

Intro. to machine learning (CSI 5325)

Lecture 13: Bayesian learning

Greg Hamerly

Some content from Tom Mitchell.

- 1 Learning to predict probabilities
- 2 Minimum description length principle
- 3 Bayes optimal classifier
- 4 Naive Bayes learner

Learning to predict probabilities

Consider predicting survival probability from patient data

- Examples $\langle x_i, d_i \rangle$, where $d_i \in \{0, 1\}$
- Want to train a neural network to output a *probability* given x_i (not a 0 or 1)

Set up

$$P(x_i, d_i | h) = P(d_i | x_i, h) P(x_i | h) = P(d_i | x_i, h) P(x_i)$$

by standard probability rules, and assuming x and h independent.
Further,

$$P(d_i | x_i, h) = h(x_i)^{d_i} (1 - h(x_i))^{1-d_i}$$

Learning to predict probabilities

In this case we can show that

$$\begin{aligned}
 h_{ML} &= \operatorname{argmax}_{h \in H} \prod_{i=1}^m P(d_i | x_i, h) P(x_i) \\
 &= \operatorname{argmax}_{h \in H} \prod_{i=1}^m P(d_i | x_i, h) \\
 &= \operatorname{argmax}_{h \in H} \prod_{i=1}^m h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} \\
 &= \operatorname{argmax}_{h \in H} \sum_{i=1}^m d_i \ln h(x_i) + (1 - d_i) \ln(1 - h(x_i))
 \end{aligned}$$

This 'error' function is known as 'cross-entropy.'

Training a neural network to predict probabilities

$$h_{ML} = \operatorname{argmax}_{h \in H} \sum_{i=1}^m d_i \ln h(x_i) + (1 - d_i) \ln(1 - h(x_i))$$

Weight update rule for a sigmoid unit: $w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$, where

$$\Delta w_{jk} = \eta \sum_{i=1}^m (d_i - h(x_i)) x_{ijk}$$

Minimum Description Length Principle

Occam's razor: prefer the shortest hypothesis

MDL: prefer the hypothesis h that minimizes

$$h_{MDL} = \operatorname{argmin}_{h \in H} L_{C_1}(h) + L_{C_2}(D|h)$$

where $L_C(x)$ is the description length of x under encoding C

Minimum Description Length Principle

Example: H = decision trees, D = training data labels

- $L_{C_1}(h)$ is # bits to describe tree h
- $L_{C_2}(D|h)$ is # bits to describe D given h
 - Note $L_{C_2}(D|h) = 0$ if examples classified perfectly by h .
 - Only need to explicitly encode exceptions to the hypothesis.
- Hence h_{MDL} trades off tree size for training errors

Minimum Description Length principle

Re-work the MAP hypothesis definition using log:

$$\begin{aligned}h_{MAP} &= \operatorname{argmax}_{h \in H} P(D|h)P(h) \\ &= \operatorname{argmax}_{h \in H} (\log_2 P(D|h) + \log_2 P(h)) \\ &= \operatorname{argmin}_{h \in H} (-\log_2 P(D|h) - \log_2 P(h))\end{aligned}$$

Minimum Description Length principle

$$h_{MAP} = \underset{h \in H}{\operatorname{argmin}} (-\log_2 P(D|h) - \log_2 P(h))$$

Interesting fact from information theory:

The optimal (shortest expected coding length) code for an event with probability p is $-\log_2 p$ bits.

So interpret h_{MAP} :

- $-\log_2 P(h)$ is length of h under optimal code
- $-\log_2 P(D|h)$ is length of D given h under optimal code

∴ prefer the hypothesis that minimizes

$$\text{length}(h) + \text{length}(\text{misclassifications})$$

Bayes Optimal Classifier

The *Bayes optimal classifier* is a classifier that:

- uses every possible hypothesis in H
- uses the probability of each hypothesis

Bayes optimal classification:

- The probability of class $v_j \in V$ given data D is:

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

- Thus, the most likely class is

$$v_{BayesOpt} = \operatorname{argmax}_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

Bayes Optimal Classifier

Example:

$$P(h_1|D) = .4, \quad P(-|h_1) = 0, \quad P(+|h_1) = 1$$

$$P(h_2|D) = .3, \quad P(-|h_2) = 1, \quad P(+|h_2) = 0$$

$$P(h_3|D) = .3, \quad P(-|h_3) = 1, \quad P(+|h_3) = 0$$

therefore,

$$\sum_{h_i \in H} P(+|h_i)P(h_i|D) = .4$$

$$\sum_{h_i \in H} P(-|h_i)P(h_i|D) = .6$$

and

$$\operatorname{argmax}_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = -$$

Bayes error rate and base error rate

The classification error rate of the Bayes optimal classifier is called the **Bayes error rate** (or just *Bayes rate*)

Which sounds similar to the **base error rate**, but is not...

The naïve strategy of always choosing the most frequent class is the *base error rate* (or just *base rate*).

Bayes error rate and base error rate

Both error rates are of interest! Why?

