

AdaBoost & Multi-Arm Bandits

Lecturer: Kris Kitani

Scribes: Meghdeep Jana, Moneish Kumar

1 Review

In the last lecture, we learned about an online supervised learning algorithm, Online Support Vector Machine(SVM).

1.1 Max-Margin and Hard SVM

Support Vector Machines (SVMs) aims to find a hyperplane (decision boundary) in the feature space that separates the closest points of two different classes by as maximum distance as possible. This distance between the two extreme ends of hyperplane is called as margin (Shown in fig 1. More formally, let's say the hyperplane is:

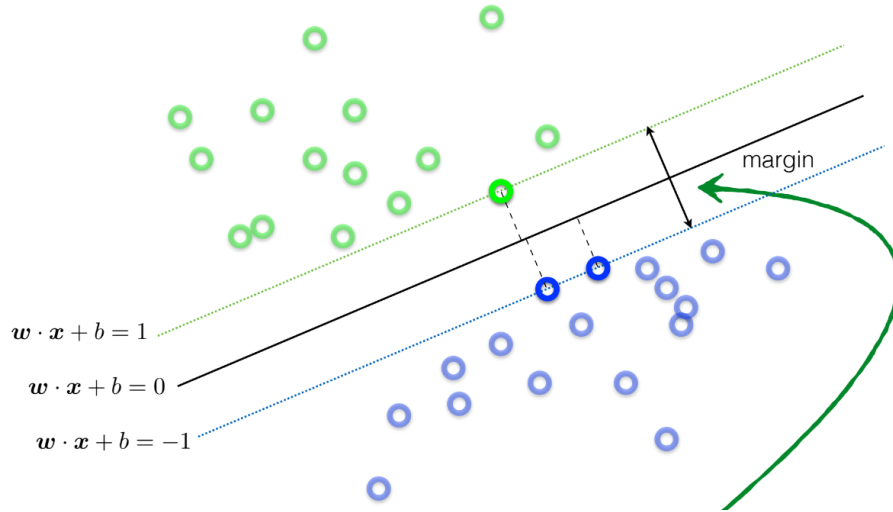


Figure 1: Max-Margin formulation of SVM. It tries to find a hyperplane that maximizes the distance between data points of two different classes

$$w \cdot x + b = 0 \tag{1}$$

For two class setting the objective of SVM can be formulated as a maximization problem:

$$\max_{\mathbf{w}} \frac{2}{\|\mathbf{w}\|}, \quad s.t. \quad \mathbf{w} \cdot \mathbf{x}_i + b \begin{cases} \geq +1 & \text{if } y_i = +1 \\ \leq -1 & \text{if } y_i = -1. \end{cases} \tag{2}$$

$$\tag{3}$$

Its form as a minimization problem would be:

$$\min_{\mathbf{w}} \|\mathbf{w}\|^2, \quad s.t. \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 (i = 1, \dots, N), \quad (4)$$

The above optimization problem can be solved using quadratic programming. The above formulation is known as "Hard-SVM" and assumes that the data is linearly separable. Further, we introduce a variant of SVMs that relaxes this assumption.

1.2 Soft SVM and Sub-gradients

The Hard SVM, as seen above tries to classify all labels correctly. This works only in the case of linearly separable data, also if we consider that usually data has some inherent noise and we would want to create a robust model that is not affected by such noise. This is achieved by soft SVM wherein we accept some classifications to create an SVM with a larger margin i.e that this SVM is more robust or has a lower variance. For each data point x_i we would want the following condition to hold.

$$y_i(w^T x_i + b) \geq 1 - \xi_i \quad \forall i \quad (5)$$

Here ξ_i is called the slack variable. The objective function now becomes:

$$\min_{w, \xi_i} \|w\|^2 + C \sum_i \xi_i \quad (6)$$

Combining both equations we get our final objective function:

$$\min_w \frac{\lambda}{2} \|w\|^2 + \frac{1}{M} \sum_{i=1}^M \max(0, 1 - y_i w^T x_i) \quad (7)$$

