Machine Learning Notes

Natali Marco

CONTENTS

LIST OF FIGURES

1 LINEAR REGRESSION

Linear Regression is in statistics a linear process to modeling the relation between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables).

The case of one explanatory variable is called *simple linear regression* and for more than one explanatory variable, the process is called *multiple linear regression*.

Suppose for example we want to predict the price of an house, given living area and the number of bedrooms, so we represent this relation as a linear regression with the hypothesis function as

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_1 x_2$$

where x_1 is the living area and x_2 is the number of bedrooms.

Here, the θ_i's are the parameters(also called *weights*) parameterizing the space of linear functions mapping from x to y and when there is no risk of confusion, we will drop the θ subscript in $h_\theta(x)$ and write it more simply as $h(x)$.

To simplify our notation, we also introduce the convention of letting $x_0 = 1$ (this is the intercept term), so we have

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

where on the right-hand side above we are viewing θ and x both as vectors, and here n is the number of input variables, without counting x_0.

1.1 LEAST SQUARES COST FUNCTION

Given the training set, to learn θ parameters a reasonable choice is to make $h(x)$ close to y, at least for the training examples we have.

To formalize this, we will define a function that measures, for each value of the θ's, how close the $h(x^{(i)})$'s are to the corresponding $y^{(i)}$'s.

We define the cost function as

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

where we divide by $\frac{1}{2}$ in order to simply derivation in future.

If you've seen linear regression before, you may recognize this as the familiar *least-squares* cost function that gives rise to the ordinary least squares regression model.
Whether or not you have seen it previously, let's keep going, and we'll eventually show this to be a special case of a much broader family of algorithms.

Our objective is to find θ that minimize $J(\theta)$ so let's consider now the *gradient descent* algorithm, which starts with some initial θ, and repeatedly performs the update:

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

This update is simultaneously performed for all values of $j = 0, 1, \dots, n$ and 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 and in order to implement this algorithm, we have to work out what is the partial derivative term on the right hand side.
Let's first work it out for the case we have only one training example (x, y), so that we can neglect the sum in the definition of J and we have

$$\begin{aligned}
\frac{\partial}{\partial\theta_j} &= \frac{\partial}{\partial\theta_j}\frac{1}{2}(h_\theta(x) - y)^2 \\
&= (h_\theta(x) - y)\frac{\partial}{\partial\theta_j}(h_\theta(x) - y) \\
&= (h_\theta(x) - y)\frac{\partial}{\partial\theta_j}(\sum_{i=0}^{n}\theta_i x_i - y) \\
&= (h_\theta(x) - y)x_j
\end{aligned}$$

For a simple training example, it provides the update rule as

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

The rule is called the *LMS update rule*(LMS stands for "least mean squares"), and is also known as the *Widrow-Hoff* learning rule.
This rule has several properties that seem natural and intuitive in fact for instance, the magnitude of the update is proportional to the error term $(y(i) - h_\theta(x^{(i)}))$; thus, for instance, if we are encountering a training example on which our prediction nearly matches the actual value of $y^{(i)}$, then we find that there is little need to change the parameters, in contrast, a larger change to the parameters will be made if our prediction $h_\theta(x^{(i)})$ has a large error.

We'd derived the LMS rule for when there was only a single training example and to modify this method for a training set of more than one example, we replace the LMS rule with the following algorithm, where we repeat until the convergence the update rule

$$\theta_j = \theta_j - \alpha(\sum_{i=0}^{m}(y^{(i)} - h_\theta(x^{(i)})x_j^{(i)} \quad \forall j = 1, 2, \dots, m$$

This is simply gradient descent on the original cost function J and this method looks at every example in the entire training set on every step, and it 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 for linear regression has only one global, and no other local optima, because J is a convex quadratic function.

Another algorithm to train the θ parameters is *stochastic descent*, where we repeatedly run through the training set, and each time we encounter a training example, we update the parameters according to the gradient of the error with respect to that single training example only.

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 can start making progress right away, and continues to make progress with each example it looks at.
Often, stochastic gradient descent gets θ "close" to the minimum much faster than batch gradient descent, but note however that it may never "converge" to the minimum, and the parameters θ will keep oscillating around the minimum of $J(\theta)$; in practice most of the values near the minimum will be reasonably good approximations to the true minimum and for these reasons, particularly when the training set is large, stochastic gradient descent is often preferred over batch gradient descent.

1.2 NORMAL EQUATION

Gradient Descent gives a way to minimize $J(\theta)$, we present now another method, where we performing the minimization explicitly and without resorting to an iterative algorithm.
In this method, we will minimize J by explicitly taking its derivatives with respect to the θ_j's, and setting them to zero.
To enable us to do this without having to write pages full of matrices of derivatives, let's introduce some notation for doing calculus with matrices.

1.2.1 Matrix derivates

For a function $f : \mathbb{R}^{m\times n} \to \mathbb{R}$ mapping from m-by-n matrices to the real numbers, we define the derivative of f with respect to A to be

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

For example suppose $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ is a 2×2 matrix and the function $f : \mathbb{R}^{2\times 2} \to \mathbb{R}$ is given by

$$f(A) = \frac{3}{2}A_{11} + 5A_{12}^2 + A_{21}A_{22}$$

so we obtain the following gradient matrix

$$\Delta_A f(A) = \begin{bmatrix} \frac{3}{2} & 10A_{12} \\ A_{22} & A_{21} \end{bmatrix}$$

We also introduce the *trace* operator defined as

$$trA = \sum_{i=1}^{n} A_{ii}$$

The trace operator has commutative, associative properties and also the following properties(proof is easy, you need only to consider the definition) with A and B as square matrices and $a \in \mathbb{R}$

$$\begin{aligned} trA &= trA^\top \\ tr(A+B) &= trA + trB \\ tr\, aA &= a\, trA \end{aligned}$$

We present now some properties about matrix derivation, without proof

$$\Delta_A trAB = B^\top \tag{1}$$
$$\Delta_{A^\top} f(A) = (\Delta_A f(A))^\top \tag{2}$$
$$\Delta_A trABA^\top C = CAB + C^\top AB^\top \tag{3}$$
$$\Delta_A |A| = |A|(A^{-1})^\top \tag{4}$$
$$\tag{5}$$

