CS 446 Machine Learning Fall 2016 Nov 01, 2016

Bayesian Learning

Professor: Dan Roth *Scribe: Ben Zhou, C. Cervantes*

Overview

- Bayesian Learning
- Naive Bayes
- Logistic Regression

1 Bayesian Learning

So far, we've discussed the paradigm of error driven learning, where our the core concept was to learn a hypothesis, make mistakes, correct the hypothesis, and repeat. Now we consider the probabilistic learning paradigm, which – rather than being driven by errors – is driven by hypothesis based on certain probabilistic assumptions.

1.1 Discriminitive vs. Generative Learning

Consider a distribution D over space $X \times Y$ (instance cross labels). We can think of the data generation process as $D(x)$ and the label process as $D(y|x)$.

$$D(x, y) = D(x)D(y|x)$$

If we know D, there is no learning; we just compute the most probably label. So far, we've been interested in finding a hypothesis that minimized the probability of mislabeling

$$h = \underset{h}{\operatorname{argmin}} E_{(x,y)\sim D} \left[h(x) \neq y\right]$$

This kind of learning is referred to as *discriminative learning.*

We are now interested in *generative learning.*

Consider the example of context-sensitive spelling correction

- I saw the girl **it** the park

where, here, we want to detect the misspelling and change $it \rightarrow in$.

According to the discriminative approach, we would model this problem as directly learning from the data how to make predictions.

1. Look at many examples
2. Discover regularities in the data
3. Use these to construct a prediction policy

In this example, the hypothesis may be *if "the" occurs after the target, choose "in"*. Approximating $h : X \to Y$, is estimating $P(Y|X)$.

According to the generative approach, we model the problem as that of generating correct sentences, where the goal is to learn a model of language and use this model to predict.

1. Learn a probability distribution over all sentences
2. Use the distribution to estimate which sentence is more likely; make a decision based on this

In this example, the model chooses the sentence that contains "in" because this sentence is more likely that the one that contains "it". This paradigm thus approximates $P(X, Y) = P(X|Y)P(Y)$

1.2 Probabilistic Learning

There are actually two different notions in probabilistic learning.

First, we could think about learning concept c that maps $X \to [0, 1]$ where $c(x)$ may be interpreted as the probability that the label 1 is assigned to x. The learning theory we learned before (linear classifiers, shattering) apply this notion.

We could instead think of learning in the Bayesian framework, where we are using probabilistic criterion in selecting a hypothesis. The hypothesis itself can be deterministic. It is the process – not the hypothesis – that we are learning.

1.3 Core Concepts

Goal: find the best hypothesis from some space H, given the observed data D.

We consider the best hypothesis to be the most probable hypothesis in H, but in order to determine that, we must assume a probability distribution over hypothesis class H. Further, we must also know something about the relationship between the observed data and the hypothesis.

Consider coin tosses. In order to determine whether the coin is fair or biased (or to what degree its biased), we must first have a notion of how fair coins behave (50% chance of heads) contrasted with how biased coins behave.

Definitions

- $P(h)$: the prior probability of a hypothesis h; this reflects the background knowledge (things we know before observing data), where if we have no prior information, $P(h)$ is the uniform distribution
- $P(D)$: the probability of seeing example D; this does not vary based on the hypothesis
- $P(D|h)$: the probability of seeing sample D, given that hypothesis h is the target
- $P(h|D)$: the posterior probability of h, or the probability of hypothesis h, given that D has been observed

Bayes Theorem
According to Bayes theorem, the posterior probability of a hypothesis h, given that we've seen data D, is equal to the probability of the data given the hypothesis times the probability of the hypothesis, divided by the probability of the data.

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Note that $P(h|D)$ increases with $P(h)$ since if the hypothesis is more likely, the hypothesis is more likely to be chosen.

$P(h|D)$ also increases with $P(D|h)$, which implies that the probability of observing this data given h is proportional to the probability of observing this h given D.

Note that $P(h|D)$ decreases with $P(D)$, since the probability of observing D goes up will decrease the probability that h is related to D.

