CMPT 210: Probability and Computing

Lecture 23

Sharan Vaswani

March 31, 2023

Randomized Load Balancing

Fussbook is a new social networking site oriented toward unpleasant people. Like all major web services, Fussbook has a load balancing problem: it receives lots of forum posts that computer servers have to process. If any server is assigned more work than it can complete in a given interval, then it is overloaded and system performance suffers. That would be bad because Fussbook users are not a tolerant bunch.

The programmers of Fussbook just randomly assigned posts to computers, and to their surprise the system has not crashed yet.

Fussbook receives 24000 forum posts in every 10-minute interval. Each post is assigned to one of several servers for processing, and each server works sequentially through its assigned tasks. It takes a server an average of $1/4$ second to process a post. No post takes more than 1 second.

This implies that a server could be overloaded when it is assigned more than 600 units of work in a 10-minute interval. On average, for $24000 \times \frac{1}{4} = 6000$ units of work in a 10-minute interval, Fussbook requires at least 10 servers to ensure that no server is overloaded (with perfect load-balancing).

Randomized Load Balancing

Q: There might be random fluctuations in the load or the load-balancing is not perfect. How many servers does Fussbook need to ensure that their servers are not overloaded with high-probability?

Let m be the number of servers that Fussbook needs to use. Recall that a server may be overloaded if the load it is assigned exceeds 600 units. Let us first look at server 1 and define T be the r.v. corresponding to the number of units of work assigned to the first server.

Let T_i be the number of seconds server 1 spends on processing post i. $T_i = 0$ if the task is assigned to a different (not the first server). The maximum amount of time spent on post i is 1-second. Hence, $T_i \in [0, 1]$.

Since there are $n := 24000$ posts in every 10-minute interval, the load (amount of units) assigned to the first server is equal to $T = \sum_{i=1}^{n} T_i$. Server 1 may be overloaded if $T \geq 600$, and hence we want to upper-bound the probability $\Pr[T \geq 600]$.

Since the assignment of a post to a server is independent of the time required to process the post, the T_i r.v's are mutually independent. Hence, we can use the Chernoff bound.

Randomized Load Balancing

We first need to estimate $\mathbb{E}[T]$.

$$\mathbb{E}[T] = \mathbb{E}[\sum_{i=1}^{n} T_i] = \sum_{i=1}^{n} \mathbb{E}[T_i] \qquad \text{(Linearity of expectation)}$$

$$\begin{aligned}\mathbb{E}[T_i] &= \sum_{i=1}^{n} \mathbb{E}[T_i|\text{server 1 is assigned post } i]\,\Pr[\text{server 1 is assigned post } i] \\ &\quad + \mathbb{E}[T_i|\text{server 1 is not assigned post } i]\,\Pr[\text{server 1 is not assigned post } i] \\ &= \frac{1}{4}\frac{1}{m} + (0)(1-1/m) = \frac{1}{4m}.\end{aligned}$$

$$\implies \mathbb{E}[T] = \sum_{i=1}^{n} \frac{1}{4m} = \frac{n}{4m} = \frac{6000}{m}.$$

Randomized Load Balancing

Recall the Chernoff Bound: $\Pr[T \geq c\mathbb{E}[T]] \leq \exp(-\beta(c)\,\mathbb{E}[T])$. In our case, $c\mathbb{E}[T] = 600 \implies c = \frac{m}{10}$. Hence,

$$\Pr[T \geq 600] \leq \exp\left(-\beta\left(\frac{m}{10}\right)\frac{6000}{m}\right)$$

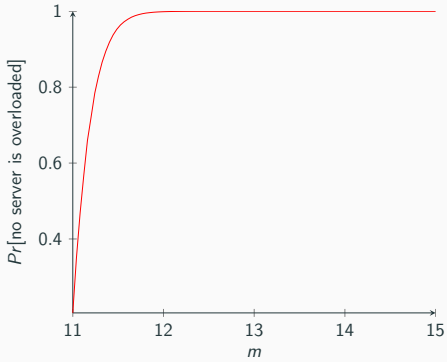
Hence, $\Pr[\text{first server is overloaded}] = \Pr[T \geq 600] \leq \exp\left(-\beta\left(\frac{m}{10}\right)\frac{6000}{m}\right)$.

