ASSIGNMENT 4

TDT4173 - Machine Learning and Case-Based Reasoning

Written by:

Håkon Ødegård Løvdal

Spring 2016

Norwegian University of Science and Technology

1 Theory

Bayesian Learning

1.

Likelihood is here defined as the probability for some data D, given any hypothesis h. This can be presented as the term $P(D|h)$. Said in other words, this is how likely it is to observe the data, D, given the hypothesis, h.

h_{MAP} is the maximum probable hypothesis $h \in H$ given the observed data, D. It uses Bayes theorem to calculate the posterior probability of each candidate hypothesis. As seen in equation 1, it accounts for the likelihood of the current hypothesis.

$$h_{MAP} = \underset{h \in H}{argmax}\ P(D|h)P(h) \tag{1}$$

h_{ML} is different from this, in the way that it assumes that every hypothesis in H is equal a priori, and therefore only finds the most likely $h \in H$ given the observed data D. This is given by equation 2

$$h_{ML} = \underset{h \in H}{argmax}\ P(D|h) \tag{2}$$

It is possible for $h_{MAP} \neq h_{ML}$. In the example on page 158 in the curriculum [1], were we shall diagnose the patient as having cancer or not. Here $H = \{\text{Cancer}, \neg\text{Cancer}\}$. Given h_{ML} we would get the following results :

$$P(\text{Cancer}) = P(\oplus|\text{Cancer}) = 0.98$$
$$P(\neg\text{Cancer}) = P(\oplus|\neg\text{Cancer}) = 0.03$$

As seen, given a positive result, we would diagnose it as cancer. But with h_{MAP} the results would be different. As seen below, tha MAP hypothesis given a positive test will result in $\neg$ cancer.

$$P(\text{Cancer}) = P(\oplus|\text{Cancer}) \cdot P(\text{Cancer}) = 0.98 \cdot 0.008 = 0.00784$$
$$P(\neg\text{Cancer}) = P(\oplus|\neg\text{Cancer}) \cdot P(\neg\text{Canter}) = 0.03 \cdot 0.992 = 0.02976$$

2.

The OptimalBayes (see equation 3) classifier classifies the optimal classification for a new instance v_j as the one which $P(v_j|D)$ is maximum. $P(v_j|D)$ is caclulated by calculating the probabilities for all hypotheses given the data. The optimal classification is then chosen from the classifications $v_j \in V$ with the highest prediction.

$$\underset{v_j \in V}{argmax} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) \tag{3}$$

The NaiveBayes (see equation 4) classifier classifies by the assumption that the probabilities for observing a conjunction $a_1, a_2 \cdots a_n$ is the product of each attribute, where the probability is given by $P(a_i|v_j)$.

$$v_{NB} = \underset{v_j \in V}{argmax}\ P(v_j) \prod_i P(a_i|v_j) \tag{4}$$

The BruteForceMap learning algorithm is of use in OptimalBayes. Like the BruteForceMap, OptimalBayes calculates the probability for every hypothesis. The NaiveBayes classifier is an example of the OptimalBayes classifier. This is because of an assumption that the attributes are conditionally independent given the target value, instead of calculating their full conditional distribution given the target.

3.

Hypothesis space: Given equation 4 it is possible to see that NaiveBayes does not search through the hypothesis space at all. OptimalBayes on the other hand, searches through the entire hypothesis space.

Performance: In terms of the performance, OptimalBayes acheives the best performance, but it quickly becomes very expensive in terms of computational cost. NaiveBayes has proven to produce good performance, but it depends on the conditional independent probability assumption to hold.

Computational cost: As soon as a problem reaches some complexity, OptimalBayes becomes impracticable to perform. Due to the fact it has to compute the posterior probability for every hypothesis $h \in H$ and then combine the predictions of each hypothesis to classify each new instance, the computational cost becomes large.

4.

I'm not sure on specific details about how to improve the algorithms in terms of the three facets above, but I think I can come up with some ideas.

Hypothesis space: OptimalBayes may prove more efficient if it could reduce the search, to not include the entire hypothesis space.

Performance: In terms of performance OptimalBayes is best, as stated before. NaiveBayes may give better performance, if it were possible to add some additional attributes, making the result more precise.

Computational cost: As stated above, OptimalBayes may prove more efficient if it didn't include the entire hypothesis space.

5.

