Statistical Techniques in Robotics 16-831 - The Robotics Institute

Project 2: List Prediction and Online SVM

Deadline: Nov 9th, 2015 11:59 PM (submit via Blackboard)

Jiaji Zhou and Sanjiban Choudhury

1 List Prediction

1.1 Introduction

In this assignment, you will explore the basic theory and implementation of list prediction. The theoretical questions are designed to introduce the notion of submodularity and its connection to modelling list prediction problems. Then we will look at an application to robot manipulator trajectory library optimization, where we also draw connection between list prediction algorithms and imitation learning. For details, you are encouraged to read [1, 2].

1.2 Theory Questions (25 points)

1.2.1 Monotone Submodularity (5 points)

Let S be the set of all candidate items and f be a function defined over lists L whose elements belong to S. Define marginal benefit $b(s|L) = f(L \oplus s) - f(L)$, where the operator $\oplus$ means list concatenation. In class, we looked at f that is monotone and submodular.

1. Monotonicity: $f(L_1) \leq f(L_1 \oplus L_2)$ and $f(L_2) \leq f(L_2 \oplus L_1)$ for any list L_1 and L_2;
2. Submodularity (dimishing return): $b(s|L_1) \geq b(s|L_1 \oplus L_2)$ for any list L_1 and L_2 and element s.

A common monotone submodular function is the *"multiple guess"* function $f(L;T) = \min(1, |L \cap T|)$, where T is a subset of S. For example, f could be whether a particular user clicks the given displayed ads or not, T is the set of ads the user would click based on his personal interests, S is the set of all candidate ads, and L is a list of ads displayed on the users' phone screen. We are happy if the user clicks at least one ad, and we are not happy if the user ignores all of them.

Show the *"multiple guess"* function is monotone and submodular.

1.2.2 Greedy Guarantee (20 points)

In class, we talked about that fact that f (monotone submodular) can be greedily optimized with a near-optimal multiplicative guarantee. Now let's prove it. Note that the employed proof techniques are pretty common for iterative algorithms.

Let L^G be the size-k list constructed greedily by choosing the item with the highest marginal benefit, namely $L^G_{i+1} = L^G_i \oplus \arg\max_{s \in S} b(s|L^G_i)$, where L^G_i is the prefix list consisting of the first i items. Denote as L^* the optimal size-k list in terms of maximizing f.

First, let's show that the gap $\Delta_i = f(L^*) - f(L^G_{i-1})$ between the optimal list L^* and L^G is shrinking in a multiplicative fashion.

- Step 1: Prove that $\Delta_i \leq \sum_{j=1}^{k} \{f(L^G_{i-1} \oplus l^*_j) - f(L^G_{i-1})\}$, where l^*_j is the jth element of L^*. (7 points)
- Step 2: Prove that $\Delta_{i+1} \leq (1 - 1/k)\Delta_i$. (8 points)

Note that $\Delta_1 = f(L^*)$ and $\Delta_{k+1} = f(L^*) - f(L^G)$ and we have shown $\Delta_{i+1} \leq (1 - 1/k)\Delta_i$, there is only more step to prove the final theorem.

- Step 3: Prove that $f(L^G) \geq (1 - 1/e) f(L^*)$. (5 points)

(Hint: You may need to use this inequality: $(1 - 1/n)^n \leq 1/e$.)

1.3 Coding (30-35 points)

Submission instructions (Important)

Please include the required plots and text analysis in the write-ups! We cannot grade this question if they do not appear in the write-up. Make sure the captions are related to the figures instead of being all same. You may choose between Matlab or Python. Dependencies (if any) need to be included so we can run your code. Provide a script for us to generate the plots in your write-up. **Name the script as "gen_plots.m" or "gen_plots.py"**.

1.3.1 Introduction

In this coding problem, you will apply list prediction techniques to a manipulator trajectory selection problem. The goal of your algorithm is to predict a list of initial seed trajectories that maximizes the chance of containing at least one collision-free trajectory after local optimization, which is equivalent to maximizing a multiple-guess function.

We use the dataset from [1]. It consists of 310 training and 212 test environments of random obstacle configurations around a target object, and 30 initial seed trajectories. In each environment, each seed trajectory has 17 features describing the spatial properties of the trajectory relative to obstacles.

The feature and label files ("feat_train", "feat_test", "result_train", "result_test") are in the "csv_data" folder. If you use Matlab, you can simply load "data.mat" which contains everything.