Product Rule

$$P(A, B) = P(A|B)P(B) = P(B|A)P(A)$$

Joint probability, if A and B are independent

$$P(A, B) = P(A)P(B); P(A|B) = P(A); P(A|B, C) = P(A|C)$$

Sum Rule

$$P(A \vee B) = P(A) + P(B) - P(A, B)$$

Total Probability
If events $A_1, A_2 ... A_n$ are mutually exclusive ($A_i \cap A_j = \phi$), $\sum_i P(A_i) = 1$

$$P(B) = \sum P(B, A_i) = \sum_i P(B|A_i)P(A_i)$$

Total Conditional Probability
If events $A_1, A_2 ... A_n$ are mutually exclusive ($A_i \cap A_j = \phi$), $\sum_i P(A_i) = 1$

$$P(B|C) = \sum P(B, A_i|C) = \sum_i P(B|A_i, C)P(A_i|C)$$

1.4 Learning Scenario

In Bayesian Learning, a learner tries to find the most probably hypothesis h from a set of hypotheses H, given the observed data. This maximally probable hypothesis is called the maximum a posteriori hypothesis (MAP), and we use Bayes theorem to compute it. This is the basic concept of Bayesian Learning; everything else is just examples and implementations.

$$\begin{aligned} h_{map} &= \underset{h \in H}{\text{argmax}} P(h|D) \\ &= \underset{h \in H}{\text{argmax}} \frac{P(D|h)P(h)}{P(D)} \\ &= \underset{h \in H}{\text{argmax}} P(D|h)P(h) \end{aligned} \tag{1}$$

Note that because we're finding the argmax, we can drop the probability of the data, since this does not vary across hypotheses.

Note that in many cases we do not have a prior probability, so often we say that there is a uniform distribution over H

$$P(h_i) = P(h_j), \ \forall h_i, h_j \in H$$

Using this assumption, we can drop the probability of h from the equation above, producing the *maximum likelihood hypothesis*

$$h_{ml} = argmax_{h \in H} P(D|h)$$

1.5 Examples

Fair vs. Biased Coin
Consider a coin that is known to be either fair or biased 60% in favor of heads. We see a seies of trials, and want to determine which kind of coin it is, based on our observations.

Hypotheses: $h_1 : P(head) = 0.5$ and $h_2 : P(head) = 0.6$

Prior Distribution: $P(h_1) = 0.75$ and $P(h_2) = 0.25$

Note that the prior distribution is informed by our beliefs about the hypotheses, rather than anything intrinsic about the problem.

After the first trial, the coin comes up heads. Now $P(D|h_1) = 0.5$ and $P(D|h_2) = 0.6$. We then compute the probability of the data, and thus we need to sum the two possible conditions: that observation came from h_1 and h_2.

$$P(D) = P(D|h_1)P(h_1) + P(D|h_2)P(h_2) = 0.5 * 0.75 + 0.6 * 0.25 = 0.525$$

Then we can compute

$$P(h_1|D) = \frac{P(D|h_1)P(h_1)}{P(D)} = \frac{0.5 * 0.75}{0.525} = 0.714$$

$$P(h_2|D) = \frac{P(D|h_2)P(h_2)}{P(D)} = \frac{0.6 * 0.25}{0.525} = 0.286$$

Note that we need not compute $P(D)$ since our goal is to compare $P(h_1|D)$ and $P(h_2|D)$, rather than knowing their exact values.

Given these results, after the first toss it is more likely that the coin is fair. If we had used the maximum likelihood approach (and thus thought both hypotheses were equally likely), however, we would have believed the coin was biased.

Regardless, if after trying 100 tosses, observing 70 heads will indicate that the coin is biased.

Language Model
Assume a language that contains five characters and a space, as in {A, B, C, D, E, _}. These characters are generated according to $P(A) = p_1$, $P(B) = p_2$, $P(C) = p_3$, $P(D) = p_4$, $P(E) = p_5$, $P(\_) = p_6$ and $\sum_i p_i = 1$

Consider learning the values for each p_i; that is, given a family of distributions and observed data, determine which distribution generated the data.

