin{aligned} p(y=i|x) &= \phi_i \\ &= \frac{e^{m_i}}{\sum_{j=0}^{n-1} e^{m_j}} \\ &= \frac{e^{\theta_i^T x}}{\sum_{j=0}^{n-1} e^{\theta_j^T x}} \end{aligned}$$

- This model, which applies to classification problems where $y \in \{1, \dots, K\}$, is called softmax regression.

- It is a generalization of logistic regression, where $K = 2$.

- Our hypothesis will output

$$h_\theta(x) = E[T(y)|x;\theta]$$

$$= E \left[\begin{array}{c} 1_{\{y=1\}} \\ 1_{\{y=2\}} \\ \vdots \\ 1_{\{y=K-1\}} \end{array} \middle| x; \theta \right]$$

$$= \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{K-1} \end{bmatrix} = \begin{bmatrix} \frac{\exp(\theta_1^T x)}{\sum_{j=0}^{n-1} \exp(\theta_j^T x)} \\ \frac{\exp(\theta_2^T x)}{\sum_{j=0}^{n-1} \exp(\theta_j^T x)} \\ \vdots \\ \frac{\exp(\theta_{K-1}^T x)}{\sum_{j=0}^{n-1} \exp(\theta_j^T x)} \end{bmatrix}$$

- Thus, our hypothesis will output the estimated probability that $p(y=i|x;\theta)$ for $i = 1, \dots, K$

- Finally, we discuss parameter fitting

- Given a training set of m examples $\{(x^{(i)}, y^{(i)}) : i=1, \dots, m\}$, we

would like to learn the parameters θ_i

- The log likelihood is

$$\begin{aligned} l(\theta) &= \sum_{i=1}^m \log p(y^{(i)} | x^{(i)}, \theta) \\ &= \sum_{i=1}^m \log \prod_{k=1}^K \phi_k^{y^{(i)} = k} \\ &= \sum_{i=1}^m \log \prod_{k=1}^K \left(\frac{e^{\theta_k^T x^{(i)}}}{\sum_{j=1}^K e^{\theta_j^T x^{(i)}}} \right)^{y^{(i)} = k} \end{aligned}$$

- We can now obtain the maximum likelihood estimate of the parameters by maximizing $l(\theta)$ in terms of θ , using a method such as gradient descent or Newton's method.

Lecture notes 2: Generative Algorithms

Part IV: Generative Learning algorithms

- So far, we have mainly talked about learning algorithms that model $p(y|x, \theta)$

- Consider a ^{binary} classification problem. Given a training set, an algorithm like logistic regression or the perceptron algorithm tries to find a straight line, i.e. a decision boundary, that separates the two classes

- Then to classify a new input, it checks on which side of the decision boundary it falls.

- Consider a different approach: build separate models for each class and to classify a new input, pass it through both models and see in which one it fits best.

- Algorithms that try to learn $p(y|x)$ directly (such as logistic regression) or algorithms that try to learn mappings directly from the input space X to the labels $\{0, 1\}$ (such as the perceptron algorithm) are called discriminative learning algorithms.

- Algorithms that instead try to model $p(x|y)$ (and $p(y)$) are called generative learning algorithms.

• $p(x|y=1)$ models the distribution of the features of class 1 examples

- After modeling $p(y)$, called the class priors, and $p(x|y)$, our algo can then use Bayes rule to derive the posterior distribution on y given x :

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

• Here $p(x) = p(x|y=1)p(y=1) + p(x|y=0)p(y=0)$ and thus, it can be calculated from $p(x|y)$ and $p(y)$

- But notice that in order to make a prediction by calculating $p(y|x)$, we don't need the denominator!

$$\begin{aligned}\arg \max_y p(y|x) &= \arg \max_y \frac{p(x|y)p(y)}{p(x)} \\ &= \arg \max_y p(x|y)p(y)\end{aligned}$$

§1: Gaussian discriminant analysis (GDA)

