COMP 652: Machine Learning

Lecture 9

Today

- Finishing discrete distribution estimation
- Generative learning example with multiclass output
- Dealing with multivariate / heterogeneous input vectors
  - Motivation from text classification
  - The naive Bayes assumption

Recall: estimating discrete distributions

- We assume data $x_1, x_2, \ldots, x_m \in \mathcal{X}$ where $\mathcal{X}$ is a finite set.
- Without loss of generality, suppose that $\mathcal{X} = \{1, 2, \ldots, k\}$.
- Let m_j be the number of data items that are j.
- The then maximum likelihood estimates are just the empirical frequencies:

$$P(x = j) = \frac{m_j}{m}$$

Proof for the general case

- Let p_i be the probability $P(x = i)$, for $i \in \{1, \ldots, k\}$.
- For data $x_1, x_2, \ldots, x_m$, the likelihood is

$$l(x_1, \ldots, x_m | p_1, \ldots, p_k) = \Pi_{i=1}^{m} p_{x_i}$$

- Collecting by possible values of x, we obtain

$$l(x_1, \ldots, x_m | p_1, \ldots, p_k) = \Pi_{j=1}^{k} p_j^{m_j}$$

  where m_j is the number of times j appears in the data set.
- The log likelihood is thus

$$\log l(x_1, \ldots, x_m | p_1, \ldots, p_k) = \sum_{j=1}^{k} m_j \log p_j$$

- We want to maximize this w.r.t. the p_j, but the p_j are constrained. They must be between zero and one, and they must sum to one...

Lagrange multipliers

- Suppose we want to solve the optimization:

$$\max_{\mathbf{u}} f(\mathbf{u})$$

$$\text{subject to: } g(\mathbf{u}) = 0$$

- The theory or Lagrange multipliers says that it is equivalent to solving the unconstrained optimization problem:

$$\max_{\mathbf{u},\lambda} f(\mathbf{u}) + \lambda g(\mathbf{u})$$

Returning to the proof

- Let us seek to maximize the log likelihood:

$$\log l(x_1, \ldots, x_m | p_1, \ldots, p_k) = \sum_{j=1}^{k} m_j \log p_j$$

subject to the constraint that $\sum_{j=1}^{k} p_j = 1$.

- This ignores the restriction that $0 \leq p_j \leq 1$. However, it turns out this will be satisfied anyway if we solve the above problem.
- The Lagrange formulation gives us:

$$\max_{p_j, \lambda} \left(\sum_{j=1}^{k} m_j \log p_j \right) + \lambda \left(\left(\sum_{j=1}^{k} p_j \right) - 1 \right)$$

- Let f denote this function.

Setting partial derivatives to zero

Partial derivative with respect to λ is

$$\begin{aligned}\frac{\partial}{\partial \lambda} f &= \left(\sum_{j=1}^{k} p_j\right) - 1 = 0 \\ &\Rightarrow \sum_{j=1}^{k} p_j = 1\end{aligned}$$

Partial derivative with respect to p_i is

$$\begin{aligned}\frac{\partial}{\partial p_i} f &= \frac{m_i}{p_i} + \lambda = 0 \\ &\Rightarrow m_i + \lambda p_i = 0\end{aligned}$$

We have $k+1$ linear equations in $k+1$ unknowns.

Solving...

