



## CMSC726 Spring 2006: Linear Classifiers

readings: available from class page

sources: course slides are based on material from a variety of sources, including **Tom Dietterich**, **Carlos Guestrin**, **Terran Lane**, Rich Maclin, Ray Mooney, Andrew Moore, **Andrew Ng**, Jude Shavlik, and others.



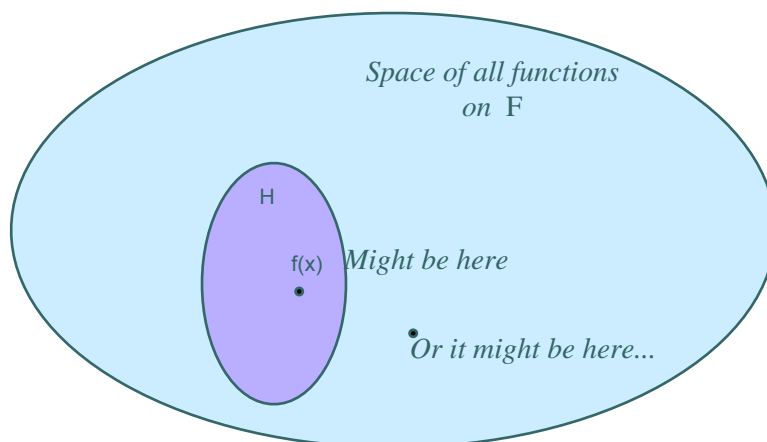
### Roadmap

- Today we'll look at learning algorithms (Naïve Bayes, Logistic Regression) for the case in which the  $x_j$  in the feature vectors are discrete
- Next time, we'll look at learning algorithms (Linear Regression, Gaussian Naïve Bayes/Linear Discriminant Analysis, Logistic Regression) for the case where the feature vectors,  $\mathbf{x}$ , are continuous, real-valued vectors.
- Along the way, we'll discuss important topics such as:
  - optimal classifiers
  - generative vs. discriminative classifiers
  - evaluating classifiers
  - overfitting
  - bias/variance trade-off

## ● ● ● | Some terminology

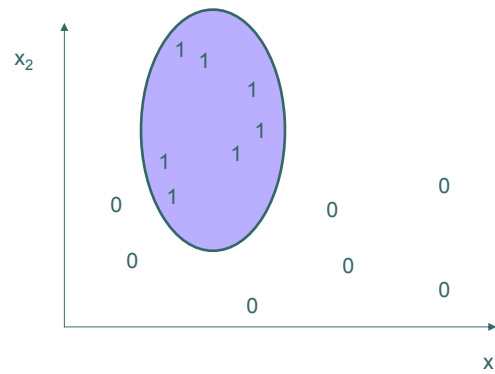
- Features (attributes, independent variables) can come in different flavors:
  - Continuous
  - Discrete
  - Categorical or nominal
- Typically assume the set of features is fixed & of finite dimension  $n$
- The set of all possible instances is the instance space or feature space  $F$
- Training set: set of instances  $X$  with class label  $Y$
- Unbiased Hypothesis Space – set of all possible functions on  $F$
- $f(X)$  true concept, target function
- The hypothesis space  $H$ , set of possible outputs of concept learner

## ● ● ● | Visually...



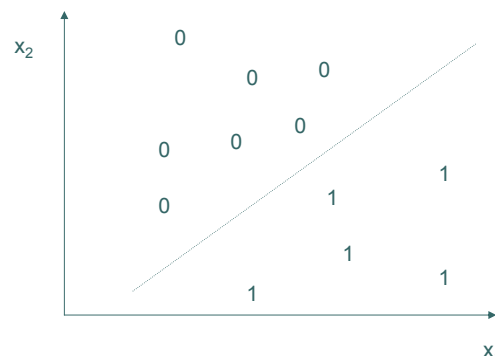
## ● ● ● | Decision Boundaries

- A classifier can be viewed as partitioning the input space or feature space  $X$  into decision regions



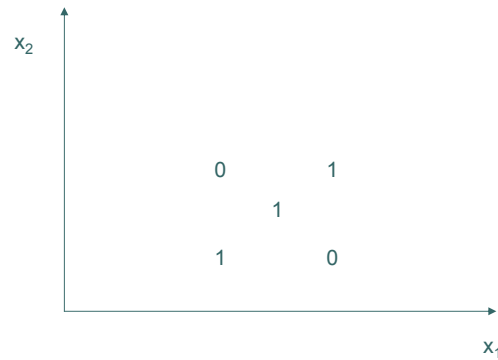
## ● ● ● | Linear Decision Boundaries

- A classifier can be viewed as partitioning the input space or feature space  $X$  into decision regions



- A set of points that can be separated by a linear decision boundary is **linearly separable**.
- Many **different** common models result in linear decision boundaries.

## ● ● ● | Non-linearly separable example



## ● ● ● | Aside

- While initially linear methods may seem overly simple:
  - Many non-linear methods are direct generalizations of linear methods
  - A variety of techniques, such as change of variables, or augmenting our attribute set to include quadratic and cross-product terms (a basis transformation), can be applied such that linear methods will be applicable

## ● ● ● | Supervised Learning = Optimization

- Given:
  - **Dataset:** Instances  $\{\langle \mathbf{x}^1; y^1 \rangle, \dots, \langle \mathbf{x}^m; y^m \rangle\}$ 
    - e.g.,  $\langle \mathbf{x}; y \rangle = \langle (\text{GPA}=3.9, \text{IQ}=120, \text{MLscore}=99); 150\text{K} \rangle$
  - **Hypothesis space:**  $H$ 
    - e.g., polynomials of degree 8
  - **Loss function:** measures quality of hypothesis  $h \in H$ 
    - e.g., 0/1 error for classification, squared error for regression
- Obtain:
  - **Learning algorithm:** obtain  $h \in H$  that minimizes loss function
    - e.g., using closed form solution, gradient descent, etc.
    - Want to minimize prediction error, but can only minimize error in dataset

## ● ● ● | Types of supervised learning problems

- **Regression**, e.g.,
  - **dataset:**  $\langle \text{position}; \text{temperature} \rangle$
  - **hypothesis space:**
  - **Loss function:**
- **Density estimation**, e.g.,
  - **dataset:**  $\langle \text{tack flips} \rangle$
  - **hypothesis space:**
  - **Loss function:**
- **Classification**, e.g.,
  - **dataset:**  $\langle \text{smiley face features} \rangle$
  - **hypothesis space:**
  - **Loss function:**

## ● ● ● | Learning as function approximation

- The general (supervised) learning problem:
  - Given some data (including features), hypothesis space, loss function
  - Learning is not magic
  - Simply trying to find a function that fits the data
- **Regression**      $h: X \rightarrow \mathbb{R}$
- **Density estimation**      $h: X \rightarrow [0..1]$
- **Classification**      $h: X \rightarrow \{y_1, y_2, \dots, y_k\}$
- (Not surprisingly) Seemly different problem, very similar solutions...

## ● ● ● | Classification

- **Learn:**  $h: X \mapsto Y$ 
    - $X$  – features
    - $Y$  – target classes
  - Suppose you know  $P(Y|X)$  exactly, how should you classify?
    - Bayes classifier:
- $$y^* = h_{\text{bayes}}(x) = \arg \max_y P(Y = y | X = x)$$
- **Why?**

## ● ● ● | Optimal classification

- **Theorem:** Bayes classifier  $h_{\text{Bayes}}$  is optimal
- That is  $error_{true}(h_{\text{Bayes}}) \leq error_{true}(h), \forall h(\mathbf{x})$

## ● ● ● | Generative vs. Discriminative Classifiers - Intuition

- Want to Learn:  $h: \mathbf{X} \rightarrow Y$ 
  - $\mathbf{X}$  – features
  - $Y$  – target class
- **Bayes optimal classifier** –  $P(Y|\mathbf{X})$
- **Generative classifier**, e.g., Naïve Bayes:
  - Assume some **functional form for  $P(\mathbf{X}|Y), P(Y)$**
  - Estimate parameters of  $P(\mathbf{X}|Y), P(Y)$  directly from training data
  - Use Bayes rule to calculate  $P(Y|\mathbf{X}=\mathbf{x})$
  - This is 'generative' model
    - Indirect computation of  $P(Y|\mathbf{X})$  through Bayes rule
    - But, can generate a sample of the data,  $P(\mathbf{X}) = \sum_y P(y)P(\mathbf{X}|y)$
- **Discriminative classifier**, e.g., Logistic Regression:
  - Assume some **functional form for  $P(Y|\mathbf{X})$**
  - Estimate parameters of  $P(Y|\mathbf{X})$  directly from training data
  - This is the 'discriminative' model
    - Directly learn  $P(Y|\mathbf{X})$
    - But cannot sample data, because  $P(\mathbf{X})$  is not available

## ● ● ● | Generative Models via **Bayes Rule**

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

Which is shorthand for:

$$(\forall i, j) P(Y = y_i | X = x_j) = \frac{P(X = x_j | Y = y_i) P(Y = y_i)}{P(X = x_j)}$$

## ● ● ● | How hard is it to learn the optimal classifier?

○ Data =

Sky	Temp	Humid	Wind	Water	Foreest	EnjoySpt
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

○ How do we represent these? How many parameters?

