

Weighted Majority and the Online Learning Approach

Lecturer: Drew Bagnell Scribe:Chris Cunningham, Sidd Swaminathan, Shihyun Lo ¹

1 Gibbs Sampling Example

Carl Wellington's thesis work ²is used as a practical example of gibbs sampling. The motivation behind this work is the problem of safely navigating an autonomous tractor through tall grass and vegetation using a combination of cameras, lidar, and infrared cameras. The crux of the problem is determining the height of the terrain supporting surface beneath vegetation, which is then used to determine traversability. Prior methods for ground height estimation include taking the lowest height of lidar pass through. However, this does not in general work very effectively. Wellington's work posed the problem as inference on a probabilistic graphical model. Example images showing the work are shown in Figure 1



Fig. 6. View from tractor of tall weeds, low grass, person, and small dirt mound

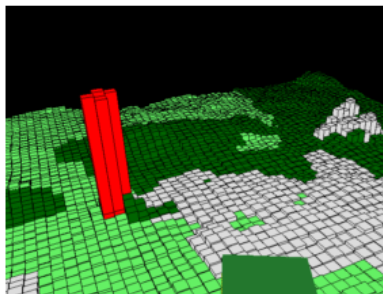


Fig. 7. System output, including ground heights and classification

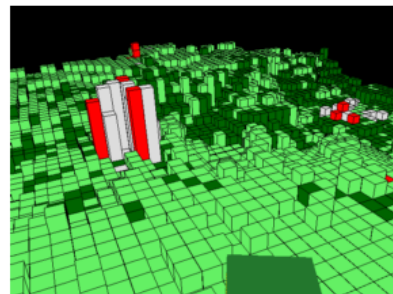


Fig. 8. Lowest hit or pass for ground height in vegetation and independent classification, showing poor ground height estimates and misclassifications

Figure 1: Example output from a tractor trying to estimate the ground plane of a vegetated area. The left image is the raw camera image. The center image is the terrain classification and ground plane estimate using gibbs sampling. The right image shows classification and ground plane estimation based on lowest hit or pass-through from lidar data.

The graphical model is a hybrid of a directed and undirected graphical model and is shown in Figure 2. The directed edges can just be thought of as undirected edges that happen to normalize locally, containing more information. There are essentially two interacting graphical models. One of the models is an undirected 2 cm square grid overlaid on the ground where each vertex has a ground height and a class. The classes included bare earth, light vegetation, heavy vegetation, and obstacle. This part of the model affects the result by enforcing similar ground heights and classes for neighboring cells in the grid. One thing to note is that for a specific vertex in the graph, if you know the Markov blanket, then it knowing heights and classes of other vertices does not affect the result. This terrain model assumes that there is a smoothness associated with the gradient.

¹Content adapted from previous scribes by Sankalp Arora, Nathaniel Barshay, and Narek Melik-Barkhudarov

This means that sharp gradients (such as trees and anthills) are not easily represented. We saw an example where initially, the robot thought that an ant-hill was an obstacle (something it couldn't drive over), but as more data was obtained, it realized that it was just part of the terrain.

The other part of the graphical model is a Hidden Semi-Markov Model used for determining ground height and terrain class. There is a vertex in the graph for each vertex above the ground. This model assumes that there free space above either vegetation, an obstacle, or nothing, which is in turn above the ground. This model works well for people, buildings, grass, and other light vegetation. However, it fails in the case of trees or other obstacles that have free space between other obstacles or vegetation in the voxel column.