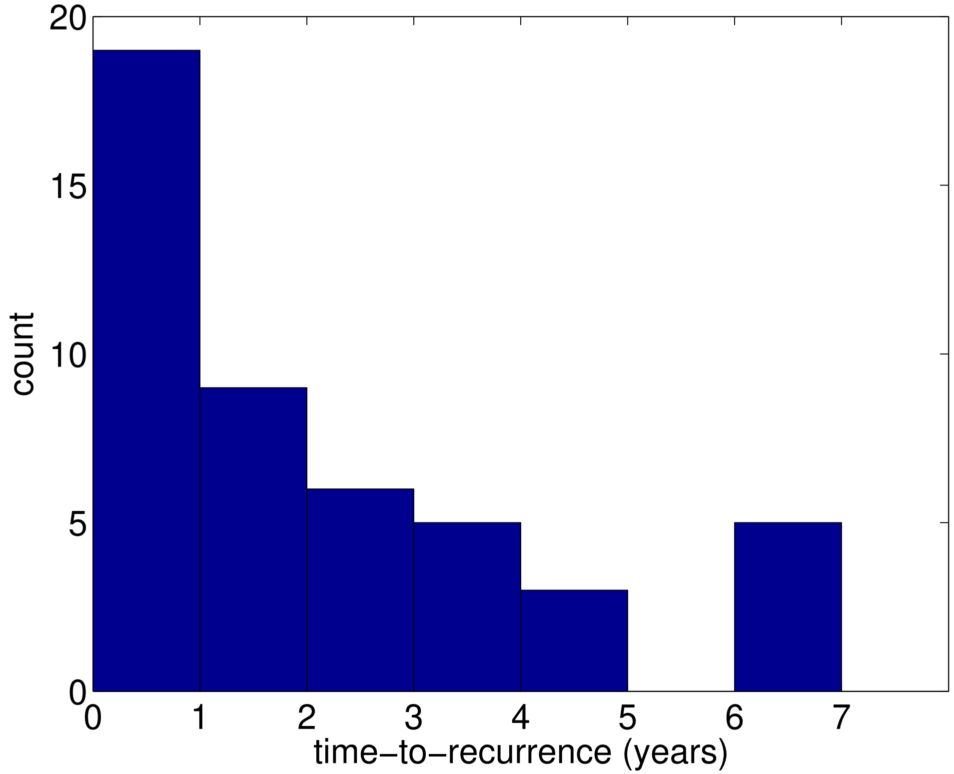
Summing the last formula over i, we can solve for λ:

$$\begin{aligned} & \sum_{i=1}^{k}(m_i + \lambda p_i) = 0 \\ \Rightarrow\quad & (\sum_{i=1}^{k} m_i) + \lambda(\sum_{i=1}^{k} p_i) = 0 \\ \Rightarrow\quad & m + \lambda = 