Let's assume a generative model of independent characters (fixed k)

$$P(U) = P(x_1, x_2...x_k) = \prod_{i=1}^{k} P(x_i|x_{i+1}...x_k) = \prod_{i=1}^{k} P(x_i)$$

If we consider two strings (U and V), the goal is to determine which is more likely. In the Bayesian framework, we can compute the probability of each string and choose the most likely. Learning, then, becomes the task of finding the parameters of a known model family.

It is important to note that in discriminative learning we frame problems as making one decision – predicting y, given x – but generative learning can be somewhat more expressive than that. In this example, we not only can predict whether U or V is more likely, given the training data, but we've also learned the distribution that governs the characters, potentially enabling us to make more decisions.

Biased Coin
Assume a coin with bias $(p, 1-p)$ for heads and tails, respectively. Consider m tosses with k heads.

To find p, we use the maximum likelihood estimate. Given what we know, the probability of data observed is $P(D|p) = p^k(1-p)^{m-k}$

The log likelihood is given by $L(p) = \log P(D|p) = k\log(p) + (m-k)\log(1-p)$

To maximize this likelihood, we can use the derivative

$$\frac{\partial L(p)}{\partial p} = \frac{k}{p} - \frac{m-k}{1-p} = 0 \Rightarrow p = \frac{k}{m}$$

When sample sizes are very small – like when two out of two toesses are heads – $p = 1$, which is unlikely. To account for this, we must smooth our distributions, and one way to do so is by assuming a prior.

1.6 Probability Distributions

Bernoulli Distribution
Random Variable X takes values $\{0, 1\}$ so that $P(X = 1) = p = 1 - P(X = 0)$.

Binomial Distribution
Random Variable X takes values $\{1, 2, ...n\}$ representing the number of successes $(X = 1)$ in n Bernoulli trails.

$$P(X = k) = f(n, p, k) = C_n^k p^k (1-p)^{n-k}$$

Note that if $X \sim Binom(n, p)$ and $Y \sim Bernoulli(p)$,

$$X = \sum_{i=1}^{n} Y$$

thus, you can consider a binomial distribution as the sum of Bernoilli distributions.

Categorical Distribution
Random Variable X takes on values in $\{1, 2...k\}$, so that $P(X = i) = p_i$ and $\sum_1^k p_i = 1$. (Think of a dice).

Multinomial Distribution
Let the random variables $X_i, (i = 1, 2, ...k)$ indicates the number of times outcome i was observed over n trials.

The vector $X = (X_1..X_k)$ follows a multinomial distribution (n, p) where $p = (p_1, p_2...p_k)$ and $\sum_1^k p_i = 1$.

$$f(x_1, x_2...x_k, n, p) = P(X_1 = x_1...X_k = x_k) = \frac{n!}{x_1!...x_k!} p_1^{x_1}...p_k^{x_k}$$

where $\sum_{i=1}^k x_i = n$. (Thinks of n tosses of a k sided dice).

1.7 Additional Examples

Multinomial Bag of Words
Consider a collection of documents written in a three word language $\{a, b, c\}$, where each document has exactly n words.

We are given labels on each document (good, bad), and our goal – given $\{D_1, D-2...D_m\}$ – is to predict the label y for an unseen document.

In the discriminative setting, we know how to solve this problem: train a model over features a, b, c to predict a label y. In the generative setting, we're going to assume a probability distribution that generates these documents and we will then try to learn that distribution, enabling us to perform various tasks (including predicting the label).

Let's assume a multinomial distribution. Let a_i, b_i, c_i be the number of a, b, c's appearances in document i, respectively, where $a_i + b_i + c_i = n$.

In this generative model

$$P(D_i|y = 1) = \frac{n!}{a_i!b_i!c_i!} \alpha_1^{a_i} \beta_1^{b_i} \gamma_1^{c_i}$$

where $\alpha_1, \beta_1, \gamma_1$ are the probability that a, b, c appears in a good document, respectively, and

$$P(D_i|y = 0) = \frac{n!}{a_i!b_i!c_i!} \alpha_0^{a_i} \beta_0^{b_i} \gamma_0^{c_i}$$

where $\alpha_0, \beta_0, \gamma_0$ are the probability that a, b, c appears in a bad document.

