Machine Learning

Linear Models

Fabio Vandin October 20th, 2023

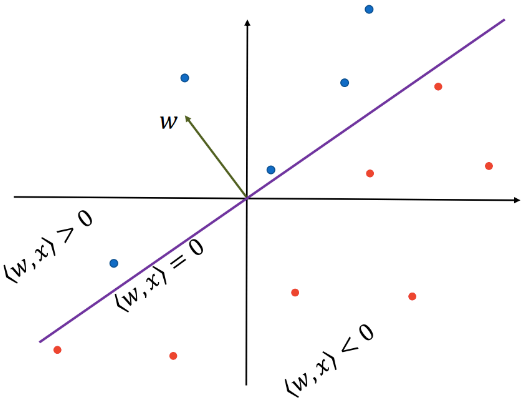
Linear Classification

$\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{-1, 1\}$, 0-1 loss

Hypothesis class = *halfspaces*

$$HS_d = \text{sign} \circ L_d = \{\mathbf{x} \to \text{sign}(h_{\mathbf{w},b}(\mathbf{x})) : h_{\mathbf{w},b} \in L_d\}$$

Example: $\mathcal{X} = \mathbb{R}^2$



w
$\langle w, x \rangle > 0$
$\langle w, x \rangle = 0$
$\langle w, x \rangle < 0$

Finding a Good Hypothesis

Linear classification with hypothesis set $\mathcal{H}$ = halfspaces.

How do we find a good hypothesis? Good = small generalization error

Finding a Good Hypothesis

Linear classification with hypothesis set $\mathcal{H}$ = halfspaces.

How do we find a good hypothesis?

Good = minimizes the training error (ERM)

$|\mathcal{H}| = +\infty$

$\Rightarrow$ "brute force" approach does not work

Finding a Good Hypothesis

Linear classification with hypothesis set $\mathcal{H}$ = halfspaces.

How do we find a good hypothesis?

Good = minimizes the training error (ERM)

$\Rightarrow$ Perceptron Algorithm (Rosenblatt, 1958)

Training set: $S = \{ (\vec{x}_i, y_i) : 1 \leq i \leq m \}$

Note:

if $\underbrace{y_i \langle \mathbf{w}, \mathbf{x}_i \rangle > 0}$ for all $i = 1, \ldots, m$

hypothesis

$h_{\vec{w}}$

$h_{\vec{w}}(\vec{x}_i) = y_i$ (prediction made by $h_{\vec{w}}$ on $\vec{x}$ is "correct", i.e., equal to y_i)

Finding a Good Hypothesis

Linear classification with hypothesis set $\mathcal{H}$ = halfspaces.

How do we find a good hypothesis?

Good = minimizes the training error (ERM)

$\Rightarrow$ Perceptron Algorithm (Rosenblatt, 1958)

Note:
if $y_i \langle \mathbf{w}, \mathbf{x}_i \rangle > 0$ for all $i = 1, \dots, m \Rightarrow$ all points are classified correctly by model $\mathbf{w} \Rightarrow$ *realizability assumption* for training set

Linearly separable data: there exists $\mathbf{w}$ such that: $y_i \langle \mathbf{w}, \mathbf{x}_i \rangle > 0$

$\forall \; i = 1, \dots, m$

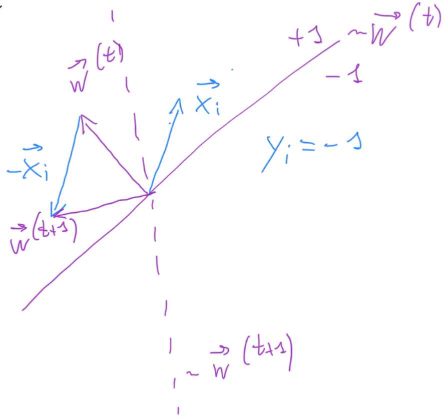
Perceptron

```
Input: training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
initialize $\mathbf{w}^{(1)} = (0, \ldots, 0)$;
for $t = 1, 2, \ldots$ do
    if $\exists i$ s.t. $y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle \leq 0$ then $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y_i \mathbf{x}_i$;
    else return $\mathbf{w}^{(t)}$;
```

$\vec{x}_i$ is missclassified by $\vec{w}(t)$

$\rightarrow$ ($\vec{w}(t)$ correctly classifies all points in the training set)

