CS229 Lecture notes

Andrew Ng

## Part IV

# 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 $h_\theta(x) = g(\theta^T x)$ where $g$ is the sigmoid function. In these notes, we'll talk about a different type of learning algorithm.

假设有两类，A类和B类，生成学习算法是指对两类分别建立一个模型。然后看样本能更好的匹配哪个模型。

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. Given a training set, an algorithm like logistic regression or the perceptron algorithm (basically) 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.

Here's 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. 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 $\mathcal{X}$ to the labels $\{0, 1\}$, (such as the perceptron algorithm) are called **discriminative** learning algorithms. Here, we'll talk about algorithms that instead try to model $p(x|y)$ (and $p(y)$). These algorithms are called **generative** learning algorithms. For instance, if $y$ indicates whether a 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.

判别式算法和生成式算法的区别

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 $x$:

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

Here, the denominator is given by $p(x) = p(x|y = 1)p(y = 1) + p(x|y = 0)p(y = 0)$ (you should be able to verify that this is true from the standard properties of probabilities), 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

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

# 1 Gaussian discriminant analysis 高斯判别分析

The first generative learning algorithm that we'll look at is Gaussian discriminant analysis (GDA). In this model, we'll assume that $p(x|y)$ is distributed according to a multivariate normal distribution. Lets talk briefly about the properties of multivariate normal distributions before moving on to the GDA model itself.

## 1.1 The multivariate normal distribution 多元高斯分布

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 semi-definite. Also written "$\mathcal{N}(\mu, \Sigma)$", its density is given by:

x样本（注意是个向量）
u均值（注意是个向量）
∑协方差矩阵

n 个变量的均值组成 mean vector

n 个变量，两两变量求协方差构成协方差矩阵 covariance matrix

|∑|协方差矩阵的行列式的值

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

其实就是协方差矩阵
协方差矩阵是对称阵，并且是半正定的

In the equation above, "$|\Sigma|$" denotes the determinant of the matrix $\Sigma$.

For a random variable $X$ distributed $\mathcal{N}(\mu, \Sigma)$, the mean is (unsurprisingly,) given by $\mu$:

$$\mathrm{E}[X] = \int_x x\, p(x; \mu, \Sigma)dx = \mu$$

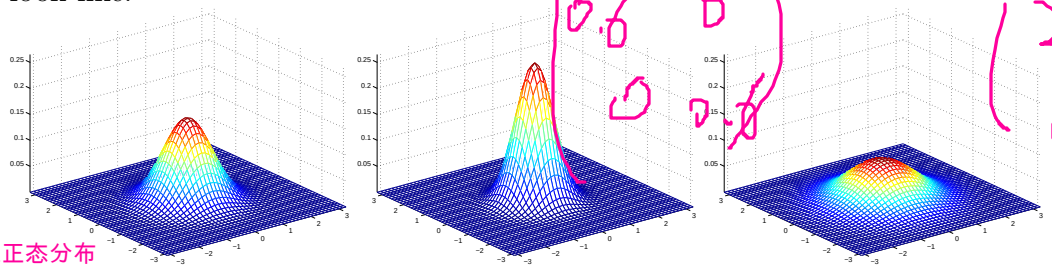
相关证明
https://blog.csdn.net/qcyfred/article/details/71598815

The **covariance** of a vector-valued random variable $Z$ is defined as $\mathrm{Cov}(Z) = \mathrm{E}[(Z - \mathrm{E}[Z])(Z - \mathrm{E}[Z])^T]$. This generalizes the notion of the variance of a

real-valued random variable. The covariance can also be defined as $\mathrm{Cov}(Z) = \mathrm{E}[ZZ^T] - (\mathrm{E}[Z])(\mathrm{E}[Z])^T$. (You should be able to prove to yourself that these two definitions are equivalent.) If $X \sim \mathcal{N}(\mu, \Sigma)$, then

