Artificial Intelligence Programming

Bayesian Learning

Cindi Thompson

Department of Computer Science

University of San Francisco

Learning and Classification

- We've spent a fair amount of time so far thinking about classification problems and algorithms.
  - Decision trees, K-nearest-neighbor, ZeroR
- So far, all of the classifiers we've learned have deterministically predicted a class based on a set of features.
  - $overcast \rightarrow playTennis$
  - $crowded \wedge FridayOrSaturday \rightarrow WillWait$
- But how can we handle uncertainty?
- Probability to the rescue!

Reminders

Axioms of Probability

- All probabilities are between 0 and 1.
  $0 \leq P(a) \leq 1$
- Propositions that are necessarily true have probability 1.
  $P(true) = 1$
- Propositions that are unsatisfiable have probability 0.
  $P(false) = 0$
- The probability of $(a \vee b)$ is
  $P(a, b) = P(a) + P(b) - P(a \wedge b)$

More Reminders

- Conditional probability: $P(a|b) = \frac{P(a,b)}{P(b)}$
- When a variable has no effect on another, we say they are independent: $P(rain|cloudy, monday) = P(rain|cloudy, tuesday) = \ldots P(rain|cloudy)$
- Bayes Theorem: $P(b|a) = \frac{P(a|b)P(b)}{P(a)}$

Incorporating Uncertainty

- So returning to classification: we predicted a class based on some features.
- Typically, this is the most likely classification given the data.
- What if we want to know how likely a hypothesis is?
- We can apply our knowledge of probability to learn a hypothesis.

Bayes' Theorem

- Recall the definition of Bayes' Theorem
- $P(b|a) = \frac{P(a|b)P(b)}{P(a)}$
- Let's rewrite this a bit.
- Let D be the data we've seen so far.
- Let h be a possible hypothesis
- $P(h|D) = \frac{P(D|h)P(h)}{P(D)}$

Example

- Imagine that we have a large bag of candy. We want to know the ratio of cherry to lime in the bag.
- We start with 5 hypotheses:
  1. h_1: 100% cherry
  2. h_2 75% cherry, 25% lime.
  3. h_3 50% cherry, 50% lime
  4. h_4 25% cherry, 75% lime
  5. h_5 100% lime
- Our agent repeatedly draws pieces of candy.
- We want it to correctly pick the type of the next piece of candy.

Example

- Let's assume our priors for the different hypotheses are:
- $(0.1, 0.2, 0.4, 0.2, 0.1)$
- Also, we assume that the observations are i.i.d.
  - This means that each choice is independent of the others. (order doesn't matter)
- In that case, we can multiply probabilities.
- $P(D|h_i) = \Pi_j P(d_j|h_i)$
- Suppose we draw 10 limes in a row. $P(D|h_3)$ is $(\frac{1}{2})^{10}$, since the probability of drawing a lime under h_3 is $\frac{1}{2}$.

Example

- How do the hypotheses change as data is observed?
- Initially, we start with the priors: $(0.1, 0.2, 0.4, 0.2, 0.1)$
- Then we draw a lime.
  - $P(h_1|lime) = \alpha P(lime|h_1)P(h_1) = 0.$
  - $P(h_2|lime) = \alpha P(lime|h_2)P(h_2) = \alpha\frac{1}{4} * 0.2 = \alpha 0.05.$
  - $P(h_3|lime) = \alpha P(lime|h_3)P(h_3) = \alpha\frac{1}{2} * 0.4 = \alpha 0.2$
  - $P(h_4|lime) = \alpha P(lime|h_4)P(h_4) = \alpha\frac{3}{4} * 0.2 = \alpha 0.15.$
  - $P(h_5|lime) = \alpha P(lime|h_5)P(h_5) = \alpha 1 * 0.1 = \alpha 0.1.$
  - $\alpha = 2.$

Example

- Then we draw a second lime.
  - $P(h_1|lime, lime) = \alpha P(lime, lime|h_1)P(h_1) = 0.$
  - $P(h_2|lime, lime) = \alpha P(lime, lime|h_2)P(h_2) = \alpha \frac{1}{4}\frac{1}{4} * 0.2 = \alpha 0.0125.$
  - $P(h_3|lime, lime) = \alpha P(lime, lime|h_3)P(h_3) = \alpha \frac{1}{2}\frac{1}{2} * 0.4 = \alpha 0.1$
  - $P(h_4|lime, lime) = \alpha P(lime, lime|h_4)P(h_4) = \alpha \frac{3}{4}\frac{3}{4} * 0.2 = \alpha 0.1125.$
  - $P(h_5|lime) = \alpha P(lime|h_5)P(h_5) = \alpha 1 * 0.1 = \alpha 0.1.$
  - $\alpha = 3.07.$

Bayesian Learning

- Eventually, the true hypothesis will dominate all others.
  - Caveat: assuming the data is noise-free, or noise is uniformly distributed.
- Notice that we can use Bayesian learning (in this case) either as a batch algorithm or as an incremental algorithm.
- We can always easily update our hypotheses to incorporate new evidence.
  - This depends on the assumption that our observations are independent.