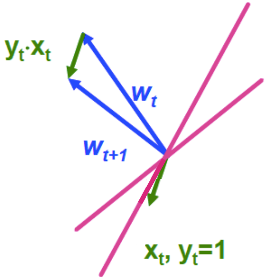
$\mathcal{X} = \mathbb{R}^2$



Perceptron

```
Input: training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
initialize $\mathbf{w}^{(1)} = (0, \ldots, 0)$;
for $t = 1, 2, \ldots$ do
    if $\exists i$ s.t. $y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle \leq 0$ then $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y_i \mathbf{x}_i$;
    else return $\mathbf{w}^{(t)}$;
```

Interpretation of update:



Note that:

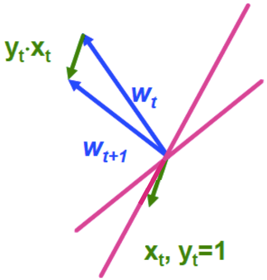
$$y_i \langle \mathbf{w}^{(t+1)}, \mathbf{x}_i \rangle = y_i \langle \mathbf{w}^{(t)} + y_i \mathbf{x}_i, \mathbf{x}_i \rangle$$
$$= \underbrace{y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle}_{<0} + \underbrace{||\mathbf{x}_i||^2}_{>0}$$

$\Rightarrow$ update guides $\mathbf{w}$ to be "more correct" on $(\mathbf{x}_i, y_i)$.

Perceptron

```
Input: training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
initialize $\mathbf{w}^{(1)} = (0, \ldots, 0)$;
for $t = 1, 2, \ldots$ do
    if $\exists i$ s.t. $y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle \leq 0$ then $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y_i \mathbf{x}_i$;
    else return $\mathbf{w}^{(t)}$;
```

Interpretation of update:



Note that:

$$\begin{aligned} y_i \langle \mathbf{w}^{(t+1)}, \mathbf{x}_i \rangle &= y_i \langle \mathbf{w}^{(t)} + y_i \mathbf{x}_i, \mathbf{x}_i \rangle \\ &= y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle + ||\mathbf{x}_i||^2 \end{aligned}$$

$\Rightarrow$ update guides $\mathbf{w}$ to be "more correct" on $(\mathbf{x}_i, y_i)$.

Termination? Depends on the realizability assumption!

Perceptron with Linearly Separable Data

If data is linearly separable one can prove that the perceptron terminates.

Proposition

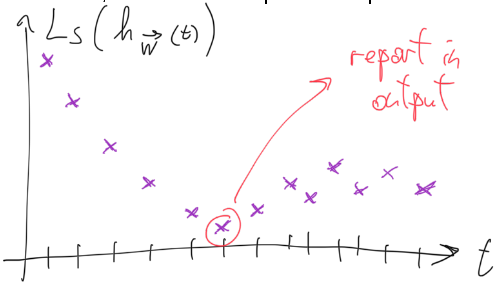
Assume that $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ is linearly separable, let:

- $B = \min\{||\mathbf{w}|| : y_i\langle \mathbf{w}, \mathbf{x}_i\rangle \geq 1 \;\; \forall i, i = 1, \ldots, m, \}$, and
- $R = \max_i ||\mathbf{x}_i||$.

Then the Perceptron algorithm stops after at most $(RB)^2$ iterations (and when it stops it holds that $\forall i, i \in \{1, \ldots, m\} : y_i\langle \mathbf{w}^{(t)}, \mathbf{x}_i\rangle > 0$).

Perceptron: Notes

- simple to implement (but some details are not described in the pseudocode...)
- for separable data
  - termination is guaranteed
  - may require a number of iterations that is exponential in d...
    $\Rightarrow$ other approaches (e.g., ILP - Integer Linear Programming) may be better to find ERM solution in such cases
  - potentially multiple solutions, which one is picked depends on starting values
- non separable data?



Perceptron: Notes

- simple to implement (but some details are not described in the pseudocode...)

- for separable data
  - termination is guaranteed
  - may require a number of iterations that is exponential in d...
    $\Rightarrow$ other approaches (e.g., ILP - Integer Linear Programming) may be better to find ERM solution in such cases
  - potentially multiple solutions, which one is picked depends on starting values

- non separable data?
  - run for some time and keep best solution found up to that point (*pocket algorithm*)

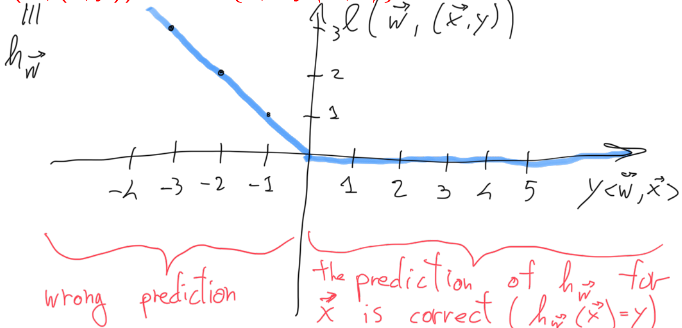
Perceptron: A Modern View

The previous presentation of the Perceptron is the standard one.