0 \\ \Rightarrow\quad & \lambda = -m \end{aligned}$$

Returning to the p_i:

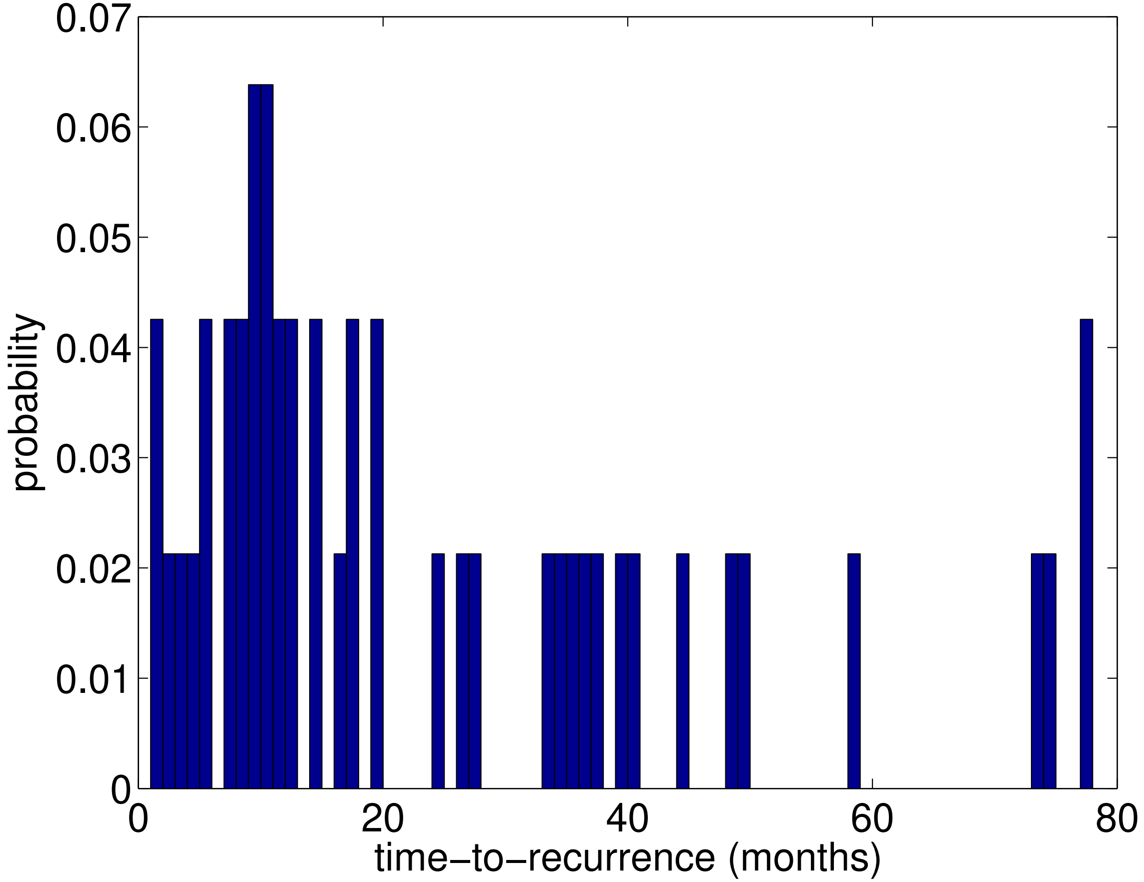
$$\begin{aligned} & m_i + \lambda p_i = 0 \\ \Rightarrow\quad & p_i = \frac{-m_i}{\lambda} \\ \Rightarrow\quad & p_i = \frac{m_i}{m} \end{aligned}$$