MAP Hypothesis

- Often, we're not so interested in the particular probabilities for each hypothesis.
  - Strictly speaking, we did't really care what α was in the candy example
- Instead, we want to know: Which hypothesis is most likely, given the data?
  - Which classification is the most probable?
  - Is $PlayTennis$ or $\neg PlayTennis$ more likely?
- We call this the *maximum a posteriori hypothesis* (MAP hypothesis).
- In this case, we can ignore the denominator $(P(D))$ in Bayes' Theorem, since it will be the same for all h.
- $h_{MAP} = argmax_{h \in H} P(D|h)P(h)$

MAP Hypothesis

- Advantages:
  - Simpler calculation
  - No need to have a prior for $P(D)$

ML Hypothesis

- In some cases, we can simplify things even further.
- What are the priors $P(h)$ for each hypothesis?
- Without any other information, we'll often assume that they're equally possible.
  - Each has probability $\frac{1}{H}$
- In this case, we can just consider the conditional probability $P(D|h)$.
- We call the hypothesis that maximizes this conditional probability the *maximum likelihood* hypothesis.
- $h_{ML} = argmax_{h \in H} P(D|h)$

Learning bias

- What sort of bias does Bayesian Learning use?
- Typically, simpler hypotheses will have larger priors.
- More complex hypotheses will fit data more exactly (but there's many more of them).
  - Under these assumptions, h_{MAP} will be the simplest hypothesis that fits the data.
  - This is Occam's razor, again.
  - Think about the deterministic case, where $P(h_i|D)$ is either 1 or 0.

Bayesian Concept Learning

- Bayesian Learning involves estimating the likelihood of each hypothesis.
- In a more complex world where observations are not independent, this could be difficult.
- Our first cut at doing this might be a brute force approach:
  1. For each h in H, calculate $P(h|D) = \frac{P(D|h)P(h)}{P(D)}$
  2. From this, output the hypothesis h_{MAP} with the highest posterior probability.
- This is what we did in the example.
  - Challenge - Bayes' Theorem can be computationally expensive to use when observations are not i.i.d.
  - $P(h|o_1, o_2) = \frac{P(o_1|h,o_2)P(h|o_2)}{P(o_1|o_2)}$

Bayesian Optimal Classifiers

- There's one other problem with the formulation as we have it.
- Usually, we're not so interested in the hypothesis that fits the data.
- Instead, we want to classify some unseen data, given the data we've seen so far.
- One approach would be to just return the MAP hypothesis.
- We can do better, though.

Bayesian Optimal Classifiers

Given new instance x, what is its most probable *classification*?

- $h_{MAP}(x)$ is not the most probable classification!

Consider:

- Three possible hypotheses:

  $$P(h_1|D) = .4,\ P(h_2|D) = .3,\ P(h_3|D) = .3$$

- Given new instance x,

  $$h_1(x) = +,\ h_2(x) = -,\ h_3(x) = -$$

- What's most probable classification of x?

Bayesian Optimal Classifiers

$\arg\max_{y \in Y} \sum_{h_i \in H} P(y|h_i)P(h_i|D)$

Our 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

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = -$$

Bayesian Optimal Classifiers

- By combining the predictions of each hypothesis, we get a Bayesian optimal classifier.
- The probability $P(v_j|D)$ that our new instance belongs to class v_j is:
- $\sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$
- Intuitively, each hypothesis gives its prediction, weighted by the likelihood that that hypothesis is the correct one.
- This classification method is provably optimal - on average, no other algorithm can perform better.

Problems

- However, the Bayes optimal classifier is mostly interesting as a theoretical benchmark.
- In practice, computing the posterior probabilities is exponentially hard.
- This problem arises when instances or data are conditionally dependent upon each other.
- Can we get around this?

Naive Bayes classifier

- The Naive Bayes classifer makes a strong assumption that makes the algorithm practical:
  - Each attribute of an example is independent of the others.
  - $P(a \wedge b) = P(a)P(b)$ for all a and b.
- This makes it straightforward to compute posteriors.