● Prior,  $P(Y)$ :

- Suppose  $Y$  is composed of  $k$  classes

● Likelihood,  $P(\mathbf{X}|Y)$ :

- Suppose  $\mathbf{X}$  is composed of  $n$  binary features

## ● ● ● | Conditional Independence

- X is **conditionally independent** of Y given Z, if the probability distribution governing X is independent of the value of Y, given the value of Z

$$(\forall i, j, k) P(X = i | Y = j, Z = k) = P(X = i | Z = k)$$

- e.g.,  $P(\text{Thunder} | \text{Rain}, \text{Lightning}) = P(\text{Thunder} | \text{Lightning})$

- Equivalent to:

$$P(X, Y | Z) = P(X | Z)P(Y | Z)$$

## ● ● ● | The Naïve Bayes assumption

- Naïve Bayes assumption:
  - Features are independent given class:

$$\begin{aligned} P(X_1, X_2 | Y) &= P(X_1 | X_2, Y)P(X_2 | Y) \\ &= P(X_1 | Y)P(X_2 | Y) \end{aligned}$$

- More generally:

$$P(X_1 \dots X_n | Y) = \prod_i P(X_i | Y)$$

- How many parameters now?
  - Suppose  $\mathbf{X}$  is composed of  $n$  binary features

## ● ● ● | The Naïve Bayes Classifier

- Given:
  - Prior  $P(Y)$
  - $n$  conditionally independent features  $\mathbf{X}$  given the class  $Y$
  - For each  $X_i$ , we have likelihood  $P(X_i|Y)$

- Decision rule:

$$\begin{aligned} y^* = h_{NB}(\mathbf{x}) &= \arg \max_y P(y) P(x_1, \dots, x_n | y) \\ &= \arg \max_y P(y) \prod_i P(x_i | y) \end{aligned}$$

- **If assumption holds, NB is optimal classifier!**

## ● ● ● | MLE for the parameters of NB

- Given dataset
  - $N(A=a, B=b) \leftarrow$  number of examples where  $A=a$  and  $B=b$
- MLE for NB, simply:
  - Prior:  $P(Y=y) =$
  
  - Likelihood:  $P(X_i=x_i|Y_i=y_i) =$

## ● ● ● | NB Subtleties – Violating the NB assumption

- Usually, features are not conditionally independent:

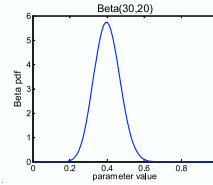
$$P(X_1 \dots X_n | Y) \neq \prod_i P(X_i | Y)$$

- Actual probabilities  $P(Y|\mathbf{X})$  often biased towards 0 or 1
- Nonetheless, NB is the single most used classifier out there
  - NB often performs well, even when assumption is violated
  - [Domingos & Pazzani '96] discuss some conditions for good performance

## ● ● ● | NB Subtleties 2 – Insufficient training data

- What if you never see a training instance where  $X_1=a$  when  $Y=b$ ?
  - e.g.,  $Y=\{\text{SpamEmail}\}$ ,  $X_1=\{\text{'Viagra'}\}$
  - $P(X_1=a | Y=b) = 0$
- Thus, no matter what the values  $X_2, \dots, X_n$  take:
  - $P(Y=b | X_1=a, X_2, \dots, X_n) = 0$
  
- What now???

## ● ● ● | MAP for Beta distribution



$$P(\theta | \mathcal{D}) = \frac{\theta^{\beta_H + \alpha_H - 1} (1 - \theta)^{\beta_T + \alpha_T - 1}}{B(\beta_H + \alpha_H, \beta_T + \alpha_T)} \sim \text{Beta}(\beta_H + \alpha_H, \beta_T + \alpha_T)$$

- MAP: use most likely parameter:

$$\hat{\theta} = \arg \max_{\theta} P(\theta | \mathcal{D}) =$$

- Beta prior equivalent to extra thumbtack flips
- As  $N \rightarrow \infty$ , prior is “forgotten”
- **But, for small sample size, prior is important!**

## ● ● ● | Bayesian learning for NB parameters – a.k.a. smoothing

- Dataset of  $N$  examples
- Prior
  - “distribution”  $Q(X_i, Y)$ ,  $Q(Y)$
  - $m$  “virtual” examples
- MAP estimate
  - $P(X_i | Y)$
- **Now, even if you never observe a feature/class, posterior probability never zero**

## ● ● ● | Example: Text Classification

- Application: Spam filtering
  - We want to classify messages according to whether they are junk mail (spam) or non-spam.
  - Once we learn the classifier, we can use it to filter out spam messages and put them in a separate folder
- Suppose we have a training set of email labeled as spam or not spam. What are the features  $x_i$ ?
- Approach #1: We represent the email as a feature vector over all the possible words in the dictionary. If an email contains the  $i$ -th word in the dictionary,  $x_i$  is 1, otherwise it is 0:

$$x = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \begin{array}{l} a \\ \text{aardvark} \\ \\ \text{viagra} \\ \\ \text{zylephone} \end{array}$$

## ● ● ● | Vocabulary

- $V$  = Set of words used in the feature vector
  - Use all the words in the dictionary
  - More common – use only words that occur at least once in the training set
  - May do other pruning/processing of the words
    - Remove stop words (a, the, of)
    - Stem words (computer, computing -> comput)
- The dimension of the feature vector  $x$  depends on the size of the vocabulary  $|V|$

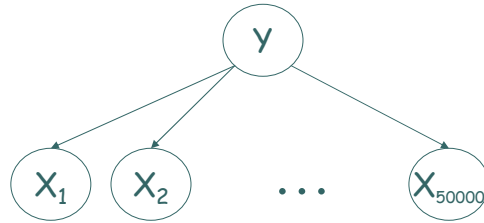
## ● ● ● | Generative Model for Text

- We want to model  $p(x|y)$
- If we have a vocabulary of 50000 words, then  $x \in \{0, 1\}^{50000}$
- If we model  $x$  as a multinomial over all  $2^{50000}$  words, we'd need  $2^{50000}-1$  parameters
- Instead, we'll make a very strong assumption about the model
  - we'll assume that the  $x_i$  are independent, given  $y$
  - $P(x_i|y, x_j) = P(x_i|y)$  or  $P(x_i, x_j|y) = P(x_i|y) P(x_j|y)$
  - E.g. let  $x_i$ =cheapest,  $y=1$  (spam),  $x_j$ =viagra
  - Clearly the above are not independent. But they may be conditionally independent (probably not though).
  - Regardless, the resulting model, called the **Naïve Bayes Model**, and the resulting classifier, the **Naïve Bayes classifier** often perform quite well

## ● ● ● | Naïve Bayes Assumption

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

## ● ● ● | Naïve Bayes Model



Parameters:

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

$$\phi_{x_i|\bar{y}} = p(x_i = 1 | y = 0)$$

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

## ● ● ● | Likelihood of the Data

$$L(\phi_y, \phi_{x_i|y}, \phi_{x_i|\bar{y}}) = \prod_{j=1}^N p(x^j, y^j)$$

Maximizing this wrt to the parameters gives MLEs:

$$\hat{\phi}_y = \frac{n_y}{N} \quad \hat{\phi}_{x_i|y} = \frac{n_{x_i y}}{n_y} \quad \hat{\phi}_{x_i|\bar{y}} = \frac{n_{x_i \bar{y}}}{n_{\bar{y}}}$$

$n_y$  is the number of documents with label  $y$   
 $n_{x_i y}$  is the number of documents with label  $y$  in which word  $i$  occurs

## ● ● ● | Using NB for Prediction

compare

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

with

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

and choose most probable

## ● ● ● | Modifications

- Straight forward to modify for features  $x_i$  that are non-binary
  - Suppose  $x_i \in \{1, \dots, k_i\}$
  - Model  $p(x_i|y)$  as multinomial rather than Bernoulli
  - MLE estimate

$$\hat{\phi}_{x_{ik}|y} = \frac{n_{x_{ik}y}}{n_y}$$

$n_{x_{ik}y}$  is the number of items with label  $y$  in which  $x_i=k$

## ● ● ● | Dealing with Continuous Attributes

- Common approach: discretize
- For example salary might be discretized as follows:

salary	<30K	30K-60K	60K-100K	100K-200K	>200K
$x_i$	0	1	2	3	4

- Issue: choosing discretization
- When the continuous-valued attributes are not modeled well by a multivariate normal, discretizing the variables and using NB can outperform a NB which models them directly (which we'll see shortly).

