TDT4173 Assignment 4

Bayesian Learning & Expectation Maximization

Iver Jordal

1. Theory

Bayesian Learning

1.1

At first, my hunch was that likelihood was the same as probability. However, it seems that is not true. "Likelihood is the hypothetical probability that an event that has already occurred would yield a specific outcome. The concept differs from that of a probability in that a probability refers to the occurrence of future events, while a likelihood refers to past events with known outcomes." [0]

In other words, *likelihood* is the probability of observing data d given a hypothesis h_i.

What is the difference between a *maximum a posteriori* hypothesis and a *maximum likelihood* hypothesis?

The maximum a posteriori hypothesis is the single most probable hypothesis given the evidence, i.e. the h_i that maximizes $P(h_i|d)$.

The Maximum Likelihood hypothesis is the hypothesis h_i that maximizes $P(d|h_i)$.

In the case of a uniform prior, "MAP learning reduces to choosing an h_i that maximizes $P(d|h_i)$" [1]. So in some cases, h_{MAP} and h_{ML} are the same hypothesis, but not always. Let me give an example where they are not equal: Let's say that we're flipping a coin. The coin is either a normal coin (h_2) or one of those special coins that don't have normal flip probabilities (h_1, h_3). In an experiment we try find out by flipping the coin several times.

$$h_1 = 90\%\ tails, 10\%\ heads$$
$$h_2 = 50\%\ tails, 50\%\ heads$$
$$h_3 = 10\%\ tails, 90\%\ heads$$

Example of non-uniform prior distribution over h_1, h_2, h_3: $\langle 0.1, 0.8, 0.1 \rangle$

Uniform prior distribution over h_1, h_2, h_3: $\langle 0.\overline{33}, 0.\overline{33}, 0.\overline{33} \rangle$

In the latter case (uniform) $h_{MAP} = h_{ML}$ but in the former case (non-uniform) $h_{MAP} \neq h_{ML}$

To explain this better, let's try to express h_{MAP} differently:

$$h_{MAP} = argmax_{h \in H}\ P(h|D) = argmax_{h \in H}\ \frac{P(D|h) * P(h)}{P(D)}$$

Because P(D) is constant in this context, we can disregard it. And when we have a uniform prior, $P(h)$ is also a constant. So in that case, $h_{MAP} = h_{ML}$ because $argmax_{h \in H}\ P(h|D) = argmax_{h \in H}\ P(D|h)$

1.2

Naive Bayes: You assume that the attributes are conditionally independent of each other

Building a Bayes classifier is all about learning naïve bayes distributions from observations. So how does one train that model? Actually, it's not that complicated, because it is basically a counting problem. This is the algorithm (copied from a slide in lecture 7):

`Naïve_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 $P(a_i|v_j)$

Classifying a new instance with naive bayes:

`Classify_New_Instance(`x`)`

$$v_{\text{NB}} = \underset{v_j \in V}{\text{argmax}}\, \hat{P}(v_j) \prod_{a_i \in x} \hat{P}(a_i|v_j)$$

OptimalBayes classifier:

$$\underset{v_j \in V}{\text{argmax}} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

Is the Brute-force MAP learning algorithm of any use?

BRUTE-FORCE MAP LEARNING algorithm

1. For each hypothesis h in H, calculate the posterior probability

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

2. Output the hypothesis h_{MAP} with the highest posterior probability

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

In other words, this algorithm "calculates the probability for each possible hypothesis, then outputs the most probable one"[2]. This can be very slow when H is a very large set of hypotheses, which is typically the case for OptimalBayes.

The naive Bayes classifier is an approximation of the Bayes classifier

1.3

	NaiveBayes	OptimalBayes
Computational cost	Moderate	Very high, thus this method is rarely used on real-world problems
Performance	Usually suboptimal. Optimal if the variables are truly conditionally independent.	Optimal

Considered hypothesis space	A subspace	All of them

1.4

I don't have deep knowledge about classifiers based on Bayes' theorem, so frankly, I don't have sound ideas about how to improve OptimalBayes and NaiveBayes. But anyway, I'll *try* to think of something:

NaiveBayes: Assumes that all variables are conditionally independent, while they are actually not. Performance could probably be improved by adding more hypotheses, somehow. That would probably hurt computational cost, though. It's a trade-off. There is a variant called *Augmented Naive Bayes*[3] which approximates the dependence among features. That could be useful.

