Machine Learning
(机器学习)-Regression

Prof.Shuyuan Yang, Zhixi Feng

E-mail： syyang@xidianedu.cn

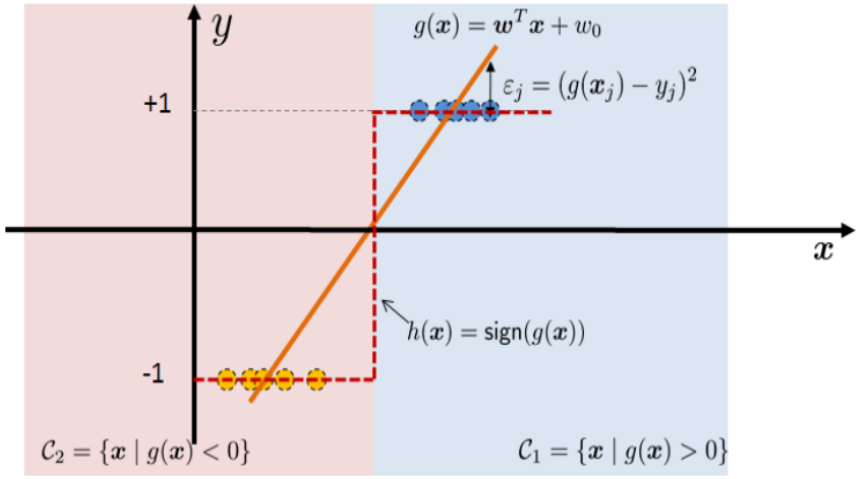
zxfeng@xidian.edu.cn

Contents

- From Linear to Logistic Regression
- Generative Learning
- Naïve Bayes Classifier

From Linear to Logistic Regression

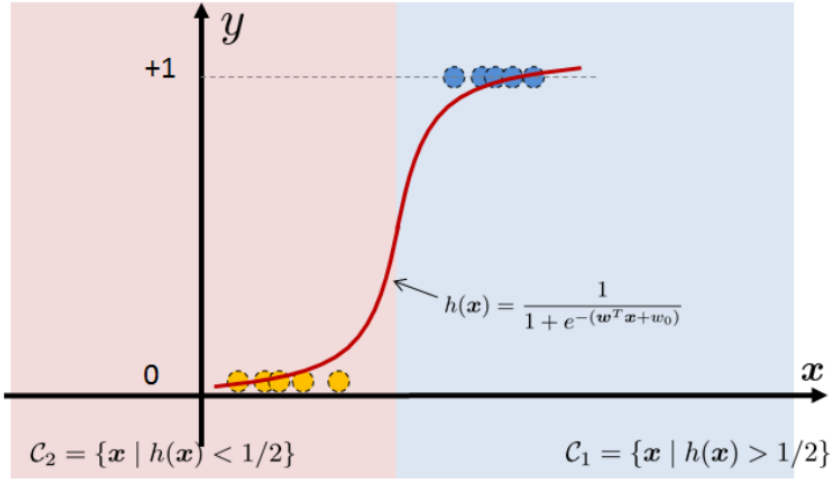
Geometry of Linear Regression



$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(g(\boldsymbol{x}_n), y_n).$$
$$= \sum_{n=1}^{N}(h_{\boldsymbol{\theta}}(\boldsymbol{x}_n) - y_n)^2$$

Geometry of Logistic Regression

soft-version



$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N}\mathcal{L}(h_\theta(\boldsymbol{x}_n), y_n)$$
$$= \sum_{n=1}^{N} -\Big\{y_n \log h_{\boldsymbol{\theta}}(\boldsymbol{x}_n) + (1-y_n)\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_n))\Big\}$$

Classification

- Model: the conditional distribution of y given x.

$$p(y|x;\theta)$$

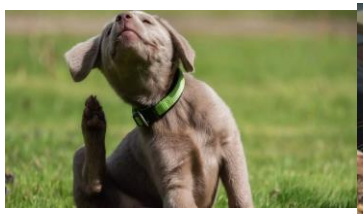
- Logistic regression:→ $p(y|x;\theta)$ $h_\theta(x) = g(\theta^t x)$

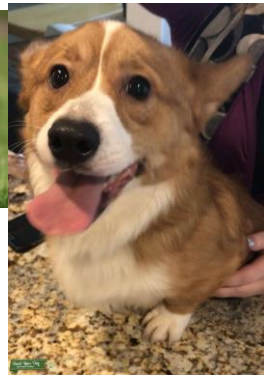
- Another way??

- 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.

Classification

- 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.

Generative Learning algorithms

- Discriminative learning algorithms.
  - learn $p(y|x)$ directly
  - learn mappings directly from the space of inputs x to the labels $\{\mathbf{0}, \mathbf{1}\}$

- Generative learning algorithms
  - Model：$p(x|y)$ and $p(y)$

$p(x|y=0)$ models the distribution of dogs‘ features,

$p(x|y=1)$ models the distribution of elephants' features.