1.2.2 Least squares revised

Given a training set, define the *design matrix* X to be the $m \times n$ matrix(actually the $m \times n + 1$ if we include the intercept term), that contains the training example in this way

$$X = \begin{bmatrix} \dots & (x^{(1)})^\top & \dots \\ \dots & (x^{(2)})^\top & \dots \\ \dots & \vdots & \dots \\ \dots & (x^{(m)})^\top & \dots \end{bmatrix}$$

Also let y be the m dimensional vector that containing all the target values from the training set

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

Now since $h_\theta(x) = (x^{(i)})^\top \theta$ we can easily verify that

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

Using the fact that for a vector z, we have $z^\top z = \sum_i z_i^2$, so we obtain

$$\begin{aligned} \frac{1}{2}(X\theta - y)^\top (X\theta - y) &= \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \\ &= J(\theta) \end{aligned}$$

To minimize $J(\theta)$ we consider now the derivative with the respect to θ

$$\begin{aligned}
\Delta_\theta J(\theta) &= \Delta_\theta \frac{1}{2}(X\theta - y)^\mathsf{T}(X\theta - y) \\
&= \frac{1}{2}\Delta_\theta(\theta^\mathsf{T}X^\mathsf{T}X\theta - \theta^\mathsf{T}X^\mathsf{T}y - \theta X y^\mathsf{T} + y^\mathsf{T}y) \\
&= \frac{1}{2}\Delta_\theta tr(\theta^\mathsf{T}X^\mathsf{T}X\theta - \theta^\mathsf{T}X^\mathsf{T}y - \theta X y^\mathsf{T} + y^\mathsf{T}y) \\
&= \frac{1}{2}\Delta_\theta(tr\theta^\mathsf{T}X^\mathsf{T}\theta X - 2try^\mathsf{T}\theta X) \\
&= \frac{1}{2}(X^\mathsf{T}X\theta + X^\mathsf{T}X\theta - 2X^\mathsf{T}y) \\
&= X^\mathsf{T}X\theta - X^\mathsf{T}y
\end{aligned}$$

In the third step, we used the fact that the trace of a real numer is just the real number, the fourth step used the fact that $trA = trA^\mathsf{T}$ and the fifth step we use the second and third properties given in matrix derivation.
To minimize J, we set its derivatives to zero, and obtain the normal equations

$$X^\mathsf{T}X\theta = X^\mathsf{T}y$$

so the value of θ that minimize $J(\theta)$ is given by

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

1.3 PROBABILISTIC INTERPRETATION OF LINEAR REGRESSION

In this section, we will give a set of probabilistic assumptions, under which least-squares regression is derived as a very natural algorithm and let us assume that the target variables and the inputs are related via the equation

$$y^{(i)} = \theta^\mathsf{T}x + \epsilon^{(i)}$$

where $\epsilon(i)$ is an error term that captures either unmodeled effects, such as if there are some features very pertinent to predicting housing price, but that we'd left out of the regression, or random noise.

Let us further assume that the $\epsilon^{(i)}$ are distributed IID (independently and identically distributed) according to a Gaussian distribution as $\epsilon^{(i)} \sim N(0, \sigma^2)$, that implies that $P(y^{(i)}|x^{(i)};\theta) \sim N(\theta^\mathsf{T}x^{(i)}, \sigma^2)$.

Given X, the design matrix, which contains all the $x^{(i)}$'s, and θ, the probability of the data is given by $P(y|X;\theta)$ and this quantity is typically viewed as a function of y, and perhaps X, for a fixed value of θ.

When we wish to explicitly view this as a function of θ, we will instead call it the likelihood function

$$\begin{aligned} L(\theta) &= L(\theta|X;y) = P(y|X;\theta) \\ &= \prod_{i=1}^{m} P(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(\frac{-(y^{(i)} - \theta^\intercal x^{(i)})^2}{2\sigma^2}\right) \end{aligned}$$

Now, given this probabilistic model relating the $y^{(i)}$'s and the $x^{(i)}$'s, a reasonable way of choosing our best guess of the parameters θ, we should choose θ to maximize $L(\theta)$.

Instead of maximizing $L(\theta)$, we can also maximize any strictly increasing function of $L(\theta)$, in particular, the derivations will be a bit simpler if we instead maximize the log likelihood $l(\theta)$

$$\begin{aligned} l(\theta) &= \log L(\theta) \\ &= \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(\frac{-(y^{(i)} - \theta^\intercal y^{(i)})^2}{2\sigma^2}\right) \\ &= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(\frac{-(y^{(i)} - \theta^\intercal y^{(i)})^2}{2\sigma^2}\right) \\ &= m \log \frac{1}{\sqrt{2\pi\sigma}} - \frac{1}{\sigma^2}\frac{1}{2}\sum_{i=1}^{m}(y^{(i)} - \theta^\intercal x^{(i)})^2 \end{aligned}$$

Here maximizing $l(\theta)$ gives the same answer that minimizing

$$\frac{1}{2}\sum_{i=1}(y^{(i)} - \theta^\intercal x^{(i)})^2$$

which we recognise to be $J(\theta)$, our original least-squares cost function.

Under the previous probabilistic assumptions on the data, least-squares regression corresponds to finding the maximum likelihood estimate of θ and this is thus one set of assumptions under which least-squares regression can be justified as a very natural method that's just doing maximum likelihood estimation.

Note however that the probabilistic assumptions are by no means necessary for least-squares to be a perfectly good and rational procedure, and there may, and indeed there are, other natural assumptions that can also be used to justify it.

1.4 NEWTON OPTIMIZATION TO MAXIMISE $l(\theta)$

Specifically, suppose we have some function $f : \mathbb{R} \to \mathbb{R}$, and we wish to find a value of θ so that $f(\theta) = 0$.

Here, $\theta \in \mathbb{R}$ is a real number and Newton's method performs the following update

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

This method has a natural interpretation in which we can think of it as approximating the function f via a linear function that is tangent to f at the current guess θ, solving for where that linear function equals to zero, and letting the next guess for θ be where that linear function is zero.

