

GI01/4C55: Supervised Learning

3. Optimization and Learning Algorithms

October 17, 2005

Massimiliano Pontil

1

Today's plan

- Gradient descent
- Newton's method
- Empirical error minimization algorithms
- Perceptron algorithm
- Nonlinear hypothesis spaces

Bibliography: These lecture notes are available at:

<http://www.cs.ucl.ac.uk/staff/M.Pontil/courses/index-SL05.htm>

Lectures are in part based on Chapter 9 of Boyd and Vandenberghe

2

Empirical error minimization

$$E(\mathbf{w}) \equiv \mathcal{E}_{\text{emp}}(\mathbf{w}) = \sum_{i=1}^m V(y_i, \mathbf{w} \cdot \mathbf{x}_i)$$

How to minimize the empirical error?

- Least squares: $V(y, z) = (y - z)^2$
- Logistic regression: $V(y, z) = -\left\{y \log p(z) + (1 - y) \log (1 - p(z))\right\}$,
with

$$p(z) = \frac{1}{1 + e^{-z}}$$

Note: we have used the equivalent notation $\mathbf{w} \cdot \mathbf{x} \equiv \mathbf{w}^\top \mathbf{x}$

3

Empirical error minimization

$$E(\mathbf{w}) = \sum_{i=1}^m V(y_i, \mathbf{w} \cdot \mathbf{x}_i)$$

We have seen that the optimality equations, $\nabla E(\mathbf{w}) = 0$, are

- for Least squares: $\sum_{i=1}^m \mathbf{x}_i (y_i - \mathbf{x}_i \cdot \mathbf{w}) = 0$ (linear in $\mathbf{w}$)
- for Logistic regression: $\sum_{i=1}^m \mathbf{x}_i (y_i - p(\mathbf{x}_i \cdot \mathbf{w})) = 0$

4

Iterative learning algorithms (I)

Suppose we want to minimize some function $E(\mathbf{w})$ (think of a general function, not necessarily a sum of losses)

Iterative algorithm

- Choose a starting value $\mathbf{w}^{(0)}$ for $\mathbf{w}$
- Compute a sequence of points $\{\mathbf{w}^{(t)}\}_{t \geq 0}$ using the update rule

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta^{(t)} \Delta(\mathbf{w}^{(t)}), \quad \eta^{(t)} > 0, \quad t \geq 0$$

Typically a stopping criterion on $\mathbf{w}^{(t)}$ is checked after each iteration

5

Iterative learning algorithms (II)

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta^{(t)} \Delta(\mathbf{w}^{(t)}), \quad t \geq 0$$

Let $E^* := \min_{\mathbf{w}} E(\mathbf{w})$

We hope that $E(\mathbf{w}^{(t)}) \rightarrow E^*$ as $t \rightarrow \infty$

We will present some natural algorithms which are used in practice and discuss their application to Least squares, Logistic regression and the perceptron algorithm

Note: We will comment on but not prove the convergence properties of the algorithms. For more information see Chapter 9 of Boyd and Vanderberghe

6

Iterative learning algorithms (III)

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta^{(t)} \Delta(\mathbf{w}^{(t)}), \quad t \geq 0$$

- The vector $\Delta(\mathbf{w}^{(t)})$ is called the **search direction** (for simplicity, we let it depend on the previous update $\mathbf{w}^{(t)}$ only)
- The positive parameter η is called the **step length**

What are candidate choices for the search direction and step length?

7

Iterative learning algorithms (IV)

Suppose you have made your choice for Δ

A natural choice for η is the minimizer of $E(\mathbf{w} + \eta \Delta(\mathbf{w}))$

$$\eta := \operatorname{argmin} \{ E(\mathbf{w} + \rho \Delta(\mathbf{w})) : \rho > 0 \}$$

This procedure is called **line search** (or, sometimes, exact line search to distinguish it from approximate versions of it)

8

Iterative learning algorithms (V)

To choose the direction $\Delta(\mathbf{w})$ we require that

$$E(\mathbf{w}^{(t+1)}) < E(\mathbf{w}^{(t)}) \quad (*)$$

unless $\mathbf{w}^{(t)}$ is optimal

If E is a differentiable convex function (this is true in most cases we'll consider), we know that, for all $\mathbf{w}, \mathbf{v} \in \mathbb{R}^d$,

$$E(\mathbf{w} + \mathbf{v}) \geq E(\mathbf{w}) + \nabla E(\mathbf{w}) \cdot \mathbf{v}$$