The Bayesian Learning Problem

- Given: a set of labeled, multivalued examples.
- Find a function $F(x)$ that correctly classifies an unseen example with attributes $(a_1, a_2, ..., a_n)$.
- Call the most probable category v_{map}.
- $v_{map} = argmax_{v_i \in V} P(v_i | a_1, a_2, ..., a_n)$
- We rewrite this with Bayes' Theorem as:
  $v_{map} = argmax_{v_i \in V} P(a_1, a_2, ..., a_n | v_i) P(v_i)$
- Estimating $P(v_i)$ is straightforward with a large training set; count the fraction of the set that are of class v_i.
- However, estimating $P(a_1, a_2, ..., a_n | v_i)$ is difficult unless our training set is *very* large. We need to see every possible attribute combination many times.

Naive Bayes assumption

- Naive Bayes assumes that all attributes are conditionally independent of each other.
- In this case, $P(a_1, a_2, ..., a_n|v_i) = \Pi_i P(a_i|v_i)$.
- This can be estimated from the training data.
- The classifier then picks the class with the highest probability according to this equation.
- Interestingly, Naive Bayes performs well even in cases where the conditional independence assumption fails.

Naive Bayes Classifier

$$v_{map} = \arg\max_{v_i \in V} P(a_1, a_2 \ldots a_n | v_j) P(v_j)$$

Naive Bayes assumption:

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

which gives

Naive Bayes classifier: $v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_i P(a_i | v_j)$

Example

- Recall our tennis-playing problem
- We want to use the training data and a Naive Bayes classifier to classify the following instance:
- Outlook = Sunny, Temperature = Cool, Humidity = high, Wind = Strong.

Example

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Example

- Our priors are:
  - $P(PlayTennis = yes) = 9/14 = 0.64$
  - $P(PlayTennis = no) = 5/14 = 0.36$
- We can estimate:
  - $P(wind = strong|PlayTennis = yes) = 3/9 = 0.33$
  - $P(wind = strong|PlayTennis = no) = 3/5 = 0.6$
  - $P(humidity = high|PlayTennis = yes) = 3/9 = 0.33$
  - $P(humidity = high|PlayTennis = no) = 4/5 = 0.8$
  - $P(outlook = sunny|PlayTennis = yes) = 2/9 = 0.22$
  - $P(outlook = sunny|PlayTennis = no) = 3/5 = 0.6$
  - $P(temp = cool|PlayTennis = yes) = 3/9 = 0.33$
  - $P(temp = cool|PlayTennis = no) = 1/5 = 0.2$

Example

- $v_{yes} = P(yes)P(sunny|yes)P(cool|yes)P(high|yes)P(strong|yes) = 0.005$
- $v_{no} = P(no)P(sunny|no)P(cool|no)P(high|no)P(strong|no) = 0.0206$
- So we see that not playing tennis is the maximum likelihood hypothesis.
- Further, by normalizing, we see that the classifier predicts a $\frac{0.0206}{0.005+0.0206} = 0.80$ probability of not playing tennis.

Estimating Probabilities

- Let's use n_c for the number of (positive or negative) examples with the characteristic of interest (say Wind = strong)
- We estimated $P(wind = strong|yes)$ as frequency of seeing both over frequency of yes days, or n_c/n
- Where we generically use n to be either the number of positive OR negative examples, depending on which one we're calculating
- As we can see from this example, estimating probabilities through frequency is risky when our data set is small.
- We only have 5 negative examples, so we may not have an accurate estimate.

Estimating Probabilities

- A better approach is to use the following formula, called an m-estimate: $\frac{n_c+mp}{n+m}$
- p is our prior estimate of n_c/n
- m is a constant called the *equivalent sample size*.
- m determines how heavily to weight p.
- p is assumed to be uniform

Estimating Probabilities

- $\frac{n_c+mp}{n+m}$
- So, in the Tennis example,
  $P(wind = strong|playTennis = no) = \frac{3+0.5m}{5+m}$
- Where m is set to some small number, smaller if we have lots of examples
- Intuition: add a bunch of examples to start you off, specifically, m of a given class, and $p \times m$ of them having $wind = strong$

Naive Bayes in practice

Along with decision trees, neural networks, nearest nbr, 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:

- Diagnosis
- Classifying text documents

Using Naive Bayes to classify spam

- One are where Naive Bayes has been very successful is in text classification.
  - Despite the violation of independence assumptions.
- Classifying spam is just a special case of text classification.
- Problem - given some emails labled ham or spam, determine the category of new and unseen documents.
- Our features will be the tokens that appear in a document.
- Based on this, we'll predict a category.