Newton's method gives a way of getting to $f(\theta) = 0$ and to maximise $l(\theta)$ we have to consider that the maxima of l correspond to points where its first derivative $l'(\theta)$ is zero.
So, by letting $f(\theta) = l'(\theta)$, we can use the same algorithm to maximize l, and we obtain the update rule

$$\theta = \theta - \frac{l'(\theta)}{l''(\theta)}$$

The generalization of Newton's method to multidimensional setting(also called the *Newton-Raphson* method) is given by

$$\theta = \theta - H^{-1}\Delta_\theta \, l(\theta)$$

where H^{-1} is the second derivative called *Hessian*.

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 can, however, be more expensive than one iteration of gradient descent, since it requires finding and inverting an $n \times n$ Hessian; but so long as n is not too large, it is usually much faster over all.
When Newton's method is applied to maximize the logistic regression log likelihood function $l(\theta)$, the resulting method is also called *Fisher scoring*.

1.5 LOCALLY WEIGHTED REGRESSION

Learning algorithms can be divided in two different categories:

PARAMETRICS: fit fixed set of parameters θ to data

NON PARAMETRICS: you have to keep an amount of data/parameters, that grows with size of data

Locally Weight Regression is our first example of non parametric learning algorithm, where to predict from a point x_i you have to watch their neighbors.

The objective is to fit θ to minimize the objective function

$$\sum_{i=1}^{n} w^{(i)}(y^{(i)} - \theta^\top x^{(i)})^2$$

where $w^{(i)}$ is a weight function and a common choice of weight function is the following

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

Note that the weights depend on the particular point x at which we're trying to evaluate x so if $|x^{(i)} - \hat{x}|$ is small the weight is close to 1, instead if $|x^{(i)} - \hat{x}|$ is large the weight is close to 0.

Hence, θ is chosen giving a much higher "weight" to the (errors on) training examples close to the query point x and note also that while the formula for the weights takes a form that is cosmetically similar to the density of a Gaussian distribution, the $w(i)$'s do not directly have anything to do with Gaussians, and in particular the $w(i)$ are not random variables, normally distributed or otherwise.

The parameter τ controls how quickly the weight of a training example falls off with distance of its $x^{(i)}$ from the query point x and it is called the *bandwidth* parameter, which choice of value is important to avoid overfitting and underfitting.

It is commonly used when we have a lot of data and we don't want to think about which features to use.

2 CLASSIFICATION

In statistics *Classification* is the problem of identifying to which of a set of categories(sub-populations) a new observation belongs, on the basis of a training set of data containing observations whose category membership is known.

Examples are assigning a given email to the "spam" or "non-spam" class, assigning a diagnosis to a given patient based on observed characteristics of the patient(sex, blood pressure, presence or absence of certain symptoms, etc.) and classification is an example of pattern recognition.

In the terminology of machine learning, classification is considered an instance of supervised learning, in which we learn based on training set of correctly identified observations.

For now, we will focus on the binary classification problem in which y can take on only two values, 0 and 1, but most of what we say here will also generalize to the multiple-class case.
For instance, if we are trying to build a spam classifier for email, then $x^{(i)}$ may be some features of a piece of email, and y may be 1 if it is a piece of spam mail, and 0 otherwise.
0 is also called the *negative class*, and 1 the *positive class* and given $x^{(i)}$, the corresponding $y^{(i)}$ is also called the *label* for the training example.

2.1 LOGISTIC REGRESSION

We could approach the classification problem ignoring the fact that y is discrete-valued, and use our old linear regression algorithm to try to predict y given x.
However, it is easy to construct examples where this method performs very poorly, in fact intuitively, it also doesn't make sense for $h_\theta(x)$ to take values larger than 1 or smaller than 0 when we know that $y \in \{0, 1\}$.

To fix this, let's change the form for our hypotheses $h_\theta(x)$ and we will choose

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

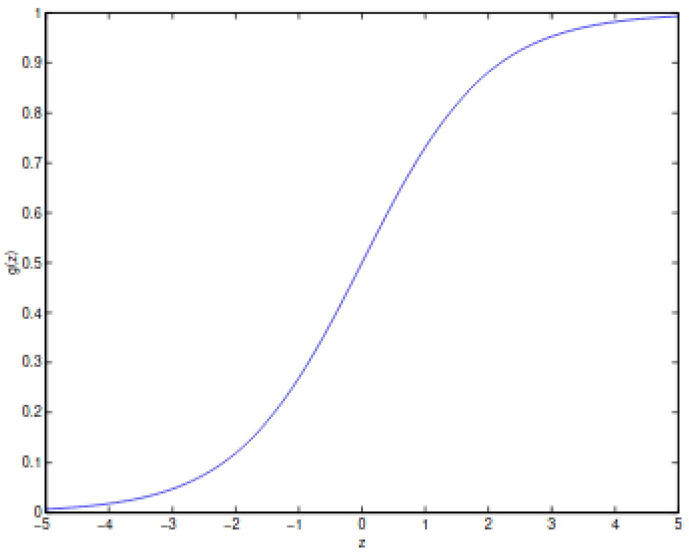
where is called *sigmoid function* the equation

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

In figure 1 is showed the function $g(z)$, where $g(z)$ tends to 1 as $z \to \infty$ and tends to 0 as $z \to -\infty$.

For now, let's take the choice of g as given, in fact other functions that smoothly increase from 0 to 1 can also be used, but for a couple of reasons that we'll see later, when we talk about GLMs, and when we talk about

Figure 1: Representation of Sigmoid function



generative learning algorithms, the choice of the logistic function is a fairly natural one.

Before moving on, here's a useful property of the derivative of the sigmoid function, which we write as g'

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

So, given the logistic regression model, to fit θ we follow how we saw least squares regression could be derived as the maximum likelihood estimator under a set of assumptions, let's endow our classification model with a set of probabilistic assumptions, and then fit the parameters via maximum likelihood.