Hence, to satisfy $(*)$ we must require that

$$\Delta(\mathbf{w}) \cdot \nabla E(\mathbf{w}) < 0$$

9

Gradient descent

Hence, a natural choice is to take Δ to be the negative gradient of E

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta^{(t)} \nabla E(\mathbf{w}^{(t)})$$

This choice reflects our intuition that if we take a little step down the gradient, E will decrease

- If $\eta^{(t)}$ is chosen via line search and E is strongly convex, algorithm converges: $E(\mathbf{w}^{(t)}) \rightarrow E^*$ as $t \rightarrow \infty$
- If the standard stopping criterion $\|\nabla E(\mathbf{w}^{(t)})\| \leq \epsilon$ is used, the algorithm terminates in a finite number of updates (which depends in particular on $\mathbf{w}^{(0)}$ and ϵ)

10

Newton's method (I)

Let us consider the problem of solving the eq. $G(w) = 0$, $w \in \mathbb{R}$

- Choose a starting point $w^{(0)}$ for w
- Compute a sequence $\{w^{(t)}\}_{t \geq 0}$ using the update rule
 - ◊ Compute $G'(w^{(t)})$
 - ◊ Let $w^{(t+1)} = w^{(t)} - \frac{G(w^{(t)})}{G'(w^{(t)})}$

If the method converges, $w^{(0)}$ is called an approximate zero of G

11

Newton's method (II)

To find the minimum of a function $E(w)$ we need to solve the equation $E'(w) = 0$

Applying Newton's method to $G = E'$, we get the iterative rule

$$w^{(t+1)} = w^{(t)} - \frac{E'(w^{(t)})}{E''(w^{(t)})}$$

Other interpretation: the quantity $-\frac{E'(w)}{E''(w)}$ minimizes the 2nd order Taylor approximation of E at w

$$E(w + v) \approx E(w) + E'(w)v + \frac{1}{2}E''(w)v^2$$

12

Newton's method (III)

More generally, in $\mathbb{R}^d$, the **Newton direction** is

$$\Delta(\mathbf{w}) = -\mathbf{H}^{-1}(\mathbf{w})\nabla E(\mathbf{w})$$

minimizes the quadratic approximation of E at $\mathbf{w}$

$$E(\mathbf{w} + \mathbf{v}) \approx E(\mathbf{w}) + \nabla E(\mathbf{w}) \cdot \mathbf{v} + \frac{1}{2}\mathbf{v} \cdot (\mathbf{H}(\mathbf{w})\mathbf{v})$$

where $\mathbf{H}(\mathbf{w})$ is the $d \times d$ Hessian matrix

$$H_{jk}(\mathbf{w}) = \frac{\partial^2 E}{\partial w_j \partial w_k}, \quad j, k = 1, \dots, d$$

Note: if we considered a linear approximation $E(\mathbf{w} + \mathbf{v}) \approx E(\mathbf{w}) + \nabla E(\mathbf{w}) \cdot \mathbf{v}$ then the optimal direction would be given by the gradient of E

13

Newton's method (IV)

In summary, Newton's method works as follows

- Choose a starting value $\mathbf{w}^{(0)}$ for $\mathbf{w}$
- Compute a sequence $\{\mathbf{w}^{(t)}\}_{t \geq 0}$ using the update rule

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta^{(t)} \mathbf{H}^{-1}(\mathbf{w}^{(t)}) \nabla E(\mathbf{w}^{(t)})$$

where $\eta^{(t)}$ is again chosen via line search

- A standard stopping criterion is

$$\left\| \nabla E(\mathbf{w}^{(t)}) \cdot (\mathbf{H}(\mathbf{w}^{(t)}) \nabla E(\mathbf{w}^{(t)})) \right\| < \epsilon$$

- This modified version of Newton's method (with line search) is sometimes called damped Newton's method
- Newton's method has typically faster convergence than gradient descent

14

Empirical error minimization via gradient descent

$$\Delta(\mathbf{w}) = -\nabla E(\mathbf{w}) = -\sum_{i=1}^m V'(y_i, \mathbf{w} \cdot \mathbf{x}_i) \mathbf{x}_i$$

Here, V' denotes (abusing some notation) the partial derivative of V wrt. its second argument. Our update rule is

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta^{(t)} \sum_{i=1}^m V'(y_i, \mathbf{w} \cdot \mathbf{x}_i) \mathbf{x}_i$$

This is an example of a **batch** learning algorithm (all training examples are used to compute the gradient).

