Online Learning

Lorenzo Rosasco

MIT, 9.520

About this class

Goal To introduce theory and algorithms for online learning.

Plan

- Different views on online learning
- From batch to online least squares
- Other loss functions
- Theory

(Batch) Learning Algorithms

A learning algorithm $\mathcal{A}$ is a map from the data space into the hypothesis space and

$$f_S = \mathcal{A}(S),$$

where $S = S_n = (x_0, y_0) \dots . (x_{n-1}, y_{n-1})$.

We typically assume that:

- $\mathcal{A}$ is deterministic,
- $\mathcal{A}$ does not depend on the ordering of the points in the training set.

notation: note the weird numbering of the training set!

Online Learning Algorithms

The pure online learning approach is:

let $f_1 = \text{init}$
for $n = 1, \ldots$
$f_{n+1} = \mathcal{A}(f_n, (x_n, y_n))$

- The algorithm works sequentially and has a recursive definition.

Online Learning Algorithms (cont.)

A related approach (similar to transductive learning) is:

let $f_1 = \text{init}$
for $n = 1, \ldots$
$f_{n+1} = \mathcal{A}(f_n, S_n, (x_n, y_n))$

- Also in this case the algorithm works sequentially and has a recursive definition, but it requires storing the past data S_n.

Why Online Learning?

Different motivations/perspectives that often corresponds to different theoretical framework.

- Biologically plausibility.
- Stochastic approximation.
- Incremental Optimization.
- Non iid data, game theoretic view.

Our goal is to minimize the expected risk

$$I[f] = \mathbb{E}_{(x,y)}\left[V(f(x), y)\right] = \int V(f(x), y) d\mu(x, y)$$

over the hypothesis space $\mathcal{H}$, but the data distribution is not known.

The idea is to use the samples to build an approximate solution and to update such a solution as we get more data.

Our goal is to minimize the expected risk

$$I[f] = \mathbb{E}_{(x,y)}\left[V(f(x), y)\right] = \int V(f(x), y) d\mu(x, y)$$

over the hypothesis space $\mathcal{H}$, but the data distribution is not known.

The idea is to use the samples to build an approximate solution and to update such a solution as we get more data.

Online Learning and Stochastic Approximation (cont.)

More precisely if

- we are given samples $(x_i, y_i)_i$ in a sequential fashion
- at the $n-th$ step we have an approximation $G(f, (x_n, y_n))$ of the gradient of $I[f]$

then we can define a recursion by

let $f_0 = \text{init}$
for $n = 0, \ldots$
$f_{n+1} = f_n + \gamma_n(G(f_n, (x_n, y_n))$

Online Learning and Stochastic Approximation (cont.)

More precisely if

- we are given samples $(x_i, y_i)_i$ in a sequential fashion
- at the $n-th$ step we have an approximation $G(f, (x_n, y_n))$ of the gradient of $I[f]$

then we can define a recursion by

let $f_0 = \text{init}$
for $n = 0, \ldots$
$f_{n+1} = f_n + \gamma_n (G(f_n, (x_n, y_n))$

Incremental Optimization

Here our goal is to solve empirical risk minimization

$$I_S[f],$$

or regularized empirical risk minimization

$$I_S^\lambda[f] = I_S[f] + \lambda \|f\|^2$$

over the hypothesis space $\mathcal{H}$, when the number of points is so big (say $n = 10^8 - 10^9$) that standard solvers would not be feasible.

Memory is the main constraint here.

Incremental Optimization (cont.)

In this case we can consider

let $f_0 = \text{init}$
for $t = 0, \ldots$
$f_{t+1} = f_t + \gamma_t(G(f_t, (x_{n_t}, y_{n_t}))$

where here $G(f_t, (x_{n_t}, y_{n_t}))$ is a pointwise estimate of the gradient of I_S or I_S^λ.

Epochs

Note that in this case the number of iteration is decoupled to the index of training set points and we can look at the data more than once, that is consider different *epochs*.

Non i.i.d. data, game theoretic view

If the data are not i.i.d. we can consider a setting when the data is a finite sequence that we will be disclosed to us in a sequential (possibly adversarial) fashion.

Then we can see learning as a two players game where

- at each step *nature* chooses a samples (x_i, y_i)
- at each step a *learner* chooses an estimator f_{i+1}.

The goal of the learner is to perform as well as if he could view the whole sequence.

Non i.i.d. data, game theoretic view

If the data are not i.i.d. we can consider a setting when the data is a finite sequence that we will be disclosed to us in a sequential (possibly adversarial) fashion.

Then we can see learning as a two players game where

- at each step *nature* chooses a samples (x_i, y_i)
- at each step a *learner* chooses an estimator f_{i+1}.