Generative Learning algorithms

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

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

Gaussian discriminant analysis

- 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.

Multivariate normal distribution 多元正态分布

- 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 $N(\mu, \Sigma)$ its density is given by:

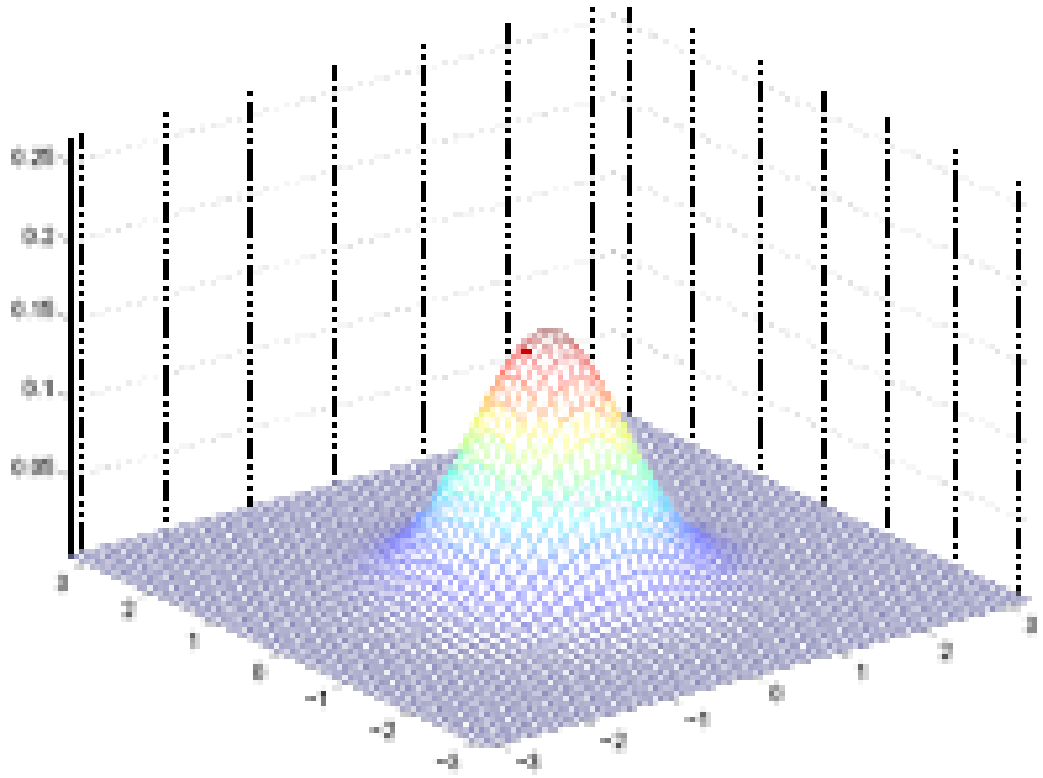
$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$