$$\begin{aligned}
&\Pr[\text{some server is overloaded}] \\
&= \Pr[\text{server 1 is overloaded} \cup \text{server 2 is overloaded} \cup \ldots \cup \text{server m is overloaded}] \\
&\leq \sum_{j=1}^{m} \Pr[\text{server j is overloaded}] && \text{(Union Bound)} \\
&= m\Pr[\text{server 1 is overloaded}] \leq m\exp\left(-\beta\left(\frac{m}{10}\right)\frac{6000}{m}\right) && \text{(All servers are equivalent)}
\end{aligned}$$

$$\implies \Pr[\text{no server is overloaded}] \geq 1 - m\exp\left(-\beta\left(\frac{m}{10}\right)\frac{6000}{m}\right).$$

Randomized Load Balancing

Plotting Pr[no server is overloaded] as a function of m.



Hence, as $m \geq 12$, the probability that no server gets overloaded tends to 1 and hence none of the Fussbook servers crash!

Questions?

Machine Learning 101 – Tossing coins

Q: We have a coin such that $\Pr[\text{heads}] = q$. We toss this coin 5 times independently and record the observations. What is the probability that we get 3H and 2T in the 5 tosses.

If X is the r.v. equal to the number of heads in 5 tosses, then $X \sim \text{Bin}(5, q)$ and hence, $\Pr[\text{3H and 2T}] = \binom{5}{3} q^3 (1-q)^2$.

Q: We have a coin such that $\Pr[\text{heads}] = q$. We toss this coin 5 times independently. What is the probability that we get the following sequence of observations – HTHHT?

$$\begin{aligned}\Pr[\text{observe HTHHT}] &= \Pr[\text{Toss 1 is H} \cap \text{Toss 2 is T} \cap \ldots \cap \text{Toss 5 is T}] \\ &= \Pr[\text{Toss 1 is H}]\,\Pr[\text{Toss 2 is T}] \ldots \Pr[\text{Toss 5 is H}] = q^3(1-q)^2 \\ & \qquad \text{(Tosses are independent)}\end{aligned}$$

Q: If I use a different coin that has $\Pr[\text{heads}] = r$ and repeat the same experiment, what is the probability that we get the following sequence of observations – HTHHT?

Hence, $\Pr[\text{observe HTHHT sequence} | \Pr[\text{heads} = q]] = q^3(1-q)^2$.

Machine Learning 101 – Estimating the bias of a coin

Similar to the voter poll example, let us "invert" this reasoning – suppose we took a coin and want to estimate its bias i.e. figure out what is the Pr[heads]. To do this, we perform an experiment – toss the coin 5 times and record the observations. Suppose we get $HTHHT$ as the sequence of observations.

This sequence of observations is referred to as the **data** and denoted by $\mathcal{D}$. Using $\mathcal{D}$, we wish to **estimate** the bias of the coin.

If we estimate the (unknown) bias of the coin to be p, then the probability that we would see $\mathcal{D}$ is equal to $p^3(1-p)^2$ (by the same reasoning as on the last slide). Formally,

$$\Pr[\mathcal{D}|p] = p^3(1-p)^2$$

This is referred to as the **likelihood** of seeing the data if we were to estimate the bias to be p.

Machine Learning 101 – Estimating the bias of a coin

The standard way to "fit" the data and estimate the bias is **maximum likelihood estimation**. For this, we compute the estimate ($\hat{p}$) that maximizes the likelihood of observing $\mathcal{D}$. Formally,

$$\hat{p} = \arg\max_{p} \Pr[\mathcal{D}|p]$$

Here, $\arg\max_p$ returns the value of p that maximizes the likelihood. $\hat{p}$ is the **statistical estimate** of the unknown bias (similar to what we saw in the Voter Poll example) and is also referred to as the **maximum likelihood estimator (MLE)**.