Now, we only need a_i, b_i, c_i to learn the parameters α, β, γ, where it's important to note that $\alpha_0 + \beta_0 + \gamma_0 = \alpha_1 + \beta_1 + \gamma_1 = 1$.

We now want to determine $P(y|D)$; that is, given a document D, determine if it is good or bad. This can be computed through the Bayes rule, and to do so we must first derive the most likely value of the parameters defined above by maximizing the log likelihood of the observed data.

$$PD = \prod_i P(y_i, D_i) = \prod_i P(D_i|y_i)P(y_i)$$

We denote $P(y_1) = \eta$, the probability that an example is "good". Thus

$$PD = \prod_i [(\frac{\eta n!}{a_i!b_i!c_i!}\alpha_1^{a_i}\beta_1^{b_i}\gamma_1^{c_i})^{y_i} \cdot (\frac{(1-\eta)n!}{a_i!b_i!c_i!}\alpha_0^{a_i}\beta_0^{b_i}\gamma_0^{c_i})^{1-y_i}]$$

Note that the above formulation is an important trick to writing down the joint probability without knowing the outcome of the experiment; depending on the label (because of the exponent), different parameters are included in the final product.

$\log(PD) = \sum_i y_i[\log(\eta) + C + a_i \log(\alpha_1) + b_i \log(\beta_1) + c_i \log(\gamma_1)] + (1-y_i)[\log(1-\eta) + C + a_i \log(\alpha_0) + b_i \log(\beta_0) + c_i \log(\gamma_0)]$

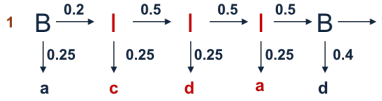
$$\frac{d \log(PD)}{d\eta} = \sum_i [\frac{y_i}{\eta} - \frac{1-y_i}{1-\eta}] = 0 \Rightarrow \eta = \sum_i \frac{y_i}{m}$$

Hidden Markov Models
Consider two states (two different distributions) that govern character generation. In this example there are five characters – $x = a, b, c, d, e$ – and two states – $s = B, I$. You can think the two states as chunking a sentence into phrases, B is the beginning of each phrase and I is inside a phrase.

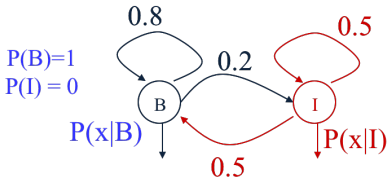
We then generate characters according to

Initial State: $P(B) = 1, P(I) = 0$
State transition: $P(B \to B) = 0.8$
$P(B \to I) = 0.2$
$P(I \to B) = 0.5$
$P(I \to I) = 0.5$
Output: $P(a|B) = 0.25$
$P(b|B) = 0.1 ... P(a|I) = 0.25$
$P(b|I) = 0 ...$

These state transitions are shown in the diagrams below.



(a) State transition



(b) State transition

Using the definitions above we can do the same thing. Given data $\{x_1, x_2 ... s_1, s_2\}^n$, we want to find the most likely parameters of the model $P(x_i|s_i), P(s_{i+1}|s_i), P(s_i)$.

Thus, given an unlabeled example $x = (x_1, x_2...x_m)$, we can use Bayes rule to predict the label $l = (s_1, s_2...s_m)$, where $l^* = \underset{l}{\operatorname{argmax}} P(l|x) = \underset{l}{\operatorname{argmax}} \frac{P(x|l)P(l)}{P(x)}$.

Note that we still have a computational issue here, since there are 2^m possible values of l. The argmax thus would take time to compute, in the simplest way. Note that in this setup, though, no states were hidden, making learning an easy problem. When the states are hidden, which is typical with HMMs, learning is more difficult.

1.8 Bayes Optimal Classifier

Using the Bayesian framework, one remaining question is the exact definition of our hypothesis space H. As noted before, Bayesian learning requires making assumptions about the distribution that generates our data before we can learn the most likely $h \in H$.