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

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

The covariance of a vector-valued random variable Z

Some examples of Gaussian distribution

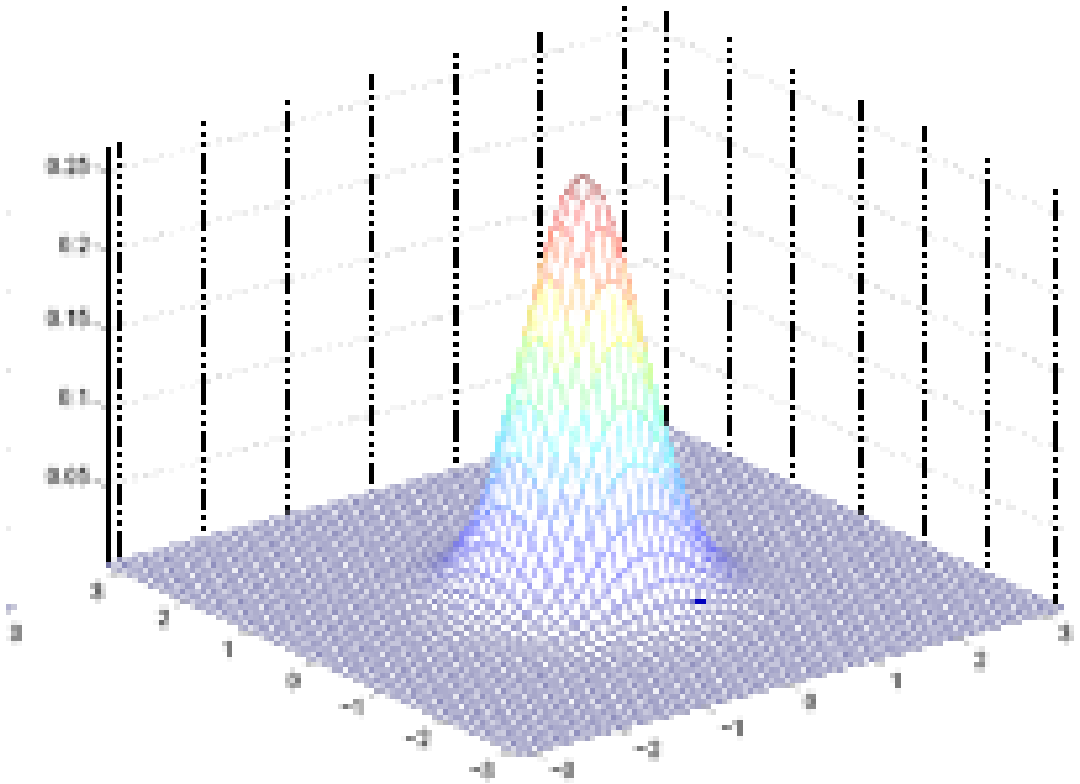


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

standard normal distribution

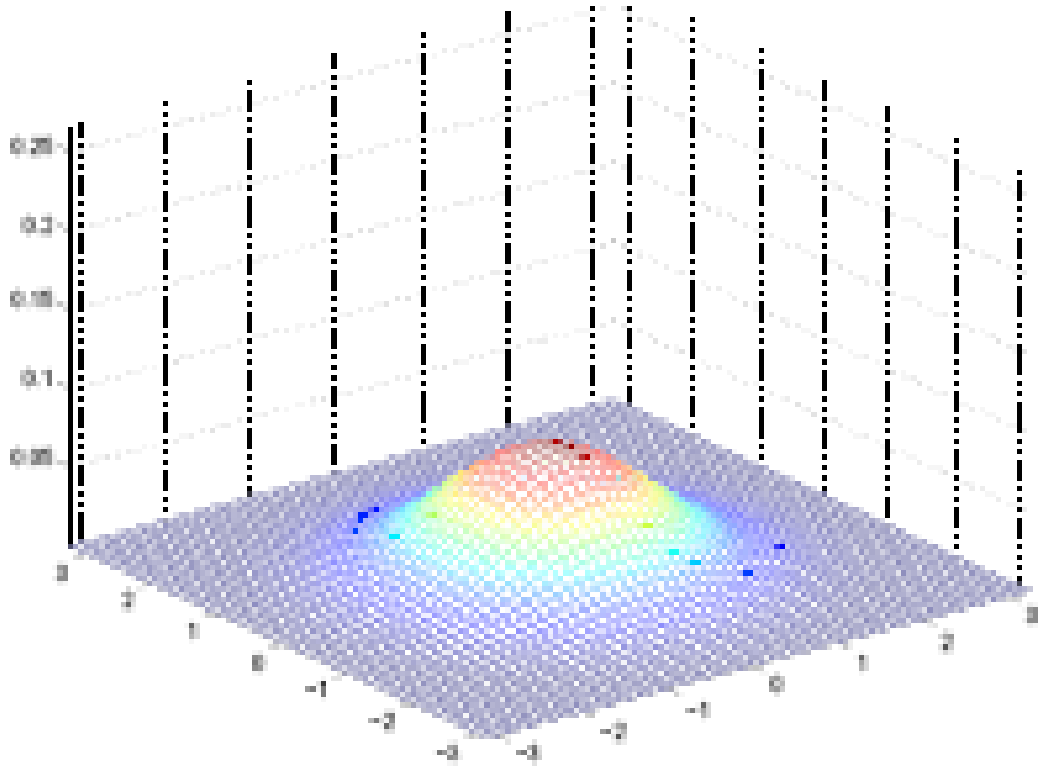
Some examples of Gaussian distribution



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$$

