Machine Learning

Linear Models

Fabio Vandin

October 21^{st}, 2022

Linear Predictors and Affine Functions

Consider $\mathcal{X} = \mathbb{R}^d$

"Linear" (affine) functions:

$$L_d = \{h_{\mathbf{w},b} : \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$$

parameters

where

$$h_{\mathbf{w},b}(\mathbf{x}) = \underbrace{\langle \mathbf{w}, \mathbf{x} \rangle}_{=} + b = \underbrace{\left(\sum_{i=1}^{d} w_i x_i \right)}_{} + b$$

Note:

- each member of L_d is a function $\mathbf{x} \to \langle \mathbf{w}, \mathbf{x} \rangle + b$
- b: *bias*

Linear Models

Hypothesis class $\mathcal{H}$: $\phi \circ L_d$, where $\phi : \mathbb{R} \to \mathcal{Y}$

- $h \in \mathcal{H}$ is $h : \mathbb{R}^d \to \mathcal{Y}$

ϕ depends on the learning problem

Example

- binary classification, $\mathcal{Y} = \{-1, 1\} \Rightarrow \phi(z) = \mathrm{sign}(z)$
- regression, $\mathcal{Y} = \mathbb{R} \Rightarrow \phi(z) = z$

Equivalent Notation

$\vec{w} = [w_1, w_2, \dots, w_d]$

Given $\mathbf{x} \in \mathcal{X}$, $\mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$, define:

- $\mathbf{w}' = (b, w_1, w_2, \dots, w_d) \in \mathbb{R}^{d+1}$
- $\mathbf{x}' = (1, x_1, x_2, \dots, x_d) \in \mathbb{R}^{d+1}$

Then:

$$h_{\mathbf{w},b}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b \cdot 1 = \langle \mathbf{w}', \mathbf{x}' \rangle \qquad (1)$$

$\Rightarrow$ we will consider bias term as part of $\mathbf{w}$ and assume $\mathbf{x} = (1, x_1, x_2, \dots, x_d)$ *when needed*, with $h_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$

Linear Classification

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

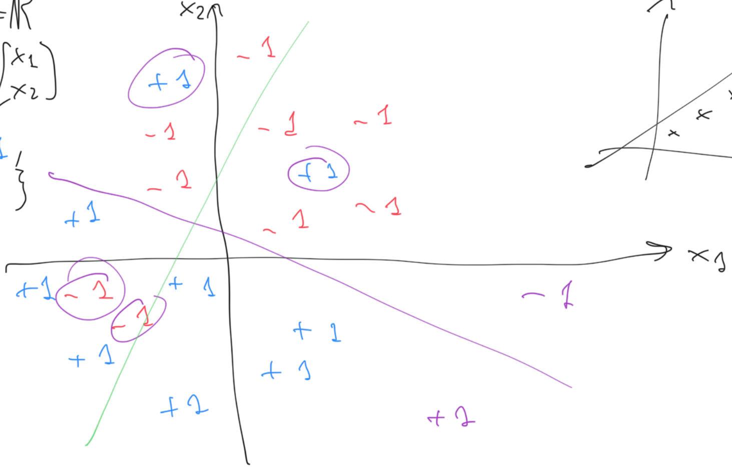
Hypothesis class = *halfspaces*

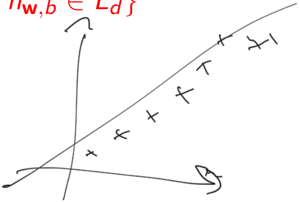
$$HS_d = \text{sign} \circ L_d = \{\mathbf{x} \to \text{sign}(h_{\mathbf{w},b}(\mathbf{x})) : h_{\mathbf{w},b} \in L_d\}$$

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

$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

$\mathcal{Y} = \{+1, -1\}$





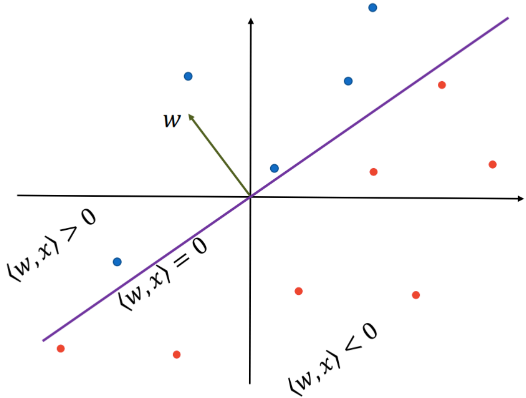
Linear Classification

$\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{-1, 1\}$, 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: $\mathcal{X} = \mathbb{R}^2$



Finding a Good Hypothesis

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

How do we find a good hypothesis? $|\mathcal{H}| = +\infty$

Good = minimizes the training error (ERM)

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 $\underbrace{y_i \langle \mathbf{w}, \mathbf{x}_i \rangle > 0}_{h_{\vec{w}}(\vec{x}_i) = y_i}$ for all $i = 1, \ldots, m$

(prediction made by $h_{\vec{w}}$ on $\vec{x}_i$ is correct)

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 $i = 1, \ldots, m \Rightarrow$ all points are classified correctly by model $\mathbf{w} \Rightarrow$ *realizability assumption* for training set