Classifying spam

- Naive Bayes is only one possible way to classify spam.
  - Rule-based systems (SpamAssassin)
  - Examining headers (broken From or Content-Type)
  - Blacklist
  - Challenge/response

Classifying spam with Naive Bayes

- Naive Bayes has several properties that make it nice as a spam classifier:
  - We don't need to encode specific rules
  - We can adapt as the types of spam change
  - Somewhat robust to spammers adding in extra text

Using Naive Bayes to classify spam

- For a given email, we'll want to compute the MAP hypothesis - that is, is:
  - $P(spam|t1, t2, .., tn)$ greater than
  - $P(ham|t1, t2, .., tn)$
- We can use Bayes' rule to rewrite these as:
  - $\alpha P(t1, t2, ..., tn|spam)P(spam)$
  - $\alpha P(t1, t2, ..., tn|ham)P(ham)$

Using Naive Bayes to classify spam

- We can then use the Naive Bayes assumption to rewrite these as:
  - $\alpha P(t1|spam)P(t2|spam)...P(tn|spam)P(spam)$
  - $\alpha P(t1|ham)P(t2|ham)...P(tn|ham)P(ham)$
- And this we know how to compute.

Using Naive Bayes to classify spam

- We can get the conditional probabilities by counting tokens in the training set.
- We can get the priors from the training set, or through estimation.

Using Naive Bayes to classify spam

- This is a case of a problem where we can tolerate occasional false negatives (spam classified as ham) but we cannot tolerate false positives (ham classified as spam).
- Plain old vanilla Naive Bayes will do fairly well, but there's a lot of tuning and tweaking that can be done to optimize performance.

Using Naive Bayes to classify spam

- There are a lot of wrinkles to consider:
  - What should be treated as a token? All words? All strings? Only some words?
  - Should headers be given different treatment? Greater or less emphasis? What about subject?
  - What about HTML?
  - When classifying an email, should you consider all tokens, or just the most significant?
  - When computing conditional probabilities, should you could the fraction of documents a token appear in, or the fraction of words represented by a particular token?

Log-likelihood

- One challenge with using Naive Bayes on data with lots of features (such as an email) is this:
  - $P(x_1|c)$ is probably small. Definitely < 1, often less than 0.001.
  - What happens if we have hundreds of $P(x_i|c)$ and we multiply them together?

Log-likelihood

- One challenge with using Naive Bayes on data with lots of features (such as an email) is this:
  - $P(x_1|c)$ is probably small. Definitely < 1, often less than 0.001.
  - What happens if we have hundreds of $P(x_i|c)$ and we multiply them together?
- Underflow!

Log-likelihood

- Remember, we don't really care about the exact values $P(spam)$ and $P(ham)$. We just want to know which one is larger.
- So, if $P(c_1) > P(c_2)$, then $log(P(c_1)) > log(P(c_2))$ since log is monotonically increasing.
- log also has the nice property that:
  - $log(a * b) = log(a) + log(b)$
- So, $log(P(x_1|c)P(x_2|c)...P(x_n|c)) = log(P(x_1|c)) + log(P(x_2|c)) + ... + log(P(x_n|c))$
- This is called *log-likelihood*

F-measure

- In this sort of application, we really need high recall, but also very few false positives, so we need a metric that captures both of these.
- The standard way to do this in IR is *f-measure*, or *F1 score*.
- $F = \frac{2 * recall * precision}{recall + precision}$
- This is the *harmonic mean* of recall and precision.

How to build a Naive Bayes classifier

- Estimate priors for each class, either given or from training data.
- Filter data to remove irrelevant features.
- For each word, compute $P(w|c)$ by counting its frequency in documents of class c. Store this in a dictionary. (you may also want to store $log(P(w|c))$ to avoid repeated calculation.)
- To classify a new document, for each class c_i
  - compute $P(log(w_1|c_i)) + P(log(w_2|c_i)) + ... + P(c_i)$
- The largest value will be the most likely classification.

Advantages of Naive Bayes

- Scales to large numbers of features and large data
- Reported to perform as well as decision trees and neural network classifiers
- Able to mix continuous and discrete features
- Can be applied to small or sparse datasets (i.e. with missing data)

Disadvantages of Naive Bayes

- May require unreasonable assumptions for continuous features
- Assumes that features are independent – not always true
- Learned distributions not likely to be good *estimates*

Bayesian Learning Summary

- MAP and ML hypotheses
- Bayesian Concept Learning
- Naive Bayes
- Applying Naive Bayes to practical problems