E.g., time-to-recurrence in years



years	0	1	2	3	4	5	6	7
t-t-r count	19	9	6	5	3	0	5	0
prob	0.4043	0.1915	0.1277	0.1064	0.0638	0	0.1064	0

E.g., time-to-recurrence in months



- Discretization/binning can affect your result strongly!
- Zero bins are problematic for interpretation, prediction, cross-validation.

“Smoothing” discrete distributions

To avoid the zero-bins problem, we might:

- Aggregate bins / choose bin ”width” so that there are no zeros bins
- More generally, we may use cross-validation to choose bin width
  - However, the correct way of doing this is a bit subtle. See HW3 – or, we may return later in the lecture.
- Another solution is to add pseudocounts – fake observations – to the data
  - We could add a constant number of counts to each bin, using cross-validation to determine the best constant
  - Or, we may use a domain-specific method to add pseudocounts

Generative learning example

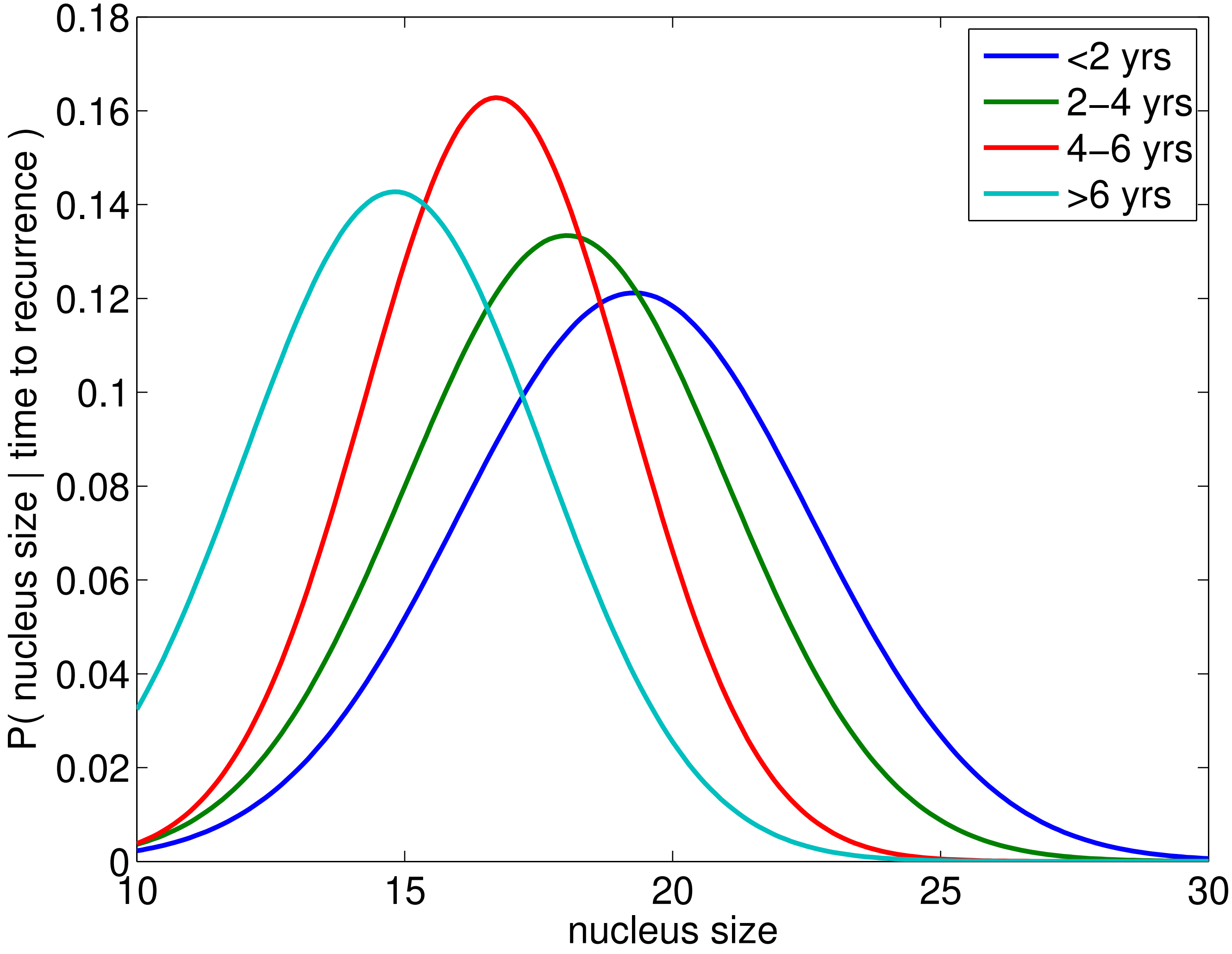
- Suppose we want to predict time-to-recurrence (y), discretized into four ranges: < 2 yrs, $2-4$ yrs, $4-6$ yrs, > 6 yrs.
- We base our predictions on real-valued nucleus size (x)
- The maximum likelihood estimates of the class priors, $P(y)$, are:

Class:	< 2 yrs	$2-4$ yrs	$4-6$ yrs	> 6 yrs
Count:	27	11	3	5
Prob:	0.59	0.24	0.06	0.11