- `feat_train` contains 310*30 = 9300 rows and 17 columns. The features of the seed trajectory i for training environment j is the $30*(j-1)+i$-th row vector.
- `result_train` contains the binary label of each trajectory for training environments. Label 1 indicates collision-free result. Label 0 indicates collision.
- `feat_test` is similar to `feat_train`. It contains 212*30 = 6360 rows.
- `result_test` is similar to `result_train`.
- `evaluateSingleSlotPrediction.m` is provided as a reference matlab file for evaluating success probablity given predictions of the first single slot.

Our objective is to optimize the average value of multiple-guess functions (average success ratio) over all environments.

1.3.2 Naive prediction strategy (10 points)

Train a learner π that predicts whether a particular seed trajectory for an environment with given features will be collision-free or not. For each test environment, a naive strategy is to construct a size-k list using the top-k trajectories sorted by the learner's predictive scores. You may use your favorite learning algorithm for π (e.g., linear regression, linear/kernel svm, random forest, etc.) as long as it gives some kind of predictive/confidence scores.

Plot your results with the x-axis as the size of list k (ranges from 1 to 8) and y-axis as the success ratio (averaged over the environments) for both training and testing. Give brief interpretations for the results.

1.3.3 List prediction strategy (20-25 points)

One key aspect for improving the result is to increase the diversity of the predicted list. The intuition for list prediction algorithms like ConSeqOpt[1] or SCP[2] is to let the greedy algorithm explicitly guide/teach the training procedure. Greedy algorithm will try to avoid picking similar items that already appeared in the list since their marginal benefits are likely to be small. In class, we have talked about using a sequence of k learner π_i to produce the list. Algorithm 1 summarizes the training procedure. Suppose that for each round of training, π_i preserves the exact ranking of the labels/marginal benefit, it will act exactly as what greedy algorithm does. However, the ranking would usually be deviated from perfect, but again you can intuitively imagine that the learning algorithm tries hard to maximize marginal benefit by following what greedy algorithm would do given the current predicted imperfect list L_{i-1}.

As for prediction on the test data, we simply call π_i sequentially to pick the top item, with optional feature update for each slot, until a full list of size-k is produced.

1. Plot your results with the x-axis as the size of list k (ranges from 1 to 8) and y-axis as the success ratios (averaged over the environments) for both training and testing. Give brief interpretation of the results.

Algorithm 1 Training procedure for list prediction algorithm (ConSeqOpt) to produce a sequence of learners.

Input: M numbers of environment E_j, each with training features X_j and label Y_j.
Output: a sequence of k learners $\{\pi_1, \pi_2, \ldots, \pi_k\}$.
for $i = 1$ **to** k **do**
  Train π_i with all X_j and Y_j (use features and labels across all environments).
  For each environment E_j: let π_i pick the top item (trajectory) $e_i^j \in E_j$ and append it to the list, i.e., $L_i^j = L_{i-1}^j \oplus e_i^j$.
  For each environment E_j: (1) update the label y for each item e as the new marginal benefit $b(e|L_i^j)$. (2) optionally update the feature x for each item e to incorporate similarity with respect to the items already in L_i^j.
end for

2. Your code should consist of modules or functions that include: 1) Training a new learner for a new slot (list position) with updated features and labels; 2) predicting the marginal utilities for all items given features and pick the best; 3) labels update; 4) optional features update.

3. Extra credit: If you implement feature update to incorporate similarity with respect to existing items in the list, you can get up to 5 points of extra credit. In practice, designs of similarity features are key factors to performance improvement. Show in the plots how similarity features help. (Hint: you can define some similarity metric in the feature space between one single item to a set/list of items and append it/them to the existing feature vector for new learner training.)

2 Online Support Vector Machines

2.1 Introduction

Support vector machines are powerful algorithms for solving the binary classification problem. The vanilla version of SVM is a batch algorithm. Thus when the size of the dataset is immensely large, the algorithm runs into computation / memory issues. We will now see how framing this as an online learning problem helps us overcome these issues.

2.2 Derivation (20 points)

We will now focus on the binary classification problem. We have a dataset $\mathscr{D} = (f_i, y_i)$ where $i = 1, 2, \ldots, T$. Each element is a pair of feature vector, $f_i \in \mathbb{R}^n$, and label $y_i \in \{-1, 1\}$. Our objective is to learn a linear classifier w such that

$$\begin{aligned} w^T f_i > 0 \quad \text{if} \quad y_i = +1 \\ w^T f_i < 0 \quad \text{if} \quad y_i = -1 \end{aligned} \tag{1}$$

In otherwords, we want to ensure $y_i w^T f_i \geq 0$.