## ● ● ● | Laplace smoothing

- Often times NB works surprisingly well as is.
- However, for some domains, especially text classification, we have to be aware of the following problem:
  - If a word is never seen in the training set, what happens?

$$\hat{\phi}_{x_i|y} = \frac{n_{x_i y}}{n_y} = 0 \quad \hat{\phi}_{x_i|\bar{y}} = \frac{n_{x_i \bar{y}}}{n_{\bar{y}}} = 0$$

- and then

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

## ● ● ● | Zero Probabilities are Bad

- In general, statistically it is a bad idea to ever give anything 0 probability
- We can solve this for the multinomial case by changing our estimate from

$$\hat{\phi}_{x_{ik}|y} = \frac{n_{x_{ik}y}}{n_y}$$

- to

$$\hat{\phi}_{x_{ik}|y} = \frac{n_{x_{ik}y} + 1}{n_y + k}$$

- this is called **Laplace smoothing**
- for Naïve Bayes this gives

$$p(y | x) = \frac{\left( \prod_{i=1}^n p(x_i | y) \right) p(y) + 1}{\left( \prod_{i=1}^n p(x_i | y) \right) p(y) + \left( \prod_{i=1}^n p(x_i | \bar{y}) \right) p(\bar{y}) + 2}$$

## ● ● ● | Alternate Event Models for Text Classification

- The model just presented is called the **multi-variate Bernoulli event model**
- Consider the SPAM example, we assume the way that an email is generated is that first we determine (according to  $p(y)$ ) whether your next email will be spam or not-spam. Then, the person sending the email goes through the dictionary, and decides whether to include each word  $i$  independently with probability  $p(x_i | y)$ . Then the probability of a message is

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

- An alternate model is called the **multinomial event model**
- Here, the  $x_i$  is the identity of the  $i$ th word in the email,  $x_i \in \{1, \dots, |V|\}$
- An email is a vector  $(x_1, \dots, x_n)$ , where  $n$  can vary for different emails
- In this model, email is generated by a random process where spam/not-spam is determined as before, according to  $p(y)$ . Then, each word in the message is chosen according to  $p(x_i | y)$ , independently from the previous words.
- The overall probability is

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

## ● ● ● | Likelihood of the Data

$$L(\phi_y, \phi_{k|y}, \phi_{k|\bar{y}}) = \prod_{i=1}^N p(x^i, y^i)$$

$$L(\phi_y, \phi_{k|y}, \phi_{k|\bar{y}}) = \prod_{i=1}^N \left( \prod_{j=1}^{n_i} p(x_j^i | y^i; \phi_{k|y}, \phi_{k|\bar{y}}) \right) p(y^i; \phi_y)$$

Maximizing this wrt to the parameters gives MLEs:

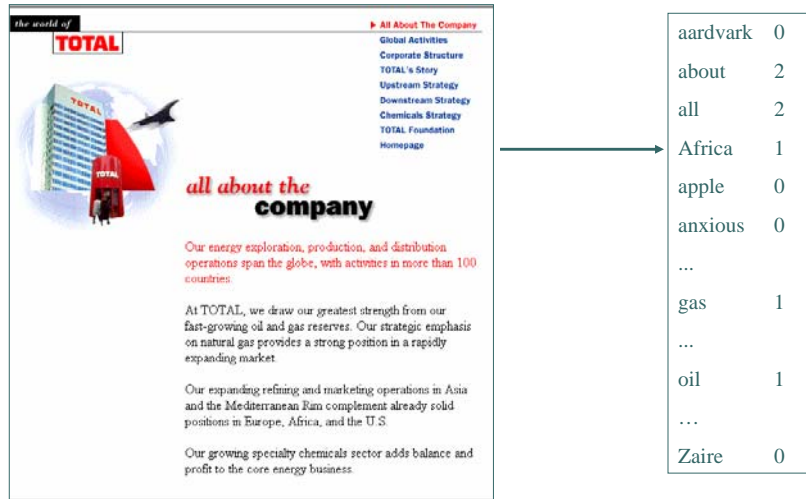
$$\hat{\phi}_y = \frac{n_y}{N} \quad \hat{\phi}_{k|y} = \frac{\sum_{y^i} n_{ky}^i}{\sum_{y^i} n^i} \quad \hat{\phi}_{k|\bar{y}} = \frac{\sum_{\bar{y}^i} n_{k\bar{y}}^i}{\sum_{\bar{y}^i} n^i}$$

$n_{ky}^i$  is the number of times word k is seen in documents with label y

## ● ● ● | Multi-variate Bernoulli vs Multinomial Event

- Multinomial Event model is also called 'bag-of-words' model, unigram model
- Multinomial Event captures word frequencies
- Multi-variate Bernoulli captures non-occurrence of words
- Multinomial Event tends to work best in text domains, where there is a large vocabulary
- Multi-variate Bernoulli works well in non-text domains. Common application: diagnosis
- See McCallum and Nigam, *A Comparison of Event Models for Naive Bayes Text Classification* (1998) for an additional details

## ● ● ● | Bag of Words Approach



## ● ● ● | NB w/ Bag of Words for text classification

### ○ Learning phase:

- Prior  $P(Y)$

- Count how many documents you have from each topic (+ prior)

- $P(X_i|Y)$

- For each topic, count how many times you saw word in documents of this topic (+ prior)

### ○ Test phase:

- For each document

- Use naïve Bayes decision rule

$$h_{NB}(x) = \arg \max_y P(y) \prod_{i=1}^{LengthDoc} P(x_i|y)$$

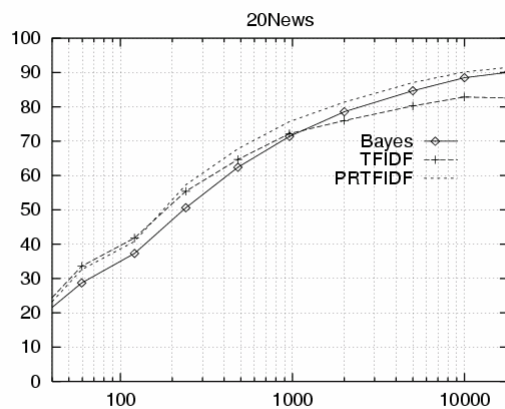
## ● ● ● | Twenty News Groups results

Given 1000 training documents from each group  
 Learn to classify new documents according to  
 which newsgroup it came from

comp.graphics	misc.forsale
comp.os.ms-windows.misc	rec.autos
comp.sys.ibm.pc.hardware	rec.motorcycles
comp.sys.mac.hardware	rec.sport.baseball
comp.windows.x	rec.sport.hockey
alt.atheism	sci.space
soc.religion.christian	sci.crypt
talk.religion.misc	sci.electronics
talk.politics.mideast	sci.med
talk.politics.misc	
talk.politics.guns	

Naive Bayes: 89% classification accuracy

## ● ● ● | Learning curve for Twenty News Groups



Accuracy vs. Training set size (1/3 withheld for test)

## ● ● ● | Generative Models for Continuous Features

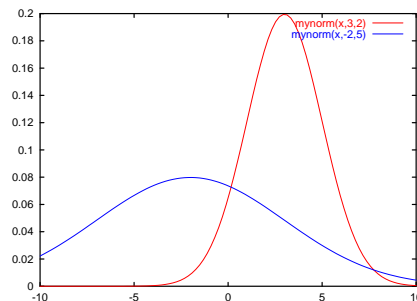
- X's are continuous, not 0/1 or 0,...,k-1
- Gaussian Discriminant Analysis (Ng notes)
  - In GDA, we learn the distribution  $P(\mathbf{x}|y)$
  - We assume  $P(\mathbf{x}|y)$  is distributed according to a multivariate normal distribution and  $P(y)$  is Bernoulli
  - other names: Linear Discriminant Analysis
    - Assumes all classes have the same  $\Sigma$
- Gaussian Naïve Bayes (Mitchell notes)
  - Assume features  $x_i$  are independent

## ● ● ● | Multivariate normal distribution

- also called multivariate Gaussian
- First, recall the univariate normal distribution:

$$p(x; \mu, \sigma) = \frac{1}{(2\pi)^{1/2} \sigma} \exp\left[-\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2}\right]$$

- where  $\mu$  is the mean and  $\sigma^2$  is the variance; written  $p(x) \sim \mathcal{N}(\mu, \sigma)$



## ● ● ● | Multivariate normal distribution

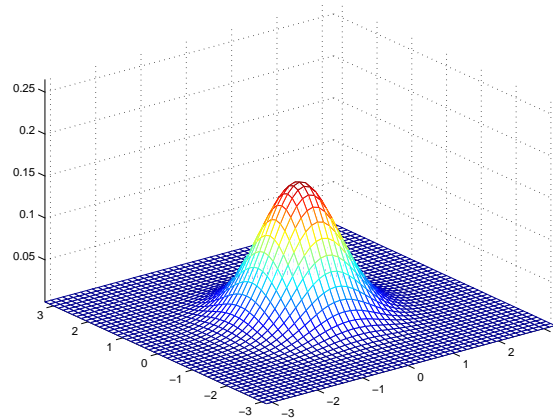
- A 2-dimensional gaussian is defined by a mean vector  $\mu = (\mu_1, \mu_2)$  and a covariance matrix:

$$\Sigma = \begin{bmatrix} \sigma_{1,1}^2 & \sigma_{1,2}^2 \\ \sigma_{2,1}^2 & \sigma_{2,2}^2 \end{bmatrix}$$