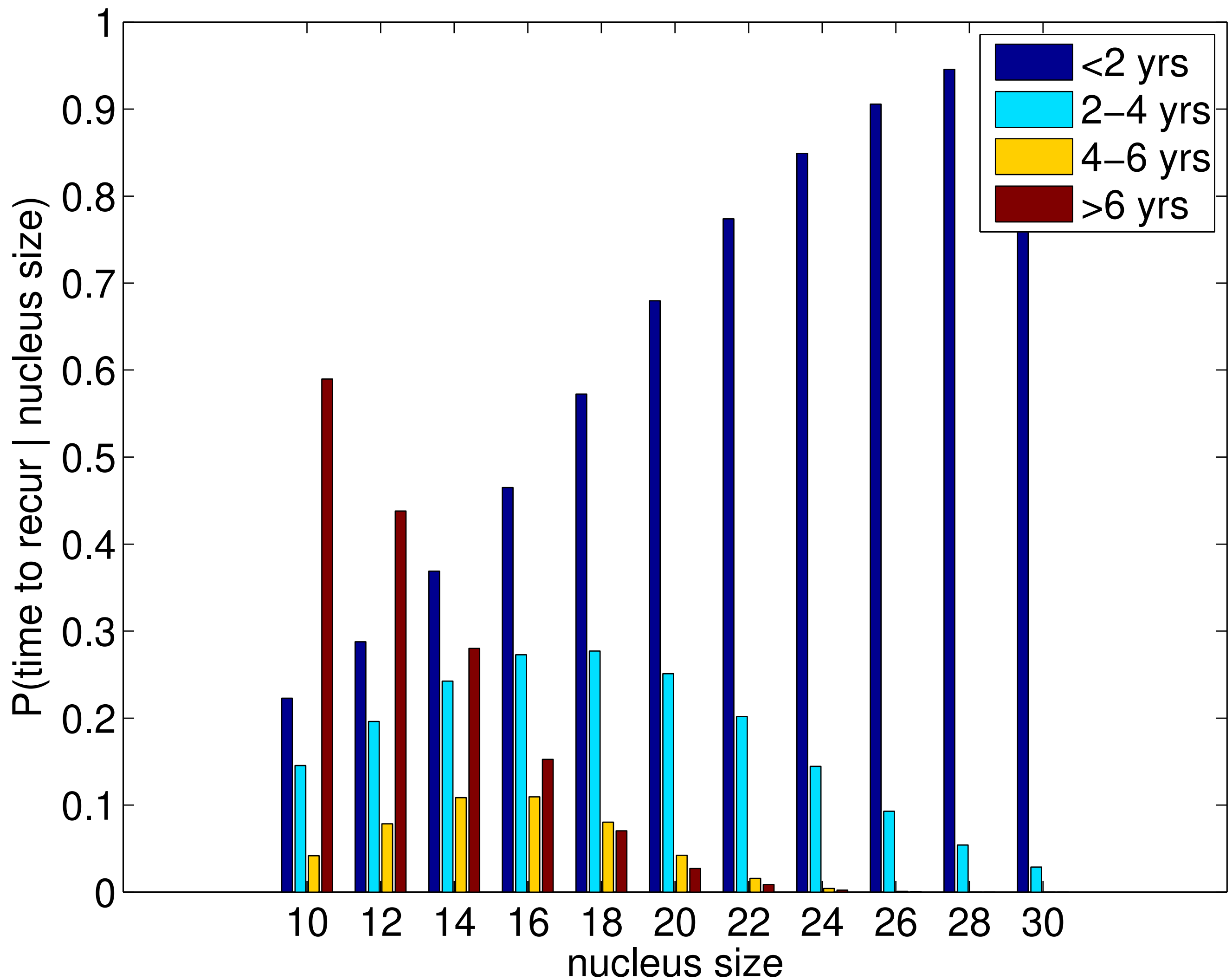
- If we assume the class-conditionals, $P(x|y)$, are Gaussian, the maximum-likelihood parameter estimates are:

Class:	< 2 yrs	$2-4$ yrs	$4-6$ yrs	> 6 yrs
Nucleus size mean:	19.3	18.0	16.7	14.8
Nucleus size std:	3.29	2.99	2.45	2.79

The class conditional densities



The posterior, $P(y|x)$



Summary of parametric density estimation

- To estimate continuous densities:
  - Assume a parametric form for the density (e.g., Gaussian)
  - Find the maximum-likelihood parameters for the data
  - We showed: univariate Gaussian, multivariate Gaussian, univariate exponential
  - Similar derivations work for many other distributions
- To estimate discrete distributions:
  - The maximum likelihood probabilities are just the empirical frequencies (We proved fairly directly, using Lagrange multipliers.)
  - Zero counts can be a problem, which "smoothing" of some sort can fix

What's next?

Returning to generative learning for classification:

- We now know how to estimate $P(y)$
- We know how to estimate $P(\mathbf{x}|y)$ when the distribution is discrete, univariate or multivariate Gaussian, or exponential.
- What if $\mathbf{x}$ contains a variety of features distributed in different ways? Some discrete, some continuous, some Gaussian, some exponential, some uniform, etc.?
- That is, how do we handle a joint conditional, $P(\mathbf{x}|y)$, when $\mathbf{x}$ may have many and/or different variables?

