Online convex optimization 2

Source: David Childer’s 73-423 note

Review

- Last time, we talked about two online learning algorithm: Perceptron and Follow the Leader (FTL)

- We showed that FTL can have a worst-case regret of order T. We now introduce a new algorithm that, under some conditions, can produce regret of order $\sqrt{T}$

Follow the Regularized Leader (FTRL)

- Previously, we showed that FTL can be unstable when $\ell_t(\theta) - \ell_{t+1}(\theta)$ is large

Follow the Regularized Leader (FTRL)

- Previously, we showed that FTL can be unstable when $\ell_t(\theta) - \ell_{t+1}(\theta)$ is large

- To avoid instability, we modify FTL by **regularization**:

Follow the Regularized Leader (FTRL)

- Previously, we showed that FTL can be unstable when $\ell_t(\theta) - \ell_{t+1}(\theta)$ is large

- To avoid instability, we modify FTL by **regularization**:
    - We choose a sequence of vectors $\theta_1, \ldots, \theta_T$ that minimizes the regret

Follow the Regularized Leader (FTRL)

- Previously, we showed that FTL can be unstable when $\ell_t(\theta) - \ell_{t+1}(\theta)$ is large

- To avoid instability, we modify FTL by **regularization**:
    - We choose a sequence of vectors $\theta_1, \ldots, \theta_T$ that minimizes the regret
    - After choosing θ_t, we observe a loss $\ell_t(\theta_t)$

        Example: $\ell_t(\theta) = (\theta^T x_t - y_t)^2$

Follow the Regularized Leader (FTRL)

- Previously, we showed that FTL can be unstable when $\ell_t(\theta) - \ell_{t+1}(\theta)$ is large

- To avoid instability, we modify FTL by **regularization**:
  - We choose a sequence of vectors $\theta_1, \ldots, \theta_T$ that minimizes the regret
  - After choosing θ_t, we observe a loss $\ell_t(\theta_t)$

    Example: $\ell_t(\theta) = (\theta^T x_t - y_t)^2$

At step t:

$$\text{Choose } \theta_t \text{ that minimizes } \sum_{s=1}^{t-1} \ell_s(\theta_t) + \lambda R(\theta_t)$$

where R is a **regularizer**

Follow the Regularized Leader (FTRL)

Input: A regularization parameter $\lambda > 0$

Follow the Regularized Leader (FTRL)

Input: A regularization parameter $\lambda > 0$

Initialize $\theta_1 \in \mathbb{R}^d$

Follow the Regularized Leader (FTRL)

Input: A regularization parameter $\lambda > 0$

Initialize $\theta_1 \in \mathbb{R}^d$

for $t = 2$ **to** T **do**

1. Choose θ_t that minimizes $\sum_{s=1}^{t-1} \ell_s(\theta_t) + \lambda R(\theta_t)$

Follow the Regularized Leader (FTRL)

Input: A regularization parameter $\lambda > 0$

Initialize $\theta_1 \in \mathbb{R}^d$

for $t = 2$ **to** T **do**

1. Choose θ_t that minimizes $\sum_{s=1}^{t-1} \ell_s(\theta_t) + \lambda R(\theta_t)$
2. Receive new data point

Follow the Regularized Leader (FTRL)

Input: A regularization parameter $\lambda > 0$

Initialize $\theta_1 \in \mathbb{R}^d$

for $t = 2$ **to** T **do**

1. Choose θ_t that minimizes $\sum_{s=1}^{t-1} \ell_s(\theta_t) + \lambda R(\theta_t)$
2. Receive new data point
3. Compute loss $\ell_t(\theta_t)$

end

Follow the Regularized Leader (FTRL)

Input: A regularization parameter $\lambda > 0$

Initialize $\theta_1 \in \mathbb{R}^d$

for $t = 2$ **to** T **do**

1. Choose θ_t that minimizes $\sum_{s=1}^{t-1} \ell_s(\theta_t) + \lambda R(\theta_t)$
2. Receive new data point
3. Compute loss $\ell_t(\theta_t)$

end

Examples of R:

- Square regularizer: $R(x) = \frac{1}{2}\|x\|_2^2$

Follow the Regularized Leader (FTRL)