The goal of the learner is to perform as well as if he could view the whole sequence.

Plan

- Different views on online learning
- From batch to online least squares
- Other loss functions
- Theory

Recalling Least Squares

We start considering a linear kernel so that

$$I_S[f] = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - x_i^T w) = \|Y - Xw\|^2$$

Remember that in this case

$$w_n = (X^T X)^{-1} X^T Y = C_n^{-1} \sum_{i=0}^{n-1} x_i y_i.$$

(Note that if we regularize we have $(C_n + \lambda I)^{-1}$ in place of C_n^{-1}.

notation: note the weird numbering of the training set!

Recalling Least Squares

We start considering a linear kernel so that

$$I_S[f] = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - x_i^T w) = \|Y - Xw\|^2$$

Remember that in this case

$$w_n = (X^T X)^{-1} X^T Y = C_n^{-1} \sum_{i=0}^{n-1} x_i y_i.$$

(Note that if we regularize we have $(C_n + \lambda I)^{-1}$ in place of C_n^{-1}.

notation: note the weird numbering of the training set!

A Recursive Least Squares Algorithm

Then we can consider

$$w_{n+1} = w_n + C_{n+1}^{-1} x_n [y_n - x_n^T w_n].$$

Proof

- $w_n = C_n^{-1}(\sum_{i=0}^{n-1} x_i y_i)$
- $w_{n+1} = C_{n+1}^{-1}(\sum_{i=0}^{n-1} x_i y_i + x_n y_n)$
- $w_{n+1} - w_n = C_{n+1}^{-1}(x_n y_n) + C_{n+1}^{-1}(C_n - C_{n+1}) C_n^{-1} \sum_{i=0}^{n-1} x_i y_i$
- $C_{n+1} - C_n = x_n x_n^T$.

A Recursive Least Squares Algorithm

Then we can consider

$$w_{n+1} = w_n + C_{n+1}^{-1} x_n [y_n - x_n^T w_n].$$

Proof

- $w_n = C_n^{-1}(\sum_{i=0}^{n-1} x_i y_i)$
- $w_{n+1} = C_{n+1}^{-1}(\sum_{i=0}^{n-1} x_i y_i + x_n y_n)$
- $w_{n+1} - w_n = C_{n+1}^{-1}(x_n y_n) + C_{n+1}^{-1}(C_n - C_{n+1}) C_n^{-1} \sum_{i=0}^{n-1} x_i y_i$
- $C_{n+1} - C_n = x_n x_n^T$.

A Recursive Least Squares Algorithm

Then we can consider

$$w_{n+1} = w_n + C_{n+1}^{-1} x_n [y_n - x_n^T w_n].$$

Proof

- $w_n = C_n^{-1} (\sum_{i=0}^{n-1} x_i y_i)$
- $w_{n+1} = C_{n+1}^{-1} (\sum_{i=0}^{n-1} x_i y_i + x_n y_n)$
- $w_{n+1} - w_n = C_{n+1}^{-1} (x_n y_n) + C_{n+1}^{-1} (C_n - C_{n+1}) C_n^{-1} \sum_{i=0}^{n-1} x_i y_i$
- $C_{n+1} - C_n = x_n x_n^T$.

A Recursive Least Squares Algorithm

Then we can consider

$$w_{n+1} = w_n + C_{n+1}^{-1} x_n [y_n - x_n^T w_n].$$

Proof

- $w_n = C_n^{-1} (\sum_{i=0}^{n-1} x_i y_i)$
- $w_{n+1} = C_{n+1}^{-1} (\sum_{i=0}^{n-1} x_i y_i + x_n y_n)$
- $w_{n+1} - w_n = C_{n+1}^{-1} (x_n y_n) + C_{n+1}^{-1} (C_n - C_{n+1}) C_n^{-1} \sum_{i=0}^{n-1} x_i y_i$
- $C_{n+1} - C_n = x_n x_n^T$.

A Recursive Least Squares Algorithm (cont.)

We derived the algorithm

$$w_{n+1} = w_n + C_{n+1}^{-1} x_n [y_n - x_n^T w_n].$$

The above approach

- is recursive;
- requires storing all the data;
- requires inverting a matrix $(C_i)_i$ at each step.

A Recursive Least Squares Algorithm (cont.)

The following matrix equality allows to alleviate the computational burden.

Matrix Inversion Lemma

$$[A + BCD]^{-1} = A^{-1} - A^{-1}B[DA^{-1}B + C^{-1}]^{-1}DA^{-1}$$

Then

$$C_{n+1}^{-1} = C_n^{-1} - \frac{C_n^{-1}x_n x_n^T C_n^{-1}}{1 + x_n^T C_n^{-1} x_n}.$$

A Recursive Least Squares Algorithm (cont.)