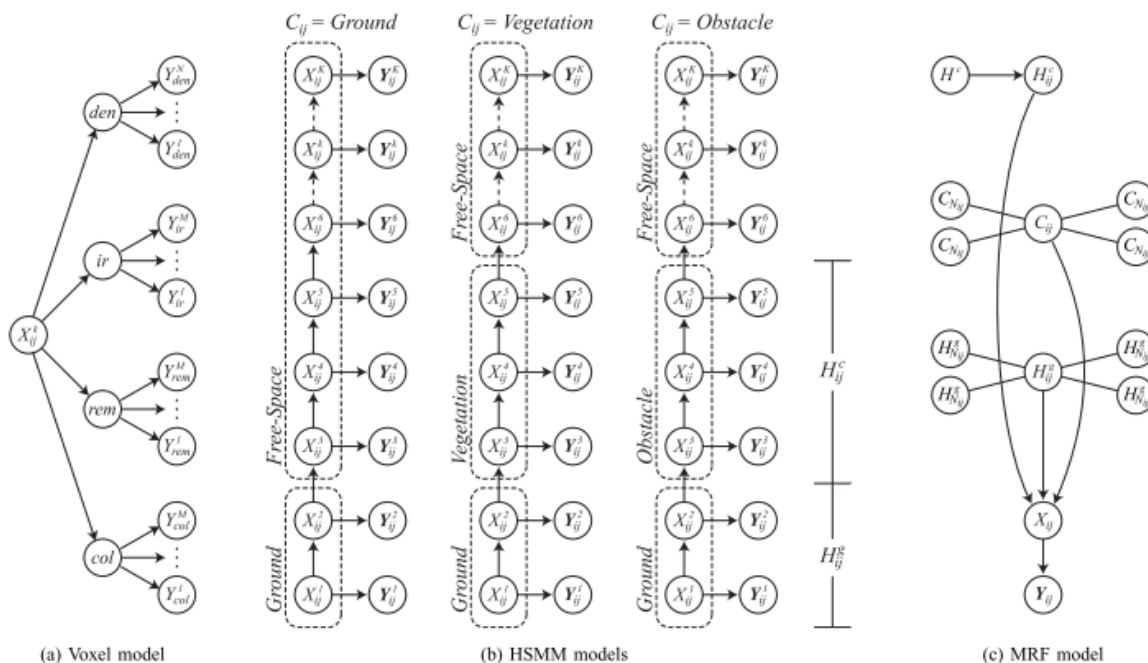


Fig. 2. A graphical description of the model showing (a) the voxel, (b) the voxel column, and (c) the connections between voxel columns. For each voxel column ij , the model contains voxel states X_{ij}^k , observations Y_{ij}^k , and a class C_{ij} , class height H_{ij}^c , and ground height H_{ij}^g that interact with neighbors N_{ij} and the common class height H^c

Figure 2: The probabilistic graphical model for the terrain classification and ground plane estimation algorithm.

Gibb's sampling is used to determine statistics for probable configurations of the graphical model. The algorithm is shown in Figure 3. This algorithm is manually tuned to work with this graphical model and is not generally applicable to all graphical models. It runs quickly enough to run in real time. Note that for a graphical model the most probably configuration does not necessarily equal the most probable assignment for all values. Additionally, Gibb's sampling does not give you the most probable configuration, just samples that can then be used to determine statistics on the graphical model.

More details can be found in the thesis.

²<http://www.carllwellington.com/research/pub/wellingtonThesis.pdf>

Algorithm 1 Gibbs sampling from the model

Sample common class heights h^c from $P(H^c \mid H_{ij \in I, J}^c)$
using all the column class height samples of the same class
for all MRF voxel columns ij **do**
 Find ground and class priors from neighbors:
 $P(H_{ij}^g \mid H_{N_{ij}}^g)$
 $P(C_{ij} \mid C_{N_{ij}})$
 for all Classes c **do**
 Find class height prior from common class height of
 same class:
 $P(H_{ij}^c \mid H^c)$
 Use class HSMM to find probability of the data and
 distributions over the ground and class height:
 $P(Y_{ij} \mid C_{ij} = c, H_{N_{ij}}^g, H^c)$
 $P(H_{ij}^g \mid C_{ij} = c, Y_{ij}, H_{N_{ij}}^g, H^c)$
 $P(H_{ij}^c \mid C_{ij} = c, Y_{ij}, H_{N_{ij}}^g, H^c)$
 end for
 Compute class distribution:
 $P(C_{ij} \mid Y_{ij}, C_{N_{ij}}, H_{N_{ij}}^g, H^c)$
 $\propto P(Y_{ij} \mid C_{ij}, H_{N_{ij}}^g, H^c)P(C_{ij} \mid C_{N_{ij}})$
 Sample c_{ij} from $P(C_{ij} \mid Y_{ij}, C_{N_{ij}}, H_{N_{ij}}^g, H^c)$
 Sample h_{ij}^g from $P(H_{ij}^g \mid C_{ij} = c_{ij}, Y_{ij}, H_{N_{ij}}^g, H^c)$
 Sample h_{ij}^c from $P(H_{ij}^c \mid C_{ij} = c_{ij}, Y_{ij}, H_{N_{ij}}^g, H^c)$
end for

Figure 3: The inference algorithm using gibbs sampling on the graphical model shown in Figure 2

2 Recap

So far we have established that the best way to represent our beliefs are probabilities.

Second. From the book "Probability: The Logic of Science" we know that probabilities are a natural extension of logic by adding a "degree of belief" to it. More specifically the claim is, that probability is the only system which is consistent with logic and continuous. This is called the Normative view of the world: logic is rational, therefore probabilities are also, and you would be irrational not to use them. Interestingly, this view does not say anything as to what happens when we do approximations. It also gives no guarantees on the performance of probabilities in practice.

Third. Bayes rule which allows us to do induction:

1. use hypothesis about the world
2. use data to refine hypothesis
3. make predictions

Fourth. Bayes filters allow us to apply the Bayes rule recursively. Filtering is expensive, since sums and integrals are exponential in state variables, however we can make approximations to deal with this (e.g. particle filters).