Let us assume that

$$
\begin{aligned}
P(y=1|x;\theta) &= h_\theta(x) \\
P(y=0|x;\theta) &= 1 - h_\theta(x)
\end{aligned}
$$

Assuming that the m training examples were generated independently, we can then write down the likelihood of the parameters as

$$\begin{aligned} L(\theta) &= p(y|x;\theta) \\ &= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{m} (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \end{aligned}$$

As before, it is more easy to maximise the log lihelikood

$$\begin{aligned} l(\theta) &= \log L(\theta) \\ &= \prod_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \end{aligned}$$

Similar to our derivation in the case of linear regression, we can use *gradient ascent* and written invectorial notation, our updates will therefore be given by

$$\theta = \theta + \alpha\, \Delta_\theta\, l(\theta)$$

Let's start by working with just one training example (x, y), and take derivatives to derive the stochastic gradient ascent rule

$$\begin{aligned} \frac{\partial}{\partial\theta_j} l(\theta) &= \left(y \frac{1}{g(\theta^\intercal x)} - (1-y) \frac{1}{1 - g(\theta^\intercal x)} \right) \frac{\partial}{\partial\theta_j} g(\theta^\intercal x) \\ &= \left(y \frac{1}{g(\theta^\intercal x)} - (1-y) \frac{1}{1 - g(\theta^\intercal x)} \right) g(\theta^\intercal x)(1 - g(\theta^\intercal x) \frac{\partial}{\partial\theta_j} \theta^\intercal x \\ &= (y(1 - g(\theta^\intercal x)) - (1-y) g(\theta^\intercal x)) x_j \\ &= (y - h_\theta(x)) x_j \end{aligned}$$

Above we used the fact that $g'(x) = g(x)(1 - g(x))$ and therefore it gives us the stochastic gradient ascent rule

$$\theta_j = \theta_j + \alpha(y - h_\theta(x)) x_j$$

If we compare this to the LMS update rule, we see that it looks identical, but this is not the same algorithm, because $h_\theta(x^{(i)})$ is now defined as a non-linear function.
Nonetheless, it's a little surprising that we end up with the same update rule for a rather different algorithm and learning problem, as how we will deeper talk in GLM(Generalized Linear Models) chapter.

2.2 PERCEPTRON ALGORITHM

We now digress to talk briefly about an algorithm that's of some historical interest, and that we will also return to later when we talk about learning

theory.
Consider modifying the logistic regression method to "force" it to output values that are either 0 or 1.

To do so, it seems natural to change the definition of g to be the threshold function

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

In the 1960s, this "perceptron" was argued to be a rough model for how individual neurons in the brain work and given how simple the algorithm is, it will also provide a starting point for our analysis when we talk about learning theory later.

Note however that even though the perceptron maybe cosmetically similar to the other algorithms we talked about, it is actually a very different type of algorithm than logistic regression and least squares linear regression; in particular, it is difficult to endow the perceptron's predictions with meaningful probabilistic interpretations, or derive the perceptron as a maximum likelihood estimation algorithm.

3 GENERALIZED LINEAR MODELS

So far, we've seen a regression example, and a classification example: in the regression example, we had $y|x;\theta \sim N(\mu, \sigma^2)$, and in the classification one $y|x;\theta \sim Bernoulli(\phi)$, for some appropriate definitions of μ and ϕ as 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*(GLMs) and we will also show how other models in the GLM family can be derived and applied to other classification and regression problems.

3.1 EXPONENTIAL FAMILY

To work our way up to GLMs, we will begin by defining exponential family distributions and we say that a class of distributions is in the exponential family if it can be written in the form

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

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

A fixed choice of T, a and b defines a family(or set) of distributions that is parameterized by η; as we vary η, we then get different distributions within this family and we now show that the Bernoulli and the Gaussian distributions are examples of exponential family distributions.

The Bernoulli distribution with mean ϕ, written $Bernoulli(\phi)$, specifies a distribution over $y \in \{0,1\}$, so that $p(y=1|\phi) = \phi$ and $p(y=0|\phi) = 1-\phi$. As we vary ϕ, we obtain Bernoulli distributions with different means and we now show that this class of Bernoulli distributions, ones obtained by varying ϕ, is in the exponential family.

We write the Bernoulli distribution as

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

Thus, the natural parameter is given by $\eta = \log(\frac{\phi}{1-\phi})$ and it is interestingly, if we invert this definition for η by solving for ϕ in terms of η, we obtain

$$\phi = \frac{1}{1+e^{-\eta}}$$

This is the familiar sigmoid function and this will come up again when we derive logistic regression as a GLM.

To complete the formulation of the Bernoulli distribution as an exponential family distribution, we also have

$$\begin{aligned} T(y) &= y \\ a(\eta) &= -\log(1-\phi) \\ &= \log(1+e^{\eta}) \\ b(y) &= 1 \end{aligned}$$

Let's now move on to consider the Gaussian distribution and recall that, when deriving linear regression, the value of σ^2 had no effect on our final choice of θ and $h_\theta(x)$.

Thus, we can choose an arbitrary value for σ^2 without changing anything and to simplify the derivation below, let's set $\sigma^2 = 1$, so we have

$$\begin{aligned} P(y|\mu) &= \frac{1}{\sqrt{2\pi}} exp\left(-\frac{1}{2}(y-\mu)^2\right) \\ &= \frac{1}{\sqrt{2\pi}} exp\left(-\frac{y^2}{2}\right) exp\left(\mu y - \frac{\mu^2}{2}\right) \end{aligned}$$

Thus, we see that the Gaussian is in the exponential family, with

$$\begin{aligned} \eta &= \mu \\ T(y) &= y \\ b(y) &= \frac{1}{\sqrt{2\pi}} exp(\frac{-y^2}{2}) \\ a(\eta) &= \frac{\mu^2}{2} = \frac{\eta^2}{2} \end{aligned}$$

There're many other distributions that are members of the exponential family: the multinomial, the Poisson (for modelling count data), the gamma and the exponential (for modelling continuous, non-negative random variables, such as time-intervals), the beta and the Dirichlet (for distributions over probabilities) and many more.
In the next section, we will describe a general "recipe" for constructing models in which y (given x and θ) comes from any of these distributions.

3.2 CONSTRUCTING GLM

Suppose you would like to build a model to estimate the number y of customers arriving in your store (or number of page views on your website) in any given hour, based on certain features x such as store promotions, recent advertising, weather, day of week, etc.

We know that the Poisson distribution usually gives a good model for numbers of visitors and knowing this, how can we come up with a model for our problem? Fortunately, the Poisson is an exponential family distribution, so we can apply a Generalized Linear Model(GLM).
In this section, we will describe a method for constructing GLM models for problems such as these and more generally, 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 about the conditional distribution of y given x and about our model:

1. $y; x, \theta \sim ExponentialFamily(\eta)$.