Input: A regularization parameter $\lambda > 0$

Initialize $\theta_1 \in \mathbb{R}^d$

for $t = 2$ **to** T **do**

1. Choose θ_t that minimizes $\sum_{s=1}^{t-1} \ell_s(\theta_t) + \lambda R(\theta_t)$
2. Receive new data point
3. Compute loss $\ell_t(\theta_t)$

end

Examples of R:

- Square regularizer: $R(x) = \frac{1}{2}\|x\|_2^2$
- Entropic regularizer: $R(x) = \sum_{i=1}^{d} x_i \log x_i$ over $\{x \in \mathbb{R}^d \ : \ \sum_i x_i = 1, x_i \geq 0\}$

Follow the Regularized Leader (FTRL)

Input: A regularization parameter $\lambda > 0$

Initialize $\theta_1 \in \mathbb{R}^d$

$\textbf{for } t = 2 \textbf{ to } T \textbf{ do}$

1. Choose θ_t that minimizes $\sum_{s=1}^{t-1} \ell_s(\theta_t) + \lambda R(\theta_t)$
2. Receive new data point
3. Compute loss $\ell_t(\theta_t)$

end

Parameter $\lambda > 0$ determines strength of the regularization: small values closer to FTL, large values closer to minimizing R

Follow the Regularized Leader (FTRL)

Input: A regularization parameter $\lambda > 0$

Initialize $\theta_1 \in \mathbb{R}^d$

for $t = 2$ **to** T **do**

1. Choose θ_t that minimizes $\sum_{s=1}^{t-1} \ell_s(\theta_t) + \lambda R(\theta_t)$
2. Receive new data point
3. Compute loss $\ell_t(\theta_t)$

end

Parameter $\lambda > 0$ determines strength of the regularization: small values closer to FTL, large values closer to minimizing R

Corresponds exactly to running regularized optimization each round

Example 1: Online Linear Optimization

- A data stream: $x_1, \ldots, x_T \in \mathbb{R}^d$

Example 1: Online Linear Optimization

- A data stream: $x_1, \ldots, x_T \in \mathbb{R}^d$

- Let $\ell_t(\theta_t) = \theta_t^T x_t$ and use quadratic regularizer $R(\theta_t) = \frac{1}{2}\|\theta\|_2^2$

Example 1: Online Linear Optimization

- A data stream: $x_1, \ldots, x_T \in \mathbb{R}^d$

- Let $\ell_t(\theta_t) = \theta_t^T x_t$ and use quadratic regularizer $R(\theta_t) = \frac{1}{2}\|\theta\|_2^2$

- FTRL update step is

$$\text{Choose } \theta_t \text{ that minimizes } \sum_{s=1}^{t-1} \theta_t^T x_s + \frac{1}{2}\lambda \|\theta_t\|_2^2$$

Example 1: Online Linear Optimization

- A data stream: $x_1, \ldots, x_T \in \mathbb{R}^d$

- Let $\ell_t(\theta_t) = \theta_t^T x_t$ and use quadratic regularizer $R(\theta_t) = \frac{1}{2}\|\theta\|_2^2$

- FTRL update step is

$$\text{Choose } \theta_t \text{ that minimizes } \sum_{s=1}^{t-1} \theta_t^T x_s + \frac{1}{2}\lambda\|\theta_t\|_2^2$$

- Solving the first-order condition for θ_t,

$$\theta_t = -\frac{1}{\lambda}\sum_{s=1}^{t-1} x_s = \theta_{t-1} - \frac{1}{\lambda}x_{t-1}$$

Algorithm: Online Linear Optimization

Input: A stream of data $x_1, x_2, , \ldots, x_T$, where $x_t \in \mathbb{R}^d$, a regularized parameter $\lambda > 0$

Initialize the coefficients $\theta_1 \in \mathbb{R}^d$

for $t = 2$ **to** T **do**

$\theta_t = \theta_{t-1} - \frac{1}{\lambda} x_{t-1}$

end

Algorithm: Online Linear Optimization

Input: A stream of data $x_1, x_2, , \dots, x_T$, where $x_t \in \mathbb{R}^d$, a regularized parameter $\lambda > 0$

Initialize the coefficients $\theta_1 \in \mathbb{R}^d$

for $t = 2$ **to** T **do**
  $\theta_t = \theta_{t-1} - \frac{1}{\lambda} x_{t-1}$
end