Finally, we have graphical models, which allow us to add structure to our beliefs. This allows us to reason more easily, as well as leads to a compact representation of beliefs. We have seen that this still is computationally expensive and we have shown ways of doing exact inference. The fallback solution for cases when we cannot do exact inference is using a Gibbs sampler.

The question now is whether we can formulate a normative view of induction?

2.1 Three views of induction

1. David Hume view. This view effectively argues that induction doesn't work. Consider a chicken in a farm. For 1000 days, every day, the farmer comes and feeds the chicken. Given the well known fact that all chickens are perfect Bayesians, what does the chicken think is going to happen to it on the 1001st day? Using induction it should assume that it will be fed, however it might be completely wrong and the farmer might have decided to eat it. The chicken might have a model that accounts for its imminent death, however using induction it will be unable to guess its death on day 1001. No matter what model the chicken uses, an adversarial farmer can always thwart it and force the chicken to make false inductive predictions. This view is correct, but not very useful.
2. The Goldilocks/Panglossian/Einstein Position. It just so happens that our world is set up in a way such that induction works. It is the nature of the universe that events can be effectively modeled through induction to a degree that allows us to make consistently useful predictions about it. In other words, we're lucky. Once again, we do not find this view very satisfying or particularly useful.
3. The No Regret view. This view has evolved from game theory and computer science and is unique to the late 20th century. It postulates that while induction can fail, it is a near optimal strategy.

3 On-line learning Algorithms

To formalize the argument of the No Regret view we consider the problem of predicting from expert advice. Assume we are given a set of n experts, each of which predicts a binary variable (for example whether the sun will rise or not tomorrow). These can be very simple experts, for example:

E1: Always predicts sunrise E2: Always predicts no sunrise E3: Flips a coin to decide whether the sun rises or not E4: Markov Expert: predicts previous days outcome. E5: Running average over past few days. ...

Similarly our algorithm predicts the same variable. After that we wait for the next day to come and observe whether the sun rose or not. We then compare the output of our algorithm to that of the best expert so far. For this purpose we define a *loss* function.

$$L(\text{you}, \text{world}) = \begin{cases} 0 & \text{if } \text{you} = \text{world} \\ 1 & \text{otherwise} \end{cases}$$

From the Hume view it follows that there is nothing that makes

$$L_t \rightarrow 0$$

We even have no way of enforcing a weaker condition

$$\frac{1}{t} \sum L_t \rightarrow 0$$

We can however try to minimize a different quantity, something we call *regret*. We define regret, as our divergence from the best expert so far:

$$R = \sum_t [L_t(\text{algorithm}) - L_t(e^*)]$$

where e^* is the best expert at time t . We will show that it is possible to make the regret scale slower than time:

$$\lim_{t \rightarrow \infty} \frac{R_t}{t} = 0$$

and this will happen even if the world is trying to trick you and even if the experts are trying to trick you.

An algorithm that satisfies the above formula is called “no regret.” Note that even though we cannot minimize loss, we can minimize regret. A side effect of this statement is that a no regret algorithm can still have high loss.

Simplest algorithm: Follow the Leader

Also called “best in hindsight” algorithm. This algorithm naively picks the expert that has done the best so far and use that as our strategy for the next timestep. Unfortunately, blindly picking the leader can lead to overfitting.

Let us assume there are 2 experts:

E1: Sun always rises E2: Sun never rises

World:	1	0	1
E1:	1	1	1
E2:	0	0	0
You:	1	0	1

In this particular example you are doing worse than either expert.

Majority Vote

Given many experts, this algorithm goes with the majority vote. In a sense, it never gets faked out by observations, because it doesn’t pay attention to them. In a statistical sense, we can say that this algorithm is “high bias, low variance,” since it is robust but not adaptive, while algorithm #1 is “low bias, high variance,” since it is adaptive but not robust. This algorithm can clearly perform much worse than the best expert as it may be the case that majority of the experts are always wrong but one is always correct. To overcome this fallacy we need a method that adapts

and decides what experts to respect while predicting, based on the past performance of the experts. To this end we try weighted majority algorithm.

3.1 Weighted Majority

Another approach is to notice that at each timestep, each expert is either right or wrong. If it's wrong, we can diminish the voting power of that particular expert then predicting the next majority vote.

More formally, this algorithm maintains a list of weights w_1, \dots, w_n (one for each expert x_1, \dots, x_n), and predicts based on a weighted majority vote, penalizing mistakes by multiplying their weight by half.