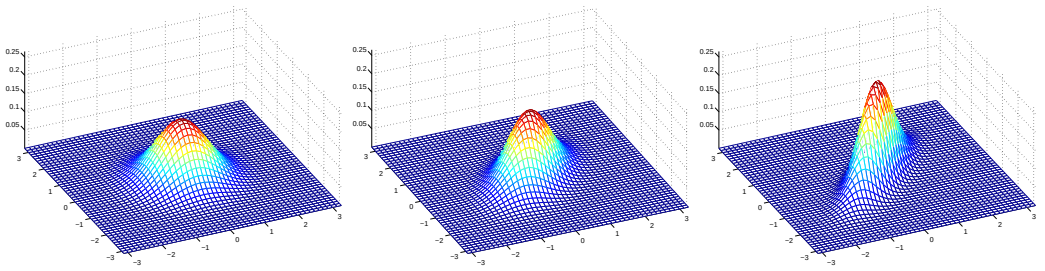
$$\mathrm{Cov}(X) = \Sigma.$$

Here're some examples of what the density of a Gaussian distribution look like:



The left-most figure shows a Gaussian with mean zero (that is, the 2x1 zero-vector) and covariance matrix $\Sigma = I$ (the 2x2 identity matrix). A Gaussian with zero mean and identity covariance is also called the **standard normal distribution**. The middle figure shows the density of a Gaussian with zero mean and $\Sigma = 0.6I$; and in the rightmost figure shows one with , $\Sigma = 2I$. We see that as $\Sigma$ becomes larger, the Gaussian becomes more "spread-out," and as it becomes smaller, the distribution becomes more "compressed."

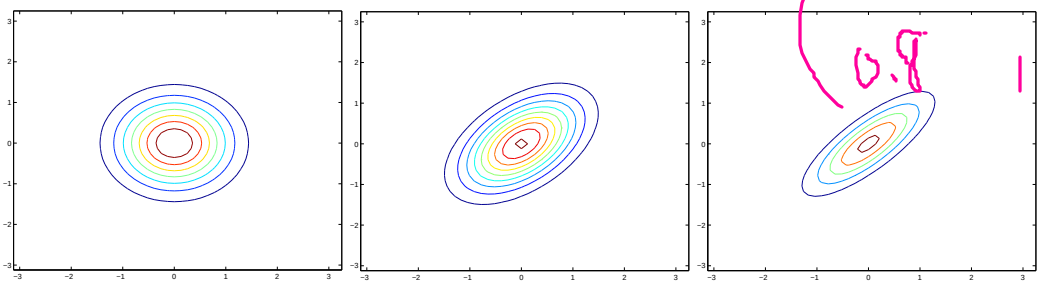
Lets look at some more examples.



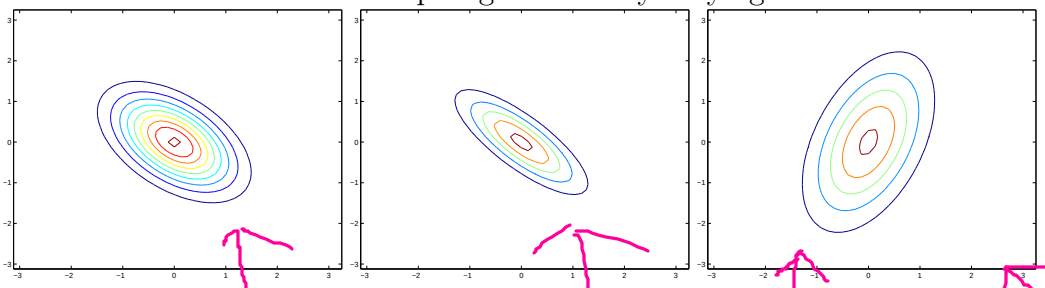
The figures above show Gaussians with mean 0, and with covariance matrices respectively

$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}; \quad .\Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}.$$

The leftmost figure shows the familiar standard normal distribution, and we see that as we increase the off-diagonal entry in $\Sigma$, the density becomes more "compressed" towards the 45° line (given by $x_1 = x_2$). We can see this more clearly when we look at the contours of the same three densities:



Here's one last set of examples generated by varying $\Sigma$:

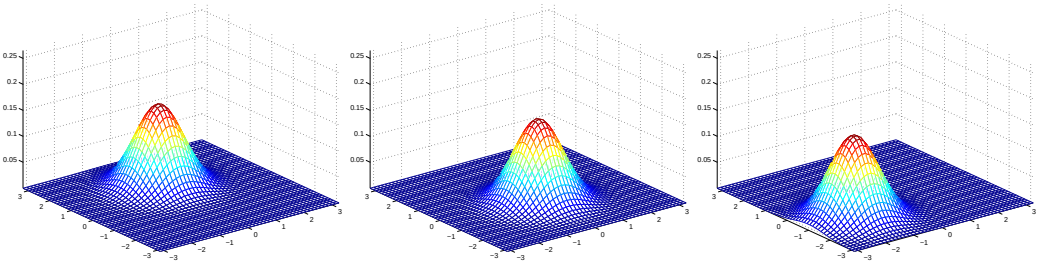


The plots above used, respectively,

$$\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}; \quad .\Sigma = \begin{bmatrix} 3 & 0.8 \\ 0.8 & 1 \end{bmatrix}.$$

From the leftmost and middle figures, we see that by decreasing the diagonal elements of the covariance matrix, the density now becomes "compressed" again, but in the opposite direction. Lastly, as we vary the parameters, more generally the contours will form ellipses (the rightmost figure showing an example).

As our last set of examples, fixing $\Sigma = I$, by varying $\mu$, we can also move the mean of the density around.



The figures above were generated using $\Sigma = I$, and respectively

$$\mu = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \quad \mu = \begin{bmatrix} -0.5 \\ 0 \end{bmatrix}; \quad \mu = \begin{bmatrix} -1 \\ -1.5 \end{bmatrix}.$$

高斯判别分析模型 特征 x 是连续的

## 1.2 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. The model is:

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

Writing out the distributions, this is:

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

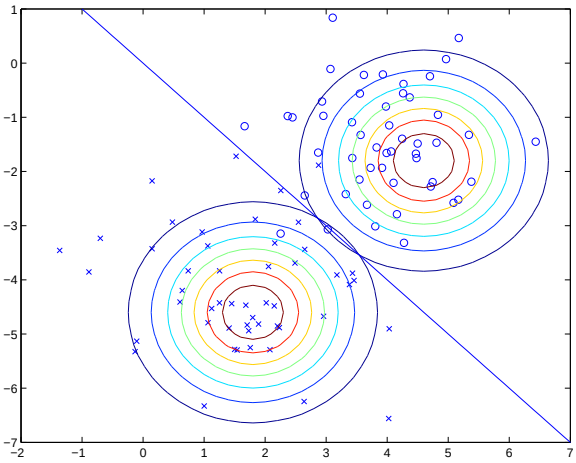
Here, the parameters of our model are $\phi$, $\Sigma$, $\mu_0$ and $\mu_1$. (Note that while there're two different mean vectors $\mu_0$ and $\mu_1$, this model is usually applied using only one covariance matrix $\Sigma$.) The log-likelihood of the data is given by

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

By maximizing $\ell$ with respect to the parameters, we find the maximum likelihood estimate of the parameters (see problem set 1) to be:

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

Pictorially, what the algorithm is doing can be seen in as follows:




Shown in the figure are the training set, as well as the contours of the two Gaussian distributions that have been fit to the data in each of the two classes. Note that the two Gaussians have contours that are the same shape and orientation, since they share a covariance matrix $\Sigma$, but they have different means $\mu_0$ and $\mu_1$. Also shown in the figure is the straight line giving the decision boundary at which $p(y = 1|x) = 0.5$. On one side of the boundary, we'll predict $y = 1$ to be the most likely outcome, and on the other side, we'll predict $y = 0$.

## 1.3 Discussion: GDA and logistic regression

The GDA model 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'll find that it

can be expressed in the form

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

where $\theta$ is some appropriate function of $\phi, \Sigma, \mu_0, \mu_1$.[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 logistic regression will, in general, give different decision boundaries when trained on the same dataset. Which is better?

We just argued that if $p(x|y)$ is multivariate gaussian (with shared $\Sigma$), then $p(y|x)$ necessarily follows a logistic function. The converse, however, is not true; i.e., $p(y|x)$ 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. 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 $\Sigma$), then GDA is **asymptotically efficient**. Informally, this means that in the limit of very large training sets (large $m$), there is no algorithm that is strictly better than GDA (in terms of, say, 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.

GDA对数据分布做了更强的假设，认为数据分布是符合高斯分布的，但逻辑回归没有做这样的假设，所以逻辑回归的鲁棒性会更好，但如果数据分布真的服从高斯分的话，那么GDA的效果可能会更好

In contrast, by making significantly weaker assumptions, logistic regression is also 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. For example, if $x|y = 0 \sim \text{Poisson}(\lambda_0)$, and $x|y = 1 \sim \text{Poisson}(\lambda_1)$, 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.