However, we can derive the Perceptron in a different way...

Assume you want to solve a:

- binary classification problem: $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{-1, 1\}$
- with linear models
- with loss $\ell(\mathbf{w}, (\mathbf{x}, y)) = \max\{0, -y\langle \mathbf{w}, \mathbf{x}\rangle\}$.



Perceptron: A Modern View

The previous presentation of the Perceptron is the standard one.

However, we can derive the Perceptron in a different way...

Assume you want to solve a:

- binary classification problem: $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{-1, 1\}$
- with linear models
- with loss $\ell(\mathbf{w}, (\mathbf{x}, y)) = \max\{0, -y\langle \mathbf{w}, \mathbf{x}\rangle\}$.

Approach: ERM $\Rightarrow$ need to find the model/hypothesis with smallest training error

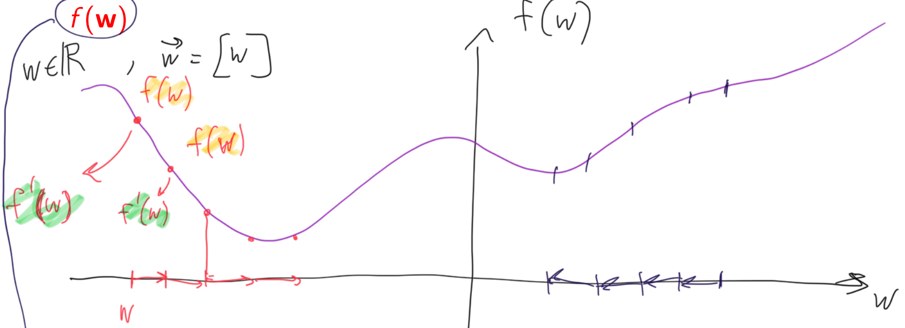
How? SGD

Note: this is a common framework in all of machine learning!

Gradient Descent (GD)

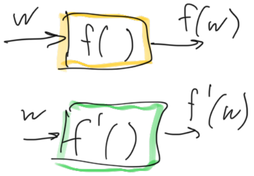
General approach for *minimizing* a differentiable convex function $f(\mathbf{w})$

$w \in \mathbb{R}$, $\vec{w} = [w]$



training error: $S = \{(\vec{x}_1, y_1), \dots, (\vec{x}_m, \vec{y}_m)\}$

$$L_S(h_{\vec{w}}) = \frac{1}{m} \sum_{i=1}^{m} \ell(\vec{w}, (\vec{x}_i, y_i)) = f(\vec{w})$$



Gradient Descent (GD)

General approach for *minimizing* a differentiable convex function $f(\mathbf{w})$

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function

Definition

The *gradient* $\nabla f(\mathbf{w})$ of f at $\mathbf{w} = (w_1, \dots, w_d)$ is

$$\nabla f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \dots, \frac{\partial f(\mathbf{w})}{\partial w_d} \right)$$

Intuition: the gradient points in the direction of the greatest rate of increase of f around $\mathbf{w}$

Let $\eta \in \mathbb{R}, \eta > 0$ be a parameter.

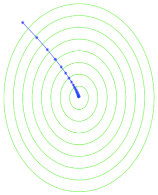
GD algorithm:

```
$\mathbf{w}^{(0)} \leftarrow \mathbf{0}$;
**for** $t \leftarrow 0$ *to* $T - 1$ **do**
    $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla f(\mathbf{w}^{(t)})$;
**return** $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$;
```

Let $\eta \in \mathbb{R}, \eta > 0$ be a parameter.