This problem is now framed as a soft margin support vector machine. This is described as

$$\begin{aligned} \min_{w,\xi} \ & \frac{\lambda}{2}\|w\|^2 + \sum_{i=1}^{T} \xi_i \\ & \xi_i \geq 0 \\ & y_i w^T f_i \geq 1 - \xi_i \end{aligned} \tag{2}$$

This is a quadratic programming problem and can be tedious to solve repeatedly in an online fashion. However, it turns out that this can be reformulated as an unconstrained optimization problem.

$$\min_{w} \frac{\lambda}{2}\|w\|^2 + \sum_{i=1}^{T} \max(0, 1 - y_i w^T f_i) \tag{3}$$

2.2.1 Prove that Equation (2) and (3) are equivalent (10 points)

We can now think about solving (3) in an online fashion. For convenience we will re-arrange the optimization problem as

$$\min_{w} \sum_{i=1}^{T} \frac{\lambda}{2T}\|w\|^2 + \max(0, 1 - y_i w^T f_i) \tag{4}$$

2.2.2 Prove that Equation (4) is a convex optimization problem (5 points)

Since it is an unconstrained optimization problem, simply showing the objective function is a convex function in w will suffice.

We will now solve this problem in an online setting using online sub-gradient descent. The online setting is a game between our learner and nature which takes place over timesteps $t = 1, 2, \ldots, T$. At each time step t, we will provide a linear classifier w_t. Nature reveals f_t and y_t. We then have a loss function

$$l_t(w) = \frac{\lambda}{2T}\|w\|^2 + \max(0, 1 - y_t w^T f_t) \tag{5}$$

We will then update w_t using the sub-gradient descent update rule

$$w_{t+1} = w_t - \alpha_t \nabla l_t(w_t) \tag{6}$$

where α_t is the step size.

Notice that we use the term sub-gradient descent. This is because the loss function is not differentiable. We define a subgradient $\nabla l(x)$ at x as any vector that is normal to $l(x)$, and is below the rest of l. We write this as:

$$l(y) \geq l(x) + \nabla l(x)^T (y - x) \qquad \forall y \tag{7}$$

2.2.3 Prove that a valid sub-gradient $\nabla l_t(w)$ for Equation 5 is as follows (5 points)

$$\begin{aligned} \nabla l_t(w) &= \frac{\lambda}{T} w - y_t f_t \quad \text{if } 1 - y_t w^T f_t > 0 \\ &= \frac{\lambda}{T} w \qquad\qquad \text{otherwise} \end{aligned} \tag{8}$$

Now we have a valid update rule for the weights w_t. Setting the learning rate $\alpha_t = \frac{\eta}{\sqrt{t}}$ results in a no-regret online learning algorithm.

2.3 Implementation (30 points)

This assignment gives you a chance to implement online support vector machines on real data.

Automated interpretation of ladar data is crucial for outdoor vehicle operation. You will be building a system to classify ladar points into one of five categories: ground (supporting surface), vegetation, facade, pole, and wire.

Implement the online support vector machine algorithm that we derived. Since we derived a binary classifier, choose 2 out of the 5 classes. Use appropriate values of λ and η.

The dataset you should use are available in **online_svm_data.zip**. They are 3D point-clouds of Oakland (taken from http://www.cs.cmu.edu/~vmr/datasets/oakland_3d/cvpr09/doc/). Features are provided courtesy of Dan Munoz. You should not distribute the data without permission.

There are five classes, their labels values are:

- 1004: Veg
- 1100: Wire
- 1103: Pole
- 1200: Ground
- 1400: Facade

Run the algorithm on both datasets and report on the performance. Make sure your report contains the following information.

1. What was the mis-classification error?
2. Are there any classes that did not get classified well? Why do you think that is?
3. How long does the learner take (in terms of data points, dimensions, classes, etc...) for training and prediction?
4. Show images/movies of the classified data. Note that MATLAB is not very good at displaying thousands of 3D points; use VRML or python or convert it to pcd and use pcd viewer.

You may choose between Matlab or Python. Dependencies (if any) need to be included so we can run your code. Provide a script for us that on execution goes through the dataset and prints out the mis-classification errror. **Name the script as "gen_results.m" or "gen_results.py"**.

References

[1] Debadeepta Dey, Tian Yu Liu, Martial Hebert, and J. Andrew (Drew) Bagnell. Contextual sequence optimization with application to control library optimization. In *RSS*, 2012.

[2] Stephane Ross, Jiaji Zhou, Yisong Yue, Debadeepta Dey, and Drew Bagnell. Learning policies for contextual submodular prediction. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1364–1372, 2013.