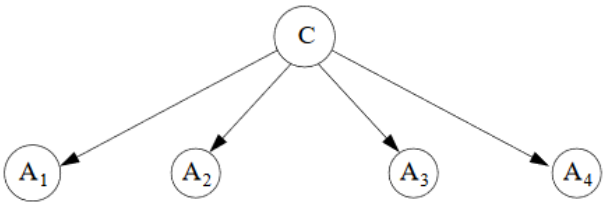

Figure 1: An example of naive Bayes

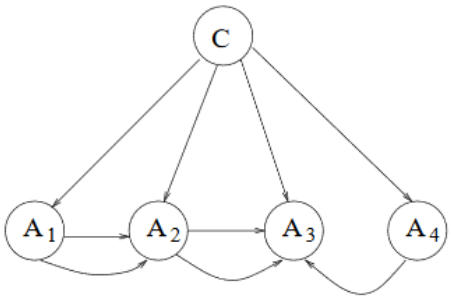

Figure 2: An example of ANB

One other method that is an improvement over NaiveBayes was proposed by M. Martinez-Arroyo and L. E. Sucar[4]:

> *To deal with dependent and irrelevant attributes, we apply a structural improvement method that eliminates and/or joins attributes, based on mutual and conditional information measures.*

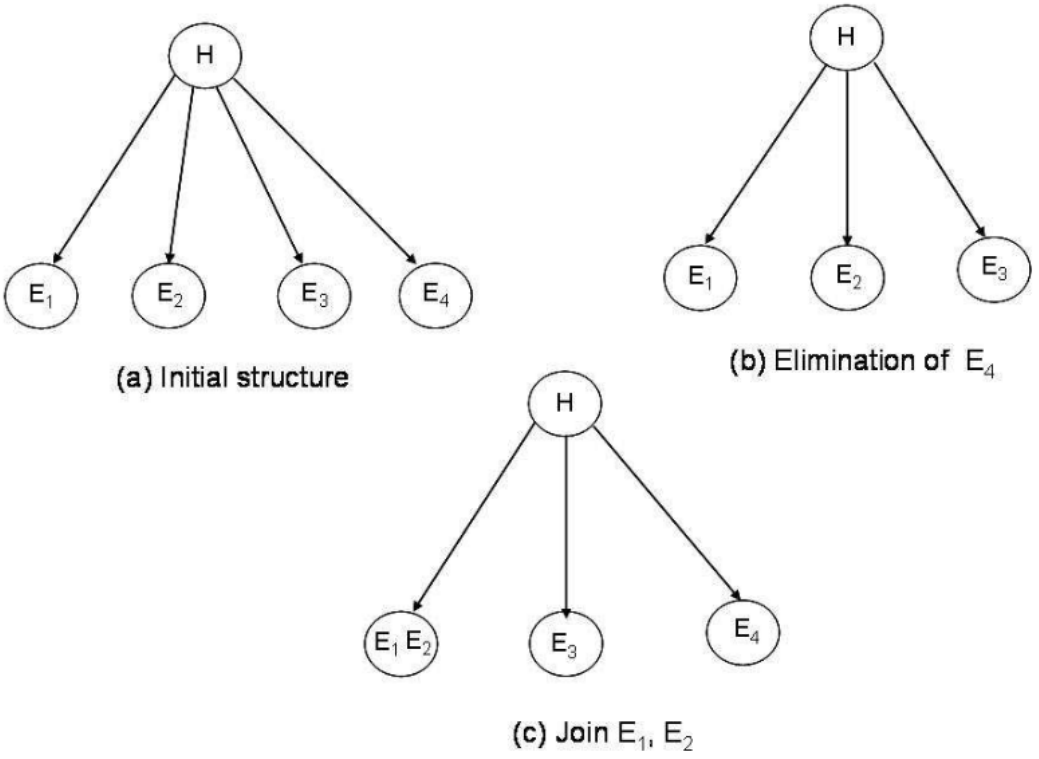

Figure 3: Structural improvement: (a) original structure (NaiveBayes), (b) one attribute is eliminated, (c) two attributes are joined into one variable.

