1.5em 0pt

Statistical Techniques in Robotics (16-831, S22) Lecture #11 (Monday, February 23)

AdaBoost & Multi-Armed Bandit

Lecturer: Kris Kitani *Scribes: Yimin Tang, Zilin Si*

1 Review

In the last lecture, we had learnt the online techniques applied on supervised learning, especially focused on online Support Vector Machine (SVM).

1.1 Hyperplanes

In geometry, a hyperplane is a subspace whose dimension is one less than that of its ambient space. For example, if a space is 3-dimensional then its hyperplanes are the 2-dimensional planes, while if the space is 2-dimensional, its hyperplanes are the 1-dimensional lines[1].

Taken the hyperplanes (lines) in 2D as example, it can be written as $w_1x_1 + w_2x_2 + b = 0$, and in the vector form, $\mathbf{w} \cdot \mathbf{x} + b = 0$, where $\mathbf{w} \in \mathbb{R}^2$. Alternatively, we can merge the bias term b into the weight vector $\mathbf{w}$ such that $\mathbf{w} \cdot \mathbf{x} = 0$, where $\mathbf{w} \in \mathbb{R}^3$.

Distance to origin : Given a line $\mathbf{w} \cdot \mathbf{x} + b = 0$, the distance from the line to the origin is $\rho = \frac{b}{||\mathbf{w}||}$.

Distance between two parallel lines : Given a line $\mathbf{w} \cdot \mathbf{x} + b = 0$ and another parallel line $\mathbf{w} \cdot \mathbf{x} + b = -1$, the distance in between is $\rho = \frac{b+1}{||\mathbf{w}||} - \frac{b}{||\mathbf{w}||} = \frac{1}{||\mathbf{w}||}$.

Similarly, the hyperplanes of 3D are planes, and we can represent them as $\mathbf{w} \cdot \mathbf{x} + b = 0$, where the distance from the plane to the origin is $\frac{b}{||\mathbf{w}||}$. And given two planes $\mathbf{w} \cdot \mathbf{x} + b = 1$ and $\mathbf{w} \cdot \mathbf{x} + b = -1$, the distance in between is $\rho = \frac{b+1}{||\mathbf{w}||} - \frac{b-1}{||\mathbf{w}||} = \frac{2}{||\mathbf{w}||}$

1.2 Support Vector Machine and max-margin classification

Support Vector Machine (SVM) [1] can be used as a max-margin classifier. It aims to separate the data given the labels by finding a hyperplane that can maximize the margin between data groups. And this maximum margin solution is most stable to perturbations of data. Assume the hyperplane with maximum margin is $\mathbf{w} \cdot \mathbf{x} + b = 0$, and the inner points are locating at the boudary of margin $\mathbf{w} \cdot \mathbf{x} + b = 1$ and $\mathbf{w} \cdot \mathbf{x} + b = -1$. Then the objection is to maximze the margin:

[1]Wikipedia

$$max_{\mathbf{w}} \frac{2}{||\mathbf{w}||} \tag{1}$$

subject to $\mathbf{w} \cdot \mathbf{x}_i + b$ is ≥ 1 if $y_i = 1$ or ≤ -1 if $y_i = -1$ for all i.

This problem is equivalent to

$$min_{\mathbf{w}} ||\mathbf{w}|| \tag{2}$$

subject to $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$ for all i.

And this is a convex quadratic programming (QP) problem where a unique solution exists.

However, due to the noise or non-separability of the data, it may not be possible to satisfy all constraints. Here we introduce the soft margin SVM which allows some misclassification to get more robust classification by using a slack variable ξ:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \tag{3}$$

Geometrically, the slack x_i is the distance from the misclassified point $\mathbf{x}_i$ to the margin.

Now the problem becomes

$$min_{w,\xi} ||\mathbf{w}||^2 + C \sum_i \xi_i \tag{4}$$

subject to $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i$ for all i.