Recall the MAP function for learning a good hypothesis

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(h|D) = \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h)$$

For example, assume three hypotheses $P(h_1|D) = 0.4$, $P(h_2|D) = 0.3$, $P(h_3|D) = 0.3$.. Therefore, $h_{MAP} = h_1$. Given a new instance, however, $h_1(x) = 1, h_2(x) = 0, h_3(x) = 0$, because we've already committed to h_1.

In this case, however, $P(f(x) = 1) = 0.4$ is less than $P(f(x) = 0) = 0.6$, so our new example is misclassified because of how we've committed to a single hypothesis.

We can account for this by combining the predictions from all hypotheses – weighted by their posterior probabilities – to determine the most probable classification.

Let V be a set of possible classifications.

$$P(v_j|D) = \sum_{h_j \in H} P(v_j|h_i, D)P(h_i|D) = \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

Bayes Optimal Classification:

$$v = argmax_{v_j \in V} P(v_j|D) = argmax_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

In the example above

$$P(1|D) = 1 \cdot 0.4 + 0 \cdot 0.3 + 0 \cdot 0.3 = 0.4$$

$$P(0|D) = 0 \cdot 0.4 + 1 \cdot 0.3 + 1 \cdot 0.3 = 0.4$$

The optimal prediction is 0.

1.9 Bayesian Classifier

Instance space: $x \in X$, where $x = (x_1, x_2...x_n)$, $x_j \in \{0, 1\}$

Label space: value $v \in V$

Given x, we want to predict most probable value in V

$$\begin{aligned} v_{MAP} &= \underset{v_j \in V}{\text{argmax}} P(v_j|x) \\ &= \underset{v_j \in V}{\text{argmax}} P(v_j|x_1, x_2, ...x_n) \\ &= \underset{v_j \in V}{\text{argmax}} \frac{P(x_1, x_2...x_n|v_j)P(v_j)}{P(x_1, x_2...x_n} \\ &= \underset{v_j \in V}{\text{argmax}} P(x_1, x_2, ...x_n|v_j)P(v_j) \end{aligned} \tag{2}$$

While $P(v)$ can be easily estimated by counting the number of times v appears in the training data, it is not feasible to estimate $P(x_1, x_2...x_n|v)$; with even a relatively small n, it is unlikely that each combination of features is seen a sufficient number of times to estimate probabilities by counting.

Therefore, we must make certain assumptions to estimate this probability. The Naive Bayes classifier does so by assuming all features are conditionally independent.

2 Naive Bayes

Recall that – in the general Bayesian classifier – we must compute $P(x_1, x_2...x_n|v_j)$.

$$\begin{aligned} P(x_1, x_2...x_n|v_j) &= P(x_1|x_2, ...x_n, v_j)P(x_2...x_n|v_j) \\ &= P(x_1|x_2...x_n, v_j)P(x_2|x_3...x_n, v_j)P(x_3...x_n|v_j) \\ &= ... \\ &= P(x_1|x_2...x_n, v_j)P(x_2|x_3...x_n, v_j)...P(x_n|v_j) \end{aligned} \tag{3}$$

Naive Bayes assumes that each feature value is conditionally independent, given a target value. This can be written as

$$\prod_{i=1}^{n} P(x_i|v_j)$$

Using this method, we now must learn $n|V| + |V|$ parameters. Doing so means we learn without search, and thus computing the hypothesis directly.

2.1 Conditional Independence

Naive Bayes assumes that feature values are conditionally independent given the target value, but does not require that they are independent.

Note that independence does not imply conditional independence. Consider the Boolean features x and y. We define $l = f(x, y) = x \vee y$ over the distribution $P(x = 0) = P(x = 1) = \frac{1}{2}$ and $P(y = 0) = P(y = 1) = \frac{1}{2}$. The distribution is defined so that x and y are independent: $P(x, y) = P(x)P(y)$.

But, given that $l = 0$, $P(x = 1|l = 0) = P(y = 1|l = 0) = \frac{1}{3}$ while $P(x = 1, y = 1|l = 0) = 0$. so x and y are not conditionally independent.