Some examples of Gaussian distribution

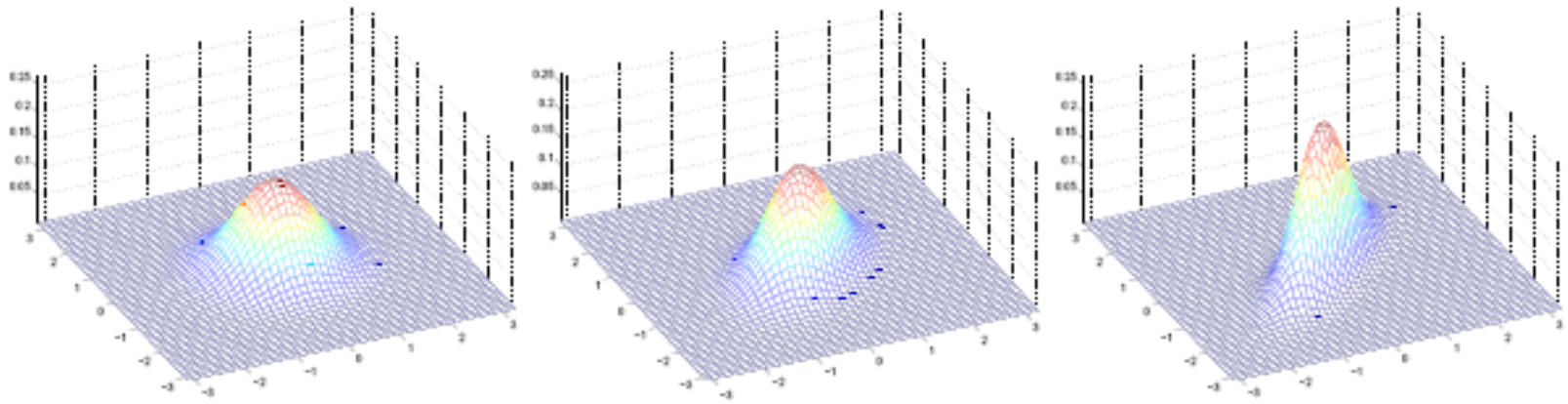


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

As Σ becomes larger, the Gaussian becomes more “spread-out,“ and as it becomes smaller, the distribution becomes more “compressed."

Some examples of Gaussian distribution

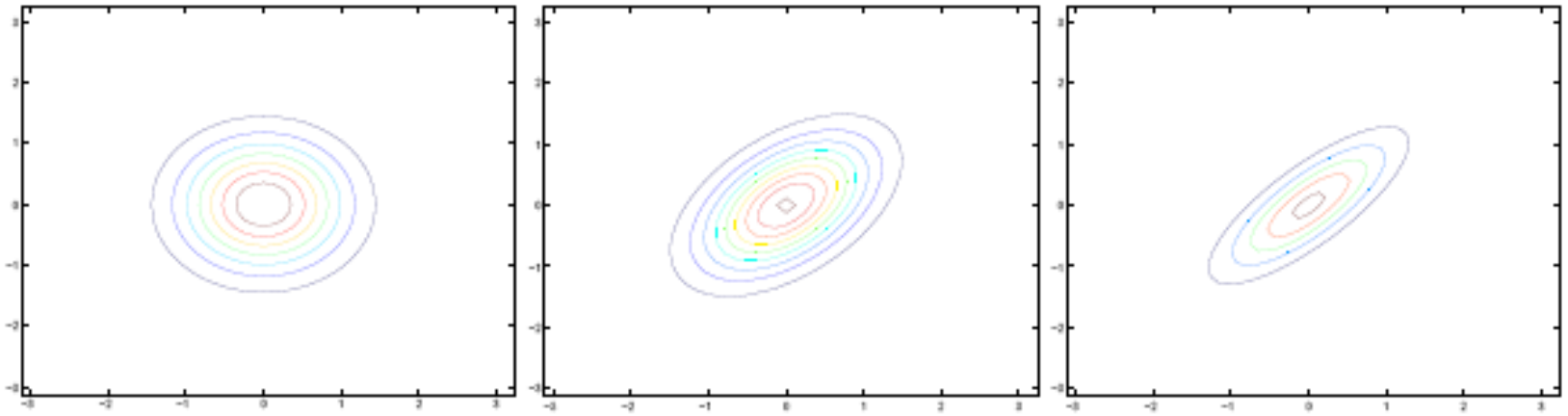


The figures above show Gaussians with mean 0, and with covariance matrices respectively