Motivating Example: Text classification

- A very important practical problem, occurring in many different applications: information retrieval, spam filtering, news filtering, building web directories etc.
- A simplified problem description:
  - *Given:* a collection of documents, classified as “interesting” or “not interesting” by people
  - *Goal:* learn a classifier that can look at the text of a new document and provide a label for it, without human intervention
- How do we represent the data (documents)?

A simple data representation

- Consider all the possible "significant" words that can occur in the documents (words in the English dictionary, proper names, abbreviations)
- Typically, words that appear in all documents (called **stopwords**) are not considered (prepositions, common verbs like "to be", "to do"...)
- In another preprocessing step, words are mapped to their root (process called **stemming**)
  E.g. learn, learned, learning are all represented by the root "learn"
- For each root, introduce a corresponding **binary feature**, specifying whether the word is present or not in the document.

Example

		a	0
		aardvark	0
		$\vdots$	$\vdots$
		fun	1
		funel	0
"Machine learning is fun"	$\Longrightarrow$	$\vdots$	$\vdots$
		learn	1
		$\vdots$	$\vdots$
		machine	1
		$\vdots$	$\vdots$
		zebra	0

What is special about this task?

- Lots of features! ≈ 100000 for any reasonable domain
- The feature vector is very sparse (a lot of 0 entries)
- It is difficult to get labeled data!
  This process is done by people, hence is very time consuming and tedious

Which of the classification algorithms studied so far apply?

Generative learning for document classification

- We can compute $P(y)$ by counting the number of interesting and uninteresting documents we have.
- How do we compute $P(\mathbf{x}|y)$?
- Assuming about $100{,}000$ words, and not too many documents, this is hopeless! There are $2^{100{,}000}$ possible $\mathbf{x}$ vectors!
  Most possible combinations of words will not appear in the data at all, and estimation of a full joint conditional distribution is impossible.
- Hence, we need to make some extra assumptions.

Naive Bayes assumption

- Assume the x_i are **conditionally independent given** y.
- In other words, assume that:

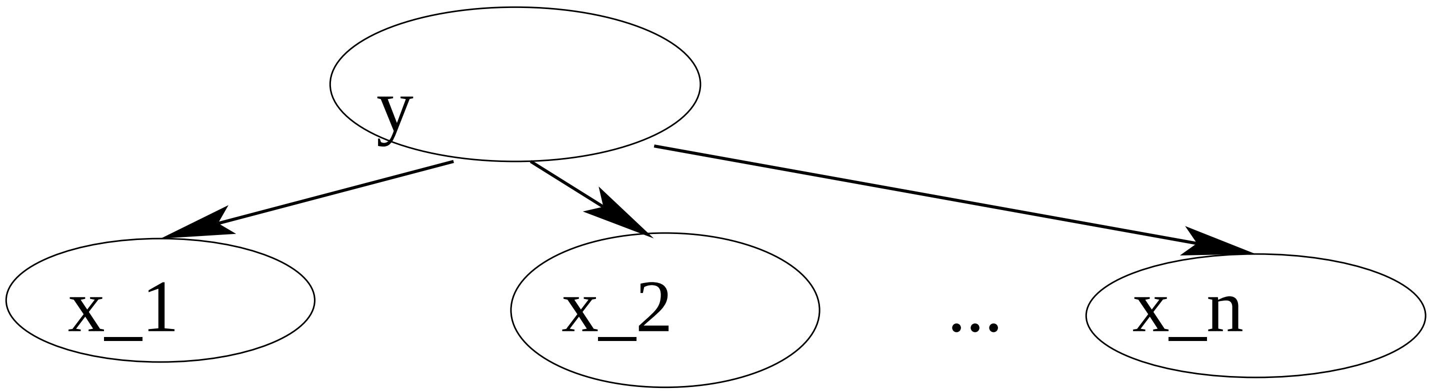
$$P(x_i|y, x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y), \forall i, j$$

- Then we have:

$$\begin{aligned} P(x_1, x_2, \ldots x_m|y) &= P(x_1|y)P(x_2|y, x_1) \cdots P(x_m|y, x_1, \ldots x_{m-1}) \\ &= P(x_1|y)P(x_2|y) \ldots P(x_m|y) \end{aligned}$$

- For binary features, instead of $O(2^n)$ numbers to describe a model, we only need $O(n)$!

