Machine Learning

Linear Models

Fabio Vandin October 16th, 2023

Linear Predictors and Affine Functions

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

“Linear” (affine) functions:

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

where

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

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

Equivalent Notation

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

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

Then:

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

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

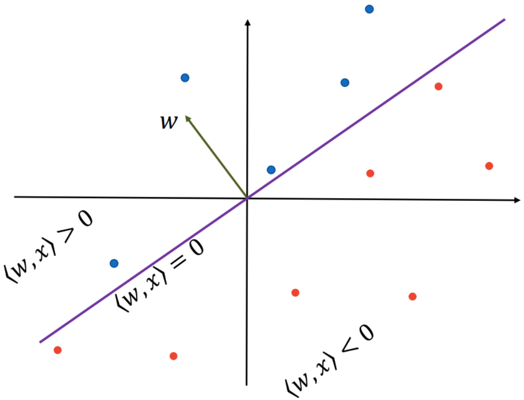
Linear Classification

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

Hypothesis class = *halfspaces*

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

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 = minimizes the training error (ERM)

$\Rightarrow$ Perceptron Algorithm (Rosenblatt, 1958)

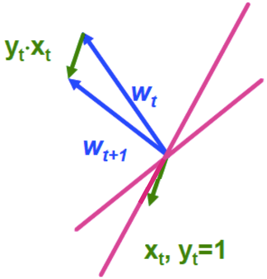
Note:
if $y_i \langle \mathbf{w}, \mathbf{x}_i \rangle > 0$ for all $i = 1, \ldots, m \Rightarrow$ all points are classified correctly by model $\mathbf{w} \Rightarrow$ *realizability assumption* for training set

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

Perceptron

```
Input: training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
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?
    - 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 $\ell(\mathbf{w}, (\mathbf{x}, y)) = \max\{0, -y\langle \mathbf{w}, \mathbf{x}\rangle\}$.

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

How?

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

Gradient Descent (GD)

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

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

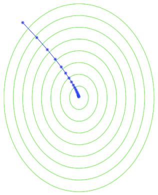
$$\nabla f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d}\right)$$

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

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)

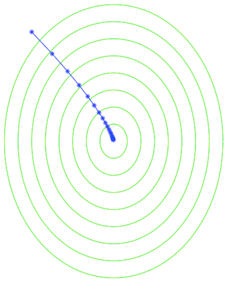
Note: there are guarantees on the number of iterations required by GD to return a *good* value of $\bar{\mathbf{w}}$ under some assumptions on f (see the book for details)

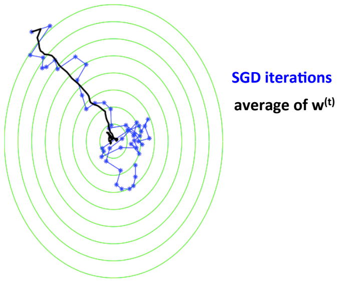
Stochastic Gradient Descent (SGD)

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

SGD algorithm:

```
$\mathbf{w}^{(0)} \leftarrow \mathbf{0}$;
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

Linear Regression

$\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \mathbb{R}$

Hypothesis class:

$$\mathcal{H}_{reg} = L_d = \{\mathbf{x} \to \langle \mathbf{w}, \mathbf{x} \rangle + b : \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$$

Note: $h \in \mathcal{H}_{reg} : \mathbb{R}^d \to \mathbb{R}$

Commonly used loss function: *squared-loss*

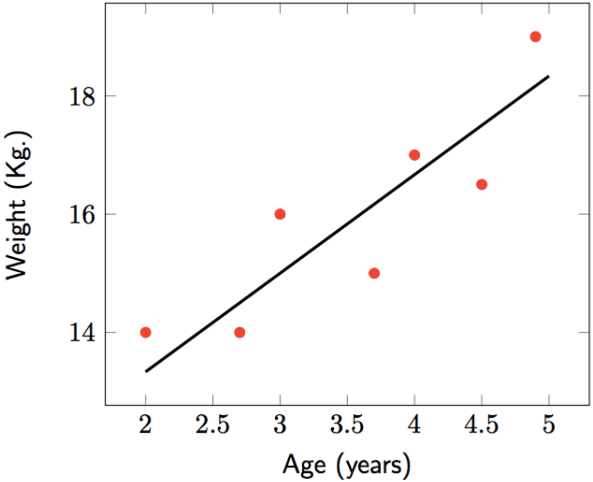
$$\ell(h, (\mathbf{x}, y)) \overset{def}{=} (h(\mathbf{x}) - y)^2$$