2. Given x, our goal is to predict the expected value of $T(y)$ given x and in most of our examples, we will have $T(y) = y$, so this means we would like the prediction $h(x)$ output by our learned hypothesis h to satisfy $h(x) = E[y|x]$.

3. The natural parameter η and the inputs x are related linearly $\eta = \theta^{\mathsf{T}} x$; if η is vector valued, then $\eta_i = \theta_i^{\mathsf{T}} x$.

The third of these assumptions might seem the least well justified of the above, and it might be better thought of as a "design choice" in our recipe for designing GLMs, rather than as an assumption per se and these three assumptions/design choices will allow us to derive a very elegant class of learning algorithms, namely GLMs, that have many desirable properties such as ease of learning.

Furthermore, the resulting models are often very effective for modelling different types of distributions over y; for example, we will shortly show that both logistic regression and ordinary least squares can both be derived as GLMs.

3.2.1 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, and we model the conditional distribution of y given x as a Gaussian $N(\mu, \sigma^2)$.

So, we let the ExponentialFamily(η) distribution above be the Gaussian distribution and as we saw previously, in the formulation of the Gaussian as an exponential family distribution, we had $\mu = \eta$ and so we have

$$\begin{aligned} h_\theta(x) &= E[y|x;\theta] \\ &= \mu \\ &= \eta \\ &= \theta^{\mathsf{T}} x \end{aligned}$$

3.2.2 Logistic Regression

We now consider logistic regression and here we are interested in binary classification, so $y \in \{0, 1\}$.

Given that y is binary valued, it therefore seems natural to choose the Bernoulli family of distributions to model the conditional distribution of y given x and in our formulation of the Bernoulli distribution as an exponential family distribution, we had $\phi = \frac{1}{1+e^{-\eta}}$.

Furthermore, note that if $y|x;\theta \sim Bernoulli(\phi)$, then $E[y|x;\theta] = \phi$, so following a similar derivation as the one for ordinary least squares, we get

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

So, this gives us hypothesis functions of the form

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

.

To introduce a little more terminology, the function g giving the distribution's mean as a function of the natural parameter ($g(\eta) = E[T(y);\eta]$) is called the *canonical response function*.
Its inverse, g^{-1}, is called the *canonical link function* and thus, the canonical response function for the Gaussian family is just the identify function and the canonical response function for the Bernoulli is the logistic function.

3.3 SOFTMAX REGRESSION

Consider a classification problem in which the response variable y can take on any one of k values, so $y \in \{1, 2, \ldots, k\}$; for example, rather than classifying email into the two classes spam or not spam, which would have been a binary classification problem, we might want to classify it into three classes, such as spam, personal mail, and work related mail.
The response variable is still discrete, but can now take on more than two values and we will thus model it as distributed according to a multinomial distribution.

Let's derive a GLM for modelling this type of multinomial data and to do so, we will begin by expressing the multinomial as an exponential family distribution.
To parameterize a multinomial over k possible outcomes, one could use k parameters $\phi_1, \ldots, \phi_k$ specifying the probability of each of the outcomes, however, these parameters would be redundant, or more formally, they would

not be independent, since knowing any $k-1$ of the ϕ_i's uniquely determines the last one, as they must satisfy $\sum_{i=1}^{k} \phi_i = 1$.

So, we will instead parameterize the multinomial with only $k-1$ parameters $\phi_1, \ldots, \phi_{k-1}$, where $\phi_i = P(y=i;\phi)$, and $P(y=k;\phi) = 1 - \sum_{i=1}^{k-1} \phi_i$.

For notational convenience, we will also let $\phi_k = 1 - \sum_{i=1}^{k-1} \phi_i$, but we should keep in mind that this is not a parameter, and that it is fully specified by $k-1$ parameters.

To express the multinomial as an exponential family distribution, we will define $T(y) \in \mathbb{R}^{k-1}$ in which T_i has 0 in all entries except 1 in i-th row.

Unlike our previous examples, here we do not have $T(y) = y$ and also, $T(y)$ is now a $k-1$ dimensional vector, rather than a real number.

We will write $(T(y))_i$ to denote the i-th element of the vector $T(y)$.

We introduce one more very useful piece of notation: an indicator function $1\{*\}$ takes on a value of 1 if its argument is true, and 0 otherwise ($1\{True\} = 1, 1\{False\} = 0$).

So, we can also write the relationship between $T(y)$ and y as $(T(y))_i = 1\{y=i\}$ and further, we have that $E[(T(y))_i] = P(y=i) = \phi_i$.

We are now ready to show that the multinomial is a member of the exponential family, infact we have

$$\begin{aligned}
P(y|\phi) &= \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} \ldots \phi_k^{1\{y=k\}} \\
&= \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} \ldots \phi_k^{1-\sum_{i=1}^{k-1} 1\{y=i\}} \\
&= \phi_1^{(T(y))_1} \phi_2^{(T(y))_2} \ldots \phi_k^{1-\sum_{i=1}^{k-1} (T(y))_i} \\
&= exp\left((T(y))_1 \log \phi_1 + (T(y))_2 \log \phi_2 + \cdots + (1 - \sum_{i=1}^{k-1} (T(y))_i) \log \phi_k \right) \\
&= exp\left((T(y))_1 \log\left(\frac{\phi_1}{\phi_k}\right) + (T(y))_2 \log\left(\frac{\phi_2}{\phi_k}\right) + \cdots + (T(y))_{k-1} \log\left(\frac{\phi_{k-1}}{\phi_k}\right) + \log \phi_k \right) \\
&= b(y) exp(\eta^T T(y) - a(\eta))
\end{aligned}$$

where we have

$$\begin{aligned}
\eta &= \begin{bmatrix} \log(\frac{\phi_1}{\phi_k}) \\ \log(\frac{\phi_2}{\phi_k}) \\ \vdots \\ \log(\frac{\phi_{k-1}}{\phi_k}) \end{bmatrix} \\
a(\eta) &= -\log \phi_k \\
b(y) &= 1
\end{aligned}$$

This completes our formulation of the multinomial as an exponential family distribution.