- This is the first generative algorithm we'll look at
- In this model, we assume that $p(x|y)$ is distributed according to a multivariate normal distribution

1.1. The multivariate normal distribution

- The multivariate normal (or Gaussian) distribution in n -dimensions is parametrized by a mean vector $\mu \in \mathbb{R}^n$ and a covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$, where

- $\Sigma \geq 0$
- Σ is symmetric, i.e. $\Sigma = \Sigma^\top$ conjugate transpose
- Σ is positive semi-definite: $\forall x \in \mathbb{C}^n, x^\top \Sigma x \geq 0$
- It is also written as $N(\mu, \Sigma)$
- Its density is given by

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} (x - \mu)^\top \Sigma^{-1} (x - \mu)\right)$$

- where $|\Sigma|$ denotes the determinant of Σ

- For a R.V. $X \sim N(\mu, \Sigma)$,

$$E[X] = \int x p(x; \mu, \Sigma) dx = \mu$$

- The covariance of a vector-valued R.V. Z is defined by

$$\text{Cov}(Z) = E[(Z - E[Z])(Z - E[Z])^\top]$$

- This generalizes the notion of the variance of a R.-valued R.V.

$$\text{Prop. } \text{Cov}(Z) = E[ZZ^T] - (E[Z])(E[Z])^T$$

Pf:

$$\text{Cov}(Z) = E[(Z - E[Z])(Z - E[Z])^T]$$

$$= E[(Z - E[Z])(Z^T - E[Z]^T)]$$

$$= E[ZZ^T - ZE[Z]^T - E[Z]Z^T + (E[Z])(E[Z])^T]$$

$$= E[ZZ^T] - (E[Z])(E[Z])^T - (E[Z])(E[Z])^T + (E[Z])(E[Z])^T \quad \text{blk}$$

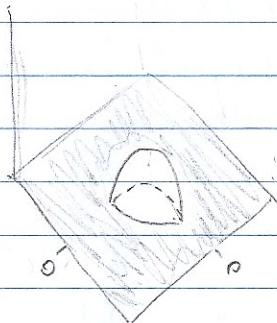
$$= E[ZZ^T] - (E[Z])(E[Z])^T$$

- If $X \sim N(\mu, \Sigma)$, Then

$$\text{Cov}(X) = \Sigma$$

- A Gaussian with zero mean and identity covariance, i.e., $\Sigma = I$, is also called the standard normal distribution.

• If $X \sim N(0, I)$ with $D \in \mathbb{R}^{2 \times d}$ and $I \in \mathbb{R}^{2 \times 2}$, it looks like



→ As Σ becomes larger, the Gaussian becomes more "spread out"; as Σ becomes smaller, it becomes more "compressed".

1.2: The Gaussian Discriminant Analysis model

- When we have a classification problem in which the input features x are continuous-valued R.V.s, we can then use GDA, which models $p(x|y)$ using a multivariate normal distribution.

- The model is:

$$y \sim \text{Bernoulli}(\phi)$$

$$x|y=0 \sim N(\mu_0, \Sigma)$$

$$x|y=1 \sim N(\mu_1, \Sigma)$$

- Here, the parameters of our model are $\phi, \mu_0, \mu_1, \Sigma$

- Notice that Σ is equal for both!!

- The log-likelihood of the data is given by

$$\ell(\phi, \mu_0, \mu_1, \Sigma) = \log \prod_{i=1}^m p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma)$$

$$= \log \prod_{i=1}^m p(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma) p(y^{(i)}; \phi)$$

- By maximizing ℓ w.r.t. the parameters, we find the maximum likelihood estimates of the parameters to be:

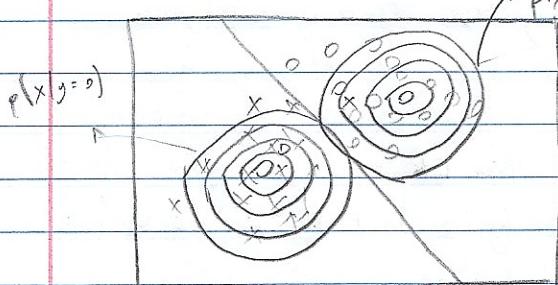
$$\hat{\phi} = \frac{1}{m} \sum 1 \sum y^{(i)} = \frac{1}{m}$$

$$\hat{\mu}_0 = \frac{\sum_{i=1}^m 1 \sum y^{(i)} = 0 \sum x^{(i)}}{\sum_{i=1}^m 1 \sum y^{(i)} = m}$$

$$\hat{\mu}_1 = \frac{\sum_{i=1}^m 1 \sum y^{(i)} = 1 \sum x^{(i)}}{\sum_{i=1}^m 1 \sum y^{(i)} = m}$$

$$\hat{\Sigma} = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \hat{\mu}_0)(x^{(i)} - \hat{\mu}_0)^T$$

- Pictorially, the algorithm does the following



$p(x|y=1)$

- The x, o 's are the training examples

- You can also be contours of the two Gaussian distributions that have fit to the data

- The straight line gives the decision boundary at which $p(y=1|x) = 0.5$ (which is equivalent to $p(y=0|x) = 0.5$)

1.3: Discussion: GDA and Logistic Regression

- The GDA has an interesting relationship to logistic regression.
- If we view the quantity $p(y=1 | X; \phi, \mu_0, \mu_1, \Sigma)$ as a function of X , we see that it can be expressed as:

$$p(y=1 | X; \phi, \mu_0, \mu_1, \Sigma) = \frac{1}{1 + \exp(-\phi^T X)}$$

where ϕ is some appropriate function of $\phi, \mu_0, \mu_1, \Sigma$

- This uses the convention of redefining the $X^{(i)}$'s on the RHS to be $(n+1)$ -vectors with $X_0^{(i)} = 1$

- This is exactly the form that logistic regression - a discriminative algorithm used to model $p(y=1 | X)$

- When would we prefer one model over another? GDA and LR will, in general, give different decision boundaries when trained on the same dataset.

- Which one is better?

- Notice that even though: $p(x|y)$ is multivariate Gaussian (with shared Σ)
 $\Rightarrow p(y|x)$ follows a logistic function, the converse is not necessarily true

- Thus GDA makes stronger modeling assumptions about the data than does logistic regression

- It turns out that when these modeling assumptions are correct, then GDA will find better fits to the data and is a better model

- Specifically, when $p(x|y)$ is indeed gaussian (with shared Σ), the GDA is asymptotically efficient.

- Informally, this means that in the limit of very large training sets, there is no algo. that is strictly better than GDA (in terms of, say, how accurately they estimate $p(y|x)$)

- Thus, in this setting, GDA will be better than LR

- Even for small training sets, we would generally expect GDA to be better.

- In contrast, by making significantly weaker assumptions, LR is more robust and less sensitive to incorrect modeling assumptions.
- There are many different sets of assumptions that would lead to $p(y|x)$ taking the form of a logistic function.
 - E.g. if $x|y=0 \sim \text{Poisson}(\lambda_0)$ and $x|y=1 \sim \text{Poisson}(\lambda_1)$, then $y|x$ will be logistic.
 - Thus LR would work on Poisson data like this
- But if we were to use GDA on such data, and fit Gaussian dists. to such non-Gaussian data, then the results would less predictable; GDA may or may not do well.
- In summary
 - GDA makes stronger modeling assumptions and is more data-efficient (i.e. requires less training data to learn "well") when the modeling assumptions are correct or at least approx. correct.
 - LR makes weaker assumptions and is significantly more robust to deviations from modeling assumptions.
 - Specifically, when the data is non-Gaussian, then in the limit of large datasets, LR will always do better than GDA.
 - For this reason, LR is used more in practice than GDA.

