Statistical Techniques in Robotics (16-831, S22) Lecture #03 (Wednesday, January 26)

Randomized Greedy and Halving Algorithms

Lecturer: Kris Kitani *Scribes: Kevin Gmelin, Jonathan Schwartz*

1 Review

In the first of the last two lectures, we discussed how to categorize feedback in a machine learning problem. In general, feedback is either Exhaustive or Sampled, either Instructive or Evaluative, and either One-Shot or Sequence. Note that feedback will fall into three of these categories, with one from each pair. In the second of the last two lectures, we looked at the Prediction With Expert Advice (PWEA) problem, in which feedback is (1) one-shot, (2) instructive, and (3) exhaustive. We investigated the Greedy algorithm for learning a solution to this problem.

1.1 Types of Feedback

The different types of feedback that we covered in class were:

1. Exhaustive vs Sampled
   - **Exhaustive:** will eventually have access to all possible states and actions
   - **Sampled:** can see results of only a sub-set of states and actions
2. Evaluative vs Instructive
   - **Evaluative:** information provided only for the action taken
   - **Instructive:** information provided for all possible actions
3. Sequence vs One-Shot
   - **Sequence:** the action taken at this time step will affect the next action
   - **One-Shot:** the action taken at this time step will not affect the next action

Applying these labels to the feedback received in a given problem will help us to categorize the problem and determine which algorithms to use.

1.2 Prediction with Expert Advice

In the most recent lecture, we were introduced to the PWEA problem, whose feedback is categorized as:

1. One-Shot: The action taken at one time step will not affect the action taken at the next time step.

2. Instructive: Information is provided on all possible actions that can be taken at each state.

3. Exhaustive: We will eventually have access to all possible states and actions.

1.3 Consistent (Greedy) Algorithm

The learning rule for this algorithm is **to pick any expert who has always been correct up to now.**

The algorithm itself looks like this:

Algorithm 1 Consistent (Greedy) Algorithm

$\mathbf{V}^{(1)} = \mathcal{H}$ ▷ Version Space initially stores all hypotheses
for $t = 1, \cdots, T$ **do**
  RECEIVE($\mathbf{x}^{(t)}$) ▷ expert advice
  $h = \text{FIRSTELEM}(V^t)$ ▷ 'greedy' selection
  $\hat{y}^{(t)} = h(\mathbf{x}^{(t)})$ ▷ predict outcome using expert
  RECEIVE($y^{(t)}$) ▷ true outcome
  $\mathbf{V}^{(t+1)} = \{h \in \mathbf{V}^{(t)} : h(\mathbf{x}^{(t)}) = y^{(t)}\}$ ▷ keep only consistent hypotheses
end for

For this algorithm to be effective, at least one of the experts needs to always be correct. That makes the problem realizable.

Definition 1. Realizability *is the assumption that the learner has access to the perfect hypothesis.*

This expression can be expressed as:

$$h^* \in \mathcal{H}$$

Where h^* is a hypothesis that is always correct, and $\mathcal{H}$ is the class of all possible hypotheses.

We also spent some time analyzing this algorithm, and observed that the number of elements in the version space $\mathbf{V}$ after M mistakes is:

$$|\mathbf{V}^{(t+1)}| \leq |\mathcal{H}| - M$$

If we assume realizability, then the maximum number of mistakes that this algorithm can make is:

$$M_C(\mathcal{H}) \leq |\mathcal{H}| - 1$$

Lastly, we summarized the strategy for deriving the mistake bounds for an algorithm. The general process is as follows:

1. Define **'potential'** function (size of the version space)

2. **Upper bound** the potential function

3. **Lower bound** the potential function
4. **Combine** bounds
5. **Algebra**/approximation to get performance bound

2 Summary

2.1 Halving (Majority) Algorithm

In this lecture, we looked at the Halving Algorithm, which improves upon the Consistent Algorithm from the last lecture.

Definition 2. Halving Algorithm *predicts using majority vote of experts who have not made any mistakes.*

The algorithm works as follows:

Algorithm 2 Halving (Majority) Algorithm

1: $\mathbf{V}^{(1)} = \mathcal{H}$ ▷ Version Space initially stores all hypotheses
2: **for** $t = 1, \cdots, T$ **do**
3: RECEIVE $(\mathbf{x}^{(t)})$ ▷ expert advice
4: $\hat{y}^{(t)} = \text{MAJORITYCONSENSUS}(\mathbf{V}^{(t)}, \mathbf{x}^{(t)})$ ▷ predict outcome using expert
5: RECEIVE$(y^{(t)})$ ▷ true outcome
6: $\mathbf{V}^{(t+1)} = \{h \in \mathbf{V}^{(t)} : h(\mathbf{x}^{(t)}) = y^{(t)}\}$ ▷ keep only consistent hypotheses
7: **end for**

In this algorithm, the version space is initially the entire hypothesis class. Each trial begins with the algorithm receiving expert advice. A prediction is then made based on the majority vote of the remaining hypotheses in the version space. The true outcome is then received, and all hypotheses that were incorrect are removed from the version space. The Halving Algorithm works very similarly to the Consistent Algorithm, with the exception of line 4. Instead of picking the first consistent expert and using their hypothesis, the Halving algorithm uses the majority vote of the consistent experts to make its decision.