And to merge the linear constraints into the objective function, this becomes

$$min_w (\frac{\lambda}{2} ||\mathbf{w}||^2 + \frac{1}{M} \sum_{m=1}^{M} 1 - y_m \mathbf{w}^T \mathbf{x}_m) \tag{5}$$

However, we only want to penalize for mistakes, weakly correct but ignore the very correct data, we replace the contraints term with a hinge loss and it turns to

$$min_w (\frac{\lambda}{2} ||\mathbf{w}||^2 + \frac{1}{M} \sum_{m=1}^{M} max\{0, 1 - y_m \mathbf{w}^T \mathbf{x}_m\}) \tag{6}$$

Now this problem is convex with linear loss fuction $max\{0, 1 - y_m \mathbf{w}^T \mathbf{x}_m\}$ and quadratic regularization $\frac{\lambda}{2}||\mathbf{w}||^2$, but it is non differentiable due to the hinge. To solve this problem, we use stochastic (Sub) gradient descent for linear SVMs.

1.3 Online sub-gradient descent (OMD)

For the hinge loss $max\{0, 1 - y_m \mathbf{w}^T \mathbf{x}_m\}$, there are many possible sub-gradients, but we use two sub-gradients:

$$\mathbf{z}_m = \begin{cases} \mathbf{0} & \text{if } y_m \mathbf{w}^T \mathbf{x}_m \geq 1 \\ -y_m \mathbf{x}_m & \text{otherwise} \end{cases} \tag{7}$$

Till here, we can write the soft SVM algorithm as

Algorithm 1 soft SVM

```
1: $\theta^{(1)} \leftarrow 0 \in \mathbb{R}^N$
2: for $t = 1...T$ do
3:     $y_d, x_d \sim D$
4:     $\theta^{(t)} = \theta^{(t-1)} + y_d x_d \cdot 1[y_d(\mathbf{w}^{(t)} \cdot \mathbf{x}_d) < 1]$
5:     $\mathbf{w}^{(t+1)} \leftarrow \frac{1}{\lambda(t+1)\theta^{(t)}}$
6: end for
7: $\bar{\mathbf{w}} = \frac{1}{T}\sum_t \mathbf{w}^t$
```

2 Summary

2.1 PAC Learning Model

A Probably Approximately Correct(PAC) [3] learning algorithm requires :

$$E_{p(x,y)}[\mathbf{1}[f(x;\mathcal{D}) \neq y]] < \epsilon \tag{8}$$

holds with probability $1-\delta$ for any dataset D with size N drawn from true distribution $P(x,y)$.

There are two types of PAC Learning Models, one is strong PAC-learning algorithm, the other is weak PAC-learning algorithm:

- **Strong PAC-learning algorithm**: Given ϵ, δ and $\mathcal{D} \sim P(x,y)$, learner outputs with probability $(1-\delta)$ a hypothesis with error at most ϵ. And run-time must be polynomial in $\frac{1}{\delta}$ and $\frac{1}{\epsilon}$ and other relevant parameters such as the size of the examples or complexity of the target concept.
- **Weak PAC-learning algorithm**: We can release the error to $\epsilon \geq 1/2 - \gamma$. And any weak learning algorithm can be boosted into a strong learning algorithm.

2.2 AdaBoost

We can call a weak PAC learner multiple times (generates a sequence of hypothesis) but present it with a different distribution at each time, then combine all hypothesis into a single learner, this learner can be a strong PAC learner.

For example, we have a binary classification problem. We can get an strong classifier by combining weighted weak learner.

The final error bound is:

$$\epsilon \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t (1-\epsilon_t)} \tag{9}$$

The main difference between WMA and Adaboost is changing weights over experts versus data. And Adaboost will learn one weak learner with re-weighting dataset, WMA just re-weighting the experts and selects one.