$\Rightarrow$ empirical risk function (training error): *Mean Squared Error*

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} (h(\mathbf{x}_i) - y_i)^2$$

Linear Regression - Example

$d = 1$



Least Squares

How to find a ERM hypothesis? *Least Squares* algorithm

Best hypothesis:

$$\arg\min_{\mathbf{w}} L_S(h_{\mathbf{w}}) = \arg\min_{\mathbf{w}} \frac{1}{m}\sum_{i=1}^{m}(\langle \mathbf{w}, \mathbf{x}_i\rangle - y_i)^2$$

Equivalent formulation: $\mathbf{w}$ minimizing *Residual Sum of Squares* (RSS), i.e.

$$\arg\min_{\mathbf{w}} \sum_{i=1}^{m}(\langle \mathbf{w}, \mathbf{x}_i\rangle - y_i)^2$$

RSS: Matrix Form

Let

$$\mathbf{X} = \begin{bmatrix} \cdots & \mathbf{x}_1 & \cdots \\ \cdots & \mathbf{x}_2 & \cdots \\ \cdots & \vdots & \cdots \\ \cdots & \mathbf{x}_m & \cdots \end{bmatrix}$$

$\mathbf{X}$: *design matrix*

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$\Rightarrow$ we have that RSS is

$$\sum_{i=1}^{m} (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

Want to find $\mathbf{w}$ that minimizes RSS (=*objective function*):

$$\arg\min_{\mathbf{w}} RSS(\mathbf{w}) = \arg\min_{\mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

How?

Compute gradient $\frac{\partial RSS(\mathbf{w})}{\partial \mathbf{w}}$ of objective function w.r.t $\mathbf{w}$ and compare it to 0.

$$\frac{\partial RSS(\mathbf{w})}{\partial \mathbf{w}} = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

Then we need to find $\mathbf{w}$ such that

$$-2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) = 0$$

$$-2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) = 0$$

is equivalent to

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y}$$

If $\mathbf{X}^T\mathbf{X}$ is invertible $\Rightarrow$ solution to ERM problem is:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Complexity Considerations

We need to compute

$$(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Algorithm:

1. compute $\mathbf{X}^T\mathbf{X}$: product of $(d+1) \times m$ matrix and $m \times (d+1)$ matrix
2. compute $(\mathbf{X}^T\mathbf{X})^{-1}$ inversion of $(d+1) \times (d+1)$ matrix
3. compute $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$: product of $(d+1) \times (d+1)$ matrix and $(d+1) \times m$ matrix
4. compute $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$: product of $(d+1) \times m$ matrix and $m \times 1$ matrix

Most expensive operation? Inversion!

$\Rightarrow$ done for $(d+1) \times (d+1)$ matrix

$\mathbf{X}^T\mathbf{X}$ not invertible?

How do we get $\mathbf{w}$ such that

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y}$$

if $\mathbf{X}^T\mathbf{X}$ is not invertible?
Let

$$\mathbf{A} = \mathbf{X}^T\mathbf{X}$$

Let $\mathbf{A}^+$ be the *generalized inverse* of $\mathbf{A}$, i.e.:

$$\mathbf{A}\mathbf{A}^+\mathbf{A} = \mathbf{A}$$

Proposition

If $\mathbf{A} = \mathbf{X}^T\mathbf{X}$ is not invertible, then $\hat{w} = \mathbf{A}^+\mathbf{X}^T\mathbf{y}$ is a solution to $\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y}$.

Computing the Generalized Inverse of $\mathbf{A}$

Note $\mathbf{A} = \mathbf{X}^T\mathbf{X}$ is symmetric $\Rightarrow$ eigenvalue decomposition of $\mathbf{A}$:

$$\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^T$$

with

- $\mathbf{D}$: diagonal matrix (entries = eigenvalues of $\mathbf{A}$)
- $\mathbf{V}$: orthonormal matrix ($\mathbf{V}^T\mathbf{V} = \mathbf{I}_{d\times d}$)

Define $\mathbf{D}^+$ diagonal matrix such that:

$$\mathbf{D}^+_{i,i} = \begin{cases} 0 & \text{if } \mathbf{D}_{i,i} = 0 \\ \frac{1}{\mathbf{D}_{i,i}} & \text{otherwise} \end{cases}$$