We can derive a bound on the number of mistakes that the Halving algorithm will make by using a similar method to that used for finding a bound on the number of mistakes made by the Greedy algorithm. First, we observe that the version space will be reduced to at most half of its size after a mistake, since at least half of the hypotheses would need to be incorrect for the Halving algorithm to make a mistake:

$$|\mathbf{V}^{t+1}| \leq \tfrac{1}{2}|\mathbf{V}^{t}|$$

This information can be used to derive an expression for the size of the version space after M mistakes:

$$|\mathbf{V}^{t}| \leq (\tfrac{1}{2})^{M}|\mathcal{H}|$$

Assuming realizability allows us to write an expression for the lower bound of the version space:

$$|\mathbf{V}^t| \geq 1$$

Putting these two bounds together gives the expression:

$$1 \leq |\mathbf{V}^{(t)}| \leq 2^{-M}|\mathcal{H}|$$

We can remove the middle term of this expression, since it has less than or equal to signs on both sides, leaving:

$$1 \leq 2^{-M}|\mathcal{H}|$$

Taking the logarithm of both sides of this expression gives:

$$0 \leq -M + \log_2 |\mathcal{H}|$$

This gives a bound on the total number of mistakes that will be made by the halving algorithm:

$$M_{halving}(\mathcal{H}) \leq \log_2 |\mathcal{H}|$$

This is a significantly better mistake bound than that of the Consistent algorithm, whose upper bound on total number of mistakes grows linearly with the size of the hypothesis class while the Halving Algorithm's mistake bound only grows logarithmically.

2.2 Randomized Greedy

Another prediction with expert advice algorithm is the randomized algorithm, which is shown below in Algorithm 3.

Algorithm 3 Randomized Greedy Algorithm

$\mathbf{V}^{(1)} = \mathcal{H}$ ▷ Version space initialized to hypothesis class
for $t = 1, \cdots, T$ **do**
  RECEIVE $(\mathbf{x}^{(t)})$ ▷ Receive expert advice
  $h \sim$ UNIFORM$(\mathbf{V}^{(t)})$ ▷ Select random hypothesis
  $\hat{y}^{(t)} = h(\mathbf{x}^{(t)})$ ▷ Prediction
  RECEIVE $(y^{(t)})$ ▷ Receive true outcome
  $\mathbf{V}^{(t+1)}) = \{h \in \mathbf{V}^{(t)} : h(\mathbf{x}^{(t)}) = y^{(t)}\}$ ▷ Update by keeping only consistent hypotheses
end for

In this algorithm, a version space of consistent hypotheses is maintained. The version space is first initialized to be the same as the hypothesis class. Then, during each trial, the algorithm first

recieves the expert advice. Then, a hypothesis is randomly chosen from the version space. This hypothesis is used to map the expert advice to a prediction. The algorithm then recieves the true outcome. Finally, all hypotheses in the version space that would have mapped the expert advice to a wrong prediction are removed from the version space. This ensures that the version space only contains hypotheses that would have mapped all past expert advice to the true outcome.

The randomized greedy algorithm is very similar to the greedy algorithm, with the only difference being how a hypothesis is chosen from the version space. In the regular greedy algorithm, the first hypothesis in the version space is chosen for making a prediction. In the randomized greedy algorithm, a random hypothesis in the version space is used.

Theorem 3. The expected number of mistakes made by the randomized greedy algorithm has an upper bound of $ln|\mathcal{H}|$ given realizability.

To prove this, we first create a lower bound for the size of the version space using the assumption of realizability:

$$|\mathbf{V}^{(t)}| \geq 1$$

Next, we note that after each trial, we can write the new size of the version space as:

$$|\mathbf{V}^{(t+1)}| = \alpha^{(t)} \cdot |\mathbf{V}^{(t)}|$$

where $\alpha^{(t)}$ is the ratio of remaining experts after trial t. This ratio has a subscript t because it varies from trial to trial.

After T trials, the size of the version space is:

$$|\mathbf{V}^{(T+1)}| = |\mathbf{V}^{(1)}| \cdot \prod_{t=1}^{T} \alpha^{(t)} = |\mathcal{H}| \cdot \prod_{t=1}^{T} \alpha^{(t)}$$

where the right hand equality comes from the fact that the version space is initialized to be the hypothesis class.

Using the inequality $b \leq e^{-(1-b)}$, we get:

$$|\mathbf{V}^{(T+1)}| \leq |\mathcal{H}| \cdot \prod_{t=1}^{T} e^{-(1-\alpha^{(t)})} = |\mathcal{H}| \cdot e^{-\sum_{t=1}^{T}(1-\alpha^{(t)})}$$

We have thus arrived at an upper bound for the size of the version space. To understand this upper bound, we examine the sum in the exponent:

$$1 - \alpha_t = 1 - \frac{|\mathbf{V}^{(t+1)}|}{|\mathbf{V}^{(t)}|} = \frac{|\mathbf{V}^{(t)}| - |\mathbf{V}^{(t+1)}|}{|\mathbf{V}^{(t)}|}$$

The numerand of this expression for $1 - \alpha_t$ is the number of hypotheses removed from the version space after conducting trial t, and thus this numerand is the number of inconsistent hypotheses.