It is equivalent and more convenient to calculate the minimizer of the **negative log-likelihood** (NLL) (since log is a monotonic function). The NLL is also referred to as the **loss function**. Formally,

$$\hat{p} = \arg\min_{p} [-\log(\Pr[\mathcal{D}|p])]$$

Machine Learning 101 – Estimating the bias of a coin

Let us compute the MLE for the bias of the coin. Recall that $\Pr[\mathcal{D}|p] = p^3(1-p)^2$.

$$-\log(\Pr(\mathcal{D}|p)) = -3\log(p) - 2\log(1-p) \implies \hat{p} = \arg\min_p[-3\log(p) - 2\log(1-p)]$$

Taking derivatives and setting it to zero,

$$\frac{d[-3\log(p) - 2\log(1-p)]}{dp} = 0 \implies -\frac{3}{\hat{p}} + \frac{2}{1-\hat{p}} = 0 \implies 5\hat{p} = 3 \implies \hat{p} = \frac{3}{5} = 0.6.$$

Checking that this is the minimum by computing the second derivative,

$$\frac{d^2[-3\log(p) - 2\log(1-p)]}{dp^2} = \frac{d[\frac{-3}{p} + \frac{2}{1-p}]}{dp} = \frac{3}{p^2} + \frac{2}{(1-p)^2} > 0 \qquad \text{(for } p \in (0,1))$$

Hence, $\hat{p}$ is the minimum of the NLL.

For this simple example, the MLE of Pr[heads]) is equal to the average number of heads we saw in $\mathcal{D}$.

Machine Learning 101 – Estimating the bias of a coin

If our experiment consists of tossing n coins and $n \to \infty$, then the MLE will tend to the true bias of the coin. Similar to the Voter Poll example, we can calculate the number of coin tosses we need such that MLE is ϵ close to the true bias of the coin with probability $1 - \delta$.

Q: Based on the results of our experiment, what should be our "guess"/"prediction" that we get a heads in the next toss of the coin?

We have estimated the bias of the coin to be equal to 0.6. Hence, given the results of our experiment, we should **predict** that we will get a heads with probability 0.6 when we toss this coin again in the future.

Machine learning

The basic framework in machine learning is to:

- Collect (training) data from the world (in this case, by tossing the coin).
- Construct a model that can explain the observations (in this case, our model was that each toss is independent and follows the same Bernoulli distribution).
- Use the model and $\mathcal{D}$ to construct the likelihood function (in this case, $p^3(1-p)^2$).
- Compute the MLE by minimizing the negative log-likelihood. This is an optimization problem (in this case, it was just taking derivatives) and is referred to as **training** the model to compute $\hat{p}$.
- Use the trained model to make predictions about the future (in this case, predict the probability that the next toss comes up heads). This is referred to as **prediction** or **inference**.

Machine Learning 101 – Estimating the bias of a coin

Q: Suppose someone hands us a new coin and asks us the following question: I tossed this coin 10 times, I got the sequence $HHTTHTTTTH$. What is the probability that I will see $4H$ and $6T$ in the next 10 tosses of the coin?

For computing the MLE of the bias of the coin, recall that it is equal to the average number of heads we got in the 10 tosses. Hence, $\hat{p} = 0.4$.

The question is that of predicting the probability of getting $4H$ and $6T$ in the next 10 tosses (referred to as the **test data**). If X is the r.v. equal to the number of heads in the next 10 tosses of the coin, then given $\mathcal{D}$, $X \sim \text{Bin}(10, 0.4)$.

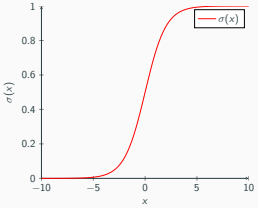
Hence, Pr[4H, 6T in the next 10 tosses$|\mathcal{D}] = \binom{10}{4}(0.4)^4(0.6)^6$.

Questions?

Digression – Sigmoid function

To extend this concept to more complicated problems, let us introduce the **sigmoid** function.

The **sigmoid** function is defined as: $\sigma : \mathbb{R} \to [0, 1]$: $\sigma(x) := \frac{1}{1+\exp(-x)}$.



Since the range of σ is $[0, 1]$, we will use it to output probabilities. Define parameter $\theta \in \mathbb{R}$ s.t. $p = \sigma(\theta) = \frac{1}{1+\exp(-\theta)}$.

$$1 - p = 1 - \frac{1}{1 + \exp(-\theta)} = \frac{\exp(-\theta)}{1 + \exp(-\theta)} = \frac{1}{1 + \exp(\theta)}$$

Hence, if $p = \sigma(\theta)$, $1 - p = \sigma(-\theta)$.

