Statistical Techniques in Robotics (16-831, F10) Lecture#09 (Tuesday September 21)

Online Learning

Lecturer: Drew Bagnell *Scribe:Ben Eckart* [1]

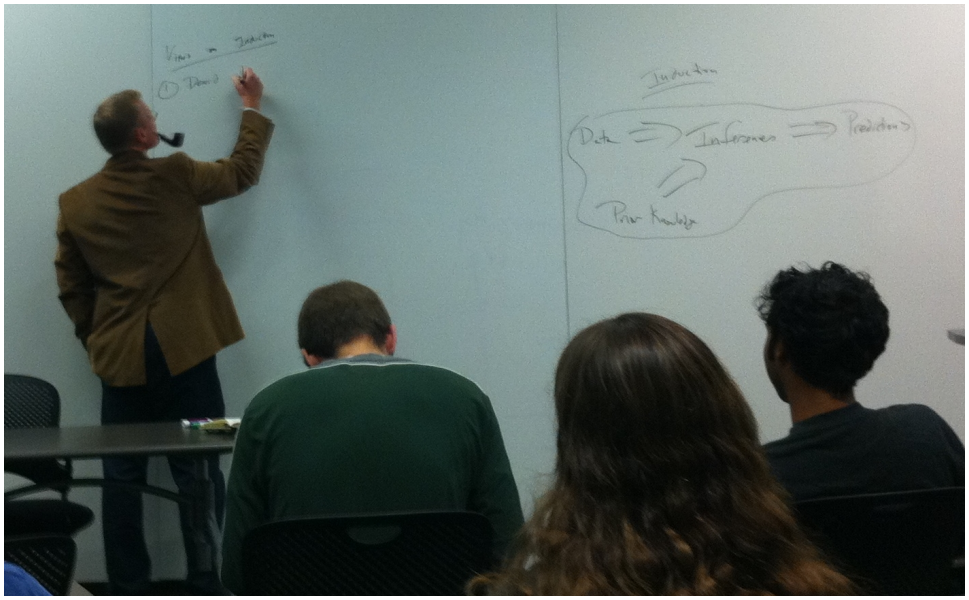

Figure 1: Drew Bagnell, with pipe and corduroy jacket, channeling the philosopher David Hume through elaborate chicken analogies.

1 Review from previous lecture

Note on Bayes Nets:

$$P(A,B) = P(A)P(B|A) = P(B)P(B|A) = \phi(A,B)/z$$

This means that $A \to B$ is equivalent to $B \to A$ and $A - -B$.

However, Bayes nets were originally made to encode causality. It is much easier for a doctor to tell the symptoms given a disease, than to tell the disease given the symptoms (since many diseases shared the same symptoms).

2 Online Learning

Online learning is concerned with making inductive decisions from limited information, using instances that are presented one at a time. This lecture begins by questioning the nature of induction

[1] Some content adapted from previous scribes: Mark Desnoyer, Andres Rodriguez

and its use as a tool for modeling and making decisions about the world. We will explore the limits of the inductive process, and in the process, prove how well different online induction algorithms perform in relation to each other through the new concepts of *loss* and *regret*.

3 Induction

Our standard justification for the use of probability theory is that it seems like the "reasonable" thing to do. Logic is certainly reasonable, and probability naturally and uniquely extends our notion of logic to include degree-of-belief type statements. Thus, if one accepts logic, one must accept probability as a way of reasoning about the world[2]. We call this the "normative" view: probabilistic logic is the way things *ought* to be. Intuitively this seems to be the case: Bayes' rule sets up a system by which we can arrive at what some would consider "common-sense" reasoning[3].

In all the probabilistic systems discussed so far (*e.g.* localization and mapping), we can momentarily discard our normative view by looking at their common elements. At the very core, something like mapping, for example, can be seen simply as an induction problem. We take in data about world, and using prior knowledge, we make inferences, and then predictions using the inferences. In fact, Bayes' rule can be seen as the "canonical" way to do these operations. Induction is a method to predict the future based on observations in the past, and to make decisions from limited information. Thus, we can look generally at induction itself in order to arrive at answers to certain fundamental questions behind many robotic learning systems. Questions such as: "Why does inductive reasoning work?", "When does it fail?", and "How good can it be?" will tell us about the nature of learning and lead us to unique notions of optimality with regard to worst-case adversarial conditions.

We begin with an illustrative example of how inductive reasoning can fail. Imagine you are a chicken. Every day of your life, the farmer comes to you at a certain time in the day and feeds you. On the 1001^{th} day, the farmer walks up to you. What do you predict? What can you predict other than that he will feed you? Unfortunately for you, this day the farmer has decided to eat you. Thus, induction fails spectacularly. You may object to this example and say that the model of the world that the chicken was using was wrong, but this is precisely the point of the discussion: perhaps we can *never* have a model that perfectly aligns with the real world. Perhaps, you (the chicken[4]) know vaguely of your impending doom, so you assign a low probability of being eaten everyday. You still will (probably) guess incorrectly on that 1001^{th} day because your guess of being fed will dominate your feeling of doom. Furthermore, for any model you bear, an adversarial or random farmer can always thwart it to force false inductive predictions about your fate each day.

