CSE599s, Spring 2014, Online Learning Lecture 2 - 04/03/2014

Online Convex Optimization Example And Follow-The-Leader

Lecturer: Brendan McMahan *Scribe: Stephen Joe Jonany*

1 Review of Online Convex Optimization

We first review the online convex optimization problem, before exploring a real-world example.

Online Convex Optimization

```
for $t = 1, 2, \ldots, T$
    player chooses $w_t$
    adversary chooses $f_t$ (simultaneously with the player's decision)
    player's loss $f_t(w_t)$ is computed
    player observes $f_t$
```

2 Real World Example - Google Search

2.1 Overview

In class we explored a real world setting where
- A user typed in the query "The joy of cooking".
- Google's Ad servers responded with a list of ads to show to the user.
- The user may possibly click one of the ads.
We might then ask the following question.
How does Google decide which ads to show to a user?

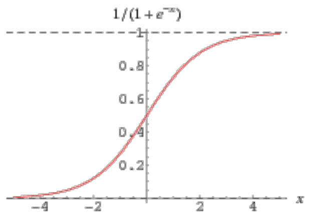
To identify the online learning aspect of this setting, we first explore some of the processes run by the ad server. The ad server does the following processes:

1. Matches ads on keywords. For instance, the keyword "cooking" might be extracted from the user's query, and may be matched with ads related to cooking.
2. For each ad, we compute features $x \in \mathbb{R}^n$, then query prediction model to estimate the probability of an ad being clicked. Note that this means that we match and predict on many more ads than we actually show.
3. Run auction, which considers the click probabilities computed from the previous step, as well as the monetary value of the ad.
4. Choose which ads to show.

We focus on the learning problem where we are trying to predict the probability of an ad being clicked by a particular user.

2.2 Example Model

In class, we assumed a linear model $w \in \mathbb{R}^n$. Given the weights, we might decide to make a prediction like so: $\hat{p} = w \cdot x$. However, since we are trying to predict probabilities, we want to normalize the predictions to be between 0 and 1. That is, we need a function $f(x)$ such that $f : \mathbb{R} \to [0, 1]$. We also want this function to be increasing, so that higher x will correspond to a higher probability. A common function that achieves this is the sigmoid function, $\sigma(x) = \frac{1}{1+e^{-x}}$.



Thus, we predict the probability like so: $\hat{p} = \sigma(w \cdot x)$.
Note that this is exactly how a logistic regression model calculates an input example's class probabilities. To learn the model, we have a training system that takes in (x, y) pairs, where x is the feature vector of the ad, and y is the corresponding label, which in this case indicates whether or not the user has clicked the ad. Still using the logistic regression model, if we were to use online gradient descent to optimize the loss function, the training process will look like so. (We use 1 to denote an ad getting clicked, and 0 otherwise to make the math work out when using online gradient descent for logistic regression).

Training Process - SGD for Logistic Regression

```
for each $(x, y)$
    $\hat{p} = \sigma(w_t \cdot x)$
    $g = (\hat{p} - y)x$
    $w_{t+1} \leftarrow w_t - \eta g$
```

Note that in the real setting, we will have to update the w used by the ad server with the new weight vectors obtained from the training process.

2.3 Problems and Challenges

In the real setting, n, the dimension of the feature vector, can be in the order of billions. In addition to this, the online setting raise some challenges:

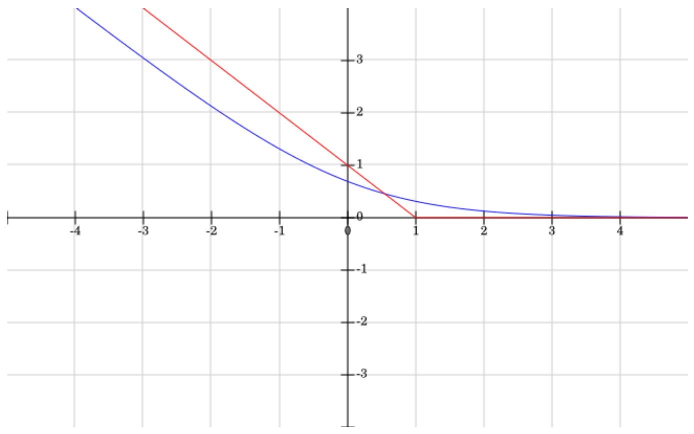
1. Just writing the model w consumes so much space. Moreover, we want to search for the best model out there.