The above objective function is convex but is not differentiable because of the second part (hinge loss). This optimisation is solved using sub-gradient descent. A sub gradient of a convex function $f(x)$ where $x \in \mathbb{R}^n$ is $g \in \mathbb{R}^n$ such that:

$$f(y) \geq f(x) + g^T (y - x) \quad (8)$$

For Hinge Loss the subgradients are:

$$g_i = \begin{cases} 0 & y_i w^t x_i \geq 1 \\ -y_i w^t x_i & \text{otherwise} \end{cases} \quad (9)$$

The algorithm for soft SVM with subgradient descent is as follows:

Algorithm 1 Soft SVM

- 1: $\theta^{(1)} \leftarrow \mathbf{0} \in \mathbb{R}^N$
 - 2: **for** $t = 1, \dots, T$ **do**
 - 3: $y_i, \mathbf{x}_i \sim D$
 - 4: $\theta^{(t)} = \theta^{(t-1)} + y_i \mathbf{x}_i \cdot \mathbf{1}[y_i(\mathbf{w}^{(t)} \cdot \mathbf{x}_i) < 1]$
 - 5: $\mathbf{w}^{(t+1)} \leftarrow \frac{1}{\lambda(t+1)} \theta^{(t)}$
 - 6: **end for**
-

2 Summary

2.1 Probably Approximately Correct(PAC) Learning

Probably Approximates Correct(PAC) learning is a theoretical framework that tries to answer the following two questions:

- What is the optimal dataset size to obtain good generalization?
- What is the computational cost of learning?

Let's consider a dataset \mathcal{D} of size N that is drawn from an underlying distribution $P(x, y)$. Here, the class labels are determined by an unknown deterministic distribution $y = f^*(x)$ this can be called as the "perfect hypothesis" in online learning domain. The dataset \mathcal{D} forms $\hat{P}(x, y)$ which is an approximation of true distribution $P(x, y)$ and more samples we draw, the better is the approximation.

We define a space of functions \mathcal{F} (hypothesis class) containing functions defined on the basis of the training set \mathcal{D} . We then define a function $f(x; \mathcal{D})$ drawn from \mathcal{F} that can be thought of as a classifier learned from the training data. A function $f(x)$ has good **generalization** if its expected error rate is less than a pre-defined threshold parameter ϵ . This can be expressed as:

$$\mathbb{E}_{P(x,y)}[\mathbf{1}[f(x; \mathcal{D}) \neq y]] < \epsilon \quad (10)$$

We can now define the PAC learning algorithm. It is a learning algorithm that requires (10) to hold with a probability of $(1 - \delta)$ for any dataset \mathcal{D} drawn from $P(x, y)$. Here, δ is a pre-defined value. There are two types of PAC learning algorithms:

Definition 1. Strong PAC Learner *This PAC learning algorithm outputs with a probability $(1 - \delta)$ a hypothesis function $f(x)$ with error at most ϵ .*

Definition 2. Weak PAC Learner *This PAC learning algorithm outputs with a probability $(1 - \delta)$ a hypothesis function $f(x)$ with error at most $\epsilon \geq \frac{1}{2} - \gamma$. Here, $\gamma > 0$.*

For both strong and weak PAC-learning algorithms, the run time must to polynomial in $1/\delta$ and $1/\epsilon$ and other relevant parameters such as the size of the examples or complexity of the target concept.

Boosting is a strategy where an ensemble of weak learners are presented with a different distribution \mathcal{D} and their hypothesis are combined into a single hypothesis which is the strong learner [6]. The key idea here is that any weak learning algorithm can be boosted into a strong learning algorithm [8]. PAC learning makes the connection between boosting and online learning. Further, we introduce a learning algorithm that leverages this concept of boosting.