Bayes rate: gives the best possible error rate of any classifier

- over the distribution of all examples
- for the chosen hypothesis space

If we know the Bayes rate, we may know when to stop training!

The base rate gives the best error for the simplest strategy.

- If a classifier is not doing better than the base rate, then it may not be worth using!
- Of course, consider the cost of making mistakes. For example, a rare cancer that occurs in only .2% of people.

Use the base rate in your experimental analysis!

Gibbs Classifier

Bayes optimal classifier provides best result, but can be expensive if there are many hypotheses.

Gibbs algorithm:

- 1 Choose one hypothesis at random, according to $P(h|D)$
- 2 Use this to classify new instance

Surprising fact: Assume target concepts are drawn at random from H according to priors on H . Then:

$$E[\text{error}_{\text{Gibbs}}] \leq 2E[\text{error}_{\text{BayesOptimal}}]$$

Suppose correct, uniform prior distribution over H , then

- Pick any hypothesis from VS , with uniform probability
- Its expected error no worse than twice Bayes optimal

Naive Bayes Classifier

Along with decision trees, neural networks, nearest neighbor, one of the most practical learning methods.

When to use

- Moderate or large training set available
- Attributes that describe instances are conditionally independent given classification

Successful applications:

- Predicting hard drive failure
- Classifying text documents

Naive Bayes Classifier

Assume target function $f : X \rightarrow V$, where each instance x described by attributes $\langle a_1, a_2 \dots a_n \rangle$.

Most probable value of $f(x)$ is:

$$\begin{aligned}v_{MAP} &= \operatorname{argmax}_{v_j \in V} P(v_j | a_1, a_2 \dots a_n) \\ &= \operatorname{argmax}_{v_j \in V} \frac{P(a_1, a_2 \dots a_n | v_j) P(v_j)}{P(a_1, a_2 \dots a_n)} \\ &= \operatorname{argmax}_{v_j \in V} P(a_1, a_2 \dots a_n | v_j) P(v_j)\end{aligned}$$

Naive Bayes assumption

$$v_{MAP} = \operatorname{argmax}_{v_j \in V} P(a_1, a_2 \dots a_n | v_j) P(v_j)$$

Naive Bayes assumption (what does this mean?):

$$P(a_1, a_2 \dots a_n | v_j) = \prod_i P(a_i | v_j)$$

which gives the **Naive Bayes classifier**:

$$v_{NB} = \operatorname{argmax}_{v_j \in V} P(v_j) \prod_i P(a_i | v_j)$$

Naive Bayes Algorithm

Naive_Bayes_Learn(*examples*)

For each target value v_j

$\hat{P}(v_j) \leftarrow$ estimate $P(v_j)$

For each attribute value a_i of each attribute a

$\hat{P}(a_i|v_j) \leftarrow$ estimate $P(a_i|v_j)$

Classify_New_Instance(x)

$$v_{NB} = \operatorname{argmax}_{v_j \in \mathcal{V}} \hat{P}(v_j) \prod_{a_i \in x} \hat{P}(a_i|v_j)$$

Naive Bayes: Example

Consider *PlayTennis* again, and new instance

$\langle \text{Outlook} = \text{sun}, \text{Temp} = \text{cool}, \text{Humid} = \text{high}, \text{Wind} = \text{strong} \rangle$

Want to compute:

$$v_{NB} = \operatorname{argmax}_{v_j \in V} P(v_j) \prod_i P(a_i | v_j)$$

$$P(y) P(\text{sun}|y) P(\text{cool}|y) P(\text{high}|y) P(\text{strong}|y) = .005$$

$$P(n) P(\text{sun}|n) P(\text{cool}|n) P(\text{high}|n) P(\text{strong}|n) = .021$$

$$\rightarrow v_{NB} = n$$

Naive Bayes: Subtleties

Conditional independence assumption is often violated

$$P(a_1, a_2 \dots a_n | v_j) = \prod_i P(a_i | v_j)$$

- ...but it works surprisingly well anyway. Note don't need estimated posteriors $\hat{P}(v_j|x)$ to be correct; need only that

$$\operatorname{argmax}_{v_j \in V} \hat{P}(v_j) \prod_i \hat{P}(a_i | v_j) = \operatorname{argmax}_{v_j \in V} P(v_j) P(a_1 \dots, a_n | v_j)$$

- see [Domingos & Pazzani, 1996] for analysis
- Naive Bayes posteriors often unrealistically close to 1 or 0

Naive Bayes: Subtleties

What if none of the training instances with target value v_j have attribute value a_i ? Then

$$\hat{P}(a_i|v_j) = 0, \text{ and } \hat{P}(v_j) \prod_i \hat{P}(a_i|v_j) = 0$$

Typical solution is Bayesian estimate for $\hat{P}(a_i|v_j)$

$$\hat{P}(a_i|v_j) \leftarrow \frac{n_c + mp}{n + m}$$

where

- n is number of training examples for which $v = v_j$,
- n_c number of examples for which $v = v_j$ and $a = a_i$
- p is prior estimate for $\hat{P}(a_i|v_j)$
- m is weight given to prior (i.e. number of “virtual” examples)