OptimalBayes: Is way too computationally expensive to be used on problems that are not very small. If there would be some way to reduce the considered hypothesis space in a smart way to reduce the computational cost, that would be nice. It would perhaps hurt predictive performance a bit.

1.5

What is the difference between NaiveBayes classifier and Bayesian belief network? What is the relation between these two methods?

Bayesian Belief networks describe conditional independence among subsets of variables. In NaiveBayes all variables are (assumed to be) conditionally independent.

2 Programming

Expectation Maximization

First, I'll visualize the data in a histogram, so I can see what kind of numbers I'm dealing with:

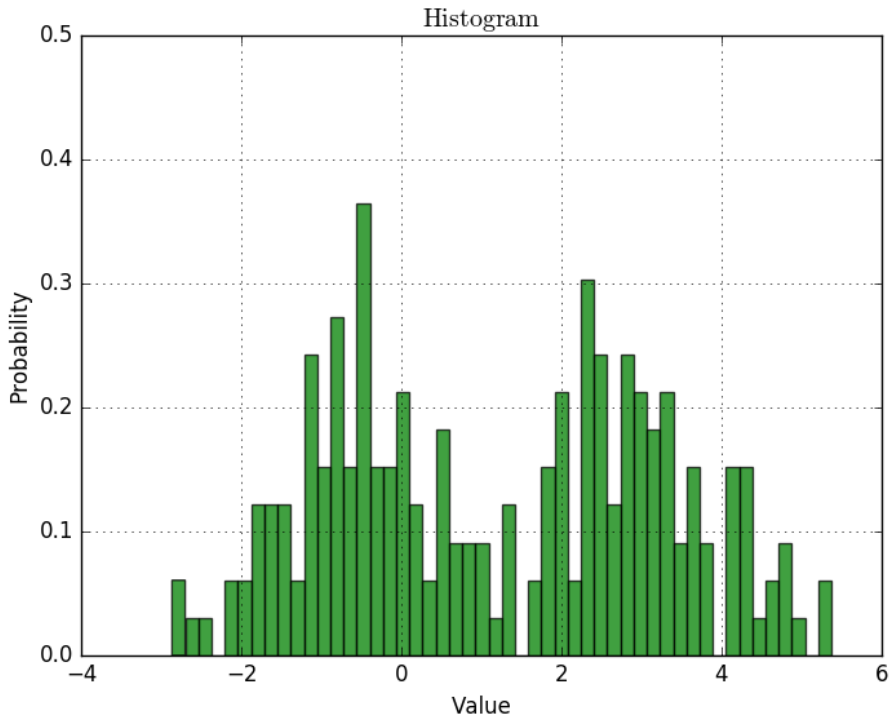

Figure 4: Histogram

That pretty much looks like two gaussians with $\sigma = 1$, as expected. Furthermore, I'm estimating that $\mu_1 \approx -0.5$ and $\mu_2 \approx 2.5$, so that's what I'll use for initial parameters. Alternatively, one could pick values randomly, but in this case I know some sane initial parameters, so it's better to use those.

```
em = ExpectationMaximization(
     k=2,
     mu=[-0.5, 2.5],
     sigma=1,
     x=data_manager.data
)
em.run(num_iterations=20)
```

After 5th iteration:	[-0.5774260146171408, 2.9776214207409355]
After 10th iteration:	[-0.5766712031819338, 2.978367647291882]
Finally (after 20 iterations):	[-0.5766685631700593, 2.978370252317221]

The numbers converge pretty quickly. There's not much going on after the first five iterations. The result makes sense. The numbers represent the mean value of each gaussian. They aren't very different from what I estimated by just looking at the histogram.