2.2 AdaBoost

We introduce Adaptive Boosting (AdaBoost), which is a machine learning that is used in conjunction with many other types of learning algorithms to improve performance. The output of the

other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier[?]. The Adaboost algorithm is shown in Algorithm 2.

Algorithm 2 AdaBoost[4]

Require: $D = \{x_n, y_n\}_{n=1}^N, \{w_n^{(0)}\}_{n=1}^N, T$

- 1: **for** $t=1, \dots, T$ **do**
- 2: $\mathbf{p}^{(t)} = \mathbf{w}^{(t-1)} / \sum_n w_n^{(t-1)}$
- 3: $h^{(t)} = \text{WEAKLEARNER}(D, \mathbf{p}^{(t)})$
- 4: $\epsilon^{(t)} = \sum_n p_n^t |h^{(t)}(x_n) - y_n|$
- 5: $\beta^{(t)} = \epsilon^{(t)} / (1 - \epsilon^{(t)})$
- 6: $w_n^{(t)} = w_n^{(t-1)} \beta^{1 - |h^{(t)}(x_n^{(t)}) - y_n^{(t)}|} \forall n$
- 7: **end for**
- 8: $h_F(x) = \mathbf{1}[\sum_{t=1}^T (\log \frac{1}{\beta^{(t)}}) h^{(t)}(x) \geq \frac{1}{2} \sum_{t=1}^T (\log \frac{1}{\beta^{(t)}})]$

Here, $x \in \mathbb{R}^M$ is the observation vector, $y \in 0, 1$ is the label, $\mathbf{w} \in \mathbb{R}^N$ is the weights of training data items (N being number of data items), n is the index of the data item and T is the final number of hypothesis and is finite. The lines 2-3 show the Prediction step and lines 4-6 show the Update step.

Prediction step: Here, $\mathbf{p}^{(t)}$ is the probability distribution over weights of data points where more weight is given to 'important/hard' data items. A weak learner takes in the dataset \mathcal{D} and the probability distribution $\mathbf{p}^{(t)}$ to return a hypothesis/classifier. The weak learner can be a decision stump, linear classifier etc.

Update step: The average error/loss of the hypothesis $h^{(t)}$ is calculated and denoted by $\epsilon^{(t)}$. We then define a penalty constant $\beta^{(t)}$ in terms of $\epsilon^{(t)}$. Note that if the hypothesis is good ($\epsilon^{(t)} = 0$), we get $\beta^{(t)} = 0$ and for a bad hypothesis ($\epsilon^{(t)} = 1$), we get $\beta^{(t)} = \infty$. In the final step 6, we update the weights $w_n^{(t)}$ using a multiplicative update that is exponential over $\beta^{(t)}$. This exponential form comes due to the regularization over the weights being an entropic regularization. So we down weight correctly classified points and we up-weight misclassified points by the weak learner. Finally at step 7, Once we have all the weak learners' hypothesis, we take the weighted average over all weak learners. And the final hypothesis is defined as a majority voting. Here, the weak learners that made fewer mistakes have a higher weight.

This AdaBoost algorithm has a error bound that is:

$$\epsilon \leq 2^T \prod_{t=1}^T \sqrt{\epsilon_t(1 - \epsilon_t)} \quad (11)$$

2.3 Multi-Armed Bandit

Now, we will be looking at the Multi-Armed Bandit (MAB) problem. In this problem we are trying to allocate a fixed set of resources amongst a set of given choices with the aim to maximize the reward that is received from the choices. In this setting the choices' properties are only partially