With $\lambda \approx \sqrt{T}$ gives $\mathrm{Regret}_T = \sqrt{T}$

Linearization

- Let θ^* minimize the total loss $\sum_{t=1}^{T} \ell_t(\theta)$

Linearization

- Let θ^* minimize the total loss $\sum_{t=1}^{T} \ell_t(\theta)$

- Recall that if ℓ_t is convex, then for any θ_t,

$$\ell_t(\theta^*) \geq \ell_t(\theta_t) + \nabla \ell_t(\theta_t)^T (\theta^* - \theta_t)$$

Linearization

- Let θ^* minimize the total loss $\sum_{t=1}^{T} \ell_t(\theta)$

- Recall that if ℓ_t is convex, then for any θ_t,

$$\ell_t(\theta^*) \geq \ell_t(\theta_t) + \nabla \ell_t(\theta_t)^T(\theta^* - \theta_t)$$

- Rearranging, we obtain

$$\ell_t(\theta_t) - \ell_t(\theta^*) \leq \nabla \ell_t(\theta_t)^T(\theta_t - \theta^*)$$

Linearization

- Let θ^* minimize the total loss $\sum_{t=1}^{T} \ell_t(\theta)$

- Recall that if ℓ_t is convex, then for any θ_t,

$$\ell_t(\theta^*) \geq \ell_t(\theta_t) + \nabla \ell_t(\theta_t)^T(\theta^* - \theta_t)$$

- Rearranging, we obtain

$$\ell_t(\theta_t) - \ell_t(\theta^*) \leq \nabla \ell_t(\theta_t)^T(\theta_t - \theta^*)$$

- Taking the sum over t,

$$\sum_{t=1}^{T} \ell_t(\theta_t) - \sum_{t=1}^{T} \ell_t(\theta^*) \leq \sum_{t=1}^{T} \nabla \ell_t(\theta_t)^T(\theta_t - \theta^*)$$

Linearization

- Define $\tilde{\ell}_t(\theta) = \nabla \ell_t(\theta_t)^T \theta$. We can rewrite the right-hand side

$$\sum_{t=1}^{T} \ell_t(\theta_t) - \sum_{t=1}^{T} \ell_t(\theta^*) \leq \sum_{t=1}^{T} \tilde{\ell}_t(\theta_t) - \sum_{t=1}^{T} \tilde{\ell}_t(\theta^*)$$

Linearization

- Define $\tilde{\ell}_t(\theta) = \nabla \ell_t(\theta_t)^T \theta$. We can rewrite the right-hand side

$$\sum_{t=1}^{T} \ell_t(\theta_t) - \sum_{t=1}^{T} \ell_t(\theta^*) \leq \sum_{t=1}^{T} \tilde{\ell}_t(\theta_t) - \sum_{t=1}^{T} \tilde{\ell}_t(\theta^*)$$

- In other words,

$$\mathrm{Regret}_T(\ell_{1:T}) \leq \mathrm{Regret}_T(\tilde{\ell}_{1:T})$$

  If the right-hand side is small, then so is the left-hand side

Linearization

- Define $\tilde{\ell}_t(\theta) = \nabla \ell_t(\theta_t)^T \theta$. We can rewrite the right-hand side

$$\sum_{t=1}^{T} \ell_t(\theta_t) - \sum_{t=1}^{T} \ell_t(\theta^*) \leq \sum_{t=1}^{T} \tilde{\ell}_t(\theta_t) - \sum_{t=1}^{T} \tilde{\ell}_t(\theta^*)$$

- In other words,

$$\mathrm{Regret}_T(\ell_{1:T}) \leq \mathrm{Regret}_T(\tilde{\ell}_{1:T})$$

  If the right-hand side is small, then so is the left-hand side

- **Linearization trick**: replace ℓ_t with $\tilde{\ell}_t$ in the FTRL algorithm

Online Gradient Descent

- We will use the square regularizer:

$$\text{Choose } \theta_t \text{ that minimizes } \sum_{s=1}^{t-1} \nabla \ell_s(\theta_s)^T \theta_t + \frac{1}{2}\lambda \|\theta_t\|_2^2$$

Online Gradient Descent

- We will use the square regularizer:

$$\text{Choose } \theta_t \text{ that minimizes } \sum_{s=1}^{t-1} \nabla \ell_s(\theta_s)^T \theta_t + \frac{1}{2}\lambda \|\theta_t\|_2^2$$