Note also that conditional independence does not imply independence. Assume

$$l = 0 : P(x = 1|l = 0) = 1, P(y = 1|l = 0) = 0$$

$$l = 1 : P(x = 1|l = 1) = 0, P(y = 1|l = 1) = 1$$

$$P(l = 0) = P(l = 1) = \frac{1}{2}.$$

Given the value of l, x and y are independent, but x and y are not independent.

2.2 Tennis Example

Consider the Naive Bayes classifier defined by

$$V_{NB} = argmax_{v_j \in V} P(v_j) \prod_i P(x_i|v_j)$$

Let's consider the tennis example, from the decision trees lecture

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Figure 2: Final Hypothesis

We must compute $P(x|v)$ and $P(v)$, i.e.

$P(playTennis = yes), P(playTennies = no)$

$P(outlook = ...|playTennis = ...)$ (6 numbers)

$P(temp = ...|playTennis = ...)$ (6 numbers)

$P(humidity = ...|playTennis = ...)$ (4 numbers)

$P(wind = ...|playTennis = ...)$ (4 numbers)

In total, we must compute 22 numbers. Given a specific example, though, we will know the values for x_i and thus will no longer need to compute all 22. We only need the relevant ones.

Now given an example of $(outlook = sunny, temp = cool, humidity = high, wind = strong)$, you want to compute the probability of each v.

$P(playTennis = yes) = 0.64, P(playTennis = no) = 0.36$

$P(outlook = sunny|yes) = \frac{2}{9}, P(outlook = sunny|no) = \frac{3}{5}$

...

$P(wind = strong|yes) = \frac{3}{9}, P(wind = strong|no) = \frac{3}{5}$

$P(yes, ...) = 0.0053, P(no, ...) = 0.0206$

$P(no|instance) = \frac{0.0206}{0.0206+0..0053} = 0.795$

There is one problem with this setup, however. If one of the $P(x|v) = 0$, e.g. $(outlook = OC)$, then the probability to predict no is zero, which is probably wrong. To account for unseen events, our probability estimates must be more robust.

2.3 Robust Estimating Probabilities

Consider predicting the most likely document label v, as in

$$v_{NB} = argmax_{v \in \{like, dislike\}} P(v) \prod_i P(x_i = word_i|v)$$

If we make the binomial assumption, the probability of a particular word x_k given label v is given by

$$P(x_k|v) = \frac{\#\ v\ documents\ containing\ x_k}{\#\ v\ documents} = \frac{n_k}{n}$$

In this setting, data sparsity becomes a serious problem. If n is small, the estimate will be inaccurate. If n_k is 0 (if, for example, x_k is only seen with v in the test data) we will never predict v at all.

To address this sparsity problem, we use *smoothing*. In smoothing, the goal is to push probabilities away from 0. There are many ways to accomplish this, and one of the simplest, Laplace smoothing, is given by

$$P(x_k|v) = \frac{n_k + mp}{n + m}$$

where n_k is the number of documents with label v in which word x_k appears, n is the number of documents with label v, p is a prior estimate of v, and m is the number of labels.

Laplace Rule
Specifically for the Boolean case, we can assume $p = \frac{1}{2}$ and $m = 2$, which gives us

$$P(x_k|v) = \frac{n_k + 1}{n + 2}$$

2.4 Two Classes

Before, we thought of Naive Bayes as a predictor, rather than a classifier. We can rewrite Naive Bayes for use in two-class classification ($v \in \{0, 1\}$) by saying it predicts $v = 1$ *iff*

$$\frac{P(v_j = 1) \cdot \prod_{i=1}^{n} P(x_i|v_j = 1)}{P(v_j = 0) \cdot \prod_{i=1}^{n} P(x_i|v_j = 0)} > 1$$