Let $\mathbf{A}^{+} = \mathbf{V}\mathbf{D}^{+}\mathbf{V}^{T}$

Then

$$\begin{aligned} \mathbf{A}\mathbf{A}^{+}\mathbf{A} &= \mathbf{V}\mathbf{D}\mathbf{V}^{T}\mathbf{V}\mathbf{D}^{+}\mathbf{V}^{T}\mathbf{V}\mathbf{D}\mathbf{V}^{T} \\ &= \mathbf{V}\mathbf{D}\mathbf{D}^{+}\mathbf{D}\mathbf{V}^{T} \\ &= \mathbf{V}\mathbf{D}\mathbf{V}^{T} \\ &= \mathbf{A} \end{aligned}$$

$\Rightarrow$ $\mathbf{A}^{+}$ is a generalized inverse of $\mathbf{A}$.

In practice: the Moore-Penrose generalized inverse $\mathbf{A}^{\dagger}$ of $\mathbf{A}$ is used, since it can be efficiently computed from the Singular Value Decomposition of $\mathbf{A}$.

Exercise

Consider a linear regression problem, where $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}$, with mean squared loss. The hypothesis set is the set of *constant* functions, that is $\mathcal{H} = \{h_a : a \in \mathbb{R}\}$, where $h_a(\mathbf{x}) = a$. Let $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$ denote the training set.

- Derive the hypothesis $h \in \mathcal{H}$ that minimizes the training error.
- Use the result above to explain why, for a given hypothesis $\hat{h}$ from the set of all linear models, the coefficient of determination $R^2 = 1 - \frac{\sum_{i=1}^{m}(\hat{h}(\mathbf{x}_i) - y_i)^2}{\sum_{i=1}^{m}(y_i - \bar{y})^2}$ where $\bar{y}$ is the average of the y_i,$i = 1, \ldots, m$ is a measure of how well $\hat{h}$ performs (on the training set).

Polynomial Models

Consider a regression problem.

Can we as hypothesis set the set of polynomials of degree r with the tools we have already developed for linear regression?

Logistic Regression

it's a regression but we will use it for classification

Learn a function h from $\mathbb{R}^d$ to $[0, 1]$.

What can this be used for?

Classification!

Example: binary classification ($\mathcal{Y} = \{-1, 1\}$) - $h(\mathbf{x}) =$ *probability* that label of $\mathbf{x}$ is 1.

For simplicity of presentation, we consider binary classification with $\mathcal{Y} = \{-1, 1\}$, but similar considerations apply for multiclass classification.

Logistic Regression: Model

Hypothesis class $\mathcal{H}$: $\phi_{\text{sig}} \circ L_d$, where $\phi_{\text{sig}} : \mathbb{R} \to [0, 1]$ is *sigmoid function*

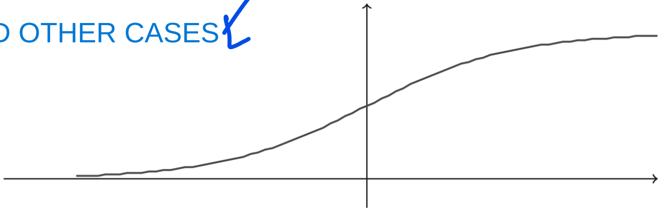
Sigmoid function = "S-shaped" function

For logistic regression, the sigmoid ϕ_{sig} used is the *logistic regression*:

$$\phi_{\text{sig}}(z) = \frac{1}{1 + e^{-z}}$$

h(x)=1 high confidence

ADD OTHER CASES



Therefore

parameters of the model/hypothesis

$$H_{\text{sig}} = \phi_{\text{sig}} \circ L_d = \{\mathbf{x} \to \phi_{\text{sig}}(\langle \mathbf{w}, \mathbf{x} \rangle) : \mathbf{w} \in \mathbb{R}^{d+1}\}$$

and $h_{\mathbf{w}}(\mathbf{x}) \in H_{\text{sig}}$ is:

$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\langle \mathbf{w}, \mathbf{x} \rangle}}$$

Main difference with binary classification with halfspaces: when $\langle \mathbf{w}, \mathbf{x} \rangle \approx 0$

- halfspace prediction is deterministically 1 or -1
- $\phi_{\text{sig}}(\langle \mathbf{w}, \mathbf{x} \rangle) \approx 1/2 \Rightarrow$ uncertainty in predicted label