Note: The step size η is called the **learning rate**

15

Online gradient descent

Rather than computing the gradient of E every time, the **online gradient** approach selects one example at a time (below, $i(t)$ is the example selected at time t)

$$\Delta(\mathbf{w}) = -V'(y_{i(t)}, \mathbf{w} \cdot \mathbf{x}_{i(t)}) \mathbf{x}_{i(t)}$$

Two standard selection rules:

- $i(t) = t$ modulo m
- $i(t)$ sampled uniformly in $\{1, \dots, m\}$

Note: a more compromising approach is select few examples at random at a time. We refer to this as the *stochastic gradient method*

16

Least squares

$$V(y, \mathbf{w} \cdot \mathbf{x}) = (y - \mathbf{w} \cdot \mathbf{x})^2 \Rightarrow V'(y, \mathbf{w} \cdot \mathbf{x}) = -2(y - \mathbf{w} \cdot \mathbf{x})$$

- Gradient descent update rule:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta^{(t)} \sum_{i=1}^m (y_i - \mathbf{w}^{(t)} \cdot \mathbf{x}_i) \mathbf{x}_i$$

- Online update rule

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta^{(t)} (y_{i(t)} - \mathbf{w}^{(t)} \cdot \mathbf{x}_{i(t)}) \mathbf{x}_{i(t)}$$

When $\eta^{(t)} = O(t^{-1})$ the algorithm converges

Note: The online algorithm is known in pattern recognition and machine learning as the **Widrow-Hoff online algorithm**

17

Logistic regression

Recalling the formula for $V(y, z)$ on page 3, we have

$$V(y, z) = y \log(1 + e^{-z}) + (1 - y) \log(1 + e^z)$$

A direct computation gives (use the trick: $p'(z) = p(z)[1 - p(z)]$)

$$V'(y, z) = (1 + e^{-z})^{-1} - y = p(z) - y$$

- Gradient descent update rule:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta^{(t)} \sum_{i=1}^m (y_i - p(\mathbf{w}^{(t)} \cdot \mathbf{x}_i)) \mathbf{x}_i$$

- Online update rule:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta^{(t)} (y_{i(t)} - p(\mathbf{w}^{(t)} \cdot \mathbf{x}_{i(t)})) \mathbf{x}_{i(t)}$$

18

Logistic regression (cont.)

If m is not too large Newton's method is preferred to gradient descent. A direct computation gives

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta^{(t)}(\mathbf{X}^\top \mathbf{W}^{(t)} \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{W}^{(t)} (\mathbf{y} - \mathbf{p}^{(t)})$$

where $\mathbf{W}^{(t)}$ is a diagonal matrix with diagonal elements

$$W_{ii}^{(t)} = p(\mathbf{w}^{(t)} \cdot \mathbf{x}_i)(1 - p(\mathbf{w}^{(t)} \cdot \mathbf{x}_i))$$

and $(\mathbf{p}^{(t)})_i = p(\mathbf{w}^{(t)} \cdot \mathbf{x}_i)$.

This algorithm is referred to as iteratively reweighted least squares:

$$\mathbf{w}^{(t+1)} = (\mathbf{X}^\top \mathbf{W}^{(t)} \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{W}^{(t)} \mathbf{z}^{(t)}$$

where $\mathbf{z}^{(t)} = \mathbf{X} \mathbf{w}^{(t)} + \eta^{(t)} \mathbf{W}^{-1} (\mathbf{y} - \mathbf{p}^{(t)})$ is the adjusted output at time t

19

Logistic regression (cont.)