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

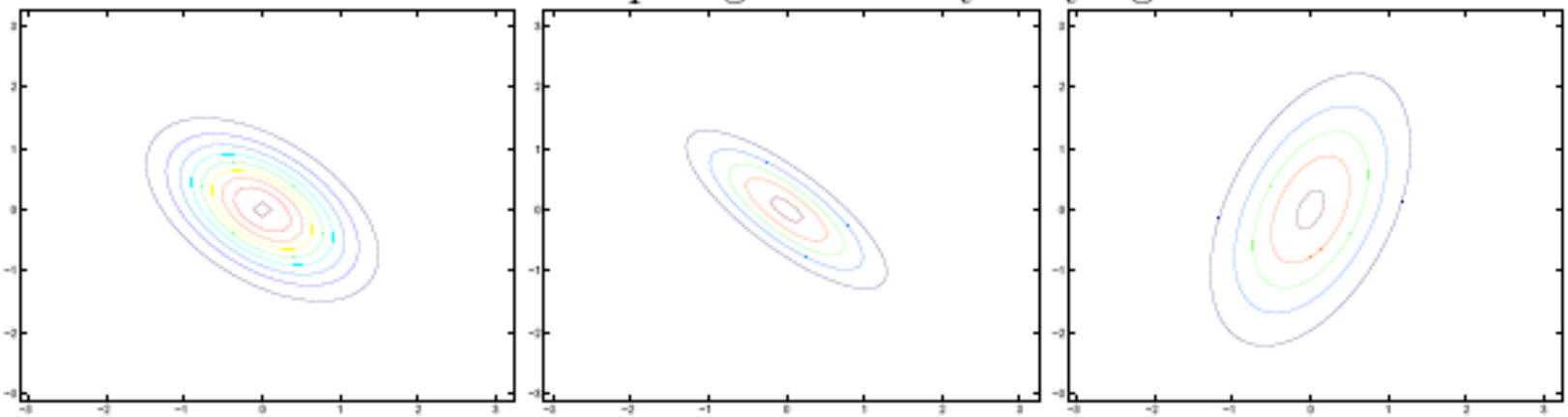
Some examples of Gaussian distribution

- We see that as we increase the o-diagonal entry in , the density becomes more “compressed" towards the 45 line (given by x1 = x2). We can see this more clearly when we look at the contours of the same three densities:



Some examples of Gaussian distribution

Here's one last set of examples generated by varying Σ:



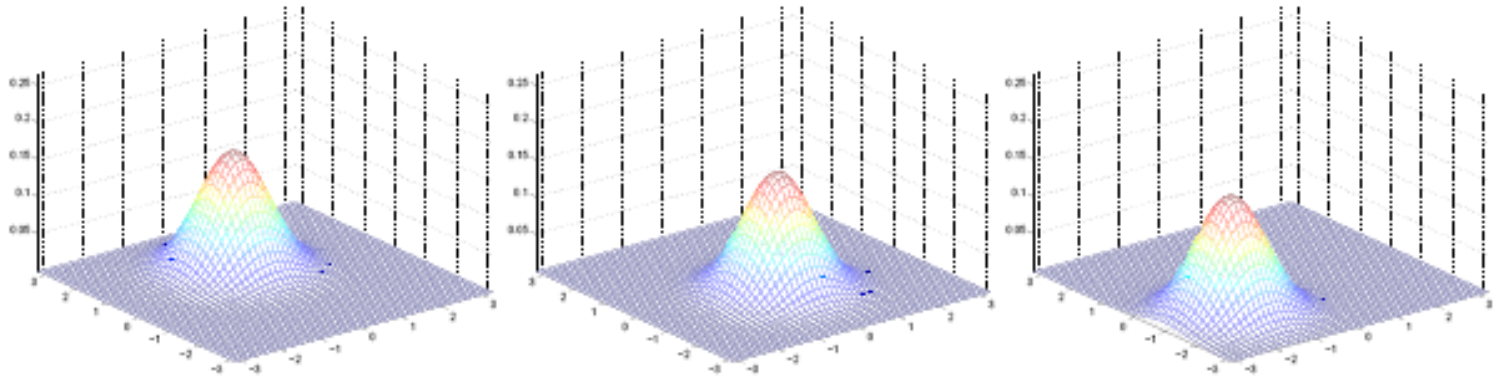
The plots above used, respectively,

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

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

Some examples of Gaussian distribution

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



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

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

Gaussian Discriminant Analysis model

- classification problem: continuous-valued random variables as input.
- use GDA model, which models $p(x|y)$ using a multivariate normal distribution. The model is:

$$y \sim Bernoulli(\phi)$$
$$x|y=0 \sim N(\mu_0, \Sigma)$$
$$x|y=1 \sim N(\mu_1, \Sigma)$$

- Writing out the distributions, this is:

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

Gaussian Discriminant Analysis model

- The log-likelihood of the data is given by

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

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

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

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

$$\mu_0 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 0\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}}$$

$$\mu_1 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}$$

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} \left(x^{(i)} - \mu_{y^{(i)}} \right) \left(x^{(i)} - \mu_{y^{(i)}} \right)^T$$

Gaussian Discriminant Analysis model

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

$$\phi = \frac{1}{m}\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}$$

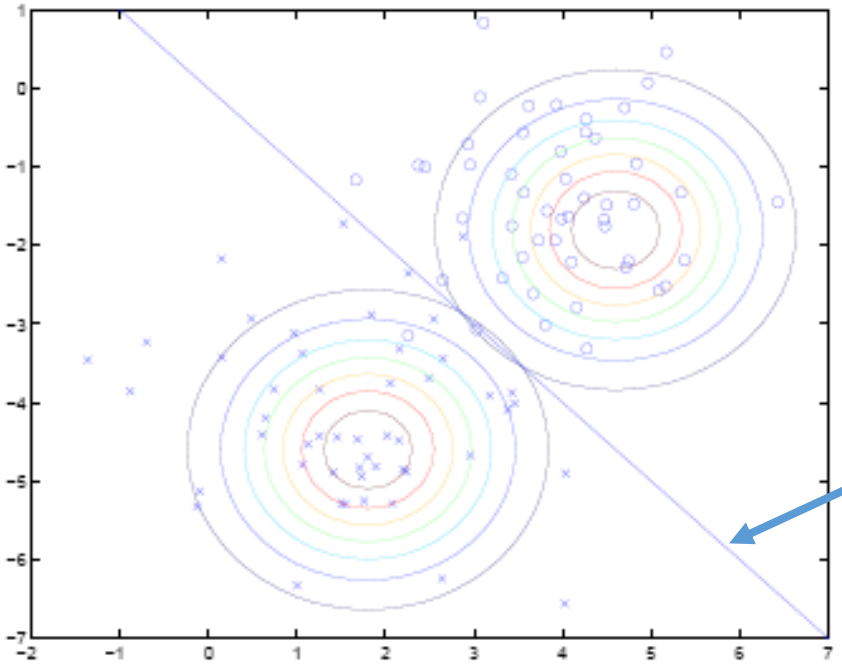
$$\mu_0 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}}$$

