Homework Assignment 2
CptS 577 Structured Prediction, Spring 2019
Abhijay Ghildyal

1

For a recurrent classifier in the case of a sequence prediction problem from a sequence ($s \in S$) of length T, the expected error at time i when following $\hat{\pi}$ can be denoted by,

$$\epsilon_i = \mathbb{E}_{s \sim d_\pi^i}[e_{\hat{\pi}}(s)]$$

The general error calculation is as follows,

$$\epsilon = \frac{1}{T} \sum_i \epsilon_i$$

We know that in the case of a recurrent classifier the total cost to go is denoted by,

$$J(\pi) = \sum_{i=1}^{T} \mathbb{E}_{s \sim d_\pi^t}[C_\pi(s)]$$

where $C(s, a)$ may or may not be true costs and seek to bound $J(\pi)$ for any cost function C based on the performance of current π compared to the expert's π^*.

Hence, the error at each step w.r.t the cost to go can be denoted as,

$$J(\hat{\pi}) \leq J(\pi^*) + \sum_{t=1}^{T} \sum_{i=1}^{t} \epsilon_i$$

i.e.,

$$J(\hat{\pi}) \leq J(\pi^*) + T^2 \epsilon$$

So, with the assumption that due to $\hat{\pi}$ there was a mistake it can be stated that an error compounds which is like an extra cost that grows quadratically in T. This, the behavior of the error is as $O(T^2\epsilon)$ for small ϵ

For **exact imitation** the expert's data is taken and a learner is trained on that data. This way the learner learns the policy of the expert over the states as seen by the expert. The assumption taken over here is that the actions of the expert seen in different trajectories are independent identically distributed (i.i.d). The issue with this approach to imitation learning is that, the learner does learn to recover from failures. For example, if the learned model deviates from an optimal trajectory at a particular time step, since it has not learned to correct itself, it will be hard for it to be able to get back to states seen by the expert thus generating a cascade of errors. This naive approach does not generalize to unseen situations.

As introduced by Ross and Bagnell in 2010, the **forward training** algorithm trains one policy π_t at each time step t over T (non-stationary policy). This means that at each t, the learner learns a policy π_t which is similar to the expert's policy π^*. This optimal policy is used to generate data from the states (which are T step trajectories) seen by the learner, induced by the previously learned policies $\pi_1, ...\pi_{t-1}$. In this if not the worst case scenario the expert's policies help in rectifying the mistakes of the learner's learned policy and thus showing better performance. The problem here is that the time horizon T can be really large or even undefined and since the policy is non-stationary, the algorithm becomes impracticable for real-world applications.

DAgger (or dataset aggregation) based on Follow-The-Leader strategy considers all errors from agreeing with the expert's policy equally. It proceeds by collecting a dataset at each iteration as per the current policy the learner has learned. It then trains the next policy after the aggregation of expert's prediction/action on the states generated by the learner's policy. So at any iteration the best policy is selected in hindsight, i.e. under all trajectories seen so far over all the iterations. The advantage with this approach is that the expert presents the learner with examples on how it can recover from mistakes it made in the past.

SEARN, when compared to DAgger reasons about cost-to-go. It does rollouts from the current rollin state reached after following the current policy. It additionally requires the use of stochastic policies. DAgger aggreagates a dataset (obtained from iterating on the dataset) to train on, while on the contrary SEARN does not aggregate a dataset.

AggreVaTe collects data by observing the expert's actions the task. It does a rollin and then a rollout of trajectories just like SEARN but then aggregates the data of the cost to go. In each trajectory, at a uniformly random time t, it explores an action a in state s, while observing the cost-to-go (Q) of the expert after performing this action. This cost to go value of the expert indicates the amount of cost it may incur in the future on taking this action. This collection of data provides an approximate estimate of the kind of policies it will be having in the future. On minimizing this cost-to-go it can be said that the learn will slowly converge to better policies in the future.