- Solving the first-order condition for θ_t,

$$\theta_t = -\frac{1}{\lambda} \sum_{s=1}^{t-1} \nabla \ell_s(\theta_s)^T = \theta_{t-1} - \frac{1}{\lambda} \nabla \ell_{t-1}(\theta_{t-1})$$

Online Gradient Descent

Input: A regularization parameter $\lambda > 0$

Initialize $\theta_1 \in \mathbb{R}^d$

for $t = 2$ **to** T **do**

1. Update $\theta_t = \theta_{t-1} - \frac{1}{\lambda}\nabla \ell_{t-1}(\theta_{t-1})$
2. Receive new data point
3. Compute loss gradient $\nabla \ell_t(\theta_t)$

end

FTRL in practice

- In practice, hard part of implementation is choosing regularization parameter λ

FTRL in practice

- In practice, hard part of implementation is choosing regularization parameter λ

- Need to know time frame T which might not be possible

FTRL in practice

- In practice, hard part of implementation is choosing regularization parameter λ

- Need to know time frame T which might not be possible

- This has led to variety of modified algorithms which choose T adaptively

FTRL in practice

- In practice, hard part of implementation is choosing regularization parameter λ

- Need to know time frame T which might not be possible

- This has led to variety of modified algorithms which choose T adaptively

- If time frame not known, simple approach is **doubling trick**
  - Set the first horizon T_1

    For $t = 1, \ldots, T_1$, run FTRL with $\lambda = \sqrt{T_1}$

FTRL in practice

- In practice, hard part of implementation is choosing regularization parameter λ

- Need to know time frame T which might not be possible

- This has led to variety of modified algorithms which choose T adaptively

- If time frame not known, simple approach is **doubling trick**
  - Set the first horizon T_1

    For $t = 1, \ldots, T_1$, run FTRL with $\lambda = \sqrt{T_1}$
  - When we reach T_k, set $T_{k+1} = 2T_k$

    For $t = T_k + 1, T_k + 2, \ldots, T_{k+1}$, run FTRL with $\lambda = \sqrt{T_k}$

FTRL in practice

- Can also use continuously-updated penalties, e.g. $\lambda_t = \sqrt{t}$

FTRL in practice

- Can also use continuously-updated penalties, e.g. $\lambda_t = \sqrt{t}$
- A popular method is **Adagrad**
  - Change Online Gradient Descent update to

$$\theta_t = \theta_{t-1} - \frac{1}{\lambda\sqrt{\sum_{s=1}^{t-1} \nabla \ell_s^2(\theta_s)}} \nabla \ell_{t-1}(\theta_{t-1})$$

FTRL in practice

- Can also use continuously-updated penalties, e.g. $\lambda_t = \sqrt{t}$
- A popular method is **Adagrad**
  - Change Online Gradient Descent update to

$$\theta_t = \theta_{t-1} - \frac{1}{\lambda\sqrt{\sum_{s=1}^{t-1} \nabla \ell_s^2(\theta_s)}} \nabla \ell_{t-1}(\theta_{t-1})$$

- For online gradient descent **with constraint**, we can enforce our update to be inside the feasible set
  - $\theta_t = \theta_{t-1} - \frac{1}{\lambda} \nabla \ell_{t-1}(\theta_{t-1})$
  - Move θ_t to the closest point in $\mathcal{K}$

This is called **projected online gradient descent**

Application: Electricity forecasting

- Toy example: predict weekly electricity consumption

- Task important for power companies, which must buy and sell excess production on interchange markets

Application: Electricity forecasting

- Toy example: predict weekly electricity consumption

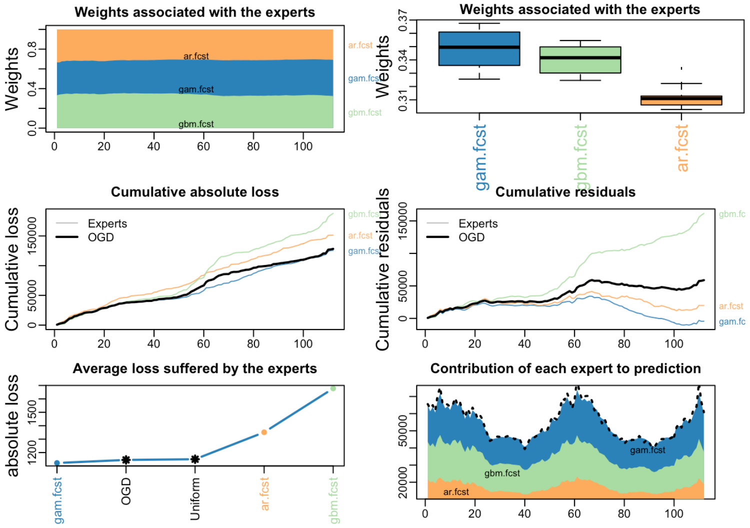
- Task important for power companies, which must buy and sell excess production on interchange markets