Loss Function

Need to define how bad it is to predict $h_{\mathbf{w}}(\mathbf{x}) \in [0, 1]$ given that true label is $y = \pm 1$

Desiderata

- $h_{\mathbf{w}}(\mathbf{x})$ "large" if $y = 1$
- $1 - h_{\mathbf{w}}(\mathbf{x})$ "large" if $y = -1$

Note that

$$\begin{aligned} 1 - h_{\mathbf{w}}(\mathbf{x}) &= 1 - \frac{1}{1 + e^{-\langle \mathbf{w}, \mathbf{x} \rangle}} \\ &= \frac{e^{-\langle \mathbf{w}, \mathbf{x} \rangle}}{1 + e^{-\langle \mathbf{w}, \mathbf{x} \rangle}} \quad \text{multiply by 1} \\ &= \frac{1}{1 + e^{\langle \mathbf{w}, \mathbf{x} \rangle}} \end{aligned}$$

Then *reasonable* loss function: increases monotonically with

$$\frac{1}{1 + e^{y\langle \mathbf{w}, \mathbf{x} \rangle}}$$

$\Rightarrow$ *reasonable* loss function: increases monotonically with

$$1 + e^{-y\langle \mathbf{w}, \mathbf{x} \rangle}$$

Loss function for logistic regression:

$$\ell(h_{\mathbf{w}}, (\mathbf{x}, y)) = \log\left(1 + e^{-y\langle \mathbf{w}, \mathbf{x} \rangle}\right)$$

Therefore, given training set $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$ the ERM problem for logistic regression is:

$$\arg\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{m}\sum_{i=1}^{m} \log\left(1 + e^{-y_i\langle \mathbf{w}, \mathbf{x}_i\rangle}\right)$$

Notes: logistic loss function is a *convex function* $\Rightarrow$ ERM problem can be solved efficiently

Definition may look a bit arbitrary: actually, ERM formulation is the same as the one arising from *Maximum Likelihood Estimation*

Maximum Likelihood Estimation (MLE) [UML, 24.1]

MLE is a statistical approach for finding the parameters that maximize the joint probability of a given dataset *assuming a specific parametric probability function*.

Note: MLE essentially assumes a *generative model* for the data

General approach:

1. given training set $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$, assume each $(\mathbf{x}_i, y_i)$ is i.i.d. from some probability distribution of parameters θ
2. consider $\mathbb{P}[S|\theta]$ (likelihood of data given parameters)
3. *log likelihood*: $L(S;\theta) = \log(\mathbb{P}[S|\theta])$
4. *maximum likelihood estimator*: $\hat{\theta} = \arg\max_\theta L(S;\theta)$

Logistic Regression and MLE

Assuming $\mathbf{x}_1, \ldots, \mathbf{x}_m$ are fixed, the probability that $\mathbf{x}_i$ has label $y_i = 1$ is

$$h_{\mathbf{w}}(\mathbf{x}_i) = \frac{1}{1 + e^{-\langle \mathbf{w}, \mathbf{x}_i \rangle}}$$

while the probability that $\mathbf{x}_i$ has label $y_i = -1$ is

$$(1 - h_{\mathbf{w}}(\mathbf{x}_i)) = \frac{1}{1 + e^{\langle \mathbf{w}, \mathbf{x}_i \rangle}}$$

for each i the probability that x_i has label y_i is:

$$\frac{1}{1+e^{-y_i \langle w, x_i \rangle}}$$

Then the likelihood for training set S is:

$$\prod_{i=1}^{m} \left(\frac{1}{1 + e^{-y_i \langle \mathbf{w}, \mathbf{x}_i \rangle}} \right)$$

i.i.d.

Therefore the log likelihood is:

prop. of log $\longrightarrow$

$$-\sum_{i=1}^{m} \log\left(1 + e^{-y_i \langle \mathbf{w}, \mathbf{x}_i \rangle}\right)$$

And note that the maximum likelihood estimator for **w** is:

$$\arg\max_{\mathbf{w}\in\mathbb{R}^d} -\sum_{i=1}^{m} \log\left(1 + e^{-y_i \langle \mathbf{w}, \mathbf{x}_i \rangle}\right) = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{m} \log\left(1 + e^{-y_i \langle \mathbf{w}, \mathbf{x}_i \rangle}\right)$$

$\Rightarrow$ MLE solution is equivalent to ERM solution!
(we justified the choice of the loss)

Bibliography

[UML] Chapter 9:

- no 9.1.1