Algorithm 1: Unified Framework for Imitation Learning

Training data S, optimal policy π^*, learning rate β ;
Data D = $\emptyset$;
for $i \leftarrow 1$ *to* N **do**
  if *Forward Training* **then**
    Sequentially learn classifier $h_i + 1$ based on the distribution induced by h_i
    for i= *1 to* T **do**
      Sample a T step trajectory by following different $\pi_{1:i-1}$
      Get dataset $D = (s_i, \pi(s_i))$ // based on expert
      $\pi_i = argmin_{\pi \in \prod} \sum_{(s,a) \in D} loss(s, a, \pi)$
    end
    $\pi(s, t) = \pi_t(s)$
    return π
  else
    $p = (1 - \beta)^{i-1}$;
    current-policy $\pi = p\pi^* + (1 - p)H_{i-1}$;
    if *SEARN* **then**
      Data D = $\emptyset$
    end
    for s *in* S **do**
      // s is a sequence in 1:T
      for $\hat{y}_t$ *in* $\pi(s)$ **do**
        // Make features $\phi_t = f(s, \hat{y}_{1:t-1})$
        if *Exact Imitation* **then**
          $D_i = (\phi_t, y_t)$
        else if *DAGGER* **then**
          // Follow the leader strategy
          $\hat{y}_t = \pi(s|\phi_t)$ // As per actions given by current policy
          $D_i = (\phi_t, \hat{y}_t)$ // i.e. in this case $D_i = (s, \pi^*(s))$
        else if *SEARN or AGGREVATE* **then**
          for *each possible action (or rollout trajectory)* y_t^j **do**
            // This is just like executing current policy π up to time T for some exploration action
            // Execute expert from time t+1 to T , and observe estimate of cost-to-go starting at time t
            $y'_{t+1:T} = \pi(s|\hat{y}_{1:T}, y_t^j)$ // Cost to go for the trajectory
            Estimate $c_t^j = loss(\hat{y}_{1:T}, y_t^j, y'_{t+1:T})$
            $D_i = (\phi_t, c_t)$ // Cost sensitive example
            // This is the same as $D_i = (s, t, a, \hat{Q})$
          end
        if *SEARN* **then**
          $h_j = CostSensitiveClassifier(D)$
        else
          $D \leftarrow D \cup D_i$ //Aggregate Dataset
      end
    end
    if *SEARN* **then**
      $H_i = \beta \sum_{j=1}^{i} \frac{(1-\beta)^{i-j}}{1-(1-\beta)^i} h_j$
    else
      $H_i = CostSensitiveClassifier(D)$
    end
end
return Best H_i on validation set

3

When applying imitation learning approaches it is necessary that the gold label is available. Especially in the case of a recurrent classifier the disadvantage is higher because the missing label in a sequence makes the whole sequence (a set of observations) not useful if not treated properly.

Remove data with missing labels: If the data is available in abundance then the sequences with missing labels can be discarded. If the data is limited then discarding some of it would lead to loss in information. In the case of imitation learning since the improvement of a classifier's accuracy depends highly on demonstration data, it is utmost importance to find a better solution than to just remove data with missing labels. **It should be useful to atleast use some part of the sequence which can help generate a feature set for the data aggregation step** but this still doesn't solve the issue of loss in valuable data.

Impute data: One method followed widely is to impute the missing value with the median observed in training data. This might not work well with a recurrent classifier as it is a multilabel setup and the error propagation due to incorrect labels will be large.

Generating a value for the missing label using a learner: A recurrent classifier can be trained on the training dataset without the missing values. If the classifier is able to predict a label (for the missing) with a certain confidence then it would be worth imputing. So, in the case of DAgger and AggreVaTe the newly available predicted data can be used to perform imitation learning but if the performance of the sub optimal classifier does not improve with the addition, then the data sequence can be discarded. In a gist, the newly learned classifier can be treated as an expert which can help generate data and help the more sub optimal classifier.

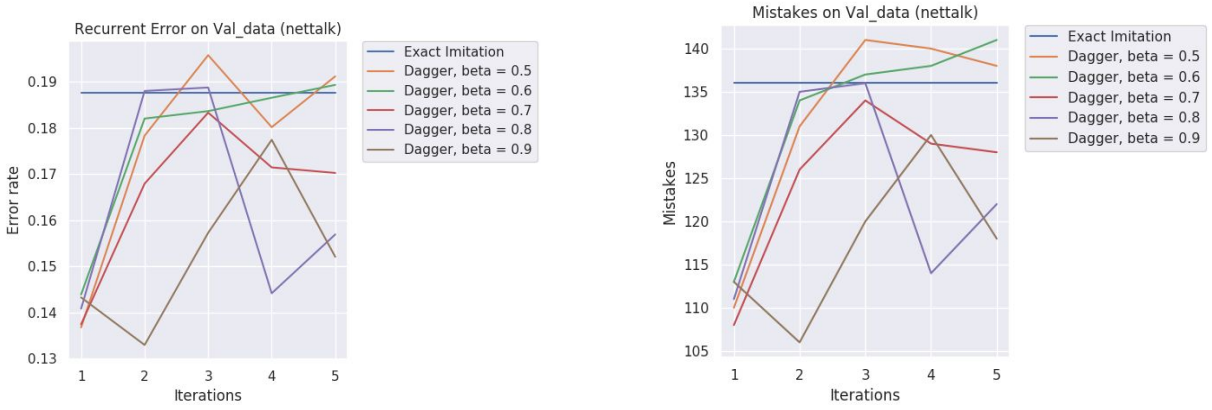
Masking the missing data: In the case of a recurrent classifier if the comb structure is formulated into a network then some portion of the structure can be masked which would be easy to implement on the network too. After a certain classifier has been trained then it can be used to predict those missing values. This way some of the data which is affected by the missing label in sequence can still be used in parts for a classifier and with the help of DAgger or AggreVaTe somewhat of a generalized classifier can trained which will predict the missing labels to generate more data for learning a more robust classifier.

