Reproducing Kernel Spaces and Regret Analysis of Multi-Armed Bandits

Anthony Della Pella

April 15, 2016

Motivating Example

Consider a "casino" which has K slot machines, which pay out some reward when played. The machines are free to play, however only one lever can be pulled at a time (that is you pull an arm, then receive a reward). Additionally, you're given that the machines each follow some probabilistic payout table.
The natural problem then is to maximize your winnings. In particular, we wish to find some strategy which, regardless of the underlying probability distribution on the payouts, will give a good reward total once we have finished pulling the arms.

Multi Armed Bandits

Definition

The Multi-Armed Bandit Setting (MAB) is characterized by a set of arms $i \in \{1, 2, \ldots, K\} = [K]$ each corresponding to a probability distribution ν_i. In addition to the K arms, we consider a sequence of time steps $t = 1, 2, \ldots$, where at each step, the player (or forecaster) chooses an arm I_t and receives some reward $X_{I_t,t}$, where $X_{I_t,t} \sim \nu_{I_t}$ independently of the past.

Variants on the MAB Setting

Note that there are many variants on this general framework. For instance in the adversarial bandit setting one has adversaries choosing the payouts $X_{I_t,t}$ on an adaptive basis based on which arm we chose previously.

Another common stipulation on the general bandit framework is that we are in the full information setting where the payouts of every arm on a given round are observed (yet we only receive the payout of the arm played).

Naive Approaches (FTL & Max Outlook)

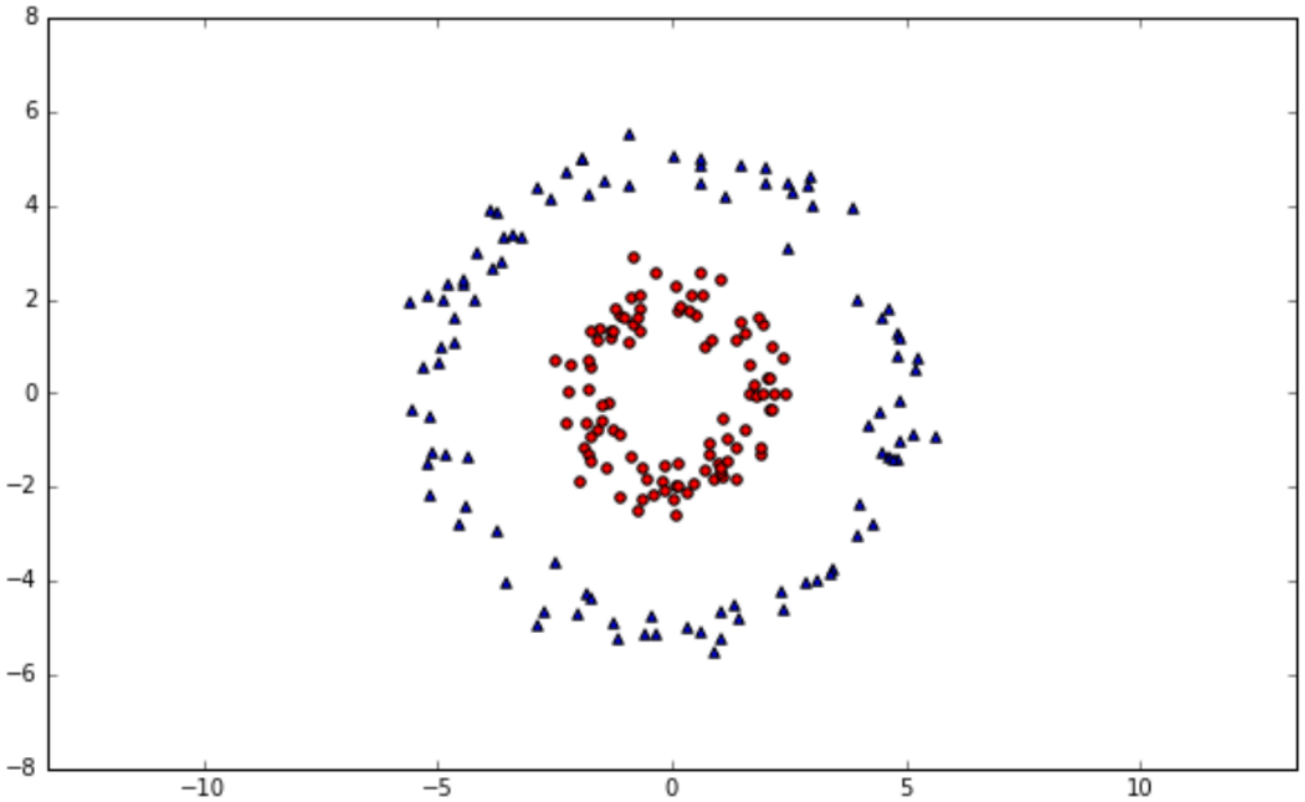
Perhaps the most naive approach in the non-full information setting is to use a Max Outlook Algorithm, which will play every arm several times and then based on what we have received play the best arm for the remainder of the game.

A similar update based approach (specific to the full information setting) is to choose the arm that has been performing best (on the whole) in previous rounds:

$$I_t = \arg\max_{i \in [K]} \sum_{i=1}^{t-1} X_{i,t}$$

Neither approach is very good, and once we try to cross over into different frameworks comparing approaches becomes difficult.