§2: Naive Bayes

- In GDA, the feature vectors x were continuous, \mathbb{R} -valued vectors.
- Let's now consider a different learning algo. in which the x_i 's are discrete.
- The problem of classifying text documents into different categories is called text classification.
- Consider the email ^{spam} classification problem
- We begin our construction of our spam filter by specifying the features x_i used to represent an email.
- We represent an email via a feature vector whose length is equal

to the no. of words in the dictionary

- If the email contains the i th word of the dictionary, set $x_i = 1$;
otherwise set $x_i = 0$

- The set of words encoded into the feature vector is called the vocabulary, so the dimension of x is equal to the size of the vocabulary

- We now want to build a generative model; so we have to model $p(x|y)$

- But suppose we have a vocabulary of 50,000 words. Then $x \in \{0, 1\}^{50000}$, and if we were to model x explicitly with a multinomial distribution over the 2^{50000} possible outcomes, then we would end up with a $(2^{50000} - 1)$ -dimensional parameter vector

- This is way too many

- Therefore, to model $p(x|y)$ we will make a very strong assumption

- The Naive Bayes (NB) assumption states that the x_i 's are conditionally independent given y

- The resulting algorithm is called the Naive Bayes classifier

- We now have that

$$p(x_1, \dots, x_{50000} | y) = p(x_1 | y) p(x_2 | y, x_1) p(x_3 | y, x_1, x_2) \dots p(x_{50000} | y, x_1, \dots, x_{50000})$$

$$= p(x_1 | y) p(x_2 | y) \dots p(x_{50000} | y) \text{ by the NB assumption}$$

$$= \prod_{i=1}^{50000} p(x_i | y)$$

- Even though the assumption is very strong, the resulting algorithm works well in practice on many problems

- Our model is parametrized by

- $\phi_{i,y=1} = p(x_i = 1 | y = 1)$

- $\phi_{i,y=0} = p(x_i = 1 | y = 0)$

- $\phi_y = p(y = 1)$

- As usual, given a training set $\{(x^{(i)}, y^{(i)}) : i = 1, \dots, n\}$, we can write the joint likelihood of the data

$$L(\phi_y, \phi_{j|y=1}, \phi_{j|y=0}) = \prod_{i=1}^n p(x^{(i)}, y^{(i)})$$

- Maximizing w.r.t. each of this gives the maximum likelihood estimates:

$$\cdot \phi_{j|y=1} = \frac{\sum_{i=1}^n \mathbb{1}\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^n \mathbb{1}\{y^{(i)} = 1\}} \Rightarrow \phi_{j|y=1} \text{ is just the fraction of the spam } (y=1) \text{ in which word } j \text{ appears}$$

$$\cdot \phi_{j|y=0} = \frac{\sum_{i=1}^n \mathbb{1}\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^n \mathbb{1}\{y^{(i)} = 0\}}$$

$$\cdot \phi_y = \frac{\sum_{i=1}^n \mathbb{1}\{y^{(i)} = 1\}}{n}$$

- Having fit all these parameters, to make a prediction on a new example with features x , we then simply calculate

$$p(y=1|x) = \frac{p(x|y=1)p(y=1)}{p(x)} \quad \text{Bayes' Thm.}$$

$$= \frac{\left[\prod_{i=1}^n p(x_i|y=1) \right] p(y=1)}{\left[\prod_{i=1}^n p(x_i|y=0) \right] p(y=0) + \left[\prod_{i=1}^n p(x_i|y=1) \right] p(y=1)}$$

and pick whichever class has the higher posterior probability

- Finally, we note that even though we have developed the NB algorithm mainly for when the features X_i are binary-valued, the generalization to where $X_i \in \{1, 2, \dots, k\}$ is straightforward

- Here we would simply model $p(x_i|y)$ as multinomial rather than Bernoulli

- Actually, even if some original input attribute were continuous valued, it is quite common to discretize it, i.e. turn it into a small set of discrete values, in order to be able to use NB.

- When the original, continuous-valued attributes are not well-modeled by a multivariate normal distribution, discretizing the features and using Naive Bayes (instead of GDA) will often result in a better classifier.