- where  $\sigma_{i,j}^2 = E[(x_i - \mu_i)(x_j - \mu_j)]$ 
  - is the variance if  $x_i = x_j$
  - covariance if  $x_i \neq x_j$

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

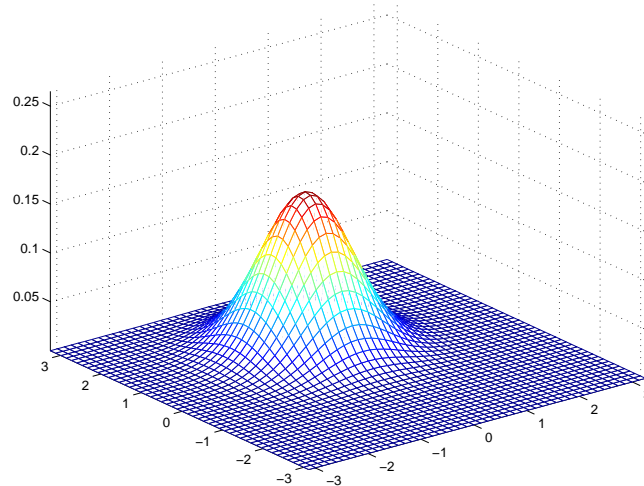
## ● ● ● | Standard normal distribution



- We get the standard normal for  $\Sigma =$  the identity matrix and  $\mu = (0,0)$

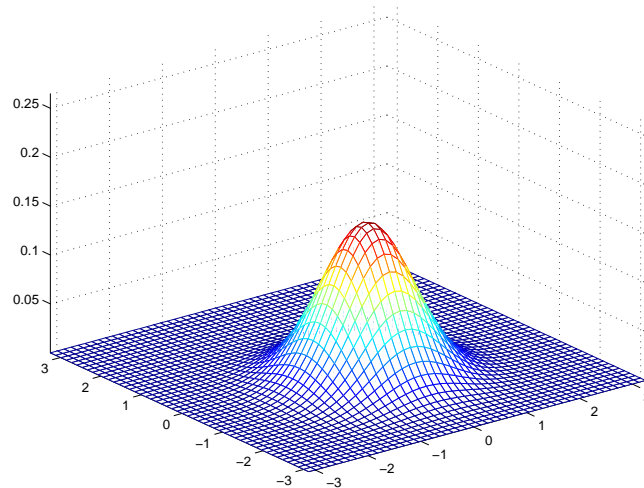
$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

● ● ● | MVG examples



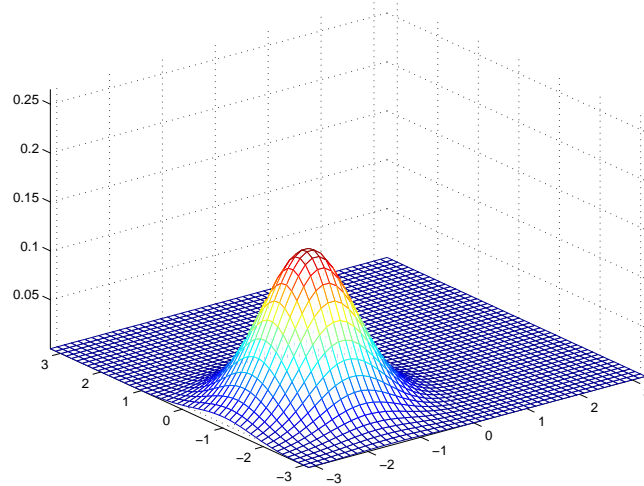
$$\mu = (1, 0) \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

● ● ● | MVG examples



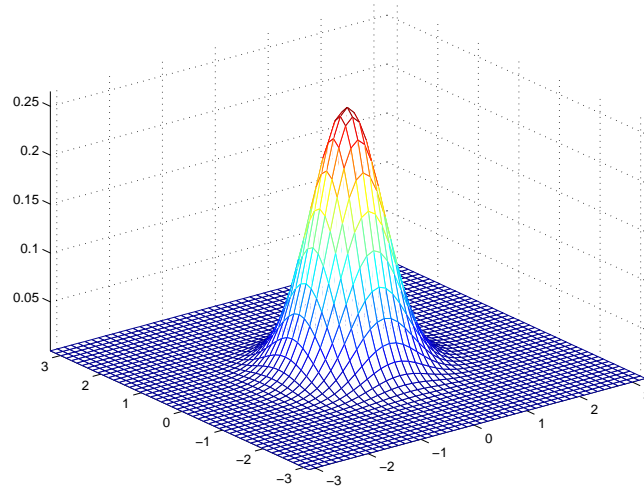
$$\mu = (-0.5, 0) \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

● ● ● | MVG examples



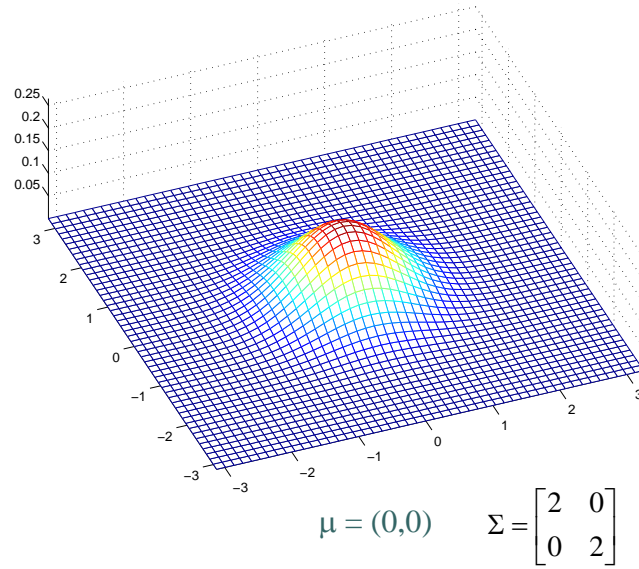
$$\mu = (-1, -1.5) \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