Three views of induction:

1. *The David Hume view.* No matter what, your model will never capture enough of the world to be correct all the time. Furthermore, induction relies heavily on priors made from assumptions, and potentially small changes in a prior can completely change a future belief. If your model is as good as your assumptions or priors, and most assumptions are not likely to be

[2]However, these arguments say nothing about the computational feasibility of the reasoning models. We almost always fall back on to approximations for tractibility reasons.

[3]Although common-sense doesn't always seem so common-sense at times. See: Monty Hall problem, or the Bagpipes/GPA example from the graphical models lectures.

[4]Remember, Drew Bagnell is not a chicken. You are a chicken.

valid at all, then induction is fundamentally broken. We find this view, although true, is not very satisfying or particularily 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.* It is true that you cannot say that induction will work or work well for any given situation, but as far as strategies go, it's provably near-optimal. This view comes from the outgrowth of game theory, computer science, and machine learning. This thinking is uniquely late 20^{th} century.

4 On-line learning Algorithms

Though we can never be *certain* of our prediction (as a chicken, we will always be eaten at some point), we *can* form a strategy that will do nearly as well as we could possibly hope given our current situation. In other words, we can always do something which is the best we can do (and we can prove this is the case).

We begin by studying the problem of "predicting from expert advice." Our notion of an expert is simply something that makes a prediction. Unfortunately, "expert" is somewhat of a misnomer, given that an expert can give bad, random, or adversarial advice. It is fruitful to think of a collection of experts as a collection of hypotheses about the world.

Lets say each day we have the same n experts that predict from the set {fed, eaten}. Or perhaps a less morbid approach would be the example of predicting whether a given day will be sunny or rainy. In this case, the experts predict from {rainy, sunny}. After they make their prediction, the algorithm makes its own prediction, and then finds out if it actually it rains. Our goal is to perform nearly as well as the best expert so far (being competitive with respect to the best single expert). To quantify these intuitions, we will need the concepts of *loss* and *regret*.

4.1 Loss and Regret

To formalize the No Regret view, let's define a loss function L where:

$$L(wrong) = 1$$

$$L(correct) = 0$$

Therefore, for the optimal play, we want to minimize

$$\sum_t L(p_t) \to 0$$

Optimal play may be impossible. In this formulation, we cannot guarantee any performance (David Hume view) because the algorithm can be arbitrarily bad if the problem is hard to predict or adversarial.

We'll assume that we have some experts that predict either 0 or 1. Remember, the experts are arbitrary and can be based on a number of things. Examples:

- Always constant
- Markov expert (repeats what it last saw)
- Random
- Something complex (consults the color of the sky)
- etc..

If we consult N experts, and we want to do nearly as well as the best expert, we can define regret as:

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

where e^* is the best expert at time t

Therefore, our goal is to make regret scale more slowly than time, such that:

$$\lim_{t \to \infty} \frac{Regret}{t} = 0$$

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. This situation occurs when David Hume's pessimism reigns and no potential model for the world is very good. Minimizing regret in this case is like saying that we can do "as good as the best of the bad things." However, in situations where no regret results in low loss, Einstein wins out and we end up with a solution that models the world well and can make good predictions.

We will now discuss three attempts to minimize regret: follow the leader, majority vote, and weighted majority.

4.2 Try #1: Follow the Leader

We can also call this algorithm the "best in hindsight" algorithm. We naively pick the expert that has done the best so far and use that as our strategy for timestep t. Unfortunately, blindingly picking the leader can lead to overfitting. Consider an example with two experts; one always predicts 1 and the other always predicts 0. If the world alternates between 0 and 1, the algorithm will get every example wrong.

4.3 Try #2: 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.

4.4 Try #3: 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 when predicting the next majority vote.

More formally, this algorithm maintains a list of weights $w_1, ..., w_n$ (one for each expert $x_1, ..., x_n$), 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.

4.5 Randomized Weighted Majority Algorithm

In this algorithm, we view the weights as probabilities, and predict each outcome with 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

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

We want β to be small if we only have a few time steps available and vice-versa. 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.

4.6 General Weighted Majority Algorithm

Algorithm

1. Set all the weights to 1.
2. Predict expert i in proportion to w_i.
3. Receive the correct value y_t.
4. Adjust weights s.t. $w_i^{t+1} \rightarrow w_i^t \exp^{-\epsilon l(i,y_t)} \forall i$, where l is the lost function, and ϵ is the penalizer.
5. Goto 2.

Analysis The bound is

$$E[R] \leq \epsilon \sum l(i^*) + \frac{1}{\epsilon} \ln(n),$$

where R is the regret. If ϵ is large, we learn quickly. If ϵ is small, we learn slowly but we'll have little regret.

5 Intro to next lecture

In a convex set, the line between any two points in the set is also in the set. A convex function can be defined s.t.

$$f(ax_1 + bx_2) \leq af(x_1) + bf(x_2),$$

where $a + b = 1$, and $a, b \geq 0$.