2.1: Laplace Smoothing

- We can make a simple change to the NB algo. to make it work much better, especially for text classification
- We first discuss a problem in the algo. in its current form

- Consider again the spam classification problem.

- Suppose that the 35000th word in the dictionary does not appear anywhere in any of your training examples

- Then, our NB spam filter will have picked its maximum likelihood estimates of the parameters $\phi_{35000y=1}$ to be $\hat{\phi}_{35000y=1} = \hat{\phi}_{35000y=0} = 0$
i.e., since it hasn't seen the 35000th in either spam or non-spam training examples, it thinks that the probability of seeing it in either type of email is zero

- Hence, when trying to decide whether a message containing the 35000th word is spam or not, it will calculate the class posterior probabilities to be:

$$p(y=1|x) = \frac{\prod_{i=1}^n p(x_i|y=1)p(y=1)}{\prod_{i=1}^n p(x_i|y=1)p(y=1) + \prod_{i=1}^n p(x_i|y=0)p(y=0)} = \frac{0}{0}$$

b/c each of the terms " $\prod_{i=1}^n p(x_i|y)$ " includes a term $p(x_{35000}|y)=0$ that is multiplied in it.

- So, the algo. doesn't know what to predict in this case

- Stated more generally, it is statistically a bad idea to estimate the probability of some event to be zero just b/c you haven't seen it before in your finite training set.

- Take the problem of estimating the mean of a multinomial random variable z taking values in $\{1, 2, \dots, K\}$

- We can parametrize our multinomial with $\phi_i = p(z=i)$
- Given a set of m independent observations $\{z^{(1)}, \dots, z^{(m)}\}$, the maximum likelihood estimates are given by

$$\hat{\phi}_j = \frac{\sum_{i=1}^m 1 \{ z^{(i)} = j \}}{m}$$

- If we use these maximum likelihood estimate, then some of the $\hat{\phi}_j$'s might end up as zero, as we saw before
- To avoid, we can use Laplace smoothing, which replaces the above estimate with

$$\hat{\phi}_j = \frac{\sum_{i=1}^m 1 \{ z^{(i)} = j \} + 1}{m + K}$$

- Note that $\sum_{j=1}^K \hat{\phi}_j = 1$:

$$\begin{aligned} \sum_{j=1}^K \hat{\phi}_j &= \frac{1}{m+K} \sum_{j=1}^K \left[\sum_{i=1}^m (1 \{ z^{(i)} = j \}) + 1 \right] \\ &= \frac{1}{m+K} \left[\sum_{j=1}^K \sum_{i=1}^m (1 \{ z^{(i)} = j \}) + K \right] \\ &= \frac{1}{m+K} \left[\sum_{i=1}^m \sum_{j=1}^K (1 \{ z^{(i)} = j \}) + K \right] \\ &= \frac{1}{m+K} \left[\sum_{i=1}^m (1) + K \right] = \frac{m+K}{m+K} = 1 \end{aligned}$$

- This is a desirable property since the $\hat{\phi}_j$'s are estimates for probabilities that we know must sum to 1
- Also $\hat{\phi}_j \neq 0$ for all j !
- Under certain (arguably quite strong) conditions, it can be shown that the Laplace smoothing actually gives the optimal estimator of the ϕ_j 's
- Returning to our NB classifier, with Laplace smoothing, we now obtain the following estimates of the parameters:

$$\phi_{j|y=1} = \frac{\sum_{i=2}^m 1 \cdot \sum x_j^{(i)} = 1 \wedge y^{(i)} = 1}{\sum_{i=2}^m 1 \cdot \sum y^{(i)} = 23 + 2}$$

$$\phi_{j|y=0} = \frac{\sum_{i=2}^m 1 \cdot \sum x_j^{(i)} = 1 \wedge y^{(i)} = 0}{\sum_{i=2}^m 1 \cdot \sum y^{(i)} = 23 + 2}$$