● ● ● | MVG examples

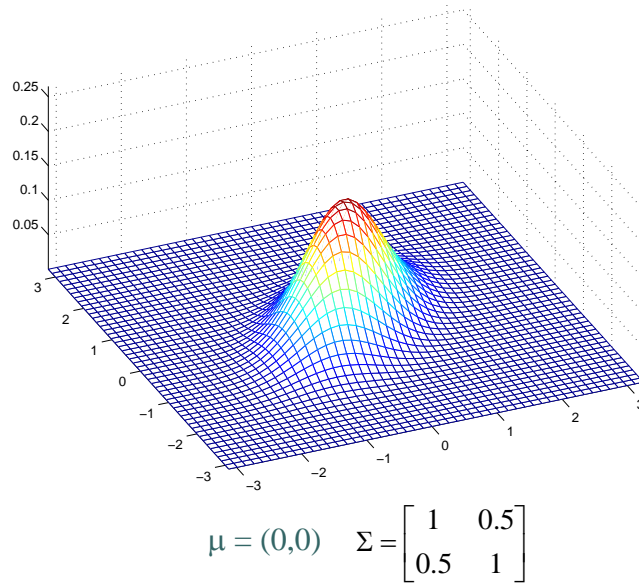


$$\mu = (0, 0) \quad \Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$$

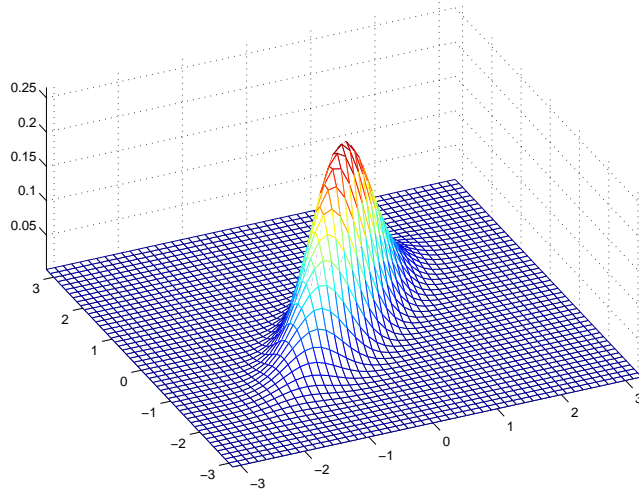
● ● ● | MVG examples



● ● ● | MVG examples

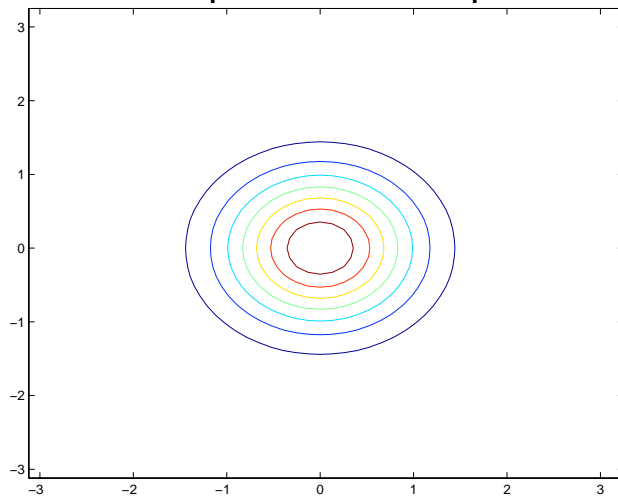


● ● ● | MVG examples

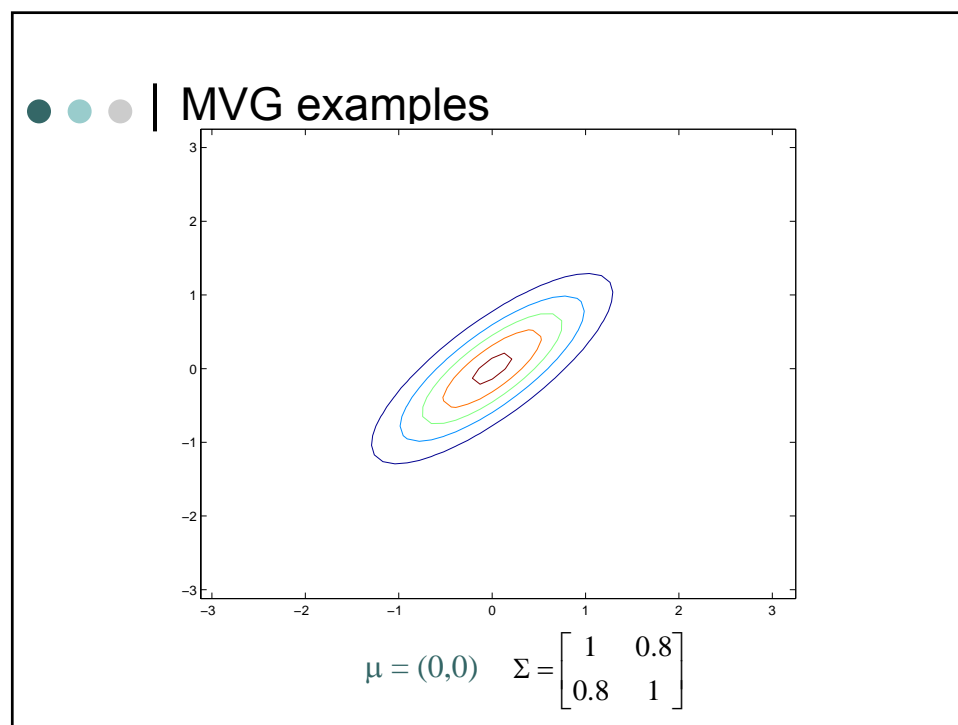
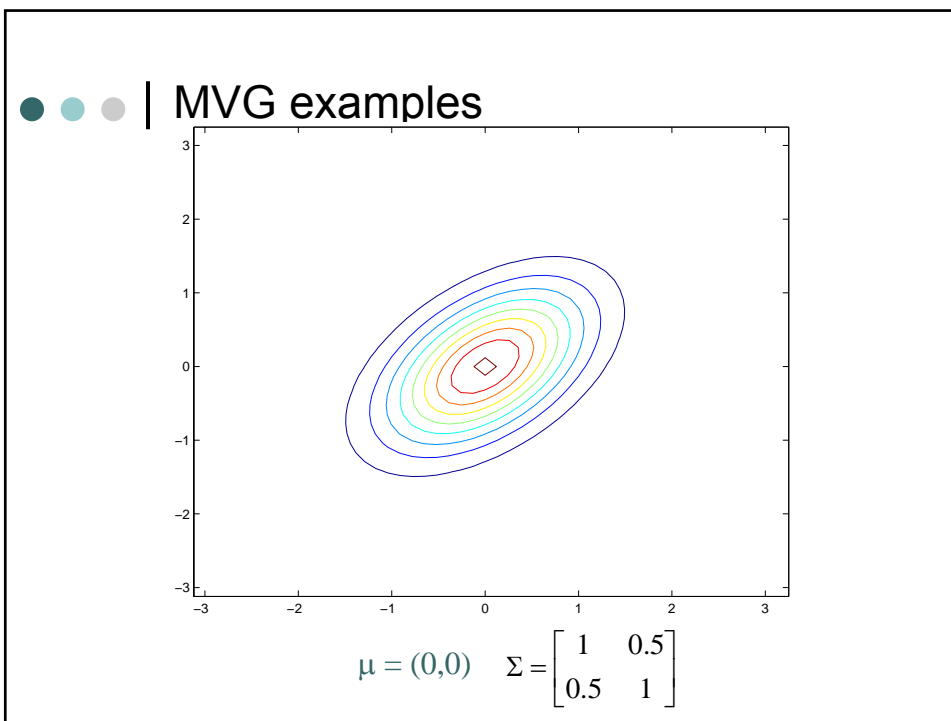


$$\mu = (0,0) \quad \Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

● ● ● | MVG examples – contour plots



$$\mu = (0,0) \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



## ● ● ● | Multivariate normal distribution

- We can generalize this to n dimensions
- parameters
  - mean vector  $\mu \in \mathbb{R}^n$
  - a covariance matrix  $\Sigma \in \mathbb{R}^{n \times n}$ , where  $\Sigma \geq 0$  is symmetric and positive semi-definite
- Written  $N(\mu, \Sigma)$ , density is

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

- where  $|\Sigma|$  is the determinant of the matrix  $\Sigma$
- For  $X \sim N(\mu, \Sigma)$ 
  - $E[X] = \int_{\mathbf{x}} \mathbf{x} p(\mathbf{x}; \mu, \Sigma) d\mathbf{x} = \mu$
  - $\text{Cov}(X) = E[XX^T] - (E[X])(E[X])^T = \Sigma$

## ● ● ● | Estimating the MVG parameters

- Given a set of data points  $\{\mathbf{x}^1, \dots, \mathbf{x}^N\}$ , the maximum likelihood estimates for the parameters of the MVG are:

$$\hat{\mu} = \frac{1}{N} \sum_i \mathbf{x}_i$$

$$\hat{\Sigma} = \frac{1}{N} \sum_i (\mathbf{x}_i - \hat{\mu})(\mathbf{x}_i - \hat{\mu})^T$$

## ● ● ● | Putting it all together in GDA

- Here
  - $y \sim \text{Bernoulli}(\phi)$
  - $x|y=0 \sim N(\mu_0, \Sigma)$
  - $x|y=1 \sim N(\mu_1, \Sigma)$
- writing this out, we get:

$$p(y) = \phi^y (1 - \phi)^{1-y}$$

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

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

## ● ● ● | Log-likelihood

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

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

## ● ● ● | GDA Maximum likelihood estimates

$$\hat{\phi} = \frac{N_{Y=1}}{N}$$

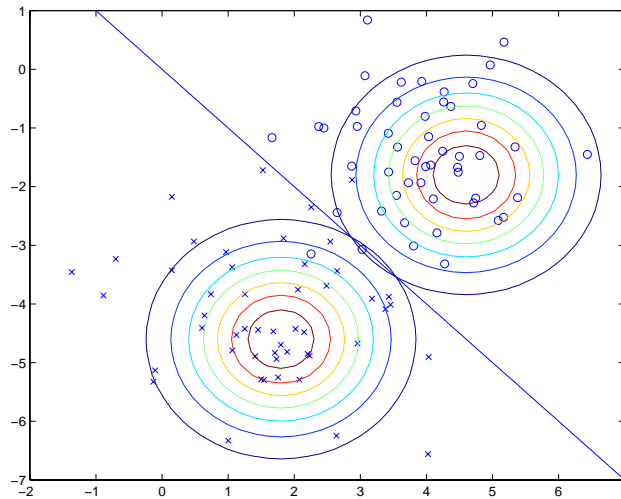
$$\hat{\mu}_0 = \frac{1}{N_{Y=0}} \sum_{i:y_i=0} \mathbf{x}_i$$

$$\hat{\mu}_1 = \frac{1}{N_{Y=1}} \sum_{i:y_i=1} \mathbf{x}_i$$

$$\hat{\Sigma} = \frac{1}{N} \sum_i (\mathbf{x}_i - \hat{\mu}_{y_i}) \cdot (\mathbf{x}_i - \hat{\mu}_{y_i})^T$$

note: this is a biased estimator for  $\Sigma$

## ● ● ● | Example



The decision boundary is at  $p(y=1|x) = 0.5$

## ● ● ● | Gaussian Naïve Bayes

Eg., character recognition:  $X_i$  is  $i^{\text{th}}$  pixel



Gaussian Naïve Bayes (GNB):

$$P(X_i = x | Y = y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} e^{-\frac{(x-\mu_{ik})^2}{2\sigma_{ik}^2}}$$

Sometimes assume variance

- is independent of  $Y$  (i.e.,  $\sigma_i$ ),
- or independent of  $X_i$  (i.e.,  $\sigma_k$ )
- or both (i.e.,  $\sigma$ )