A Machine Learning Problem – Linear Classifiers



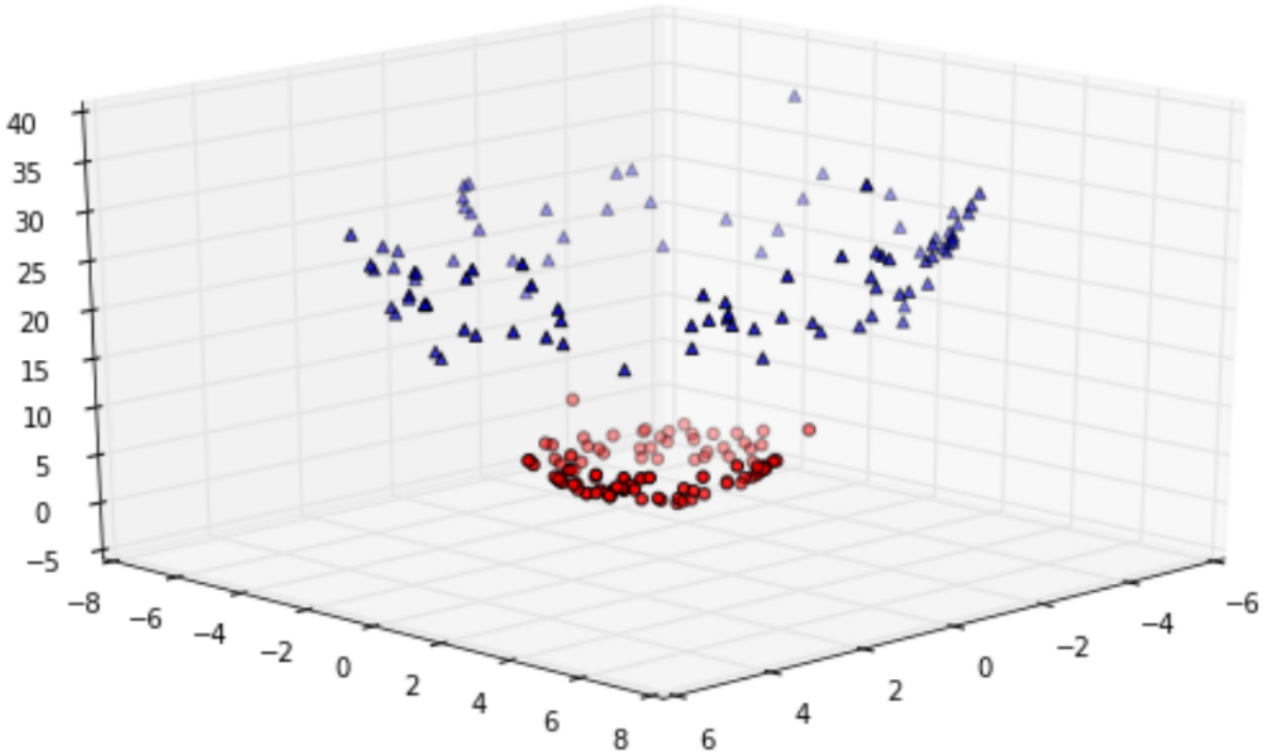
Approach I - Increase Dimensions

We could throw in an extra dimension along the z-axis of the data which would measure the radius from the origin so that our data is now represented by passing a data vector $\mathbf{x}$ under the transformation $T : \mathbb{R}^2 \to \mathbb{R}^3$ defined as:

$$T(\mathbf{x}) = (x_1, x_2, \sqrt{x_1^2 + x_2^2}).$$

I now show what happens when we have $\mathbf{x}$ represent an unlabeled data point shown in the figure. A plot of T applied to our data set is shown on the next slide.

Visualizing $T(\mathbf{x})$



The Kernel Trick

The technique of finding a linear classifier to identify differences between red and blue points by finding some other representation of our data say $\varphi(\mathbf{x})$ is known as the kernel trick. We would like to allow the possibility that $\varphi(\mathbf{x})$ lives in a higher, even infinite dimensional, space. In the case that there exists a function $\kappa(\mathbf{x}, \mathbf{y}) = \langle \varphi(\mathbf{x}), \varphi(\mathbf{y}) \rangle$, where $\langle \cdot, \cdot \rangle$ is an inherited inner product from a Hilbert space $\mathcal{H}$, we say that κ is the reproducing kernel of $\mathcal{H}$.

Back to Bandits

In the general multi armed bandit framework (which is itself just an instantiation of the even more general notion of a random process) we can use the kernel trick to do many things which would otherwise be impossible.

The main use of the kernel trick as it pertains to multi armed bandits is to show that algorithms in different settings can be transformed via some kernel to apply to another setting. For instance, one could apply an appropriate kernel to a full information setting and achieve desirable results which generalize to the adversarial setting.

One Main Caveat

The main flaw is that finding a proper reproducing kernel and its underlying Hilbert space can be challenging. In particular general random processes only require a metric space to be defined. Thus, it is beneficial to have an easier framework (namely Banach spaces) to work over.

While in general Banach spaces have no such kernel (due to the lack of an inner product) we can weaken our requirements and define an appropriate kernel using a semi-norm. See for instance the work of Zhang & Zhang *("Reproducing Kernel Banach Spaces for Machine Learning")*.