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

Perceptron

Input: training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$

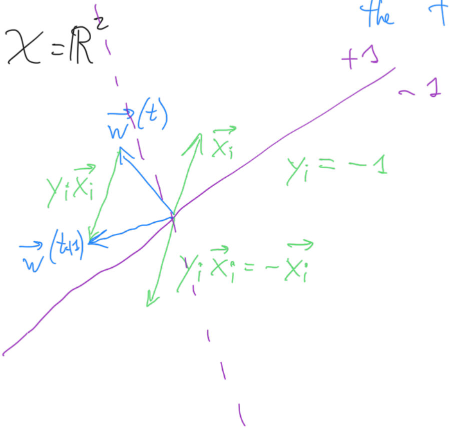
$\vec{x}_i$ is missclassified by $\vec{w}(t)$

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** $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y_i \mathbf{x}_i$;

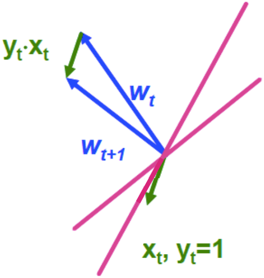
  else return $\mathbf{w}^{(t)}$; ($\vec{w}(t)$ correctly classifies all points in the training set)



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:

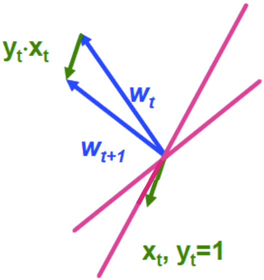
$$y_i \langle \mathbf{w}^{(t+1)}, \mathbf{x}_i \rangle = y_i \langle \mathbf{w}^{(t)} + y_i \mathbf{x}_i, \mathbf{x}_i \rangle$$
$$= \underbrace{y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle}_{<0} + \underbrace{\|\mathbf{x}_i\|^2}_{>0}$$

$\Rightarrow$ 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

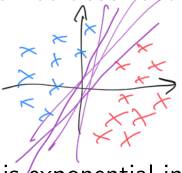
Assume that $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ is linearly separable, let:

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

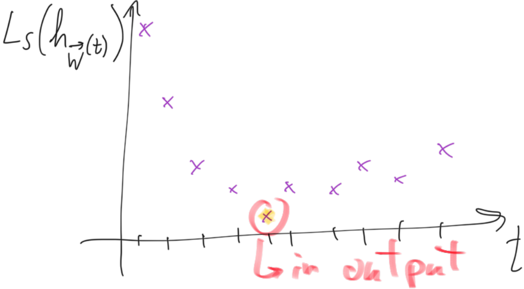
- simple to implement (but some details are not described in the pseudocode...)



- 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

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*)

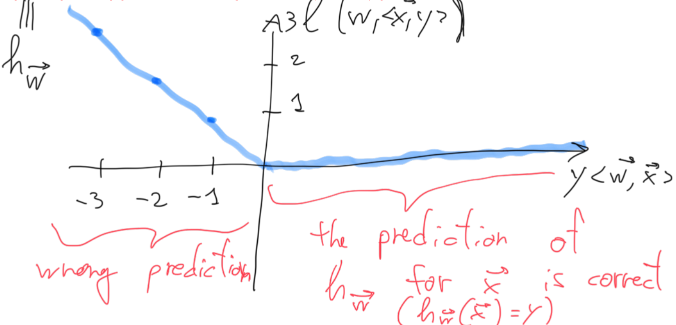
Perceptron: A Modern View

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)

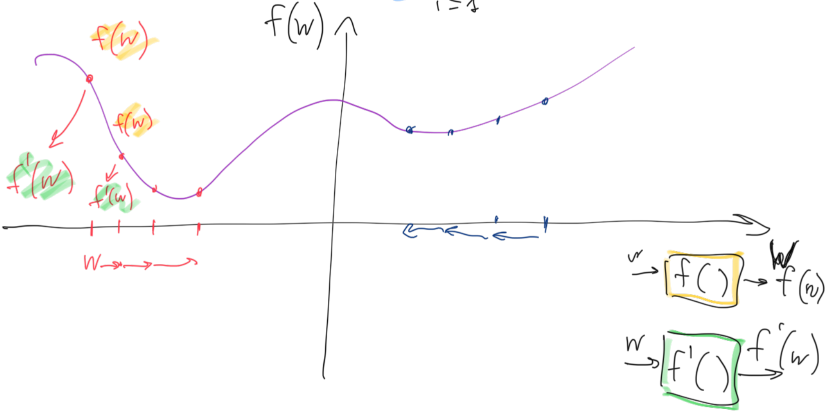
General approach for *minimizing* a differentiable convex function

$f(\mathbf{w}) \longrightarrow$ training error

$$S = \{(\vec{x}_1, y_1), \dots, (\vec{x}_m, y_m)\} \quad L_S(h_{\vec{w}}) = \frac{1}{m}\sum_{i=1}^{m} \ell\left(\vec{w}, (\vec{x}_i, y_i)\right) = f(\vec{w})$$

$\vec{w} \in \mathbb{R}$

$w \in \mathbb{R}$



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 $\mathbf{w} = (w_1, \ldots, w_d)$ is

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

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