The NaiveBayes makes an assumption that the values of all the attributes $a_1, a_2 \cdots a_n$ are conditionally independent given the target value v. Bayesian belief networks is different from this since

it describes a conditional network, allowing for conditional independence assumptions to only subsets of the variables (a conditional independence graph). This makes each variable or node to be associated with a conditional probability table.

The relation between the two methods is that the NaiveBayes classifier is a special case of Bayesian belief network, where all of the features are conditionally independent. This means that the class-nodes does not have any parents, and nodes corresponding to the attributes does not have any edges between them.

2 Programming

As my initial parameters, I set `means` to the minimum and maximum number of the provided data list. This were to ensure that the initial means were as far apart from each other as possible. I hopped this would make the two numbers be close to each their Gaussian. The problem with this though, is that both the numbers are in the far ends of the data points. Two other solutions I can think of would be two chose the means randomly, or to find the two median values of the data while sorted. Below is the intermediate results from every fifth iteration of the code and during which iteration it converged.

As seen in listing 2, the result gets very close to the final solution after 5 iterations. The convergence happens after 32 iterations. In the first iteration we can see that the initial means are far from the solution. Most likely, choosing the minimum and maximum from the data list is not the best solution.

Listing 1: "Inital parameters"

```
means = [min(data), max(data)] # Minimum and maximum of data
num_of_mixtures = len(data)  # Number of mixtures
num_of_measures = len(means)  # Number of Gaussians
```

Listing 2: "Intermediate results and final results"

```
Iteration 1:     [-2.8646000000, 5.3726000000]
Iteration 5:     [-0.5760949893, 2.9789393072]
Iteration 10:    [-0.5766665606, 2.9783722284]
Iteration 15:    [-0.5766685562, 2.9783702592]
Iteration 20:    [-0.5766685631, 2.9783702524]
Iteration 25:    [-0.5766685631, 2.9783702523]
Iteration 30:    [-0.5766685631, 2.9783702523]
Iteration 32:    [-0.5766685631, 2.9783702523]
Converged at iteration 32
```

References

[1] T. M. MITCHELL, *Machine Learning*, McGraw-Hill, 1997.

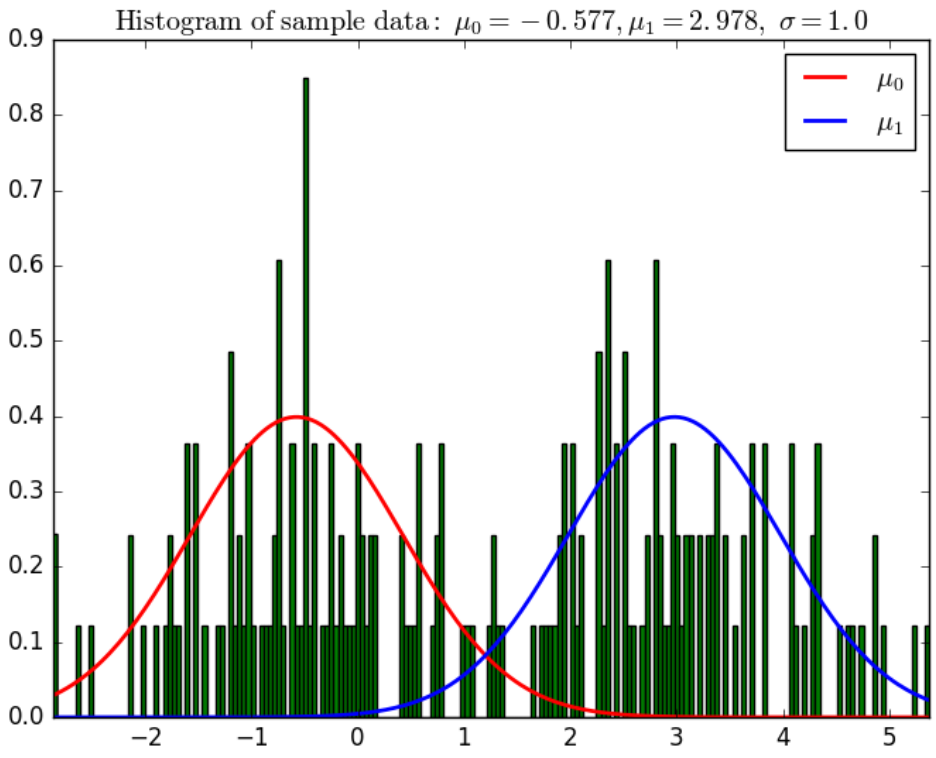

Figure 1: Visualization of sample data with fitted Gaussians