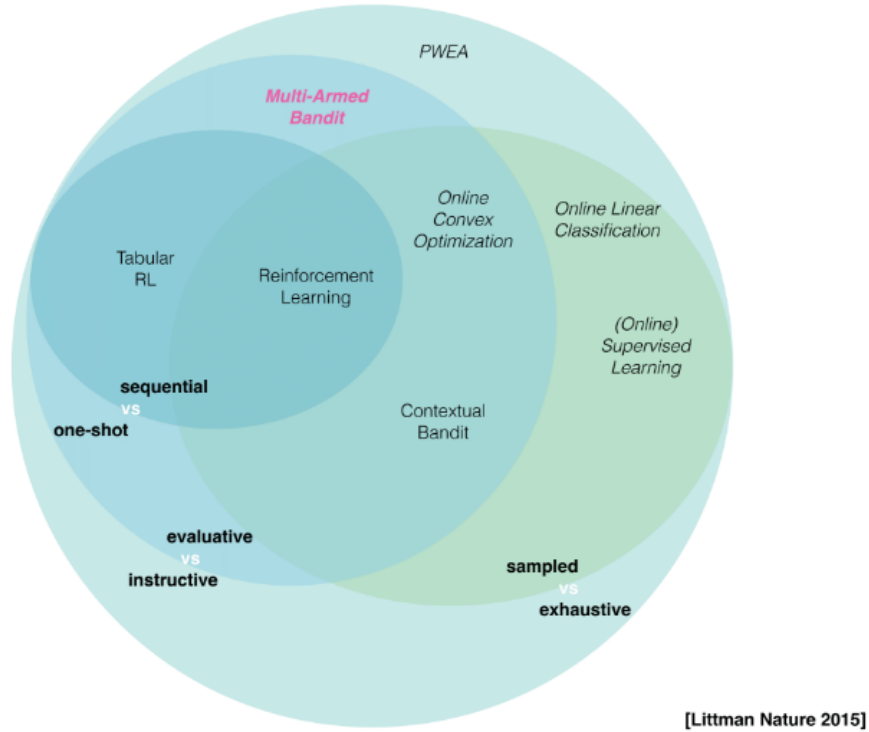


Figure 2: Concept venn diagram with Multi-Arm bandits

known at the time of allocation and may be better know as subsequent allocations are done or as time passes. We can also see where MAB are placed in the concept venn diagram in figure 2

Why Multi-Arm Bandit ? The name comes from imagining a gambler at the slot-machines (also called one arm bandits, because they take away your money!), who has to figure out which machines to play, how many times to play each machine and in which order to play them, and whether to continue with the current machine or try a different machine.

The current setting of Multi - Armed Bandits makes this problem a:

- **One shot feedback** selecting one action leads to only one reward.
- **Exhaustive feedback** actions are limited and you get to pull arms
- **Evaluative feedback** only a sample of the reward is shown (one the one that you pull),

2.3.1 PWEA vs Bandits

In PWEA, at any given time T we had access to each of the experts losses. Figure 5 shows an example of PWEA at time stamp 3, where the algorithm made a mistake and we know that experts 1, and experts 2 have made a mistake so we can reduce their weight for the next iteration. But, in the case of MAB only know the reward on the bandit that was chosen and not the others. Figure 4 shows a example of this.

Loss vs Reward

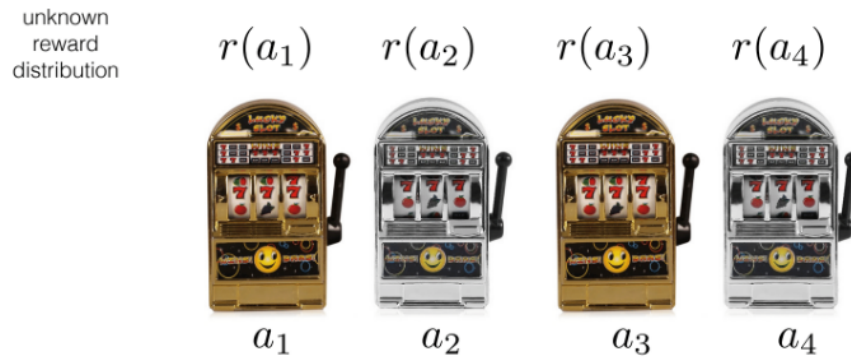






Figure 3: Multiple One-Arm bandits. Each have an unknown reward distribution and our objective is to maximize the reward when we have T pulls.