Let's go back to the online version of logistic regression

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta^{(t)} (y_{i(t)} - p(\mathbf{w}^{(t)} \cdot \mathbf{x}_{i(t)})) \mathbf{x}_{i(t)}$$

Suppose we modify $p(z)$ as

$$p(z) = \frac{1}{1 + e^{-\beta z}}$$

where β is a positive parameter. In particular, when $\beta \rightarrow \infty$ then $p(z)$ goes to

$$p(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

20

Online perceptron

If we recode the output as $\tilde{y} := 2(y - \frac{1}{2})$ (positive and negative classes) and fix the learning rate to one, then the online gradient descent gives the famous perceptron algorithm

$$\mathbf{w}^{(t+1)} = \begin{cases} \mathbf{w}^{(t)} + \tilde{y}^{(t)} \mathbf{x}^{(t)} & \text{if } \tilde{y}^{(t)} \mathbf{w} \cdot \mathbf{x}^{(t)} < 0 \\ \mathbf{w}^{(t)} & \text{otherwise} \end{cases}$$

Equivalently, this is the online algorithm for the loss function

$$V(\tilde{y}_i, \mathbf{w} \cdot \mathbf{x}) = \begin{cases} -\tilde{y} \mathbf{w} \cdot \mathbf{x} & \text{if } \tilde{y} \mathbf{w} \cdot \mathbf{x} < 0 \\ 0 & \text{otherwise} \end{cases}$$

21

Convergence of the perceptron

Let $1 \leq t_1 < t_2 < \dots < t_N$ denote the times when a mistake has been made

Suppose that the training set is linearly separable with a margin at least ρ , that is there is a unit vector $\mathbf{u} \in \mathbb{R}^d$ such that

$$\tilde{y}_i \mathbf{u} \cdot \mathbf{x}_i \geq \rho, \quad i = 1, \dots, m$$

Then we have the following important result

$$N \leq \frac{\max_i \|\mathbf{x}_i\|^2}{\rho^2}$$

- we'll see a proof of this later in the course

22

Richer hypothesis spaces

- So far we have considered linear functions for simplicity. Nothing prevents us to minimize the empirical error in a richer space of *nonlinear* functions f parameterized by a vector $\mathbf{w}$
- These function spaces provide richer models which can fit the training data better than linear functions (typically, the number of parameters may be much larger than the dimension of $\mathbf{x}$). However, to guarantee generalization (avoid overfitting) we will need to modify the empirical error to penalize the selection of “complicated functions”

23

Nonlinear features

A natural idea is to choose f to be linear in some nonlinear features of $\mathbf{x}$

$$f(\mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x})$$

We have already encountered an example where $x \in \mathbb{R}$ and

$$\phi(x) = (1, x, x^2, \dots, x^r)$$

As r increases the minimum of the empirical error decreases but overfitting may occur

We'll see that overfitting can be controlled by minimizing the penalized error $E_{\text{emp}}(\mathbf{w}) + \lambda \|\mathbf{w}\|^2$ for an appropriate choice of the positive parameter λ

24

Kernel expansions

$$f(\mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x})$$

We will see that this approximation is related to the apparently different functions

$$f(\mathbf{x}) = \sum_{i=1}^m c_i K(\mathbf{x}, \mathbf{x}_i)$$

where K is called the kernel function

Optimization algorithms can be derived similarly to the case of linear functions

An important example is the Gaussian kernel, leading to **radial basis functions**

$$f(\mathbf{x}) = \sum_{i=1}^m c_i \exp(-\beta \|\mathbf{x} - \mathbf{x}_i\|^2)$$

25

Neural networks

The above type of approximation gives nonlinear functions of x . These are still linear in the parameters $\mathbf{w}$ or the vector $c = (c_1, \dots, c_m)$ in the kernel approximation

A neural network (NNet) is an example of nonlinear approximation. For example, a one hidden layer neural network

$$f(\mathbf{x}) = \sum_{\ell=1}^L u_{\ell} h(\mathbf{w}_{\ell} \cdot \mathbf{x})$$

depends nonlinearly on the $L(d+1)$ parameters $\mathbf{w} = (u_1, \dots, u_L, \mathbf{w}_1, \dots, \mathbf{w}_L)$

A linear function (perceptron) is a NNet with no hidden layers. On the other hand, the model can be made richer by adding more hidden layers (multi-layer perceptron)

26

Neural networks (cont.)

The function h is called activation function and could be for example $h(z) = (1 + e^{-z})^{-1}$, used in logistic regression

$$E(\mathbf{w}) = \sum_{i=1}^m V\left(y_i, \sum_{\ell=1}^L u_{\ell} h(\mathbf{w}_{\ell} \cdot \mathbf{x}_i)\right)$$

The parameters $\mathbf{w}$ are usually computed via (online) gradient descent, called **back-propagation**

NNets have played an important role in the development of learning algorithms in the 60's and later in the mid 80's