The link function is given, for $i = 1, \ldots, k$ by

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

For convenience we have also that $\eta_k = \log \frac{\phi_k}{\phi_k} = 0$ and to invert the link function and derive the response function, we therefore have that

$$\begin{aligned} e^{\eta_i} &= \frac{\phi_i}{\phi_k} \\ \phi_k e^{\eta_i} &= \phi_i \\ \phi_k \sum_{i=1}^{k} e^{\eta_i} &= \sum_{i=1}^{k} \phi_i = 1 \end{aligned}$$

This implies that $\phi_k = \dfrac{1}{\sum_{i=1}^{k} e^{\eta_i}}$, that gives the response function

$$\phi_i = \frac{e^{\eta_i}}{\sum_{j=1}^{k} e^{\eta_j}}$$

This function mapping from the η's to the ϕ's is called the *softmax function*.

To complete our model, we assume that η_i's are linearly related to the x's, so, we have $\eta_i = \theta_i^{\mathsf{T}} x$ (for $i = 1, \ldots, k-1$), where $\theta_1, \ldots, \theta_{k-1} \in \mathbb{R}^{d+1}$ are the parameters of our model.
For notational convenience, we can also define $\theta_k = 0$, so that $\eta_k = \theta_k^{\mathsf{T}} x = 0$, as given previously and hence, our model assumes that the conditional distribution of y given x is given by

$$\begin{aligned} P(y = i | x; \theta) &= \phi_i \\ &= \frac{e^{\eta_i}}{\sum_{j=1}^{k} e^{\eta_j}} \\ &= \frac{e^{\theta_i^{\mathsf{T}} x}}{\sum_{j=1}^{k} e^{\theta_j^{\mathsf{T}} x}} \end{aligned}$$

This model, which applies to classification problems where $y \in \{1, \dots, k\}$, is called *softmax regression* and it is a generalization of logistic regression.
Our hypothesis will output

$$\begin{aligned} h_\theta(x) &= E[T(y)|x;\theta] \\ &= \begin{bmatrix} 1\{y=1\} \\ 1\{y=2\} \\ \dots \\ 1\{y=k-1\} \end{bmatrix} \\ &= \begin{bmatrix} \phi_1 \\ \phi_2 \\ \dots \\ \phi_{k-1} \end{bmatrix} \\ &= \begin{bmatrix} \frac{exp(\theta_1^\top x)}{\sum_{j=1}^k exp(\theta_j^\top x)} \\ \frac{exp(\theta_2^\top x)}{\sum_{j=1}^k exp(\theta_j^\top x)} \\ \dots \\ \frac{exp(\theta_{k-1}^\top x)}{\sum_{j=1}^k exp(\theta_j^\top x)} \end{bmatrix} \end{aligned}$$

In other words, our hypothesis will output the estimated probability that $P(y = i|x;\theta)$, for every value of $i = 1, \dots, k$ and even though $h_\theta(x)$ as defined above is only $k-1$ dimensional, clearly $P(y = k|x;\theta)$ can be obtained as $1 - \sum_{i=1}^{k-1} \phi_i$.
Lastly, let's discuss parameter fitting: similar to our original derivation of ordinary least squares and logistic regression, if we have a training set of n examples $\{(x(i), y(i)); i = 1, \dots, n\}$ and would like to learn the parameters θ_i of this model, we would begin by writing down the log-likelihood

$$\begin{aligned} l(\theta) &= \sum_{i=1}^n \log P(y^{(i)}|x^{(i)};\theta) \\ &= \sum_{i=1}^n \log \prod_{l=1}^k \left(\frac{e^{\theta_l^\top x^{(i)}}}{\sum_{j=1}^k e^{\theta_j^\top x^{(i)}}}\right)^{1\{y^{(i)}=l\}} \end{aligned}$$

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 ascent or Newton's method.

4 GENERATIVE LEARNING ALGORITHMS

So far, we've mainly been talking about learning algorithms that model $p(y|x;\theta)$, the conditional distribution of y given x; for instance, logistic regression modeled $p(y|x;\theta)$ as $h_\theta(x) = g(\theta^\mathsf{T} x)$ where g is the sigmoid function.

In these notes, we'll talk about a different type of learning algorithm: consider a classification problem in which we want to learn to distinguish between elephants ($y = 1$) and dogs ($y = 0$), based on some features of an animal and given a training set, an algorithm like logistic regression or the perceptron algorithm tries to find a straight line, that is, a decision boundary, that separates the elephants and dogs.

Then, to classify a new animal as either an elephant or a dog, it checks on which side of the decision boundary it falls, and makes its prediction accordingly but now we introduce a different approach.

First, looking at elephants, we can build a model of what elephants look like, then, looking at dogs, we can build a separate model of what dogs look like, so finally, to classify a new animal, we can match the new animal against the elephant model, and match it against the dog model, to see whether the new animal looks more like the elephants or more like the dogs we had seen in the training set.

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

Here, we'll talk about algorithms that instead try to model $p(x;y)$ and $p(y)$ and these algorithms are called *generative learning algorithms*; for instance, if y indicates whether an example is a dog (0) or an elephant (1), then $p(x|y = 0)$ models the distribution of dogs' features, and $p(x|y = 1)$ models the distribution of elephants' features.

After modeling $p(y)$, called the *class priors* and $p(x;y)$, our algorithm can then use Bayes rule to derive the posterior distribution on y given x

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

where the denominator is given by

$$p(x) = p(x|y = 1)p(y = 1) + p(x|y = 0)p(y = 0)$$

and thus can also be expressed in terms of the quantities $p(x|y)$ and $p(y)$ that we've learned.

Actually, if were calculating $p(y|x)$ in order to make a prediction, then we don't actually need to calculate the denominator, since

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

4.1 GAUSSIAN DISCRIMINANT ANALYSIS

The first generative learning algorithm that we'll look at is *Gaussian discriminant analysis(GDA)* and in this model, we'll assume that $p(x|y)$ is distributed according to a multivariate normal distribution.

The *multivariate normal distribution* in n-dimensions, also called the multivariate Gaussian distribution, is parameterized by a mean vector $\mu \in \mathbb{R}^n$ and a covariance matrix $\Sigma \in \mathbb{R}^{n\times n}$, where $\Sigma \geq 0$ is symmetric and positive semidefinite.