2. Computing the prediction naively by doing a normal dot product will be too slow, especially given the constraint that we want to give the feedback to the user in the order of hundreds of milliseconds. Fortunately, even though w is dense, x, the feature vector, is usually sparse. This is especially true if we are using a bag of words representation. Then, computing $w \cdot x$ will only involve a number of operations that is proportional to the number of non-zero elements in x, as opposed to n.

3. Even if we perform well on the (x, y) examples, we have to remember that these examples are obtained from ads that were actually shown. Thus, it might be the case that there might be some unshown ads that are actually good. A future topic related to this is exploration.

2.4 Mapping back to OCO

Going back the the OCO formulation, we see that w_t is the weight vector outputted by the training process. Since we chose to use logistic regression as our model, we want our loss function to be related to the conditional likelihood. The specific form chosen in class was $f_t = \log(1 + \exp(-y_t(w \cdot x_t)))$. It was commented in class that this loss function can be viewed as a soft version of the hinge loss. The plot below shows this.



However, there were also some comments that OCO does not model some aspects of the given example. For one, there is a delay before the player actually observes f_t, since this information is only available after the user has made a clicking decision.

2.5 Comments on using Regret Analysis

To review,

$$\text{Regret} = \sum_t f_t(w_t) - \min_{w \in \mathcal{W}} \sum_t f_t(w).$$

When comparing between two models A and B, we note that we can cancel out the constant term on the right:

$$\text{Regret}(A) - \text{Regret}(B) = \sum_t f_t(A_t) - \sum_t f_t(B_t) = \text{Loss}(A) - \text{Loss}(B).$$

When using regret as a metric to optimize on, we have to be wary if $\mathcal{W}$ has changed when we made changes to a model; a higher regret might mislead us into thinking that a model is worse than the other, but this is not necessarily true.

3 Follow-The-Leader (FTL) Algorithm

The first new algorithm that we explore to apply in the OCO framework is Follow-The-Leader, which was described to be the simplest, most natural, and impractical online learning algorithm. Much of this section is adapted from the survey of Shai Shalev-Shwartz [1].

```
Follow-The-Leader
$w_1$ is set arbitrarily
for $t = 1, 2, \ldots, T$
    $w_t = \operatorname{argmin}_{w \in \mathcal{W}} \sum_{s=1}^{t-1} f_s(w)$
```

There are two ways to view the algorithm.

1. We are playing on round t the weight vector that minimizes the regret if the game stopped on round $t-1$.
2. We use all the seen examples as a batch machine learning problem, and solve for the best weight vector.

We note that the longer we run this algorithm, the slower the algorithm gets, since the batch problem becomes larger and larger (Impractical).
We now prove our first regret bound. Note that regrets can also be expressed as being relative to a chosen vector, instead of just being compared against the best model in hindsight. In the lemma below, $\text{Regret}(u)$ denotes the regret with respect to the vector u.

Lemma 1. *The points $w_1, \ldots, w_T$ played by FTL satisfy* $\forall u \in \mathcal{W}, \text{Regret}(u) \le \sum_{t=1}^{T} \big(f_t(w_t) - f_t(w_{t+1})\big)$

Proof. We first restate the lemma. Substituting the definition of $\text{Regret}(u)$ gives us the following equivalent statement: $\sum_{t=1}^{T} \big(f_t(w_t) - f_t(u)\big) \le \sum_{t=1}^{T} \big(f_t(w_t) - f_t(w_{t+1})\big)$, which reduces to

$$\sum_{t=1}^{T} f_t(w_{t+1}) \le \sum_{t=1}^{T} f_t(u).$$

In class, the expression on the left side of the inequality was called the cumulative loss of the Be-The-Leader (BTL) algorithm, which cheats by looking into the next example. This term was introduced by Kalai et al. [2]

We now use induction to prove

$$\forall u \in \mathcal{W}, \sum_{t=1}^{T} f_t(w_{t+1}) \le \sum_{t=1}^{T} f_t(u).$$