A graphical representation of the naive Bayesian model



- The nodes corresponding to x_i are parameterized by $P(x_i|y)$.
- The node corresponding to y is parameterized by $P(y)$

Naive Bayes for binary features

- The parameters of the model are $\theta_{i,1} = P(x_i = 1|y = 1)$, $\theta_{i,0} = P(x_i = 1|y = 0)$, $\theta_1 = P(y = 1)$
- What is the decision surface?

$$\frac{P(y = 1|\mathbf{x})}{P(y = 0|\mathbf{x})} = \frac{P(y = 1)\prod_{i=1}^{n} P(x_i|y = 1)}{P(y = 0)\prod_{i=1}^{n} P(x_i|y = 0)}$$

- Using the log trick, we get:

$$\log\frac{P(y = 1|\mathbf{x})}{P(y = 0|\mathbf{x})} = \log\frac{P(y = 1)}{P(y = 0)} + \sum_{i=1}^{n}\log\frac{P(x_i|y = 1)}{P(x_i|y = 0)}$$

- Note that in the equation above, the x_i would be 1 or 0

Decision boundary of naive Bayes with binary features

$$
\begin{aligned}
\text{Let: } w_0 &= \log \frac{P(y=1)}{P(y=0)} \\
w_{i,1} &= \log \frac{P(x_i=1|y=1)}{P(x_i=1|y=0)} \\
w_{i,0} &= \log \frac{P(x_i=0|y=1)}{P(x_i=0|y=0)}
\end{aligned}
$$

We can re-write the decision boundary as:

$$
\begin{aligned}
\log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} &= w_0 + \sum_{i=1}^{n} (w_{i,1}x_i + w_{i,0}(1-x_i)) \\
&= w_0 + \sum_{i=1}^{n} w_{i,0} + \sum_{i=1}^{n}(w_{i,1} - w_{i,0})x_i
\end{aligned}
$$

This is a *linear decision boundary!*

Naive Bayes vs. linear discriminative classifiers

- A linear (or logistic) discriminative classifier would learn *directly* the decision boundary (e.g. using maximum likelihood)
- In this case, the direct learning problem would be dreadfully underconstrained!

- Naive Bayes learns $P(y)$, $P(x_i = 1|y = 1)$ and $P(x_i = 1|y = 0)$, for all i. Each of these estimations is well-determined by the data. (Or is it? More about that soon.)
- The coefficients that determine the linear decision boundary are *implicitly* defined, by $P(y)$ and $P(x_i|y)$.
- Moreover, the computational burden of fitting all the parameters is very low.

One problem

- In text classification, we want to estimate $P(x_i|y)$ for $y = 0$ and for $y = 1$.
- The max. likelihood estimate is (for $y = 1$, say) is:

$$
\begin{aligned}
P(x_i = 1|y = 1) &= \frac{\text{\# interesting docs with word } i}{\text{\# interesting docs}} \\
P(x_i = 0|y = 1) &= 1 - P(x_i = 1|y = 1) \\
P(x_i = 1|y = 1) &= \frac{\text{\# uninteresting docs with word } i}{\text{\# uninteresting docs}} \\
P(x_i = 0|y = 1) &= 1 - P(x_i = 1|y = 1)
\end{aligned}
$$

- What happens if some of these probabilities are zero?

One problem (II)

- Suppose word j never occurs in an interesting document, so $P(x_j = 1|y = 1) = 0$.
- Suppose we are asked to classify a new document that includes word j.

$$P(y = 1|\mathbf{x}) \propto \Pi_{i=1}^{n} P(x_i|y = 1)P(y = 1) = 0$$

- But are we *really sure* an interesting document cannot have word j?
- What if j never occurs in an interesting or an uninteresting document?
- Intuitively, a question arises as to how *confident* we are of our estimates.