## ● ● ● | GNB Maximum likelihood estimates

$$\hat{\phi}_k = \frac{N_{Y=k}}{N}$$

$$\hat{\mu}_{ik} = \frac{1}{N_{Y=k}} \sum_{j:y_j=k} x_i^j$$

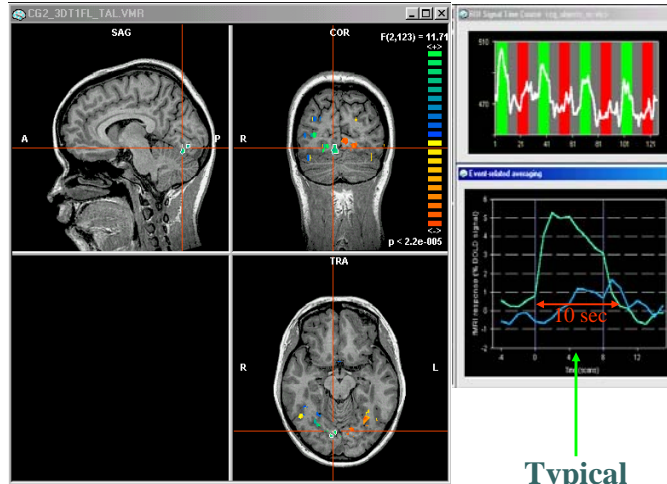
$$\hat{\sigma}_{ik}^2 = \frac{1}{N_{Y=k}} \sum_{j:y_j=k} (\mathbf{x}_i^j - \hat{\mu}_{ik})^2$$

Example: GNB for classifying mental states

[Mitchell et al.]

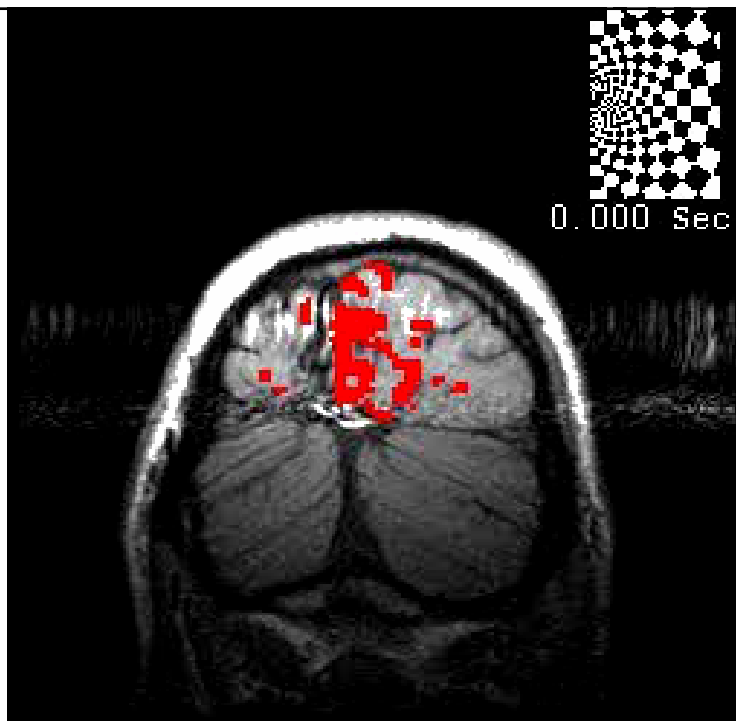
~1 mm resolution  
~2 images per sec.  
15,000 voxels/image  
non-invasive, safe

measures Blood  
Oxygen Level  
Dependent (BOLD)  
response



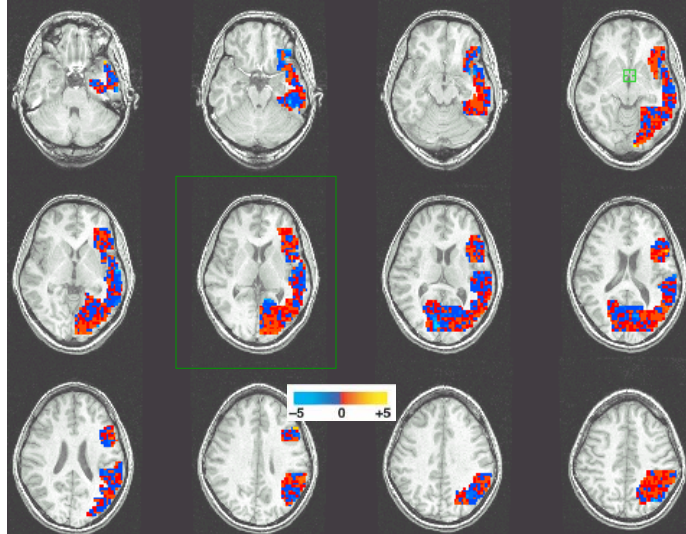
Typical  
impulse  
response

Brain scans can track  
activation with precision  
and sensitivity



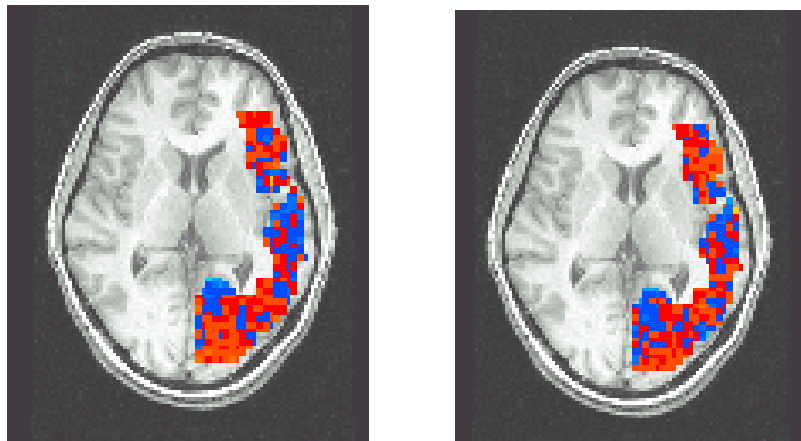
[Mitchell et al.]

- ● ● | Gaussian Naïve Bayes: Learned  $\mu_{\text{voxel,word}}$   
 $P(\text{BrainActivity} \mid \text{WordCategory} = \{\text{People,Animal}\})$  [Mitchell et al.]



- ● ● | Learned Bayes Models – Means for  
 $P(\text{BrainActivity} \mid \text{WordCategory})$  [Mitchell et al.]  
 Pairwise classification accuracy: 85%

People words      -5      0      +5      Animal words



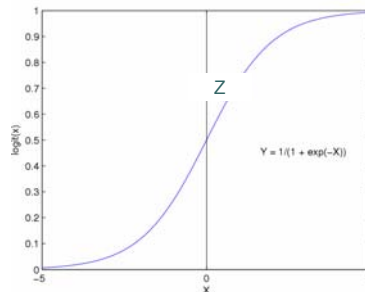
## ● ● ● | Summary of Naïve Bayes

- Simple model that makes very strong/incorrect independence assumptions
- Often works surprisingly well in practice
- Very popular, particularly in natural language processing and information retrieval where there are many features compared to the number of examples
- In application with lots of data, Naïve Bayes (and logistic regression) do not usually perform as well as more sophisticated methods.
- Good 'strawman'

## ● ● ● | Logistic Regression

- Learn  $P(Y|\mathbf{X})$  directly
  - Assume a particular functional form
  - Sigmoid applied to a linear function of the data:

$$P(Y = 1 | \mathbf{X}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$



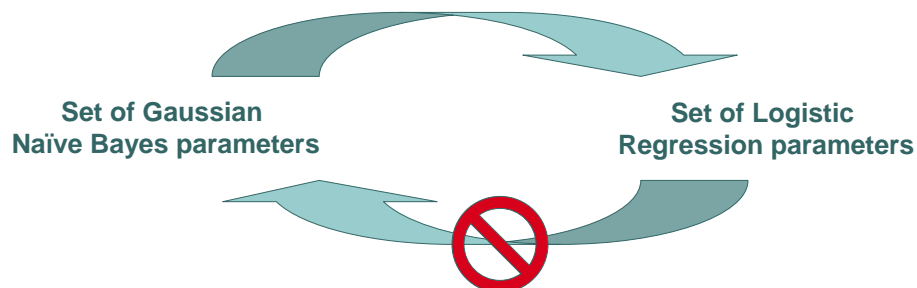
**Logistic function (or Sigmoid):**  $g(z) = \frac{1}{1 + \exp(-z)}$

## ● ● ● | Logistic regression v. Naïve Bayes

- Consider learning  $f: X \rightarrow Y$ , where
  - $X$  is a vector of real-valued features,  $\langle X_1 \dots X_n \rangle$
  - $Y$  is boolean
- Could use a Gaussian Naïve Bayes classifier
  - assume all  $X_i$  are conditionally independent given  $Y$
  - model  $P(X_i | Y = y_k)$  as Gaussian  $N(\mu_{ik}, \sigma_i)$
  - model  $P(Y)$  as Bernoulli( $\theta, 1-\theta$ )
- What does that imply about the form of  $P(Y|X)$ ?