Figure 4: Partial feedback in case of MAB. We can't see the rewards of the bandits whose arm was pulled.

Recall: PWEA

We have access to all the experts' losses ...

Experts				Predict	Outcome
t = 1					
t = 2					
t = 3					

... so we can remove or down-weight experts at every step

Figure 5: Classic PWEA, where we have access to all of the experts' losses

Previously we used loss $l \in [0, 1]$ to describe the performance of an algorithm. Similarly we can also define reward $g = 1 - l \in [0, 1]$ to describe the algorithms performance. The objective here would be to maximize this reward. The terms loss and reward can be used interchangeably in the sense that we want to minimize loss or maximize reward for an algorithm. The Regret can also be written in terms of reward:

$$R^T(h) = \underbrace{\sum_{t=1}^T g(h(x^T), y^T)}_{\text{reward of the expert}} - \underbrace{\sum_{t=1}^T g(\hat{y}^T, y^T)}_{\text{reward of the algorithm}}$$

2.3.2 “Exploration and Exploitation” trade-off

As discussed before that MAB differs from PWEA in the observability of loss. Since we only see the rewards of the arms that were pulled, we need to find balance between 1) exploiting arms that did well in the past, 2) exploring arms that might do well in the future. In PWEA we need not explore because at every instance we can see the loss of other experts. This step of exploration is necessary in case of MAB because we have some/no information about some arms.

2.3.3 Applications of Bandits

- **Which advertisement to display** Suppose you are an advertiser seeking to optimize which ads to show visitors on a particular website. For each visitor, you can choose one out of a collection of ads, and your goal is to maximize the number of clicks over time. The way to think about this in our example is that each ad has some theoretical — but unknown —

click-through-rate (CTR) that is assumed to not change over time. How do we go about solving which ad we should choose? ¹

- **Robotic grasps** In this scenario we consider the different trajectories (N) the robot can take to be like the N bandits and the reward in this case is if the robot could actually grasp the object or not.
- **Medical Treatment** MABs can be used in various situations such as to decide the optimal dosage for patients like the authors of [1] suggest, to deciding the course of the medical treatment for a patient ²
- **Packet routing** The authors of [11] study online shortest path routing over multi-hop networks, where time-varying link costs or delays are modelled as independent and identically distributed random processes, whose parameters are initially unknown.
- **AI for Go** Alpha Go which is deep-mind’s AI for playing Go simply combined the previous state of the art (Monte Carlo Tree Search [technique inspired from multi-armed bandits]) with the new deep learning techniques.

2.3.4 Types of Bandits

- **Stochastic MABs:** In this setting, each arm $k \in [n]$ is associated with an (unknown) probability distribution v_k on $[0, 1]$ with true mean μ_k , and rewards from arm k are assumed to be drawn i.i.d. from v_k .
- **Adversarial MABs:** Here there are no probabilistic assumptions on the rewards. Instead, the rewards can be generated by an adversary.
- **Markovian MABs:** Here the rewards of each arm follow a Markov process on some underlying state space. The tools used to analyze Markovian MABs are quite different from those used to analyze stochastic and adversarial MABs, and are more similar to tools used in reinforcement learning.

References

- [1] M. Aziz, E. Kaufmann, and M.-K. Riviere. On multi-armed bandit designs for dose-finding clinical trials. *Journal of Machine Learning Research*, 22(14):1–38, 2021.
- [2] D. Bouneffouf and I. Rish. A survey on practical applications of multi-armed and contextual bandits. *arXiv preprint arXiv:1904.10040*, 2019.
- [3] K. Ding, J. Li, R. Bhanushali, and H. Liu. Deep anomaly detection on attributed networks. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 594–602. SIAM, 2019.