$$\mu_1 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \mu_{y^{(i)}}\right)\left(x^{(i)} - \mu_{y^{(i)}}\right)^T$$

Gaussian Discriminant Analysis model

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



- Note that the two Gaussians have contours that are the same shape and orientation

GDA VS. logistic regression

- The likelihood is $$p(\boldsymbol{x}|i) = \frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right\}$$
- The prior is $$p_Y(i) = \pi_i.$$
- The prior is $$\begin{aligned} p(1|\boldsymbol{x}) &= \frac{p(\boldsymbol{x}|1)p_Y(1)}{p(\boldsymbol{x}|1)p_Y(1)+p(\boldsymbol{x}|0)p_Y(0)} \\ &= \frac{1}{1+\frac{p(\boldsymbol{x}|0)p_Y(0)}{p(\boldsymbol{x}|1)p_Y(1)}} = \frac{1}{1+\exp\left\{-\log\left(\frac{p(\boldsymbol{x}|1)p_Y(1)}{p(\boldsymbol{x}|0)p_Y(0)}\right)\right\}} \\ &= \frac{1}{1+\exp\left\{-\log\left(\frac{\pi_1}{\pi_0}\right)-\log\left(\frac{p(\boldsymbol{x}|1)}{p(\boldsymbol{x}|0)}\right)\right\}}. \end{aligned}$$

GDA VS. logistic regression

- We can show that the last term is

$$
\begin{aligned}
&\log\left(\frac{p(\boldsymbol{x}|1)}{p(\boldsymbol{x}|0)}\right) \\
&= \log\left(\frac{\frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}}\exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1)\right\}}{\frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}}\exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_0)\right\}}\right) \\
&= -\frac{1}{2}\left[(\boldsymbol{x}-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1) - (\boldsymbol{x}-\boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_0)\right] \\
&= (\boldsymbol{\mu}_1-\boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}^{-1}\boldsymbol{x} - \frac{1}{2}\left(\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_0\right).
\end{aligned}
$$

- Let us define

$$
\begin{aligned}
\boldsymbol{w} &= \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_0) \\
w_0 &= -\frac{1}{2}\left(\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_0\right) + \log\left(\frac{\pi_1}{\pi_0}\right)
\end{aligned}
$$

GDA VS. logistic regression

- Then,

$$\log\left(\frac{p(\boldsymbol{x}|1)}{p(\boldsymbol{x}|0)}\right) = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^T \boldsymbol{\Sigma}^{-1}\boldsymbol{x} - \frac{1}{2}\left(\boldsymbol{\mu}_1^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_0\right)$$
$$= \boldsymbol{w}^T\boldsymbol{x} + w_0 - \log \pi_1/\pi_0$$

- Therefore,

$$p(1|\boldsymbol{x}) = \frac{1}{1 + \exp\left\{-\log\left(\frac{\pi_1}{\pi_0}\right) - \log\left(\frac{p(\boldsymbol{x}|1)}{p(\boldsymbol{x}|0)}\right)\right\}}$$
$$= \frac{1}{1 + \exp\{-(\boldsymbol{w}^T\boldsymbol{x} + w_0)\}}$$
$$= h_{\boldsymbol{\theta}}(\boldsymbol{x})$$

$\phi, \Sigma, \mu_0, \mu_1$

GDA VS. logistic regression

- The hypothesis function is the posterior distribution

$$p_{Y|\boldsymbol{x}}(1|\boldsymbol{x}) = \frac{1}{1+\exp\{-(\boldsymbol{w}^T\boldsymbol{x}+w_0)\}} = h_{\boldsymbol{\theta}}(\boldsymbol{x})$$

$$p_{Y|\boldsymbol{x}}(0|\boldsymbol{x}) = \frac{\exp\{-(\boldsymbol{w}^T\boldsymbol{x}+w_0)}{1+\exp\{-(\boldsymbol{w}^T\boldsymbol{x}+w_0)\}} = 1-h_{\boldsymbol{\theta}}(\boldsymbol{x}),$$

- So logistic regression offers probabilistic reasoning which linear regression does not.