To summarize: 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 approximately correct. Logistic regression makes weaker assumptions, and is significantly more robust to deviations from modeling assumptions. Specifically, when the data is indeed non-Gaussian, then in the limit of large datasets, logistic regression will

[1] This uses the convention of redefining the $x^{(i)}$'s on the right-hand-side to be $n + 1$-dimensional vectors by adding the extra coordinate $x_0^{(i)} = 1$; see problem set 1.

almost always do better than GDA. For this reason, in practice logistic regression is used more often than GDA. (Some related considerations about discriminative vs. generative models also apply for the Naive Bayes algorithm that we discuss next, but the Naive Bayes algorithm is still considered a very good, and is certainly also a very popular, classification algorithm.)

## 2 Naive Bayes 朴素贝叶斯

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

For our motivating example, consider building an email spam filter using machine learning. 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**.

Lets say we have a training set (a set of emails labeled as spam or non-spam). We'll begin our construction of our spam filter by specifying the features $x_i$ 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. Specifically, if an email contains the $i$-th word of the dictionary, then we will set $x_i = 1$; otherwise, we let $x_i = 0$. For instance, the vector

$$x = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \text{a} \\ \text{aardvark} \\ \text{aardwolf} \\ \vdots \\ \text{buy} \\ \vdots \\ \text{zygmurgy} \end{matrix}$$

is used to represent an email that contains the words "a" and "buy," but not "aardvark," "aardwolf" or "zygmurgy."[2] The set of words encoded into the

[2]Actually, rather than looking through an english dictionary for the list of all english words, in practice it is more common to look through our training set and encode in our feature vector only the words that occur at least once there. Apart from reducing the number of words modeled and hence reducing our computational and space requirements,

feature vector is called the **vocabulary**, so the dimension of $x$ is equal to the size of the vocabulary.

Having chosen our feature vector, we now want to build a discriminative model. So, we have to model $p(x|y)$. But if we have, say, a vocabulary of 50000 words, then $x \in \{0,1\}^{50000}$ ($x$ is a 50000-dimensional vector of 0's and 1's), 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. This is clearly too many parameters.

文章里有某个词，对应位置的值就为 1

x 一共有 2 ^ 50000 种情况，每种情况都有一个出现的概率，而所有这些概率的和得为 1，计算出所有这些情况的 p(x|y)，然后存起来，之后再给一个 x，那么肯定对应 2 ^ 50000 种情况的一种，我们取其 p(x|y) 即可。但这个计算量是根本不可能完成的。

To model $p(x|y)$, we will therefore make a very strong assumption. We will assume that the $x_i$'s are conditionally independent given $y$. This assumption is called the **Naive Bayes (NB) assumption**, and the resulting algorithm is called the **Naive Bayes classifier**. For instance, if $y = 1$ means spam email; "buy" is word 2087 and "price" is word 39831; then we are assuming that if I tell you $y = 1$ (that a particular piece of email is spam), then knowledge of $x_{2087}$ (knowledge of whether "buy" appears in the message) will have no effect on your beliefs about the value of $x_{39831}$ (whether "price" appears). More formally, this can be written $p(x_{2087}|y) = p(x_{2087}|y, x_{39831})$. (Note that this is *not* the same as saying that $x_{2087}$ and $x_{39831}$ are independent, which would have been written "$p(x_{2087}) = p(x_{2087}|x_{39831})$"; rather, we are only assuming that $x_{2087}$ and $x_{39831}$ are conditionally independent *given* $y$.)