- Train 3 complicated statistical/machine learning models on training set, then every week use their forecasts with incoming predictors (temperature, season, etc) to predict that week's usage

Application: Electricity forecasting

- Toy example: predict weekly electricity consumption
- Task important for power companies, which must buy and sell excess production on interchange markets
- Train 3 complicated statistical/machine learning models on training set, then every week use their forecasts with incoming predictors (temperature, season, etc) to predict that week's usage
- Let $\lambda = \sqrt{t}$ for online gradient descent

Results


Weights associated with the experts
Weights
ar.fcst
gam.fcst
gbm.fcst
Weights associated with the experts
Weights
gam.fcst
gbm.fcst
ar.fcst
Cumulative absolute loss
Cumulative loss
Experts
OGD
Cumulative residuals
Cumulative residuals
Experts
OGD
Average loss suffered by the experts
absolute loss
gam.fcst
OGD
Uniform
ar.fcst
gbm.fcst
Contribution of each expert to prediction
gam.fcst
gbm.fcst
ar.fcst

Application: Ad-Click Prediction at Google

- Google implemented system to forecast probability of clicking on ads (McMahan et al. 2013)

- Want system to automatically give new prediction for each ad, for each customer

Application: Ad-Click Prediction at Google

- Google implemented system to forecast probability of clicking on ads (McMahan et al. 2013)

- Want system to automatically give new prediction for each ad, for each customer

- Apply online method to update continuously and automatically

Application: Ad-Click Prediction at Google

- Google implemented system to forecast probability of clicking on ads (McMahan et al. 2013)
- Want system to automatically give new prediction for each ad, for each customer
- Apply online method to update continuously and automatically
- Want to use ad and user-level attributes for prediction

Application: Ad-Click Prediction at Google

- Apply features in online logistic regression:

$$\ell_t(\theta) = -y_t \text{logit}_\theta(x_t) - (1 - y_t)(1 - \text{logit}_\theta(x_t))$$

Application: Ad-Click Prediction at Google

- Apply features in online logistic regression:

$$\ell_t(\theta) = -y_t \mathrm{logit}_\theta(x_t) - (1 - y_t)(1 - \mathrm{logit}_\theta(x_t))$$

- Individual sites and each have own attributes (ie, each person's search history, etc)

Application: Ad-Click Prediction at Google

- Apply features in online logistic regression:

$$\ell_t(\theta) = -y_t \text{logit}_\theta(x_t) - (1 - y_t)(1 - \text{logit}_\theta(x_t))$$

- Individual sites and each have own attributes (ie, each person's search history, etc)

- Need extreme speed and scale: use sparse prediction method, using only a few coefficients at a time

Application: Ad-Click Prediction at Google

- Apply features in online logistic regression:

$$\ell_t(\theta) = -y_t \text{logit}_\theta(x_t) - (1 - y_t)(1 - \text{logit}_\theta(x_t))$$

- Individual sites and each have own attributes (ie, each person's search history, etc)

- Need extreme speed and scale: use sparse prediction method, using only a few coefficients at a time

- **Approach**: Linearized FTRL with particular choice of regularizer

$$\theta_t \text{ that minimizes } \sum_{s=1}^{t-1} \nabla \ell_s^T \theta_t + \sum_{s=1}^{t-1} \sigma_s \|\theta_t - \theta_s\|^2 + \lambda \|\theta_t\|_1$$

Application: Ad-Click Prediction at Google

Approach: Linearized FTRL with particular choice of regularizer

$$\theta_t \text{ that minimizes } \sum_{s=1}^{t-1} \nabla \ell_s^T \theta_t + \sum_{s=1}^{t-1} \sigma_s \|\theta_t - \theta_s\|^2 + \lambda \|\theta_t\|_1$$

- Uses regularizer depending on whole past sequence of θ_s, plus LASSO penalty
- Latter acts like Lasso penalty; former like square penality, but leads to more computationally efficient updates