- Not true when the covariances are different

GDA VS. logistic regression

? ?? 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?

$p(x|y)$

is multivariate gaussian
(with shared Σ)

$\rightleftharpoons$

$p(y|x)$

follows a logistic function

This shows **that GDA makes stronger modeling assumptions** about the data than does logistic regression.

GDA VS. logistic regression

- 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.

$$x|y=0 \sim \text{Poisson}(\lambda_0)$$
$$x|y=1 \sim \text{Poisson}(\lambda_1)$$

- 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.

GDA VS. logistic regression

- 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 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.)

Discussions:

- In GDA, the feature vectors x were continuous, real-valued vectors. Lets now talk about a different learning algorithm in which the input x 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.

Discussions:

- 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 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, $x_i = 0$ One-hot 编码

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

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

Discussions:

Having chosen our feature vector, we now want to build a discriminative model.

Model: $p(x|y)$

But if we have 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.

Naive Bayes

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

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

if y = 1 means spam email; “buy” is word 2087 and “price” is word 39831;

$$p(x_{2087} \mid y) = p(x_{2087} \mid y, x_{39831})$$

Note that this is not the same as saying that x_{2087} and x_{39831} are independent

$$p(x_{2087}) = p(x_{2087} \mid x_{39831})$$

Naive Bayes

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

The first equality simply follows from the usual properties of mprobabilities,

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.

Naive Bayes

- 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 $\{(x^{(i)}, y^{(i)}); i = 1, \cdots, m\}$
- we can write down the joint likelihood of the data:

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

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

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

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

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

Naive Bayes

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

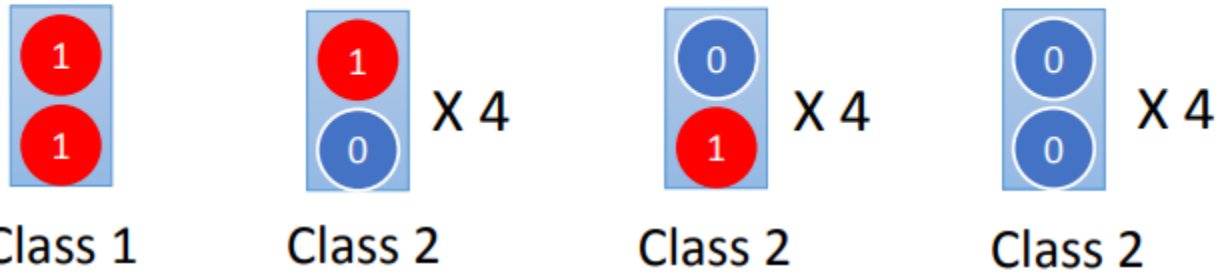
$$p(y=1|x) = \frac{p(x|y=1)\,p(y=1)}{p(x)}$$
$$= \frac{\left(\prod_{i=1}^{n} p(x_i|y=1)\right) p(y=1)}{\left(\prod_{i=1}^{n} p(x_i|y=1)\right) p(y=1) + \left(\prod_{i=1}^{n} p(x_i|y=0)\right) p(y=0)}$$

- and pick whichever class has the higher posterior probability

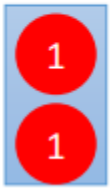
Generative v.s. Discriminative

- Example

Training Data



Testing Data



Class 1?
Class 2?

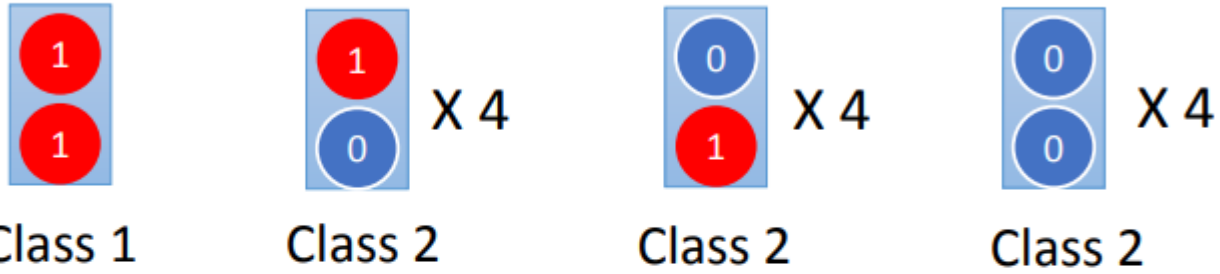
How about Naïve Bayes?