Since we are dividing this numerand by the size of the verion space, it becomes clear that $1 - \alpha_t$ is equal to the probability of choosing a wrong hypothesis and thus it is the probability of making a mistake at trial t. Thus:

$$1 - \alpha^{(t)} = p(\hat{y}^{(t)} \neq y^{(t)})$$

$$\sum_{t=1}^{T}(1 - \alpha^{(t)}) = \sum_{t=1}^{T}(p(\hat{y}^{(t)} \neq y^{(t)})) = \mathbb{E}[M^{(T)}]$$

where $\mathbb{E}[M^{(T)}]$ is the expected total number of mistakes after trial T. We can rewrite the upper bound on the version space as:

$$|\mathbf{V}^{(T+1)}| \leq |\mathcal{H}| \cdot e^{-\mathbb{E}[M^{(T)}]}$$

Combining the lower and upper bound on version space size:

$$1 \leq |\mathbf{V}^{(T+1)}| \leq |\mathcal{H}| \cdot e^{-\mathbb{E}[M^{(T)}]}$$

Removing the middle expression and performing some algebra, we get:

$$1 \leq |\mathcal{H}| \cdot e^{-\mathbb{E}[M^{(T)}]}$$

$$0 \leq ln|\mathcal{H}| - \mathbb{E}[M^{(T)}]$$

$$\mathbb{E}[M^{(T)}] \leq ln|\mathcal{H}|$$

We have arrived at an upper bound on the expected total number of mistakes. Note that unlike previous performance bounds, this is not a bound on the total number of mistakes, but rather, on the average total number of mistakes. The randomized greedy algorithm has the same worst case performance as the normal greedy algorithm, which makes sense because it is possible that the randomized algorithm ends up selecting the first hypothesis every trial and behaving identical to the non-randomized greedy algorithm.

This may raise the question of why the randomized greedy algorithm would be used over the normal greedy algorithm. Reasons for using a randomized algorithm are given in [1], which states that for randomized algorithms, "no particular input elicits its worst-case behavior". The regular greedy algorithm's performance could be affected by the initial order of hypotheses specified when the version space is initialized. It's possible you could have an initial order of hypotheses that has degraded performance for the particular data distributions you are dealing with. Furthermore, if the data is being generated by an adversary, the adversary could use knowledge of your initial order of hypotheses against you to make your normal greedy algorithm perform poorly with lots of mistakes. The randomized greedy algorithm alleviates these concerns because the initial order of hypotheses becomes irrelevant to the total number of mistakes made.

2.3 Regret

Previous theorems and analyses of performance bounds assumed realizability, where at least one hypothesis in the hypothesis class is perfect. Performance was measured in terms of the total number of mistakes made, with success tied to being lucky and quickly choosing the perfect hypothesis in the hypothesis class. However, for some prediction with expert advice problems, realizability may not hold, and thus no hypothesis in the hypothesis class would make 0 mistakes. Without realizability, the total number of mistakes of an algorithm is less informative because we don't know how the algorithm compares to the best hypothesis in the hypothesis class. In the end, we want our online learner to have similar performance as the best fixed hypothesis. This leads us to developing a new measure of performance called regret:

Definition 4. Regret *is the difference between the cumulative loss of the online learner and the cumulative loss of a single hypothesis.*

$$R^{(T)}(h) = \sum_{t=1}^{T} L(\hat{y}^{(t)}, y^{(t)}) - \sum_{t=1}^{T} L(h(x^{(t)}), y^{(t)})$$

where L is a loss function and $x^{(t)}$ is the expert advice received at trial t. It is important to note that this definition of regret is relative to a specified fixed hypothesis. If we want the performance relative to the entire hypothesis class, we use external regret:

Definition 5. External Regret *is the difference between the cumulative loss of the online learner and the cumulative loss of the best hypothesis in the hypothesis class.*

$$R^{(T)}(\mathcal{H}) = \sum_{t=1}^{T} L(\hat{y}^{(t)}, y^{(t)}) - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} L(h(x^{(t)}), y^{(t)})$$

$$R^{(T)}(\mathcal{H}) = \max_{h \in \mathcal{H}} R^{(T)}(h)$$

We can define a third performance metric, average regret:

Definition 6. Average Regret *is the difference between the average loss of the online learner and the average loss of the best hypothesis in the hypothesis class.*

$$R^{(T)}_{avg} = \frac{1}{T} R^{(T)}(\mathcal{H})$$

In general, we want online learning algorithms whose performance is similar to the performance of the best fixed hypothesis. More formally, we want algorithms whose average regret goes to zero as the number of trials goes to infinity. We call these algorithms 'no regret' algorithms:

Definition 7. No Regret Algorithms *are online learning algorithms whose average regret goes to 0 in the limit as the number of trials approaches infinity.*

In the long run, no regret algorithms perform no worse than the best hypothesis in the hypothesis class.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.