Also written as $N(\mu, \Sigma)$ its density is given by

$$P(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$

In the equation above, $|\Sigma|$ denotes the determinant of the matrix Σ and for a random variable X distributed as $N(\mu, \Sigma)$, the mean is (unsurprisingly) given by μ

$$E[X] = \int_x xp(x;\mu,\Sigma)dx = \mu$$

The covariance of a vector-valued random variable Z is defined as

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

This generalizes the notion of the variance of a real-valued random variable and the covariance can also be defined as

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

If $X \sim N(\mu, \Sigma)$ then $Cov(X) = \Sigma$

4.1.1 The Gaussian Discriminant Analysis model

When we have a classification problem in which the input features x are continuous valued random variables, we can then use the Gaussian Discriminant Analysis(GDA) model, which models $p(x;y)$ using a multivariate normal distribution, as we can see now:

$$\begin{aligned} y &\sim Bernoulli(\phi) \\ x|y = 0 &\sim N(\mu_0, \Sigma) \\ x|y = 1 &\sim N(\mu_1, \Sigma) \end{aligned}$$

Writing out the distributions, we obtain that

$$
\begin{aligned}
p(y) &= \phi^y(1-\phi)^{1-y} \\
p(x|y=0) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)) \\
p(x|y=1) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1))
\end{aligned}
$$

Here, the parameters of our model are ϕ, Σ, μ_0 and μ_1 (note that while there're two different mean vectors μ_0 and μ_1, this model is usually applied using only one covariance matrix Σ) and the log-likelihood of the data is given by

$$
\begin{aligned}
l(\phi, \mu_0, \mu_1, \Sigma) &= \log \sum_{i=1}^{m} p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) \\
&= \log \sum_{i=1}^{m} p(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma) p(y^{(i)}; \phi)
\end{aligned}
$$

By maximizing l with respect to the parameters, we find the maximum likelihood estimate of the parameters to be

$$
\begin{aligned}
\phi &= \frac{1}{m} \sum_{i=1}^{m} 1\{y^{(i)} = 1\} \\
\mu_0 &= \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 0\} x(i)}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}} \\
\mu_1 &= \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\} x(i)}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}} \\
\Sigma &= \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu_{y(i)})(x^{(i)} - \mu_{y^{(i)}})^{\mathsf{T}}
\end{aligned}
$$

Pictorially, what the algorithm is doing can be seen in figure 2.

4.1.2 Discussion: GDA and logistic regression

The GDA model has an interesting relationship to logistic regression, in fact if we view the quantity

$$p(y = 1|x; \phi, \mu_0, \mu_1, \Sigma)$$

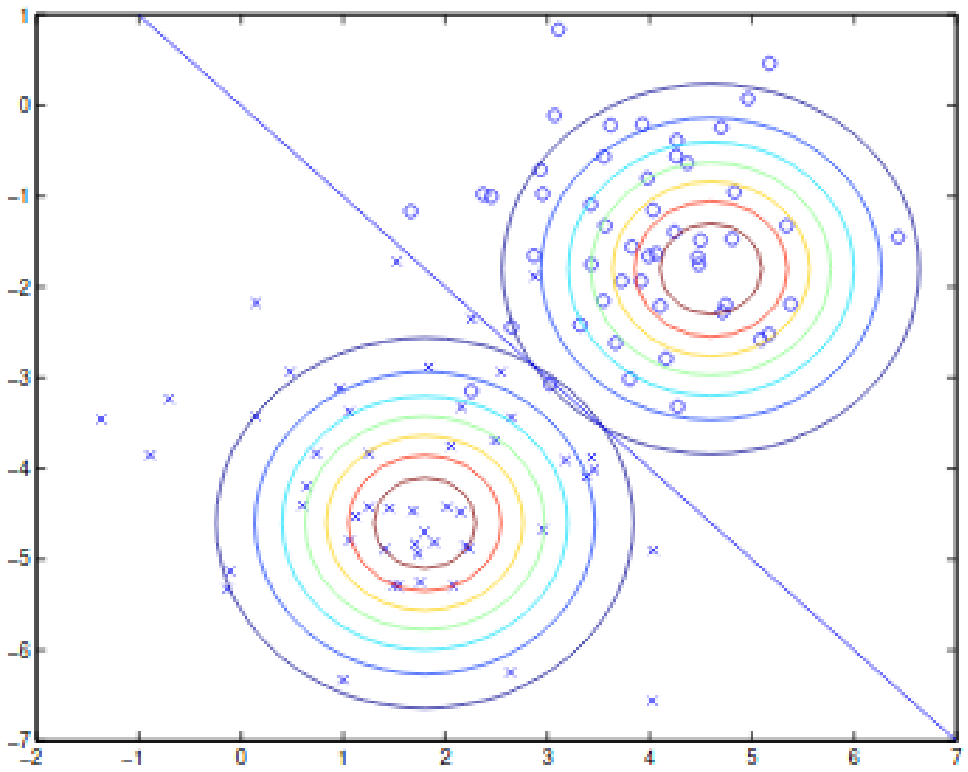
as a function of x, we'll find that it can be expressed in the form

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

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

This is exactly the form that logistic regression, a discriminative algorithm, used to model $p(y = 1|x)$ and GDA and logistic regression will, in general, give different decision boundaries when trained on the same dataset.
We just argued that if $p(x|y)$ is multivariate gaussian(with shared Σ), then $p(y|x)$ necessarily follows a logistic function and the converse, however, is not

Figure 2: GDA representation example



true, so $p(y|x)$ being a logistic function does not imply $p(x|y)$ is multivariate gaussian.

This shows that GDA makes stronger modeling assumptions about the data than does logistic regression and it turns out that when these modeling assumptions are correct, then GDA will find better fits to the data, so it is a better model.

Specifically, when $p(x|y)$ is indeed gaussian (with shared Σ), then GDA is asymptotically efficient and informally, this means that in the limit of very large training sets, there is no algorithm that is strictly better than GDA, in terms of how accurately they estimate $p(y|x)$.

In particular, it can be shown that in this setting, GDA will be a better algorithm than logistic regression and more generally, even for small training set sizes, we would generally expect GDA to better.
In contrast, by making significantly weaker assumptions, logistic regression is also more robust and less sensitive to incorrect modeling assumptions and there are many different sets of assumptions that would lead to $p(y|x)$ taking the form of a logistic function; for example, if $x|y = 0 \sim Poisson(\lambda_0)$, and $x|y = 1 \sim Poisson(\lambda_1)$, then $p(y|x)$ will be logistic.