Laplace smoothing

- Instead of the maximum likelihood estimate:

$$\theta_{i,1} = \frac{\#\ \text{interesting docs with word } i}{\#\ \text{interesting docs}}$$

use:

$$\theta_{i,1} = \frac{(\#\ \text{interesting docs with word } i) + 1}{(\#\ \text{interesting docs}) + 2}$$

- If a word appears in a lot of documents, this estimate is only slightly different from max. likelihood.
- If it appears not at all, its likelihood in a class (interesting or uninteresting) goes down with the size of the class.

Variation: Multinomial representation

- Previously, we had binary features for word presence/absence.
- A different (perhaps better) representation would take into account how many times a word occurs in a document.
- In this case, $P(x_i|y)$ is modeled as a multinomial distribution
- Estimation and prediction much as before.
- (Increased danger of overfitting, however.)

Example: 20 newsgroups

Given 1000 training documents from each group, learn to classify new documents according to which newsgroup they 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
- For text classification, Naive Bayes obtains results *comparable* to any other learning algorithm.

Naive Bayes for classification in general

- We're given supervised learning data $\langle \mathbf{x_i}, y_i \rangle$.
- We estimate $P(y)$, which is a discrete distribution, usually by max. likelihood.
- We estimate $P(x_i|y)$ for each value of y and each feature i.
  - If x_i is discrete, we use max. likelihood, possibly with smoothing.
  - If x_i is continuous, we choose a parametric form for density, and use max. likelihood to estimate its parameters.
- To classify a new instance, $\mathbf{x}$, we compute

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} = \frac{\Pi_{i=1}^{n} P(x_i|y)P(y)}{P(\mathbf{x})}$$

- Aside: It is safe to mix $P(x_i|y)$ that are probabilities with ones that are likelihoods.

Naive Bayes questions

- Q: Can we use Naive Bayes for regression instead of classification?

Naive Bayes questions

- Q: Can we use Naive Bayes for regression instead of classification?
- A: Not really. Estimating $P(y)$ is fine – this is just density estimation. But what about $P(\mathbf{x}|y)$? This requires estimating a distribution/density for $\mathbf{x}$ for every possibly real value y.

Naive Bayes questions (II)

- Q: What if the features are not conditionally independent? That is $P(\mathbf{x}|y) \neq \Pi_{i=1}^{n} P(x_i|y)$?

Naive Bayes questions (II)

- Q: What if the features are not conditionally independent? That is $P(\mathbf{x}|y) \neq \Pi_{i=1}^{n} P(x_i|y)$?
- A: The predictor can be overconfident in its estimation of $P(y|\mathbf{x})$.

Naive Bayes summary

Naive Bayes is a generative learning approach which assumes that the class conditional distribution is conditionally independent given y:
$P(\mathbf{x}|y) = \Pi_{i=1}^{n} P(x_i|y)$.

Pros:

- Parameter fitting is computationally efficient
- Good when feature vector is unstructured (i.e., features conditionally independent) and
- Applicable even when feature vector very large

Cons:

- Performance suffers when conditional independence doesn't hold
- May not give optimal classifier within all of same mathematical form

Not-so-naive Bayes

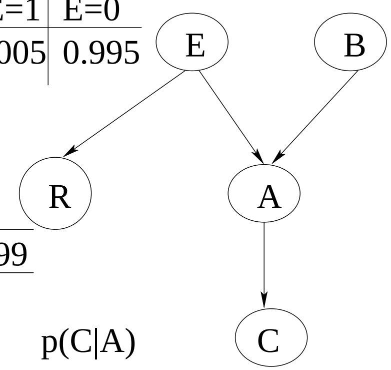
If the naive Bayes assumption does not hold, but there is information about which variables influence which others, a more complicated graph structure can be used:

p(E)

E=1	E=0
0.005	0.995

p(B)

B=1	B=0
0.01	0.99



p(R|E)

	R=1	R=0
E=0	0.0001	0.9999
E=1	0.65	0.35

p(A|B,E)

	A=1	A=0
B=0,E=0	0.001	0.999
B=0,E=1	0.3	0.7
B=1,E=0	0.8	0.2
B=1,E=1	0.95	0.05

p(C|A)

	C=1	C=0
A=0	0.05	0.95
A=1	0.7	0.3

This is a **Bayes network** (and uses similar learning algorithms; details in COMP-526)