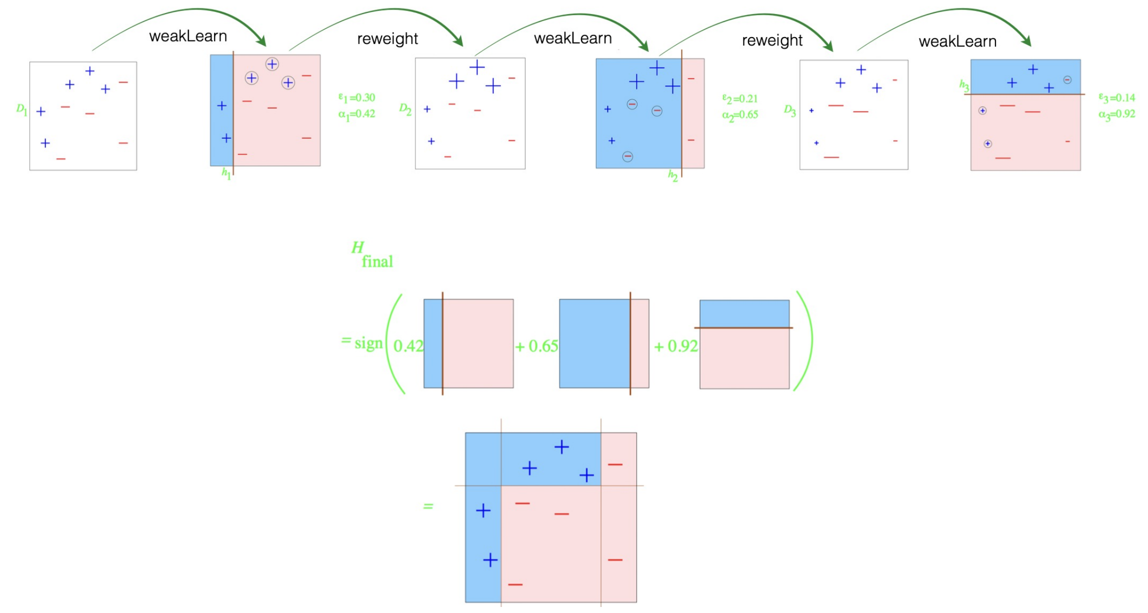

Figure 1: Toy example

Algorithm 2 Adaboost

```
1: function ADABOOST($\boldsymbol{D} = \{\boldsymbol{x}_n, y_n\}_{n=1}^N, \left\{w_n^{(0)}\right\}_{n=1}^N, T$)
2:     for $t = 1...T$ do
3:         $\boldsymbol{p}^{(t)} = \boldsymbol{w}^{(t-1)} / \sum_n w_n^{(t-1)}$
4:         $h^{(t)} = \text{WEAKLEARNER}\left(\boldsymbol{D}, \boldsymbol{p}^{(t)}\right)$                ▷ Prediction
5:         $\epsilon^{(t)} = \sum_n p_n^t \left|h^{(t)}(x_n) - y_n\right|$
6:         $\beta^{(t)} = \epsilon^{(t)} / 1 - \epsilon^{(t)}$
7:         $w_n^{(t)} = w_n^{(t-1)} \beta^{1 - \left|h^{(t)}\left(\boldsymbol{x}_n^{(t)}\right) - y_n^{(t)}\right|} \forall n$                ▷ Update
8:     end for
```