Logistic regression will also work well on Poisson data like this, but if we were to use GDA on such data, and fit Gaussian distributions to such non-Gaussian data, then the results will be less predictable, and GDA may (or may not) do well.

Specifically, when the data is indeed non-Gaussian, then in the limit of large datasets, logistic regression will almost always do better than GDA and for this reason, in practice logistic regression is used more often than GDA.

4.2 NAIVE BAYES

In GDA, the feature vectors x were continuous, real valued vectors and let's now talk about a different learning algorithm in which the x_j's are discrete-valued.

For our motivating example, consider building an email spam filter using machine learning and here, we wish to classify messages according to whether they are unsolicited commercial(spam) email, or non-spam email.

After learning to do this, we can then have our mail reader automatically filter out the spam messages and perhaps place them in a separate mail folder: classifying emails is one example of a broader set of problems called *text classification*.

Let's say we have a training set (a set of emails labeled as spam or non-spam) and we'll begin our construction of our spam filter by specifying the features x_j used to represent an email.

We will represent an email via a feature vector whose length is equal to the number of words in the dictionary and specifically, if an email contains the j-th word of the dictionary, then we will set $x_j = 1$ otherwise, we let $x_j = 0$ so for instance, the vector

$$x = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

is used to represent an email that contains the words "a" and "buy" but not "aardvark", "aardwolf" or "zygmurgy.".

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 and having chosen our feature vector, we now want to build a generative model, so we have to model $p(x|y)$.

If we have a vocabulary of 50000 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'd end up with a $(2^{50000} - 1)$-dimensional parameter vector so this is clearly too many parameters.

To model $p(x|y)$, we will therefore make a very strong assumption, so we will assume that the x_i's are conditionally independent given y and this assumption is called the Naive Bayes (NB) assumption, and the resulting algorithm is called the *Naive Bayes classifier*.

We now have

$$\begin{aligned} p(x_1, \ldots, x_{50000}|y) &= p(x_1|y)p(x_2|y, x_1)p(x_3|y, x_1, x_2) \ldots p(x_{50000}|y, x_1, \ldots, x_{49999}) \\ &= p(x_1|y)p(x_2|y)p(x_3|y) \ldots p(x_{50000}|y) \\ &= \prod_{i=1}^{n} p(x_i|y) \end{aligned}$$

The first equality simply follows from the usual properties of probabilities, the second equality used the NB assumption and we note that even though the Naive Bayes assumption is an extremely strong assumptions, the resulting algorithm works well on many problems.

Our model is parameterized by $\phi_{j|y=1} = p(x_j = 1|y = 1)$, $phi_{j|y=0} = p(x_j = 1|y = 0)$, and $\phi_y = p(y = 1)$ and as usual, given a training set $\{(x^{(i)}, y^{(i)}) \forall i = 1, \ldots, m\}$, we can write down the joint likelihood of the data as

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

Maximizing this with respect to $\phi_y, \phi_{j|y=0}$ and $\phi_{j|y=1}$ gives the maximum likelihood estimates

$$\begin{aligned} \phi_{j|y=1} &= \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^m 1\{y^{(i)} = 1\}} \\ \phi_{j|y=0} &= \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^m 1\{y^{(i)} = 0\}} \\ \phi_j &= \frac{\sum_{i=1}^m 1\{y^{(i)} = 1\}}{m} \end{aligned}$$

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

$$\begin{aligned} p(y = 1|x) &= \frac{p(x|y = 1)p(y = 1)}{p(x)} \\ &= \frac{\left(\sum_{j=1}^n p(x_j|y = 1)\right) p(y = 1)}{\left(\sum_{j=1}^n p(x_j|y = 1)\right) p(y = 1) + \left(\sum_{j=1}^n p(x_j|y = 1)\right) p(y = 0)} \end{aligned}$$

and pick whichever class has the higher posterior probability.

Lastly, we note that while we have developed the Naive Bayes algorithm mainly for the case of problems where the features x_j are binary valued, the generalization to where x_j can take values in $\{1, 2, \ldots, k_j\}$ is straightforward, so we would simply model $p(x_j|y)$ as multinomial rather than as Bernoulli.

Indeed, even if some original input attribute were continuous valued, it is quite common to discretize it, that is, turn it into a small set of discrete values, and apply Naive Bayes.

4.3 LAPLACE SMOOTHING

The Naive Bayes algorithm as we have described it will work fairly well for many problems, but there is a simple change that makes it work much better, especially for text classification.

Let's briefly discuss a problem with the algorithm in its current form, and then talk about how we can fix it so we consider spam/email classification, and let's suppose that, after completing the ML course, you decide to submit the work you did to the NIPS conference for publication (NIPS is one of the top machine learning conferences, and the deadline for submitting a paper is typically in late June or early July).

Because you end up discussing the conference in your emails, you also start getting messages with the word "nips" in it, but this is your first NIPS paper, and until this time, you had not previously seen any emails containing the word "nips"; in particular "nips" did not ever appear in your training set of spam/non-spam emails.

Assuming that "nips" was the 35000-th word in the dictionary, your Naive Bayes spam filter therefore had picked its maximum likelihood estimates of the parameters $\phi_{35000|y}$ to be

$$\phi_{35000|y=1} = \frac{\sum_{i=1}^{m} 1\{x_{35000}^{(i)} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}$$

$$\phi_{35000|y=0} = \frac{\sum_{i=1}^{m} 1\{x_{35000}^{(i)} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}}$$

Because it has never seen "nips" before in either spam or non-spam training examples, it thinks the probability of seeing it in either type of email is zero and hence, when trying to decide if one of these messages containing "nips" is spam, it calculates the class posterior probabilities, and obtains $p(y = 1|x) = \frac{0}{0}$.

This is because each of the terms $\prod_{j=1}^{n} p(x_j|y)$ includes a term $p(x_{35000}|y) = 0$ that is multiplied into it, so our algorithm obtains $0/0$, and doesn't know how to make a prediction.

Stating the problem more broadly, it is statistically a bad idea to estimate the probability of some event to be zero just because you haven't seen it before in your finite training set.

To avoid this, we can use *Laplace smoothing*, which uses

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

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

In practice, it usually doesn't matter much whether we apply Laplace smoothing to ϕ_y or not, since we will typically have a fair 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.