Naïve Bayes

Generative vs Discriminative

- Discriminative methods model P(y|**x**) directly
  - Logistic Regression
- Generative methods model P(**x**|y) and P(y)
  - Linear Discriminant analysis

- Under LDA model we can show

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

where θ is some function of p, Σ, μ_0, and μ_1

the same form used by logistic regression

- This indicates
  - If $P(\mathbf{x} \mid y)$ is a multivariate Gaussian distribution, $P(y \mid \mathbf{x})$ follows a logistic function
  - But the converse is not true
- LDA makes stronger modeling assumptions

Generative models

- Create something that can <u>generate</u> ex's
- Can create <u>complete</u> input feature vectors
  - Describes probability distributions for <u>all</u> features
  - Stochastically create a plausible feature vector
  - Example: Bayes net
- Eg, describe the class of birds
  - Probably has feathers, lays eggs, flies, etc
- Make a model that *generates* positives
- Make a model that *generates* negatives
- Classify a test example based on which is more likely to generate it
  - The Naïve Bayes ratio does this

Discriminative Models

- What <u>differentiates</u> class A from class B?
- Don’t try to model all the features, instead focus on the task of categorizing
  - Captures <u>differences</u> between categories
  - May not use all features in models
  - Examples: decision trees, SVMs, & neural nets
  - Eg, what differentiates birds and mammals?
- Typically more efficient and simpler

LDA – Continuous Inputs

- LDA is a generative model for continuous inputs and assumes a multivariate Gaussian on the entire feature space
- How do we handle discrete inputs
  - Simplest Case: Naïve Bayes classifer
  - More general Case: Bayesian Networks (We will spend considerable amount of time understanding Bayes nets in this course)

Success Story of Naïve Bayes: Spam Filter

- The naïve Bayes classifier is widely used for text data (hence this example)
- We want to classify email messages into the spam and non-spam categories
- Our training set is a set of emails that has been classified manually into the two categories
- First question: how do we represent an email using a feature vector **x** – what features should we use?

A Bayes Classifier

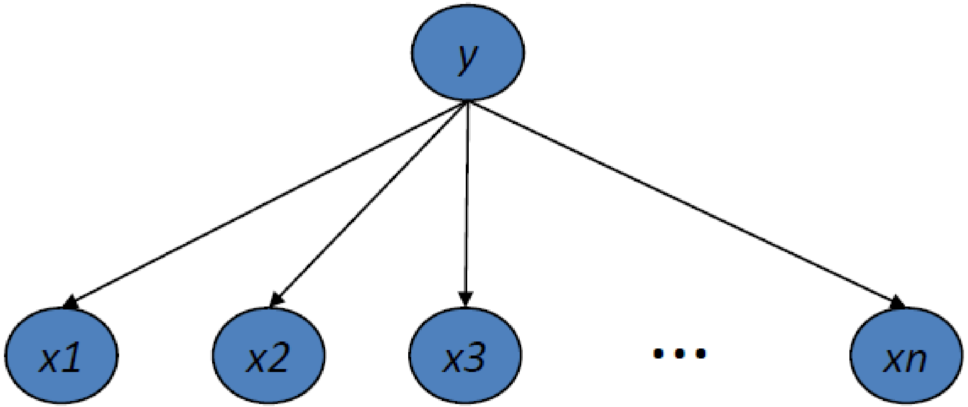
- To learn a bayes classifier, we need to model P(**x**|*y*) and P(*y*)
- If our vocabulary has n words, there are 2^n possible values for **x**
- If we model P(**x**|*y*) explicitly as a multinomial distribution over all possible values of **x**, we need to learn $2*(2^n - 1)$ parameters
- To avoid such problem, we can assume that x_i*'s* are conditionally independent given y, i.e.,

$$P(x_1, x_2, \ldots, x_n \mid y) = \prod_{i=1}^{n} P(x_i \mid y)$$

- This is called the **Naïve Bayes assumption**
- The number of parameters for P(x|y) is now 2*n (Why?)

Hence *avoids* estimating probability of compound features (eg., x1 $\wedge$ x2)

Naïve Bayes



- A generative model – an email is generated as follows:
  - Determine if it is a spam or not according to P(y) (Bernoulli)
  - Determine if each word x_i in the vocabulary is contained in the message *independently* according to P(x_i | y) (Bernoulli)
- For this model, we need to learn:
  - For y: P($y=1$)
  - For x_i: P(x_i =1| y = 1) and P(x_i =1| y = 0) "class conditional probability" for i=1,…,n

MLE for Naïve Bayes

Suppose our training set contained N emails, the maximum likelihood estimate of the parameters are :

$P(y=1)=\frac{N_1}{N}$, where N_1 is the number of spam emails

$P(x_i=1 \mid y=1)=\frac{N_{i|1}}{N_1}$,

i.e., the fraction of spam emails where x_i appeared

$P(x_i=1 \mid y=0)=\frac{N_{i|0}}{N_0}$

i.e., the fraction of the nonspam emails where x_i appeared

What if x_i is Multinomial?