We now have:

$$
\begin{aligned}
& p(x_1, \ldots, x_{50000}|y) \\
&= p(x_1|y)p(x_2|y,x_1)p(x_3|y,x_1,x_2)\cdots p(x_{50000}|y,x_1,\ldots,x_{49999}) \quad \text{链式法则} \\
&= p(x_1|y)p(x_2|y)p(x_3|y)\cdots p(x_{50000}|y) \quad \text{根据我们的假设，做了简化} \\
&= \prod_{i=1}^{n} p(x_i|y)
\end{aligned}
$$

The first equality simply follows from the usual properties of probabilities, and the second equality used the NB assumption. 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 $\phi_{i|y=1} = p(x_i = 1|y = 1)$, $\phi_{i|y=0} = p(x_i = 1|y = 0)$, and $\phi_y = p(y = 1)$. As usual, given a training set $\{(x^{(i)}, y^{(i)}); i =$

this also has the advantage of allowing us to model/include as a feature many words that may appear in your email (such as "cs229") but that you won't find in a dictionary. Sometimes (as in the homework), we also exclude the very high frequency words (which will be words like "the," "of," "and,"; these high frequency, "content free" words are called **stop words**) since they occur in so many documents and do little to indicate whether an email is spam or non-spam.

$1, \ldots, m\}$, we can write down the joint likelihood of the data:

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

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

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

这个估计的直观解释就是所有y=1的样本中有单词xj的邮件数量除以y=1样本个数

这个估计的直观解释就是所有y=0的样本中有单词xj的邮件数量除以y=0样本个数（注意公式中的符号是与符号）

这个估计得直观解释就是 y = 1 的样本数除上样本总数

In the equations above, the "$\wedge$" symbol means "and." The parameters have a very natural interpretation. For instance, $\phi_{j|y=1}$ is just the fraction of the spam ($y = 1$) emails in which word $j$ does appear.

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

$$\begin{aligned}
p(y = 1|x) &= \frac{p(x|y = 1)p(y = 1)}{p(x)} \\
&= \frac{\left(\prod_{i=1}^{n} p(x_i|y = 1)\right) p(y = 1)}{\left(\prod_{i=1}^{n} p(x_i|y = 1)\right) p(y = 1) + \left(\prod_{i=1}^{n} p(x_i|y = 0)\right) p(y = 0)},
\end{aligned}$$

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 $x_i$ are binary-valued, the generalization to where $x_i$ can take values in $\{1, 2, \ldots, k_i\}$ is straightforward. Here, we would simply model $p(x_i|y)$ as multinomial rather than as Bernoulli. Indeed, even if some original input attribute (say, the living area of a house, as in our earlier example) 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. For instance, if we use some feature $x_i$ to represent living area, we might discretize the continuous values as follows:

| Living area (sq. feet) | $< 400$ | 400-800 | 800-1200 | 1200-1600 | $>1600$ |
|---|---|---|---|---|---|
| $x_i$ | 1 | 2 | 3 | 4 | 5 |

Thus, for a house with living area 890 square feet, we would set the value of the corresponding feature $x_i$ to 3. We can then apply the Naive Bayes

algorithm, and model $p(x_i|y)$ with a multinomial distribution, as described previously. 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 拉普拉斯平滑

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. Lets briefly discuss a problem with the algorithm in its current form, and then talk about how we can fix it.

Consider spam/email classification, and lets suppose that, after completing CS229 and having done excellent work on the project, you decide around June 2003 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 35000th word in the dictionary, your Naive Bayes spam filter therefore had picked its maximum likelihood estimates of the parameters $\phi_{35000|y}$ to be

假设有一个新单词 nips，在字典中的位置是 35000，但是在邮件中从来没出现过

$$\begin{aligned}\phi_{35000|y=1} &= \frac{\sum_{i=1}^{m} 1\{x^{(i)}_{35000} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}} = 0 \\ \phi_{35000|y=0} &= \frac{\sum_{i=1}^{m} 1\{x^{(i)}_{35000} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}} = 0\end{aligned}$$

I.e., 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. Hence, when trying to decide if one of these messages containing "nips" is spam, it calculates the class posterior probabilities, and obtains

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

最终出现 0/0 的情况　　这个时候极大似然估计就不是一个合理的假设了

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

考虑用贝叶斯估计解决这个问题

,



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. Take the problem of estimating the mean of a multinomial random variable $z$ taking values in $\{1, \ldots, k\}$. We can parameterize our multinomial with $\phi_i = p(z = i)$. Given a set of $m$ independent observations $\{z^{(1)}, \ldots, z^{(m)}\}$, the maximum likelihood estimates are given by

$$\phi_j = \frac{\sum_{i=1}^{m} 1\{z^{(i)} = j\}}{m}.$$

As we saw previously, if we were to use these maximum likelihood estimates, then some of the $\phi_j$'s might end up as zero, which was a problem. To avoid this, we can use **Laplace smoothing**, which replaces the above estimate with