The following matrix equality allows to alleviate the computational burden.

Matrix Inversion Lemma

$$[A + BCD]^{-1} = A^{-1} - A^{-1}B[DA^{-1}B + C^{-1}]^{-1}DA^{-1}$$

Then

$$C_{n+1}^{-1} = C_n^{-1} - \frac{C_n^{-1}x_n x_n^T C_n^{-1}}{1 + x_n^T C_n^{-1} x_n}.$$

A Recursive Least Squares Algorithm (cont.)

Moreover

$$C_{n+1}^{-1} x_n = C_n^{-1} x_n - \frac{C_n^{-1} x_n x_n^T C_n^{-1}}{1 + x_n^T C_n^{-1} x_n} x_n = \frac{C_n^{-1}}{1 + x_n^T C_n^{-1} x_n} x_n,$$

we can derive the algorithm

$$w_{n+1} = w_n + \frac{C_n^{-1}}{1 + x_n^T C_n^{-1} x_n} x_n [y_n - x_n^T w_n].$$

Since the above iteration is equivalent to empirical risk minimization (ERM) the conditions ensuring its convergence – as $n \to \infty$ – are the same as those for ERM.

A Recursive Least Squares Algorithm (cont.)

Moreover

$$C_{n+1}^{-1} x_n = C_n^{-1} x_n - \frac{C_n^{-1} x_n x_n^T C_n^{-1}}{1 + x_n^T C_n^{-1} x_n} x_n = \frac{C_n^{-1}}{1 + x_n^T C_n^{-1} x_n} x_n,$$

we can derive the algorithm

$$w_{n+1} = w_n + \frac{C_n^{-1}}{1 + x_n^T C_n^{-1} x_n} x_n [y_n - x_n^T w_n].$$

Since the above iteration is equivalent to empirical risk minimization (ERM) the conditions ensuring its convergence – as $n \to \infty$ – are the same as those for ERM.

A Recursive Least Squares Algorithm (cont.)

Moreover

$$C_{n+1}^{-1} x_n = C_n^{-1} x_n - \frac{C_n^{-1} x_n x_n^T C_n^{-1}}{1 + x_n^T C_n^{-1} x_n} x_n = \frac{C_n^{-1}}{1 + x_n^T C_n^{-1} x_n} x_n,$$

we can derive the algorithm

$$w_{n+1} = w_n + \frac{C_n^{-1}}{1 + x_n^T C_n^{-1} x_n} x_n [y_n - x_n^T w_n].$$

Since the above iteration is equivalent to empirical risk minimization (ERM) the conditions ensuring its convergence – as $n \to \infty$ – are the same as those for ERM.

A Recursive Least Squares Algorithm (cont.)

The algorithm

$$w_{n+1} = w_{n-1} + \frac{C_n^{-1}}{1 + x_n^T C_n^{-1} x_n} x_n [y_n - x_n^T w_n].$$

is of the form

let $f_0 = \text{init}$
for $n = 1, \ldots$
$f_{n+1} = \mathcal{A}(f_n, S_n, (x_n, y_n))$.

The above approach

- is recursive;
- requires storing all the data;
- updates the inverse just via matrix/vector multiplication.

A Recursive Least Squares Algorithm (cont.)

The algorithm

$$w_{n+1} = w_{n-1} + \frac{C_n^{-1}}{1 + x_n^T C_n^{-1} x_n} x_n [y_n - x_n^T w_n].$$

is of the form

let $f_0 = \text{init}$
for $n = 1, \ldots$
$f_{n+1} = \mathcal{A}(f_n, S_n, (x_n, y_n))$.

The above approach

- is recursive;
- requires storing all the data;
- updates the inverse just via matrix/vector multiplication.

Recursive Least Squares (cont.)

How about the memory requirement?

The main constraint comes from the matrix storing/inversion

$$w_{n+1} = w_n + \frac{C_n^{-1}}{1 + x_n^T C_n^{-1} x_n} x_n [y_n - x_n^T w_n].$$

We can consider

$$w_{n+1} = w_n + \gamma_n x_n [y_n - x_n^T w_n].$$

where γ_t is a decreasing sequence.

Recursive Least Squares (cont.)

How about the memory requirement?
The main constraint comes from the matrix storing/inversion

$$w_{n+1} = w_n + \frac{C_n^{-1}}{1 + x_n^T C_n^{-1} x_n} x_n [y_n - x_n^T w_n].$$

We can consider

$$w_{n+1} = w_n + \gamma_n x_n [y_n - x_n^T w_n].$$

where γ_t is a decreasing sequence.