$$P(Y = 1 | \mathbf{X}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

## ● ● ● | Gaussian Naïve Bayes v. Logistic Regression



- Representation equivalence
  - **But only in a special case:** GNB with class-independent variances
- But what's the difference???
- **LR makes no assumptions about  $P(X|Y)$  in learning**
- **Loss function**
  - Optimize different functions  $\rightarrow$  Obtain different solutions

## ● ● ● | Logistic Regression

- In logistic regression, we learn the conditional distribution  $P(y|\mathbf{x})$
- Let  $p_y(\mathbf{x};\mathbf{w})$  be our estimate of  $P(y|\mathbf{x})$ , where  $\mathbf{w}$  is a vector of adjustable parameters.
- Assume there are two classes,  $y = 0$  and  $y = 1$  and

$$p_1(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

$$p_0(\mathbf{x}; \mathbf{w}) = 1 - p_1(\mathbf{x}; \mathbf{w})$$

- This is equivalent to

$$\log \frac{p_1(\mathbf{x}; \mathbf{w})}{p_0(\mathbf{x}; \mathbf{w})} = \mathbf{w} \cdot \mathbf{x}$$

low, the log odds of class 1 is a linear function of  $\mathbf{x}$

## ● ● ● | Constructing a Learning Algorithm

- Because we're fitting a conditional probability distribution, we find the probability distribution  $h$  that is most likely, given the data.
- Let  $S$  be the training sample. Our goal is to find  $h$  to maximize  $P(h_0|S)$ .

$$\begin{aligned} \operatorname{argmax}_{h_0} P(h_0 | S) &= \operatorname{argmax}_{h_0} \frac{P(S | h_0)P(h_0)}{P(S)} && \text{by Bayes' Rule} \\ &= \operatorname{argmax}_{h_0} P(S | h_0)P(h_0) && \text{because } P(S) \text{ doesn't depend on } h \\ &= \operatorname{argmax}_{h_0} P(S | h_0) && \text{if we assume } P(h) \text{ is uniform} \\ &= \operatorname{argmax}_{h_0} \log P(S | h_0) && \text{because log is monotone} \end{aligned}$$

- The **likelihood function** views  $P(S|h_0)$  as a function of the parameters in the model. In this case, our parameters are the weights,  $\mathbf{w}$ .  
 $L(\mathbf{w};S) = P(S|h_0)$
- The  $\mathbf{w}$  that maximizes the likelihood of the training data is the **maximum likelihood estimator**

## ● ● ● | Computing the Likelihood

- In this framework, we assume that each training example  $(\mathbf{x}^i, y^i)$  is drawn from the same underlying (but unknown) probability distribution  $P(\mathbf{x}, y)$ . This means that the log likelihood of  $S$  is the sum of the log likelihoods of the individual training examples:

$$\log P(S | h) = \log \prod_i P((\mathbf{x}_i, y_i) | h)$$

$$\log P(S | h) = \sum_i \log P((\mathbf{x}_i, y_i) | h)$$

- Any joint distribution  $P(a, b)$  can be factored as  $P(a|b)P(b)$ . So, we have:

$$\begin{aligned} \arg \max_h \log P(S | h) &= \arg \max_h \sum_i \log P((\mathbf{x}_i, y_i) | h) \\ &= \arg \max_h \sum_i \log P(y_i | \mathbf{x}_i, h) P(\mathbf{x}_i | h) \end{aligned}$$

- Now,  $p(\mathbf{x}|h)=P(\mathbf{x})$ , because it does not depend on  $h$

$$\begin{aligned} \arg \max_h \log P(S | h) &= \arg \max_h \sum_i \log P(y_i | \mathbf{x}_i, h) P(\mathbf{x}_i) \\ &= \arg \max_h \sum_i \log P(y_i | \mathbf{x}_i, h) \end{aligned}$$

- Hence, the log likelihood of  $S$  is the sum of the log conditional likelihood of the individual data points

## ● ● ● | Log Likelihood for Conditional Probability Estimators

- Take an example  $(\mathbf{x}^i, y^i)$ 
  - if  $y^i = 0$ , the log likelihood is  $\log(1 - p_1(\mathbf{x}; \mathbf{w}))$
  - if  $y^i = 1$ , the log likelihood is  $\log p_1(\mathbf{x}; \mathbf{w})$
- These two are mutually exclusive, so we can combine them to get:

$$\ell(\mathbf{w}; \mathbf{x}_i, y) = \log P(y_i | \mathbf{x}_i, \mathbf{w}) = (1 - y_i) \log[1 - p_1(\mathbf{x}; \mathbf{w})] + y_i \log p_1(\mathbf{x}; \mathbf{w})$$

- The goal of our learning algorithm will be to find  $w$  to maximize:

$$J(\mathbf{w}) = \ell(\mathbf{w}; \mathbf{x}_i, y)$$

## ● ● ● | Fitting LR by Gradient Ascent

$$\begin{aligned}
 \frac{\partial J(\mathbf{w})}{\partial w_j} &= \sum_i \frac{\partial}{\partial w_j} \ell(\mathbf{w}; y^i, \mathbf{x}^i) \\
 \frac{\partial}{\partial w_j} \ell(\mathbf{w}; y^i, \mathbf{x}^i) &= \frac{\partial}{\partial w_j} ((1 - y^i) \log[1 - p_1(\mathbf{x}^i; \mathbf{w})] + y^i \log p_1(\mathbf{x}^i; \mathbf{w})) \\
 &= (1 - y^i) \frac{1}{1 - p_1(\mathbf{x}^i; \mathbf{w})} \left( - \frac{\partial p_1(\mathbf{x}^i; \mathbf{w})}{\partial w_j} \right) + y^i \frac{1}{p_1(\mathbf{x}^i; \mathbf{w})} \left( \frac{\partial p_1(\mathbf{x}^i; \mathbf{w})}{\partial w_j} \right) \\
 &= \left[ \frac{y^i}{p_1(\mathbf{x}^i; \mathbf{w})} - \frac{(1 - y^i)}{1 - p_1(\mathbf{x}^i; \mathbf{w})} \right] \left( \frac{\partial p_1(\mathbf{x}^i; \mathbf{w})}{\partial w_j} \right) \\
 &= \left[ \frac{y^i(1 - p_1(\mathbf{x}^i; \mathbf{w})) - (1 - y^i)p_1(\mathbf{x}^i; \mathbf{w})}{p_1(\mathbf{x}^i; \mathbf{w})(1 - p_1(\mathbf{x}^i; \mathbf{w}))} \right] \left( \frac{\partial p_1(\mathbf{x}^i; \mathbf{w})}{\partial w_j} \right) \\
 &= \left[ \frac{y^i - p_1(\mathbf{x}^i; \mathbf{w})}{p_1(\mathbf{x}^i; \mathbf{w})(1 - p_1(\mathbf{x}^i; \mathbf{w}))} \right] \left( \frac{\partial p_1(\mathbf{x}^i; \mathbf{w})}{\partial w_j} \right)
 \end{aligned}$$

## ● ● ● | Gradient cont.

- So we get:

$$\begin{aligned}
 \frac{\partial p_1(\mathbf{x}^i; \mathbf{w})}{\partial w_j} &= \frac{1}{(1 + e^{-\mathbf{w} \cdot \mathbf{x}^i})^2} \frac{\partial}{\partial w_j} (1 + e^{-\mathbf{w} \cdot \mathbf{x}^i}) \\
 &= \frac{1}{(1 + e^{-\mathbf{w} \cdot \mathbf{x}^i})^2} e^{-\mathbf{w} \cdot \mathbf{x}^i} \frac{\partial}{\partial w_j} (-\mathbf{w} \cdot \mathbf{x}^i) \\
 &= \frac{1}{(1 + e^{-\mathbf{w} \cdot \mathbf{x}^i})^2} e^{-\mathbf{w} \cdot \mathbf{x}^i} (-x_{ij}) \\
 &= p_1(\mathbf{x}^i; \mathbf{w})(1 - p_1(\mathbf{x}^i; \mathbf{w}))x_{ij}
 \end{aligned}$$

## ● ● ● | Gradient cont.

- The gradient of the loglikelihood for a single point is:

$$\begin{aligned}\frac{\partial}{\partial w_j} \ell(\mathbf{w}; \mathbf{x}^i, y^i) &= \left[ \frac{y^i - p_1(\mathbf{x}^i; \mathbf{w})}{p_1(\mathbf{x}^i; \mathbf{w})(1 - p_1(\mathbf{x}^i; \mathbf{w}))} \right] \left( \frac{\partial p_1(\mathbf{x}^i; \mathbf{w})}{\partial w_j} \right) \\ &= \left[ \frac{y^i - p_1(\mathbf{x}^i; \mathbf{w})}{p_1(\mathbf{x}^i; \mathbf{w})(1 - p_1(\mathbf{x}^i; \mathbf{w}))} \right] p_1(\mathbf{x}^i; \mathbf{w})(1 - p_1(\mathbf{x}^i; \mathbf{w})) x_{ij} \\ &= (y^i - p_1(\mathbf{x}^i; \mathbf{w})) x_{ij}\end{aligned}$$