GD algorithm:

```
$\mathbf{w}^{(0)} \leftarrow \mathbf{0}$;
for $t \leftarrow 0$ to $T - 1$ do
    $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla f(\mathbf{w}^{(t)})$;
return $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$;
```



Notes:

- output vector could also be $\mathbf{w}^{(T)}$ or $\arg\min_{\mathbf{w}^{(t)} \in \{1,\dots,T\}} f(\mathbf{w}^{(t)})$
- returning $\bar{\mathbf{w}}$ is useful for nondifferentiable functions (using *subgradients* instead of gradients...) and for stochastic gradient descent...
- η: *learning rate*; sometimes a time dependent $\eta^{(t)}$ is used (e.g., "move" more at the beginning than at the end)

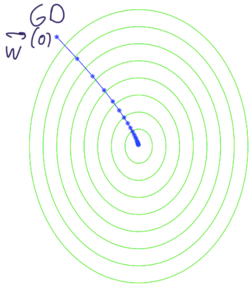
Note: there are guarantees on the number of iterations required by GD to return a *good* value of $\bar{\mathbf{w}}$ under some assumptions on f (see the book for details)

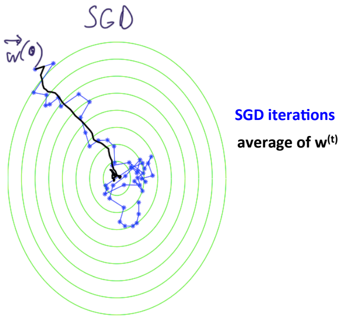
Stochastic Gradient Descent (SGD)

Idea: instead of using exactly the gradient, we take a (random) vector with *expected value* equal to the gradient direction.

SGD algorithm:

```
$\mathbf{w}^{(0)} \leftarrow \mathbf{0}$; // or $\vec{w}^{(0)} =$ random vector
for $t \leftarrow 0$ to $T-1$ do
    choose $\mathbf{v}_t$ at random from distribution such that $\mathbf{E}[\mathbf{v}_t | \mathbf{w}^{(t)}] \in \nabla f(\mathbf{w}^{(t)})$;
    /* $\mathbf{v}_t$ has expected value equal to the gradient of $f(\mathbf{w}^{(t)})$ */
    $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \mathbf{v}_t$;
return $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$;
```





Note: there are guarantees on the number of iterations required by GD to return a *good*, *in expectation*, value of $\bar{\mathbf{w}}$ under some assumptions on f (see the book for details)

Why should we use SGD instead of GD?

Question: when do we use GD in the first place?

Answer: for example to find $\mathbf{w}$ that minimizes $L_S(\mathbf{w})$

That is: we use GD for $f(\mathbf{w}) = L_S(\mathbf{w})$
$\Rightarrow \nabla f(\mathbf{w})$ depends on all pairs $(\mathbf{x}_i, y_i) \in S, i = 1, \ldots, m$: may require long time to compute it!

What about SGD?

We need to pick $\mathbf{v}_t$ such that $\mathbf{E}[\mathbf{v}_t | \mathbf{w}^{(t)}] \in \nabla f(\mathbf{w}^{(t)})$: **how**?

Pick a random $(\mathbf{x}_i, y_i) \in S \Rightarrow$ pick $\mathbf{v}_t \in \nabla \ell(\mathbf{w}^{(t)}, (\mathbf{x}_i, y_i))$:

- satisfies the requirement!
- requires much less computation than GD

Analogously we can use SGD for regularized losses, etc.

Back to Our Linear Classification Problem

- binary classification problem: $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{-1, 1\}$
- with linear models
- with loss $\ell(\mathbf{w}, (\mathbf{x}, y)) = \max\{0, -y\langle \mathbf{w}, \mathbf{x}\rangle\}$.

How to find the ERM solution? SGD!

SGD for Linear Classification

SGD: take i uniformly at random from $\{1, \dots, m\}$. Let $(\vec{x}', y')$ be the corresponding point in the training set, and consider the vector $\nabla \ell(\vec{w}, (\vec{x}', y'))$

Note that GD considers (as gradient of the function to minimize):

$$\nabla L_S(\vec{w}) = \frac{1}{m} \sum_{i=1}^{m} \nabla \ell(\vec{w}, (\vec{x}_i, y_i))$$

and for SGD we have:

$$\mathbb{E}[\nabla \ell(\vec{w}, (\vec{x}', y'))] = \sum_{i=1}^{m} \overbrace{\Pr[(\vec{x}', y') = (\vec{x}_i, y_i)]}^{\frac{1}{m} \ \forall i \in \{1, \dots, m\} \ \text{(uniform distribution)}} \cdot \nabla \ell(\vec{w}, (\vec{x}_i, y_i))$$

$$= \frac{1}{m} \sum_{i=1}^{m} \nabla \ell(\vec{w}, (\vec{x}_i, y_i))$$

$$= \nabla L_S(\vec{w})$$