Here's the histogram plot with the fitted gaussians overlayed (note that they are not appropriately scaled in the y-direction, but should be correct in the x-direction)

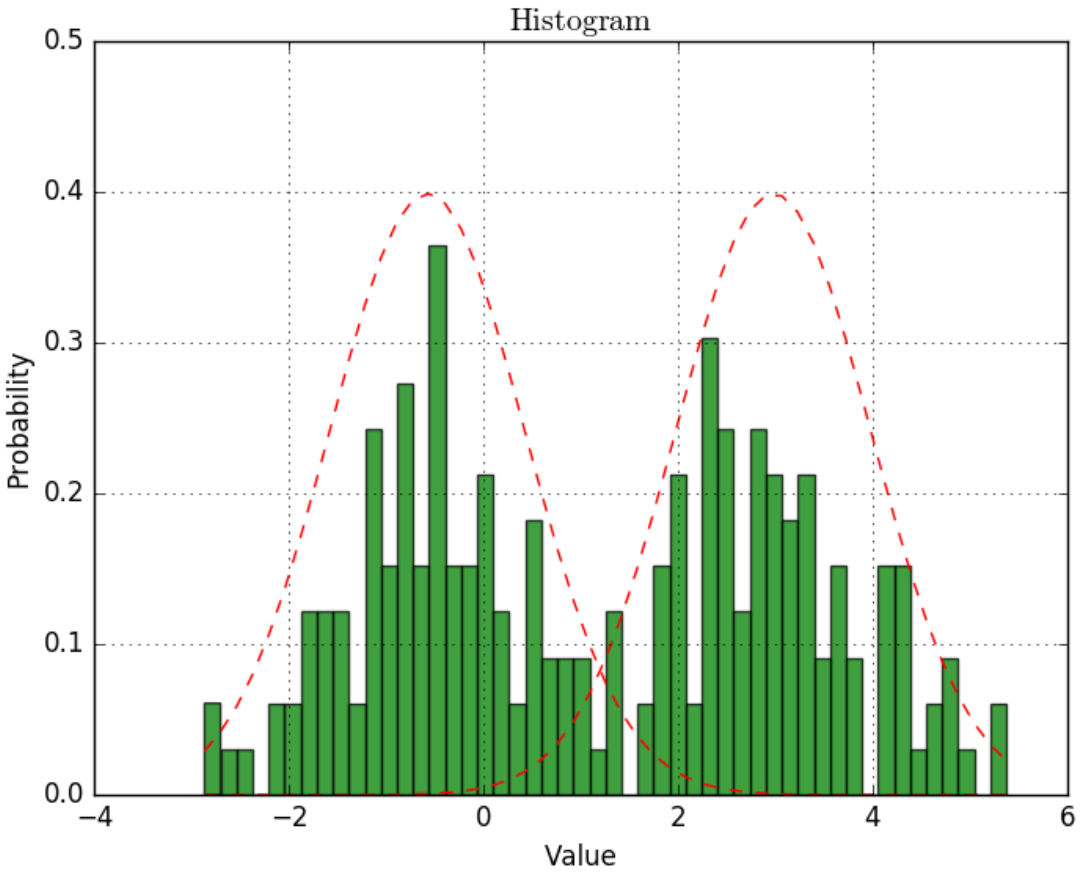

Figure 5: Histogram with fitted gaussians overlayed

References

0: Weisstein, Eric W. "Likelihood." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/Likelihood.html

1: Artificial intelligence – a modern approach 3E, Russell & Norvig, Pearson 2010

2: http://cse-wiki.unl.edu/wiki/index.php/Bayesian_Learning#Brute-force_MAP_learning_algorithm

3: http://www.cs.unb.ca/~hzhang/publications/FLAIRS04ZhangH.pdf

4: M. Martinez-Arroyo and L. E. Sucar, "Learning an Optimal Naive Bayes Classifier," *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, Hong Kong, 2006, pp. 1236-1239.