2.3 Multi-Armed Bandit

2.3.1 Overview

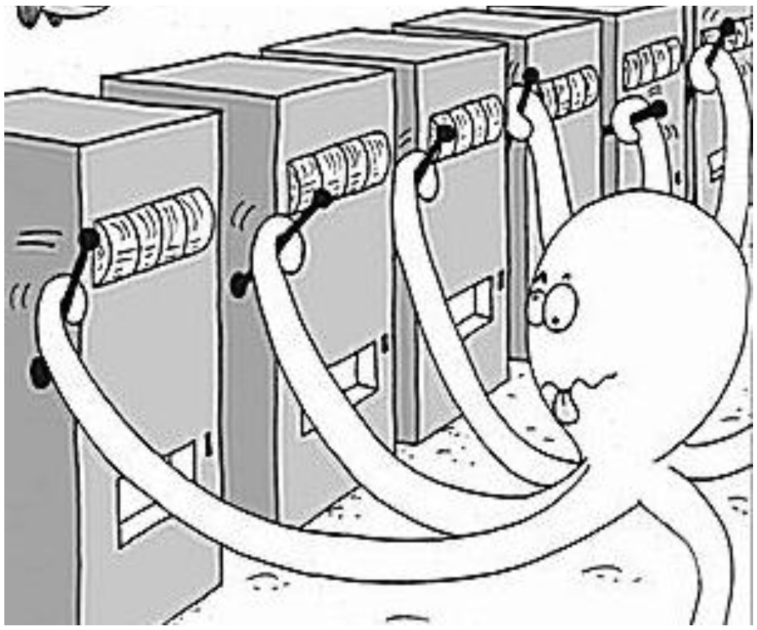

Figure 2: Multi-armed bandit problem

In the multi-armed bandit problem settings, we have multiple bandits with unknown reward distribution. Each time we are allowed to pull one arm and get this bandit's reward. Without knowing other bandits' rewards in each iteration, we should use learning strategies to maximize our total rewards over T iterations based on our historical information.

There could be different types of observations in this learning problem:

- **one-shot feedback**: In each time step, we only perform one action and get one reward. This action won't change the state of the next time step.
- **exhaustive feedback**: Since states and actions are finite, the player can pull all arms over the duration of the game.
- **evaluative feedback**: The player receives a sampled reward from the underlying reward distribution at each time step.

2.3.2 PWEA vs Bandits

PWEA and MAB problems both target to maximize the rewards or minimize the regrets, however they are different in the problem settings.

For PWEA, the player can get access to all experts' losses after taking an action so that he can remove or down-weight experts at each time step. However, for MAB, the player can only get access to the loss of the expert that's been selected. Mathematically, the player chooses an action $a^{(t)} \in \{1, 2, ..., n\}$ at time step t, then for PWEA, the player observes loss vector for all possible actions as $l^{(t)} \in [0, 1]^N$; but for MAB, the player can only observe loss for his action as $l^{(t)}_{a^{(t)}} \in [0, 1]$.

2.3.3 Exploration and Exploitation Trade-off

Since with MAB, the player only gets to see the rewards of the arm pulled, we need to find the balance between 1) exploiting arms that did well in the past and 2) exploring arms that might do well in the future. This doesn't apply to PWEA because it will explore automatically as the player can see all rewards.

2.3.4 Applications

Some applications of MAB are:

- **A/B testing**: The searching engine displays advertisements and gets rewards from the user who clicks ads.
- **Robot grasping**: The robot learns to grasp with different strategies with the reward as it can successfully pick up the object.
- **Medical treatment**: The doctor gives medicines to the patient where the reward is the patient gets healthy.
- **Packet routing**: The data packet is sent to a target location along a path where the reward comes from the low round-trip time.
- **AI for Go**: The algorithm explores the space of the future actions and gets win as a reward.

All these applications can only explore one action at each time step and get this action's reward.

2.3.5 Stochastic Bandit

For stochastic bandit [2], each arm has its own static reward distribution and each pulls gives the player a sample from this distribution. In order to maximize the rewards in a certain T rounds, the player has to consider the exploration-exploitation trade-off.

Mathematically, we assume each arm k has a static reward distribution as $v_k(r)$ with mean μ_k. By pulling, we sample a reward $r^{(t)}$ from this distribution.

Give multiple arms $k = 1, 2, ..., N$ with reward distribution $v_1(r), v_2(r), ..., v_N(r)$ and mean $\mu_1, \mu_2, ..., \mu_N$, to maximize the rewards, the player should pull the arm with the highest mean. However, during the game, the player does not know the true distribution and mean of each arm but only knows the estimated mean $\hat{\mu_1}^{(t)}, \hat{\mu_2}^{(t)}, ..., \hat{\mu_N}^{(t)}$ after number of pulls $T_1(t), T_2(t), ..., T_N(t)$. Therefore it is hard to choose which arm to pull to maximize the rewards because there might be difference between the estimated and the true mean.

References

[1] W. S. Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006.

[2] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

[3] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.