Let's denote $p_i = P(x_i = 1|v = 1)$, $q_i = P(x_i = 1|v = 0)$. Now recall that when $x_i = 1$ we want to use p_i and q_i, but if $x_i = 0$ we want to use $1 - p_i$ and $1 - q_i$. Since $x_i \in \{0, 1\}$, we can write a single equation that uses the correct probability, given the value of x_i, as

$$\frac{P(v_j = 1) \cdot \prod_{i=1}^{n} p_i^{x_i}(1 - p_i)^{1-x_i}}{P(v_j = 0) \cdot \prod_{i=1}^{n} q_i^{x_i}(1 - q_i)^{1-x_i}} > 1$$

Due to the exponents, we p_i is used when $x_i = 1$, and $1-p_i$ is used when $x_i = 0$. We can further rewrite this as below

$$\frac{P(v_j = 1) \cdot \prod_{i=1}^{n} (1 - p_i)(\frac{p_i}{1-p_i})^{x_i}}{P(v_j = 0) \cdot \prod_{i=1}^{n} (1 - q_i)(\frac{q_i}{1-q_i})^{x_i}} > 1$$

If we now take the log of both sides, the product is transformed into a sum.

$$\log \frac{P(v_j = 1)}{P(v_j = 0)} + \sum_i \log \frac{1 - p_i}{1 - q_i} + \sum_i (\log \frac{p_i}{1 - p_i} - \log \frac{q_i}{1 - q_i})x_i > 0$$

In effect, we are multiplying some coefficient with x and we add another term that is irrelevant to x. This is just a linear separator:

$$w_i = \log \frac{p_i}{q_i} \frac{1-p_i}{1-q_i}$$

If $p_i = q_i$, then $w_i = 0$ and the feature is irrelevant.

Thus, in the cases of two classes we have that

$$\log \frac{P(v_j = 1|x)}{P(v_j = 0|x)} = \sum_i w_i x_i - b$$

Thus, Naive Bayes works because it operates as a linear separator. Moreover, since we know $P(v_j = 1|x) = 1 - P(v_j = 0|x)$, we can rewrite it as

$$P(v_j = 1|x) = \frac{1}{exp(-\sum_i w_i x_i + b)}$$

which is simply the logistic function.

3 Logistic Regression

Consider the logistic regression model

$$P(y = 1/-1|x, w) = \frac{1}{1 + exp(-y(w^T x + b)}$$

This is exactly the same model as we derived for Naive Bayes, but here we do not assume independence. Rather than computing w based on observed counts, we directly find the best w that satisfies the above.

As a result, training will be more difficult, but the weight vector will be more expressive. Since Naive Bayes computes its weight vector given the $\frac{n_k}{n}$ fractions, not all the coefficients are expressible. Thus, while Naive Bayes cannot represent all linear threshold functions, it converges much faster because of the trivial way w is computed.

In *logistic regression*[1], our goal is to find a (w, b) that maximizes the log likelihood of the data $(\{x_1, x_2...x_m\})$, which can be rewritten as minimizing the negative log likelihood

$$min_{w,b} \sum_1^m \log P(y = 1/-1|x, w) = min_{w,b} \sum_1^m \log[1 + exp(-y_i(w^T x_i + b)]$$

[1]Logistic regression is sometimes called Maximum Entropy – particularly in the NLP community – but the term has fallen out of favor. The term comes from the fact that the resulting distribution is the one that has the largest entropy from among all those that activate the same features.

To get good generalization, we're going to modify our objective function by adding a regularization term (the L_2 norm of w); this is called regularized logistic regression

$$min_w f(w) = \frac{1}{2} w^T w + C \sum_1^m \log[1 + exp(-y_i(w^T x_i))]$$

where C is a user selected parameter that balances the two terms, $\frac{1}{2} w^T w$ is the regularization term, and the sum is the empirical loss.

Since the second term is the loss function, regularized logistic regression can be related to other learning methods, like SVM.

L_1 SVM solves the following optimization problem (hinge loss):

$$min_w f_1(w) = \frac{1}{2} w^T w + C \sum_1^m max(0, 1 - y_i(w^T x_i))$$

L_2 SVM solves the following optimization problem:

$$min_w f_2(w) = \frac{1}{2} w^T w + C \sum_1^m (max(0, 1 - y_i w^T x_i))^2$$