- If x_i is discrete with more than two possible values $\{v_1, \ldots, v_m\}$, $P(x_i | y)$ can be described by a conditional probability table

	$y = 0$	$y=1$
$x_i=v_1$	$P(x_i=v_1 \mid y = 0)$	$P(x_i=v_1 \mid y = 1)$
$x_i=v_2$	$P(x_i=v_2 \mid y = 0)$	$P(x_i=v_2 \mid y = 1)$
…	…	…
$x_i=v_m$	$P(x_i=v_m \mid y = 0)$	$P(x_i=v_m \mid y = 1)$

- Really only needs m-1 rows since rows sum to 1
- In multi-class cases, we just need to add more columns to the above table.

$$P(x_i = v_j \mid y = k) = \frac{N_{ij|k}}{N_k}$$

i.e., the fraction of class k examples where x_i took value v_j

Problem with MLE

- Many words are rare, particularly when considering a particular class
  - The probability estimates for such words can be poor for such words, even with a reasonably large dataset
- Consider the spam example:
  - Suppose in our training set "Mahalanobis" appears in a non-spam mail and never appears in a spam mail
  - Suppose also that "XXX" appears in a spam message but no non-spam messages
  - Now suppose we get a new message **x** that contains both words
- We will have that P(**x**|*y*) = $\prod_i P(x_i \mid y) = 0$ for both *y=0* and *y=1*
  - Because P("Mahalanobis" | **y**=1) = 0 and P("XXX" | **y**=0) = 0
- Given limited training data, MLE can result in probabilities of 0 or 1. Such extreme probabilities are "too strong" and cause problems.
  - Use Laplace smoothing to help correct this

Laplace Smoothing

- Suppose we estimate a probability P(z) and we have n_0 examples where z is false and n_1 examples where z is true. Our MLE estimate is $$P(z=1)=\frac{n_1}{n_0+n_1}$$
- Laplace Estimate. Add 1 to the numerator and 2 to the denominator $$P(z=1)=\frac{n_1+1}{n_0+n_1+2}$$

  If we don't observe any examples, we expect P(*z*=1) = 0.5, but our belief is weak (equivalent to seeing one example of each outcome).

  As n_0 and n_1 get large converges to MLE
- If z has K different outcomes, then we estimate it as

$$P(z=k)=\frac{n_k+1}{n+K}$$

Learning and Predicting

- Learning
  - Need to estimate the following probability distributions (via counting)

$p(y)$ $\boxed{\text{Prior distribution of y}}$

$p(x_i \mid y)$ $\boxed{\text{Class conditional distribution of } x_i}$

- Predicting
  - Given **x** =(x_1, x_2, …, x_d), compute p(y|**x**)

$$p(y \mid \mathbf{x}) = \frac{p(y)p(\mathbf{x} \mid y)}{p(\mathbf{x})} \propto p(y)\prod_i p(x_i \mid y)$$

  - Apply decision theory to make final prediction of *y*

NB learns a linear decision boundary

- For binary feature spaces Naïve Bayes gives a linear decision boundary

$$P(\mathrm{x}|Y=y) = P(x_1=v_1|Y=y) \cdot P(x_2=v_2|Y=y) \cdots P(x_n=v_n|Y=y)$$

- Define a discriminant function for class 1 versus class 0

$$h(\mathrm{x}) = \frac{P(Y=1|\mathrm{X})}{P(Y=0|\mathrm{X})} = \frac{P(x_1=v_1|Y=1)}{P(x_1=v_1|Y=0)} \cdots \frac{P(x_n=v_n|Y=1)}{P(x_n=v_n|Y=0)} \cdot \frac{P(Y=1)}{P(Y=0)}$$

Log of Odds Ratio

$$\frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} = \frac{P(x_1=v_1|y=1)}{P(x_1=v_1|y=0)} \cdots \frac{P(x_n=v_n|y=1)}{P(x_n=v_n|y=0)} \cdot \frac{P(y=1)}{P(y=0)}$$

$$\log\frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} = \log\frac{P(x_1=v_1|y=1)}{P(x_1=v_1|y=0)} + \ldots \log\frac{P(x_n=v_n|y=1)}{P(x_n=v_n|y=0)} + \log\frac{P(y=1)}{P(y=0)}$$

Suppose each x_j is binary and define

$$\alpha_{j,0} = \log\frac{P(x_j=0|y=1)}{P(x_j=0|y=0)}$$

$$\alpha_{j,1} = \log\frac{P(x_j=1|y=1)}{P(x_j=1|y=0)}$$

Log Odds

- Now rewrite as

$$\log \frac{P(y=1|\mathrm{x})}{P(y=0|\mathrm{x})} = \sum_j \alpha_{j,1} \cdot x_j + \alpha_{j,0} \cdot (1 - x_j) + \log \frac{P(y=1)}{P(y=0)}$$

$$\log \frac{P(y=1|\mathrm{x})}{P(y=0|\mathrm{x})} = \sum_j (\alpha_{j,1} - \alpha_{j,0}) x_j + \left(\sum_j \alpha_{j,0} + \log \frac{P(y=1)}{P(y=0)} \right)$$

- We classify into class 1 if this is ≥ 0 and into class 0 otherwise

- For arbitrary multinomial features the boundary is linear in a binary one-vs-all encoding of the features
- For numeric features the Gaussian naïve Bayes classifier does not give a linear boundary

Naïve Bayes Summary

- Generative classifier
  - learn P($\mathbf{x}|y$) and P(y)
  - Use Bayes rule to compute P($y|\mathbf{x}$) for classification
- Assumes conditional independence between features given class labels
  - Greatly reduces the numbers of parameters to learn
  - Referred to as the ***Naïve assumption***
- Batch learning but can be easily turned into online learning
  - Just incrementally update the various probability estimates
- Often works surprisingly well and a good “first thing” to try