Digression – Sigmoid function

σ is an invertible function and hence there is a one-one mapping from θ to p (every p can be specified by specifying the equivalent θ). Formally, since $p = \sigma(\theta)$ and $1 - p = \sigma(-\theta)$.

$$\frac{p}{1-p} = \frac{\sigma(\theta)}{\sigma(-\theta)} = \frac{1+\exp(\theta)}{1+\exp(-\theta)} = \frac{\exp(\theta)\ (1+\exp(\theta))}{1+\exp(\theta)} = \exp(\theta) \implies \log\left(\frac{p}{1-p}\right) = \theta$$

Recall from Tutorial 4 that $\frac{p}{1-p}$ is referred to as the **odds**. Hence the sigmoid transformation is equivalent to choosing the parameter θ to represent the **log-odds**.

Machine Learning 101 – Estimating the bias of multiple coins

Q: Suppose now we toss 5 different coins and obtain the sequence $HTHHT$. We want to estimate the bias of each of these coins, but have some additional information that the bias of the coin i depends on its weight $x_i \in \mathbb{R}$ (which is known).

We will assume a **linear model** meaning that our model for the bias of coin is:

$$p_i = \sigma(\theta x_i) = \frac{1}{1 + \exp(-\theta x_i)}$$

Here, θ is the **parameter** of our model, x_i (the known weights) for the coins are referred to as the **features**. The model is **linear** because the argument to the sigmoid function is linear in θ.

As before, we need to obtain the MLE $\hat{\theta}$. Writing down the likelihood in terms of θ,

$$\Pr[\mathcal{D}|\{x_1, x_2, \ldots, x_5\}, \theta] = [\sigma(\theta x_1)]\,[\sigma(-\theta x_2)]\,[\sigma(\theta x_3)][\sigma(\theta x_4)][\sigma(-\theta x_5)]$$

To represent the likelihood in a more compact way, let us define $y_i \in \mathbb{R}$ such that $y_i = 1$ if coin i in $\mathcal{D}$ is a heads and $y_i = -1$ if coin i in $\mathcal{D}$ is a tails. For $\mathcal{D} = HTHHT$, $y_1 = 1$, $y_2 = -1$ and so on. For example i, y_i is referred to as the **label**, hence each toss i is described by the (x_i, y_i) pair referred to as the **input-output** pair or the **feature-label** pair.

$$\Pr[\mathcal{D}|\{x_1, x_2, \ldots, x_5\}, \theta] = [\sigma(y_1\theta x_1)]\,[\sigma(y_2\theta x_2)]\,[\sigma(y_5\theta x_5)] = \prod_{i=1}^{5}[\sigma(y_i\theta x_i)]$$

$$\implies -\log(\Pr[\mathcal{D}|\{x_1, x_2, \ldots, x_5\}, \theta]) = \sum_{i=1}^{5} -\log(\sigma(y_i\theta x_i)) = \sum_{i=1}^{5} \log\left(1 + \exp(-y_i\theta x_i)\right)$$

The NLL defined above is referred to as the **logistic loss** and this model is referred to as (1-dimensional) **logistic regression**. Since we are classifying the coins as those that came up heads or tails, we are doing **binary classification**.

Logistic regression for binary classification is heavily used in machine learning. E.g. Classifying whether a patient with feature x has cancer or not.

Machine Learning 101 – Estimating the bias of multiple coins

In order to compute the MLE ($\hat{\theta}$), we need to minimize the NLL on the previous slide,

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{5} \log\left(1 + \exp(-y_i \theta x_i)\right).$$

Unfortunately, this optimization problem can not be solved directly by taking derivatives like before. We need techniques from **numerical optimization** that studies efficiently minimizing complicated functions and the related computational properties (for example, see `https://vaswanis.github.io/409_981-F22.html`).

Once we have computed $\hat{\theta}$, we can use it to predict the probability of heads for a new coin that has feature x as $\hat{p} = \sigma(\hat{\theta}x)$.

Questions?