- (In practice, it usually doesn't matter if we apply Laplace smoothing to ϕ_y or not, since we will typically have a fair and equitable fraction each of spam and non-spam messages, so ϕ_y will be a reasonable estimate of $p(y=1)$ and will be quite far from 0 anyway)

2.2: Event models for text classification

- We now discuss one model that is specific for text classification
- This model does even better than NB for text classification
- In the specific context of text classification, NB as presented uses the multi-variate Bernoulli event model
- In this model, we assumed that the way an email is generated is that first it is randomly determined (according to the class priors $p(y)$) whether a spammer or non-spammer will send you your next message
- Then, the person sending the email runs through the dictionary, deciding whether to include each word i in that email independently and according to the probabilities $p(x_i=1|y) = \phi_{iy}$
- Thus, the probability of a message was given by $p(y) \prod_{i=1}^n p(x_i|y)$
- A different model is the multinomial event model
- To describe, we use different notation
- Let x_i denote the identity of the i th word in the email. Thus, $x_i \in \{1, \dots, |V|\}$ where V is our vocabulary
- An email of n words is now represented by a vector (x_1, x_2, \dots, x_n) of length n
- Note that n can vary for different documents

- In the multinomial event model, we assume that the way an email is generated is via a random process in which spam/non-spam is first determined (according to $p(y)$) as before
- Then the sender of the email writes the email by first generating x_1 from some multinomial distribution over words ($p(x_1|y)$)
- Next, the second word x_2 is chosen independently of x_1 but from the same multinomial distribution
- Similarly for x_3, x_4, \dots, x_n .
- Thus, the overall probability of a message is given by $p(y) \prod_{i=1}^n p(x_i|y)$
- \exists Note that, even though this formula looks the same as before, they are not
 - In particular, $x_i|y$ is now a multinomial rather than a Bernoulli distribution.

- The parameters for our new model are ϕ_{yj}
- $\phi_y = p(y)$
- $\phi_{yj=1} = p(x_j = k | y = 1)$ (for all j)
- $\phi_{yj=0} = p(x_j = k | y = 0)$ (for all j)
- Note that we have assumed that $p(x_{ij}|y)$ is the same for all values of j , i.e. that the distribution according to which a word is generated doesn't depend on its position j within the email.

- Given a training set $\{(x^{(i)}, y^{(i)}) : i = 1, \dots, m\}$ where $x^{(i)} = (x_1^{(i)}, \dots, x_n^{(i)})$ and n_i is the no. of words in the i th training example, the likelihood of the data is given by

$$\begin{aligned} L(\phi, \phi_{yj=1}, \phi_{yj=0}) &= \prod_{i=1}^m p(x^{(i)}, y^{(i)}) \\ &= \prod_{i=1}^m \left(\prod_{j=1}^n p(x_{ij}^{(i)} | y^{(i)}; \phi_{yj=1}, \phi_{yj=0}) \right) p(y^{(i)}; \phi_y) \end{aligned}$$

- Maximizing this yields the maximum likelihood estimates of the parameters

$$\hat{\phi}_{k(j)=1} = \sum_{i=1}^m \sum_{j=1}^{n_i} I\{x_{ij}^{(i)} = k\} / \sum_{j=1}^{n_i} I\{x_{ij}^{(i)} = 1\}$$

$$\sum_{i=1}^m I\{y^{(i)} = 1\} / \sum_{i=1}^m n_i$$

$$\phi_{k|y=0} = \frac{\sum_{i=1}^m \sum_{j=1}^{n_i} 1 \{ X_j^{(i)} = k \wedge y^{(i)} = 0 \}}{\sum_{i=1}^m 1 \{ y^{(i)} = 0 \}}$$

$$\phi_y = \frac{1}{m} \sum_{i=1}^m 1 \{ y^{(i)} = 1 \}$$

- If we apply Laplace smoothing, when estimating $\phi_{k|y=0}$ and $\phi_{k|y=1}$, we add 1 to the numerators and $|V|$ to the denominators