Base case. $T = 1$. $f_1(w_2) \le f_1(u)$ for any u is true because w_2 minimizes f_1.
Recall that $w_2 = \underset{w\in\mathcal{W}}{\text{argmin}} \sum_{s=1}^{2-1} f_s(w) = \underset{w\in\mathcal{W}}{\text{argmin}} f_1(w)$.
Inductive Hypothesis. We assume the statement above holds true for time $T-1$, that is,

$$\forall u \in \mathcal{W}, \quad \sum_{t=1}^{T-1} f_t(w_{t+1}) \le \sum_{t=1}^{T-1} f_t(u).$$

Inductive Step. From the above I.H., we complete the inductive step by showing that the statement holds true for time T, via the following sequence of equivalent inequalities:

$$\begin{aligned}
\sum_{t=1}^{T-1} f_t(w_{t+1}) &\le \sum_{t=1}^{T-1} f_t(u) \\
\sum_{t=1}^{T-1} f_t(w_{t+1}) + f_T(w_{T+1}) &\le \sum_{t=1}^{T-1} f_t(u) + f_T(w_{T+1}) \\
\sum_{t=1}^{T} f_t(w_{t+1}) &\le \sum_{t=1}^{T-1} f_t(u) + f_T(w_{T+1}).
\end{aligned}$$

Since the statement above is true for all u, we can take $u = w_{T+1}$ which gives us

$$\sum_{t=1}^{T} f_t(w_{t+1}) \le \sum_{t=1}^{T} f_t(w_{T+1}).$$

We can view the term on the right side of the inequality as a function of vector, $f(v) = \sum_{t=1}^{T} f_t(v)$. In this case, the argument is $v = w_{T+1}$. We note that w_{T+1} is exactly the minimizer of $f(v)$. Thus, for all u, $\sum_{t=1}^{T} f_t(w_{T+1}) \le \sum_{t=1}^{T} f_t(u)$. With this final inequality, we end the inductive proof. □

We now explore an example where FTL performs poorly.

Example 1 Assume that $\mathcal{W} \in [-1,1]$, $f_t(w) = g_t w$, where $g_t \in [-1,1]$ is chosen by the adversary, and the player chooses w_t. We also abbreviated $\sum_{s=1}^{t} g_s$ as $g_{1:t}$. Below we show a game between a player with the FTL strategy that arbitrarily picks $w_1 = 0$, and an adversary that wants to cause the player to incur losses (note that the adversary also knows that the FTL strategy is deterministic). We also show the BTL strategy on the right hand side for comparison. We consider what the FTL strategy will do under this setting, where the points played are given by

$$\begin{aligned} w_t &= \underset{w\in[-1,1]}{\operatorname{argmin}} \sum_{s=1}^{t-1} f_s(w) \\ &= \underset{w\in[-1,1]}{\operatorname{argmin}} \sum_{s=1}^{t-1} g_s w \\ &= \underset{w\in[-1,1]}{\operatorname{argmin}}\, g_{1:t-1} w. \end{aligned}$$

So, the FTL strategy always chooses the sign opposite to $g_{1:t-1}$, with magnitude 1.

t	FTL w_t	g_t	FTL Loss	$g_{1:t}$	BTL $\hat{w}_t = w_{t+1}$	BTL Loss
1	0	0.5	0	0.5	-1	-0.5
2	-1	-1	1	-0.5	1	-1
3	1	1	1	0.5	-1	-1
4	-1	-1	1	-0.5	1	-1
..	..	..	..	..	..	..

The regret of FTL is $(T-1) - (-0.5) \approx T$, which is bad, because we want the regret to be $o(T)$.
Noting that the cumulative loss of BTL is $\approx -T$, using the previous bound we just proved, we get that

$$\forall u \in \mathcal{W}, \quad \text{Regret}(u) \le (T - 0.5) - (-T + 0.5) = 2T - 1.$$

The upperbound is above T, which makes this regret bound a weak one.
Towards the end of the lecture, it was hinted that with the Follow-The-Regularized-Leader algorithm, we can improve the stability of the FTL algorithm by adding a regularizing term. But that's for the next lecture!

References

[1] Shai Shalev-Shwartz, "Online Learning and Online Convex Optimization", Foundations and Trends in Machine Learning, 2012.

[2] Adam Kalai, Santosh Vempala, "Efficient Algorithms for online decision problems", Journal of Computer and System Sciences, 2005.