Algorithm

1. Set all the weights to 1.
2. Predict 1 (rain) if $\sum_{x_i=1} w_i \geq \sum_{x_i=0} w_i$, and 0 otherwise.
3. Penalize experts that are wrong: for all i s.t. x_i made a mistake, $w_i^{t+1} \leftarrow \frac{1}{2}w_i^t$.
4. Goto 2

Analysis of Algorithm The sum of the weights is $w \leq \left(\frac{3}{4}\right)^m n = \left(\frac{4}{3}\right)^{-m} n$, where m is the number of mistakes. The weight of the best expert $w_i^* = \left(\frac{1}{2}\right)^{m^*} = 2^{-m^*} \leq w$. Therefore,

$$2^{-m^*} \leq w \leq \left(\frac{4}{3}\right)^{-m} n.$$

Taking the \log_2 and solving for m gives,

$$m \leq \frac{m^* + \log_2 n}{\log_2 \frac{4}{3}} = 2.41(m^* + \log_2 n)$$

Thus, the number of mistakes by this algorithm is bounded by m^* plus an amount logarithmic in the number of experts, n .

To get a more intuitive feel for this last inequality, imagine the case when one expert makes *no* mistakes. Due to the binary search nature of the algorithm, it will take $\log_2 n$ iterations to “find” it. Furthermore, if we keep adding good experts, the term m^* will go down, but the term $\log_2 n$ will rise (a fair trade in this case). Adding bad experts will probably not change m^* much and will serve only to add noise, showing up in a higher $\log_2 n$. Thus, we can see the trade-offs involved when adding more experts to a system.

3.2 Randomized Weighted Majority Algorithm

In this algorithm, we predict each outcome with by picking an expert with a probability proportional to its weight. We also penalize each mistake by β instead of by one-half.

Algorithm

1. Set all the weights to 1.
2. Choose expert i in proportion to w_i .
3. Penalize experts that are wrong: for all i s.t. x_i made a mistake, $w_i^{t+1} \leftarrow \beta w_i^t$.
4. Goto 2.

Analysis The bound is

$$E[m] \leq \frac{m * \ln(1/\beta) + \ln(n)}{1 - \beta}.$$

It is important to note here that the expectation in the expression is caused by probabilistic nature of the method of selecting an expert for prediction. Also note that since we are picking a single expert, we don't have to worry about figuring out a way to combine multiple experts and can easily work in a space where the experts are predicting abstract things like whether the object seen is a phone, ocean or moon.

Generally, we want β to be small if we only have a few time steps available and vice-versa. Although β can also be time-varying, for example if you start with a large number of experts with a belief that some of the experts are good at predicting the world at the next timestep. In such a case we would want the β to be high initially to reach to the correct experts quickly and then decrease the value of β with time to avoid over-fitting. This algorithm serves to thwart an adversarial world that might know our strategy for picking experts, or any potential collusion between malicious experts and the world.

3.3 Generalized Weighted Majority

The Randomized Weighted Majority algorithm mentioned above assumes a $1 - 0$ loss function. This algorithm generalizes this to use any loss function in $[0, 1]$.

Algorithm

1. Set all the weights to 1. $w_i = 1 \forall i$
2. Choose expert i in proportion to the normalized weights ($w_i / \sum w_i$)
3. Adjust the weights

$$w_i \leftarrow w_i e^{-\epsilon l_t(e_i)}$$

4. Goto 2

Analysis The bound on this algorithm is

$$E[R] \leq \epsilon \sum_t l_t(e^*) + \frac{1}{\epsilon} \ln N \quad (1)$$

Ideally, we would like this algorithm to be what is called “No Regret”, defined as the average regret over time converging to 0:

$$\frac{R_T}{T} \rightarrow 0 \quad (2)$$

To do so, we need to make sure that $\epsilon(T)$ decays in such a way that the regret grows less than linearly as a function of T . Since $l(e^*) \leq 1$ by definition, we have that $\sum_t l_t(e^*) \in O(T)$. Therefore, applying this to (1), we get:

$$E[R] \in O(\epsilon T + \frac{1}{\epsilon} \ln N) \quad (3)$$

Setting $\epsilon = \frac{1}{\sqrt{T}}$, we get that

$$E[R] \in O(\sqrt{T} + \sqrt{T} \ln N) \quad (4)$$

This grows sublinearly in T , thus the ratio in (2) tends towards $\frac{1}{\sqrt{T}}$, and we have shown a no regret algorithm.

Of course, this requires knowing T beforehand, it turns out one can also achieve no regret (via a harder proof) by varying ϵ with time:

$$\epsilon_t = \frac{1}{\sqrt{t}}$$

4 Suggested Readings

1. Avrim Blum Online Learning survey (<http://www.cs.cmu.edu/~avrim/Papers/survey.ps.gz>)
2. Probability: The Logic of Science, <http://bayes.wustl.edu/etj/prob/book.pdf>, Chapters 1 and 2 (you can skim over the derivations); Probabilistic Robotics, Chapters 1 and 2