- The overall gradient is:

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \sum_i (y^i - p_1(\mathbf{x}^i; \mathbf{w})) x_{ij}$$

## ● ● ● | Batch Ascent Descent

- Training set  $\{\langle \mathbf{x}^i, y^i \rangle\}$ ,  $i = 1..N$

Repeat until convergence {  
for every j

$$w_j = w_j + \alpha \sum_{i=1}^N \left( y^i - \frac{1}{(1 + e^{-\mathbf{w}\mathbf{x}^i})} \right) x_j^i$$

}

## ● ● ● | Logistic Regression for $K > 2$

- To handle  $K > 2$  classes, we make one class the 'reference' class. Suppose it is class  $K$ . Then we represent each of the other classes as a logistic function of the odds of class  $k$  versus class  $K$ :

$$\log \frac{P(y = 1 | \mathbf{x})}{P(y = K | \mathbf{x})} = \mathbf{w}_1 \cdot \mathbf{x}$$

$$\log \frac{P(y = 2 | \mathbf{x})}{P(y = K | \mathbf{x})} = \mathbf{w}_2 \cdot \mathbf{x}$$

⋮

$$\log \frac{P(y = k-1 | \mathbf{x})}{P(y = K | \mathbf{x})} = \mathbf{w}_{k-1} \cdot \mathbf{x}$$

- The conditional probability for class  $k \neq K$  is

$$P(y = k | \mathbf{x}) = \frac{e^{\mathbf{w}_k \cdot \mathbf{x}}}{1 + \sum_{j=1}^{K-1} e^{\mathbf{w}_j \cdot \mathbf{x}}}$$

- and for class  $k = K$ :

$$P(y = K | \mathbf{x}) = \frac{1}{1 + \sum_{j=1}^{K-1} e^{\mathbf{w}_j \cdot \mathbf{x}}}$$

## ● ● ● | Loss functions: Likelihood v. Conditional Likelihood

- Generative (Naïve Bayes) Loss function:

### Data likelihood

$$\begin{aligned} \ln P(\mathcal{D} | \mathbf{w}) &= \sum_{j=1}^N \ln P(\mathbf{x}^j, y^j | \mathbf{w}) \\ &= \sum_{j=1}^N \ln P(y^j | \mathbf{x}^j, \mathbf{w}) + \sum_{j=1}^N \ln P(\mathbf{x}^j | \mathbf{w}) \end{aligned}$$

- Discriminative models cannot compute  $P(\mathbf{x} | \mathbf{w})$ !
- But, discriminative (logistic regression) loss function:

### Conditional Data Likelihood

$$\ln P(\mathcal{D}_Y | \mathcal{D}_X, \mathbf{w}) = \sum_{j=1}^N \ln P(y^j | \mathbf{x}^j, \mathbf{w})$$

- Doesn't waste effort learning  $P(X)$  – focuses on  $P(Y|X)$  all that matters for classification

● ● ● | That's all M(C)LE. How about MAP?

- One common approach is to define priors on  $\mathbf{w}$ 
  - Normal distribution, zero mean, identity covariance
  - "Pushes" parameters towards zero

$$p(\mathbf{w} | Y, \mathbf{X}) \propto P(Y | \mathbf{X}, \mathbf{w})p(\mathbf{w})$$

- Corresponds to **Regularization**
  - Helps avoid very large weights and overfitting
- MAP estimate

$$\mathbf{w}^* = \arg \max_{\mathbf{w}} \ln \left[ p(\mathbf{w}) \prod_{j=1}^N P(y^j | \mathbf{x}^j, \mathbf{w}) \right]$$

● ● ● | MLE vs MAP

- Maximum conditional likelihood estimate

$$\mathbf{w}^* = \arg \max_{\mathbf{w}} \ln \left[ \prod_{j=1}^N P(y^j | \mathbf{x}^j, \mathbf{w}) \right]$$

$$w_i \leftarrow w_i + \eta \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 | \mathbf{x}^j, \mathbf{w})]$$

- Maximum conditional a posteriori estimate

$$\mathbf{w}^* = \arg \max_{\mathbf{w}} \ln \left[ p(\mathbf{w}) \prod_{j=1}^N P(y^j | \mathbf{x}^j, \mathbf{w}) \right]$$

$$w_i \leftarrow w_i + \eta \left\{ -\lambda w_i + \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 | \mathbf{x}^j, \mathbf{w})] \right\}$$

## ● ● ● | Summary of Logistic Regression

- Learns the **Conditional Probability Distribution**  $P(y|x)$
- Local Search. Begins with initial weight vector. Modifies it iteratively to maximize an objective function. The objective function is the **conditional log likelihood** of the data – so the algorithm seeks the probability distribution  $P(y|x)$  that is most likely given the data.

## ● ● ● | GNB vs LR #1

Consider  $Y$  boolean,  $X_i$  continuous,  $X = \langle X_1 \dots X_n \rangle$

Number of parameters:

- NB:  $4n + 1$
- LR:  $n + 1$

Estimation method:

- NB parameter estimates are uncoupled
- LR parameter estimates are coupled

## ● ● ● | GNB vs. LR #2

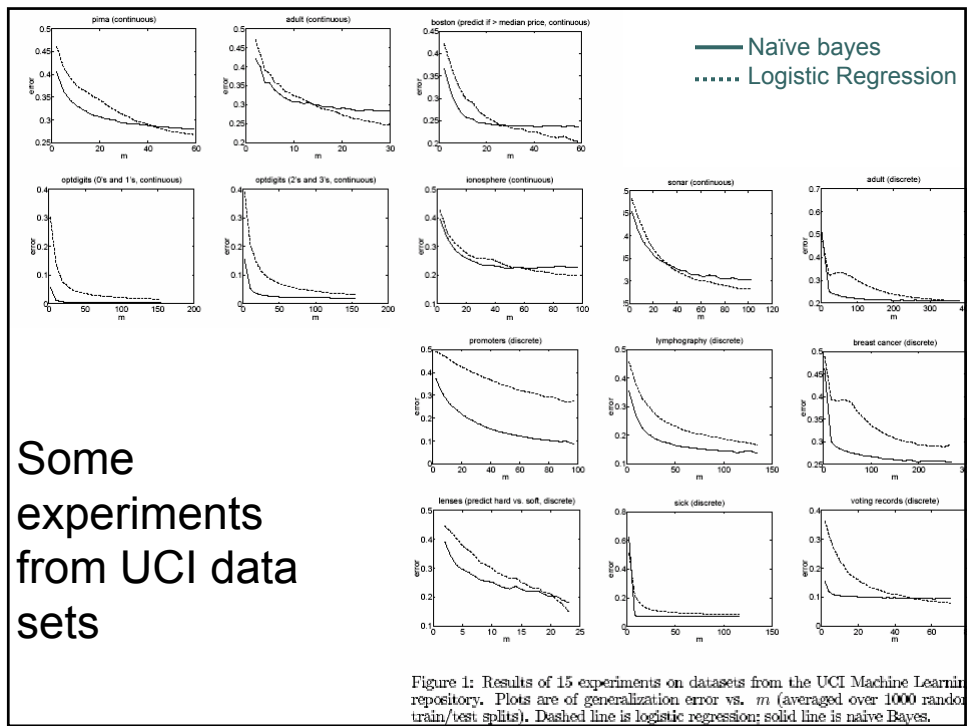
[Ng & Jordan, 2002]

- Generative and Discriminative classifiers
- Asymptotic comparison (# training examples  $\rightarrow$  infinity)
  - when model correct
    - GNB, LR produce identical classifiers
  - when model incorrect
    - LR is less biased – does not assume conditional independence
      - **therefore LR expected to outperform GNB**

## ● ● ● | GNB vs. LR #3

[Ng & Jordan, 2002]

- Generative and Discriminative classifiers
- Non-asymptotic analysis
  - convergence rate of parameter estimates,  $n = \#$  of attributes in  $X$ 
    - Size of training data to get close to infinite data solution
    - GNB needs  $O(\log n)$  samples
    - LR needs  $O(n)$  samples
  - **GNB converges more quickly to its (perhaps less helpful) asymptotic estimates**



## ● ● ● | What you should know LR

- GNB with class-independent variances representationally equivalent to LR
  - Solution differs because of objective (loss) function
- In general, NB and LR make different assumptions
  - NB: Features independent given class  $\rightarrow$  assumption on  $P(\mathbf{X}|Y)$
  - LR: Functional form of  $P(Y|\mathbf{X})$ , no assumption on  $P(\mathbf{X}|Y)$
- LR is a linear classifier
  - decision rule is a hyperplane
- LR optimized by conditional likelihood
  - no closed-form solution
  - concave  $\rightarrow$  global optimum with gradient ascent
  - Maximum conditional a posteriori corresponds to regularization
- Convergence rates
  - GNB (usually) needs more data
  - LR (usually) gets to better solutions in the limit