$$P(x|C_i) = P(x_1|C_i)P(x_2|C_i)$$

Generative v.s. Discriminative

- Example

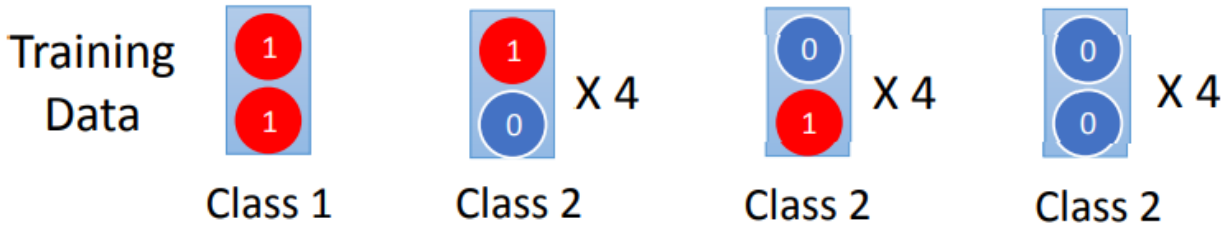
Training Data



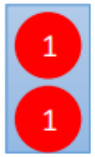
$$P(C_1) = \frac{1}{13} \qquad P(x_1 = 1|C_1) = 1 \qquad P(x_2 = 1|C_1) = 1$$

$$P(C_2) = \frac{12}{13} \qquad P(x_1 = 1|C_2) = \frac{1}{3} \qquad P(x_2 = 1|C_2) = \frac{1}{3}$$

Generative v.s. Discriminative



Testing Data



<0.5

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

$P(x|C_1)$: 1×1 — $P(C_1)$: $\frac{1}{13}$

$P(x|C_2)$: 1×1 — $P(C_2)$: $\frac{1}{13}$

$P(x|C_2)$: $\frac{1}{3} \times \frac{1}{3}$ — $P(C_2)$: $\frac{12}{13}$

$$P(C_1) = \frac{1}{13} \quad P(x_1 = 1|C_1) = 1 \quad P(x_2 = 1|C_1) = 1$$

$$P(C_2) = \frac{12}{13} \quad P(x_1 = 1|C_2) = \frac{1}{3} \quad P(x_2 = 1|C_2) = \frac{1}{3}$$

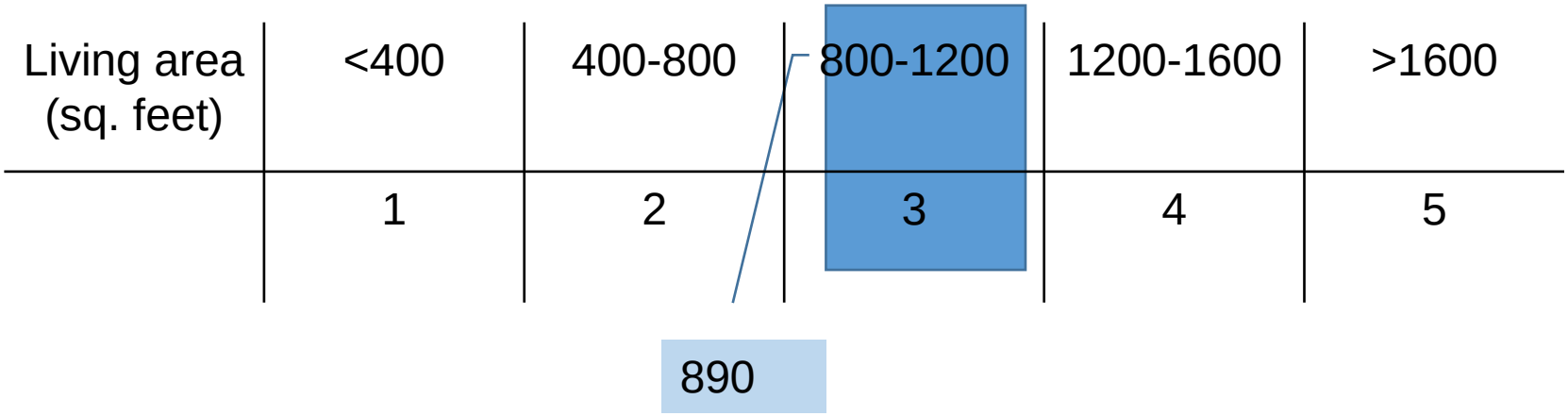
Naive Bayes

$$x_i \begin{cases} \{0,1\} \\ \{1,2,\cdots,k_i\} \end{cases}$$

- 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 to represent living area, we might discretize the continuous values as follows:

Naive Bayes



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 classier.

Laplace smoothing

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

- Example：
- NerIPS conference.

Assuming that “NerIPS” was the 35000th word in the dictionary

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

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

Laplace smoothing

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

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

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

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

$p(x_{35000}|y) = 0$

how to make a prediction?

Laplace smoothing

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

- Laplace smoothing:

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

Note that

1. $\sum_{j=1}^{k} \phi_j = 1$ still holds
2. $\phi_j \neq 0$ for all values of j

Under certain conditions, it can be shown that the Laplace smoothing actually gives the optimal estimator of the ϕ_j's.

Laplace smoothing

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

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

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