4.

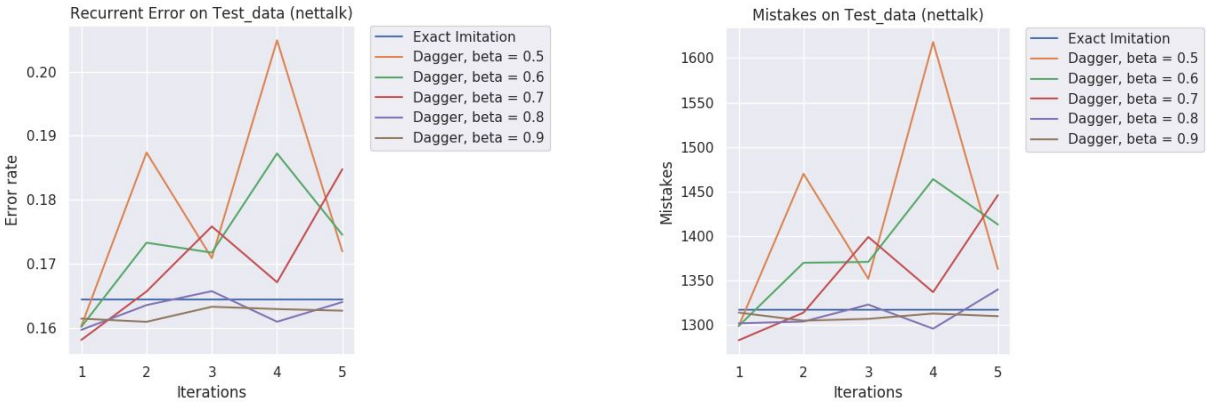
"To better leverage the presence of the expert in our imitation learning setting, we optionally allow the algorithm to use a modified policy π_i = βi π + (1 − βi) π_i at iteration i that queries the expert to choose controls a fraction of the time while collecting the next dataset. This is often desirable in practice as the first few policies, with relatively few datapoints, may make many more mistakes and visit states that are irrelevant as the policy improves."*

- A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, *Stéphane Ross, Geoffrey J. Gordon, J. Andrew Bagnell*

Though in the case of recurrent classifier that I implemented using linear SVM, with a decaying beta, the number of examples with y history predicted features getting added (i.e. mistakes) lowered the accuracy and generalizability of the model. With a constant beta the number of examples with mistakes (incorrect y_hat in history) getting added were less and this helped in increasing the performance using Dagger. A smaller y_history was used (around 2-3) in the features so as to prevent the model from overfitting. Since the y_history features were adding sparsity to the feature set (history_len * no_lables), especially in the case OCR dataset with 26 labels, SVM seems to be the ideal choice for a linear classifier being a max margin classifier, trying to create a decision boundary for a wide as possible gap in separating categories.



From the recurrent error and no. of mistakes plot on validation data it can observed that there is scope for improvement using Dagger.



With beta = 0.9 the examples added via Dagger helped the classifier to improve. Though with lesser values of beta the number of examples getting added in every iteration were a lot higher. This led to the addition of a number of bad examples which affected the generalizability and performance.

Applying Dagger on the OCR dataset seems to be bringing a lot of improvement as per the results from validation dataset

Though from the results on the test dataset it is likely that with the aggregation of more data it is only overfitting on the distribution it has in the training dataset. This is highly likely because the size of the dataset is small. More reason for the poor performance on test dataset lies in one of the major drawbacks of Dagger:

“While iterative training corrects the compounding of error effect one sees in control and decision making applications, it does not address all issues that arise. Consider, for instance, a problem of learning to drive near the edge of a cliff: methods like DAGGER consider all errors from agreeing with the expert driver equally. If driving immediately off the cliff makes the expert easy to imitate – because the expert simply chooses the go straight from then on– these approaches may learn that very poor policy. More generally, a method that only reasons about agreement with a demonstrator instead of the long term costs of various errors may poorly trade-off inevitable mistakes.” - Reinforcement and Imitation Learning via Interactive No-Regret Learning, *Stéphane Ross, J. Andrew Bagnell*