Recursive Least Squares (cont.)

How about the memory requirement?
The main constraint comes from the matrix storing/inversion

$$w_{n+1} = w_n + \frac{C_n^{-1}}{1 + x_n^T C_n^{-1} x_n} x_n [y_n - x_n^T w_n].$$

We can consider

$$w_{n+1} = w_n + \gamma_n x_n [y_n - x_n^T w_n].$$

where γ_t is a decreasing sequence.

Online Least Squares

The algorithm

$$w_{n+1} = w_n + \gamma_n x_n [y_n - x_n^T w_n].$$

- is recursive;
- does not requires storing the data;
- does not require updating the inverse, but only vector/vector multiplication.

Convergence of Online Least Squares (cont.)

When we consider

$$w_{n+1} = w_n + \gamma_n x_n [y_n - x_n^T w_n].$$

we are no longer guaranteed convergence.
In other words:

how shall we choose γ_n?

Batch vs Online Gradient Descent

Some ideas from the comparison with batch least squares.

$$w_{n+1} = w_n + \gamma_n x_n [y_n - x_n^T w_n].$$

Note that

$$\nabla (y_n - x_n^T w)^2 = x_n [y_n - x_n^T w].$$

The batch gradient algorithm would be

$$w_{n+1} = w_n + \gamma_n \frac{1}{t} \sum_{t=0}^{t-1} x_t [y_t - x_t^T w_n].$$

since

$$\nabla I(w) = \nabla \frac{1}{t} \sum^{t-1} (y_t - x_t^T w)^2 = \frac{1}{t} \sum^{t-1} x_t (y_t - x_t^T w)$$

Batch vs Online Gradient Descent

Some ideas from the comparison with batch least squares.

$$w_{n+1} = w_n + \gamma_n x_n [y_n - x_n^T w_n].$$

Note that

$$\nabla (y_n - x_n^T w)^2 = x_n [y_n - x_n^T w].$$

The batch gradient algorithm would be

$$w_{n+1} = w_n + \gamma_n \frac{1}{t} \sum_{t=0}^{t-1} x_t [y_t - x_t^T w_n].$$

since

$$\nabla I(w) = \nabla \frac{1}{t} \sum^{t-1} (y_t - x_t^T w)^2 = \frac{1}{t} \sum^{t-1} x_t (y_t - x_t^T w)$$

Convergence of Recursive Least Squares (cont.)

- If γ_n decreases too fast the iterate will get stuck away from the real solution.
- In general the convergence of the online iteratatoin is slower than the recursive least squares since we use a simpler step-size.

Polyak Averging

If we choose $\gamma_n = n^{-\alpha}$, with $\alpha \in (1/2, 1)$ and use, at each step, the solution obtained averaging all previous solutions (namely Polyak averaging), then convergence is ensured and is almost the same as recursive least squares.

Extensions

- Other Loss functions
- Other Kernels.

Other Loss functions

The same ideas can be extended to other convex differentiable loss functions, considering

$$w_{n+1} = w_n + \gamma_n \nabla V(y_n, x_n^T w_n).$$

If the loss is convex but not differentiable then more sophisticated methods must be used, e.g. proximal methods, subgradient methods.

Non linear Kernels

If we use different kernels $f(x) = \Phi(x)^T w = \sum_{i=0}^{n-1} c^i K(x, x_i)$ then we can consider the iteration

$$c_{n+1}^i = c_n^i, \quad i = 1, \dots, n$$
$$c_{n+1}^{n+1} = \gamma_n [y_n - c_n^T K_n(x_n)]),$$

where $K_n(x) = K(x, x_0), \dots, K(x, x_{n-1})$ and $c_n = (c_n^1, \dots, c_n^n)$.

Non linear Kernels (cont.)

Proof

- $f_n(x) = w_n^T \Phi(x) = K_n(x)^T c_n$.
- $$w_{n+1} = w_n + \gamma_n \Phi(x_n)[y_n - \Phi(x_n)^T w_n],$$

  gives, for all x,

  $$f_{n+1}(x) = f_n(x) + \gamma_n [y_n - K_n(x_n)^T c_n] K(x, x_n),$$

  that can be written as

  $$K_{n+1}(x)^T c_{n+1} = K_n(x)^T c_n + \gamma_n [y_n - K_n(x_n)^T c_n] K(x, x_n).$$

  QED

Non linear Kernels

$$c_{n+1}^i = c_n^i, \quad i = 1, \ldots, n$$
$$c_{n+1}^{n+1} = \gamma_n [y_n - c_n^T K_n(x_n)]).$$

The above approach

- is recursive;
- requires storing all the data;
- does not require updating the inverse, but only vector/vector multiplication – $O(n)$.