这其实是计算各个类别出现的概率

$$\phi_j = \frac{\sum_{i=1}^{m} 1\{z^{(i)} = j\} + 1}{m + k}.$$

Here, we've added 1 to the numerator, and $k$ to the denominator. Note that $\sum_{j=1}^{k} \phi_j = 1$ still holds (check this yourself!), which is a desirable property since the $\phi_j$'s are estimates for probabilities that we know must sum to 1. Also, $\phi_j \neq 0$ for all values of $j$, solving our problem of probabilities being estimated as zero. Under certain (arguably quite strong) conditions, it can be shown that the Laplace smoothing actually gives the optimal estimator of the $\phi_j$'s.

Returning to our Naive Bayes classifier, with Laplace smoothing, we therefore obtain the following estimates of the parameters:

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

类别1，各个单词出现的概率，假设0类和1类都至少出现1次，总数多了两次，各个类别分别多了1次

类别0，各个单词出现的概率

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

## 2.2 Event models for text classification

To close off our discussion of generative learning algorithms, lets talk about one more model that is specifically for text classification. While Naive Bayes

as we've presented it will work well for many classification problems, for text classification, there is a related model that does even better.

In the specific context of text classification, Naive Bayes as presented uses the what's called 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 $p(x_i = 1|y) = \phi_{i|y}$. Thus, the probability of a message was given by $p(y) \prod_{i=1}^{n} p(x_i|y)$.

Here's a different model, called the **multinomial event model**. To describe this model, we will use a different notation and set of features for representing emails. We let $x_i$ denote the identity of the $i$-th word in the email. Thus, $x_i$ is now an integer taking values in $\{1, \ldots, |V|\}$, where $|V|$ is the size of our vocabulary (dictionary). An email of $n$ words is now represented by a vector $(x_1, x_2, \ldots, x_n)$ of length $n$; note that $n$ can vary for different documents. For instance, if an email starts with "A NIPS ...," then $x_1 = 1$ ("a" is the first word in the dictionary), and $x_2 = 35000$ (if "nips" is the 35000th word in the dictionary).

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, and similarly for $x_3$, $x_4$, and so on, until all $n$ words of the email have been generated. Thus, the overall probability of a message is given by $p(y) \prod_{i=1}^{n} p(x_i|y)$. Note that this formula looks like the one we had earlier for the probability of a message under the multi-variate Bernoulli event model, but that the terms in the formula now mean very different things. In particular $x_i|y$ is now a multinomial, rather than a Bernoulli distribution.

The parameters for our new model are $\phi_y = p(y)$ as before, $\phi_{i|y=1} = p(x_j = i|y = 1)$ (for any $j$) and $\phi_{i|y=0} = p(x_j = i|y = 0)$. Note that we have assumed that $p(x_j|y)$ is the same for all values of $j$ (i.e., that the distribution according to which a word is generated does not depend on its position $j$ within the email).

If we are given a training set $\{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$ where $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_{n_i}^{(i)})$ (here, $n_i$ is the number of words in the $i$-training example),

the likelihood of the data is given by

$$\begin{aligned}\mathcal{L}(\phi, \phi_{i|y=0}, \phi_{i|y=1}) &= \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}) \\ &= \prod_{i=1}^{m} \left( \prod_{j=1}^{n_i} p(x_j^{(i)}|y; \phi_{i|y=0}, \phi_{i|y=1}) \right) p(y^{(i)}; \phi_y).\end{aligned}$$

Maximizing this yields the maximum likelihood estimates of the parameters:

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

If we were to apply Laplace smoothing (which needed in practice for good performance) when estimating $\phi_{k|y=0}$ and $\phi_{k|y=1}$, we add 1 to the numerators and $|V|$ to the denominators, and obtain:

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

分子实际上就是对训练集和中的所有垃圾邮件中词 k 出现的次数求和

分母就是对训练集中所有垃圾邮件的总长度求和

拉普拉斯平滑

While not necessarily the very best classification algorithm, the Naive Bayes classifier often works surprisingly well. It is often also a very good "first thing to try," given its simplicity and ease of implementation.

|V| 的具体值是字典的长度，也就是假设每个词在每个类别都至少出现一次。