¹<https://www.spotx.tv/resources/blog/developer-blog/introduction-to-multi-armed-bandits-with-applications-in-digital-advertising/>

²https://www.ima.umn.edu/materials/2017-2018.4/W9.14-16.17/26479/bandit_slides_IMA_2017.pdf

- [4] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.
- [5] X. Huo and F. Fu. Risk-aware multi-armed bandit problem with application to portfolio selection. *Royal Society open science*, 4(11):171377, 2017.
- [6] M. Kearns. Thoughts on hypothesis boosting. Unpublished, Dec. 1988.
- [7] A. Kumar. Introduction to multi-armed bandit problem and its applications, June 2018. [Online; posted 13-June-2018].
- [8] R. E. Schapire. The strength of weak learnability. *Mach. Learn.*, 5(2):197–227, jul 1990.
- [9] W. Shen, J. Wang, Y.-G. Jiang, and H. Zha. Portfolio choices with orthogonal bandit learning. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.
- [10] J. Sublime and S. Lefebvre. Collaborative clustering through constrained networks using bandit optimization. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.
- [11] M. S. Talebi, Z. Zou, R. Combes, A. Proutiere, and M. Johansson. Stochastic online shortest path routing: The value of feedback. *IEEE Transactions on Automatic Control*, 63(4):915–930, 2017.
- [12] P. Whittle. Multi-armed bandits and the gittins index. *Journal of the Royal Statistical Society: Series B (Methodological)*, 42(2):143–149, 1980.

3 Appendix

3.1 More Applications of MAB

A comprehensive study of the applications of MAB are nicely mentioned in [2]. A few selected uses cases are mentioned below.

Crowdsourcing: Crowdsourcing has emerged as a powerful means available to employers or service requester to get tasks completed in a timely and cost-effective manner[7]. The employer’s aim is to maximize the number of the tasks completed. The available workers(or employees or freelancers) act as arms in this case, having hidden qualities unknown to the employer. So the problem can be posed as a MAB problem where the employer can either explore the arms to learn their qualities or choose to exploit the best arm identified till now.

Recommend courses to learners

MABs are also used in recommending courses to learners in e-learning systems. There are N courses that the e-learning system can recommend to learner. Each course has an unknown rate of success p_i . a_i and b_i is the historical number of success (number of learners that finished the course) and failure (number of learners that dropped the course). Each course i is considered to be an arm of a multi-armed bandit with a_i success and b_i failure. This is a MAB with Bernoulli process and the optimal solution is the Gittins index [12].

Anomaly Detection

Performing anomaly detection on attributed networks concerns with finding nodes whose behaviors deviate significantly from the majority of nodes. Authors in [3] investigate the problem of anomaly detection in an interactive setting by allowing the system to proactively communicate with the human expert in making a limited number of queries about ground truth anomalies. Their objective is to maximize the true anomalies presented to the human expert after a given budget is used up.

Bandits for Clustering

[10] considers collaborative clustering, which is machine-learning paradigm concerned with the unsupervised analysis of complex multiview data using several algorithms working together. The authors propose a collaborative peer to peer clustering algorithm based on the principle of non stochastic multi-arm bandits to assess in real time which algorithms or views can bring useful information.

Finance

In recent years, sequential portfolio selection has been a focus of increasing interest at the intersection of the machine learning and quantitative finance. The trade-off between exploration and exploitation, with the goal of maximizing cumulative reward, is a natural formulation of the portfolio choice problems. In [9], the authors proposed a bandit algorithm for making online portfolio choices via exploiting correlations among multiple arms. In [5], the authors incorporate risk-awareness into the classic multi-armed bandit setting and introduce a novel algorithm for portfolio construction. Through filtering assets based on the topological structure of financial market and combining the optimal multi-armed bandit policy with the minimization of a coherent risk measure, they achieve a balance between risk and return.