A.Z
Machine Learning
Nov, 2023

lec 5 Linear classification

Abstract

This part introduced some basic classifiers, including Linear classifier, Perception, Logistic regression(this is a classifier). And gived a optimization method gradient descend.

1 Notations

- Input space: $X \subset \mathbb{R}^d$, each x $\in X$ is a d-dimensional real-valued vector, output space: $\mathcal{Y} = \{-1, +1\}$
- Target function or concept: $f : X \mapsto \mathcal{Y}$ assigns a (true) label to each example
- Training sample: $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, with $y_i = f(x_i)$ drawn from an unknown distribution D
- Hypothesis class: $\mathcal{H} = \left\{\mathbf{x} \mapsto \operatorname{sgn}\left(\sum_{j=1}^d w_j x_j + w_0\right) \mid \mathbf{w} \in \mathbb{R}^d, w_0 \in \mathbb{R}\right\}$ consists of functions $h(\mathbf{x}) = \operatorname{sgn}\left(\sum_{j=1}^d w_j x_j + w_0\right)$ that map each example in one of the two classes
where $\operatorname{sgn}(a) = \begin{cases} +1, & a \geq 0 \\ -1 & a < 0 \end{cases}$
- Linear classifiers:

$$h(\mathbf{x}) = \operatorname{sgn}\left(\sum_{j=1}^d w_j x_j + w_0\right) = \operatorname{sgn}\left(\mathbf{w}^T \mathbf{x} + w_0\right)$$

- The hyperplane $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = 0$ splits the input space into two half-spaces.
$\mathbf{w}$ is the normal vector of the hyperplane $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = 0$ means that $\mathbf{w}$ is perpendicular to any vector lying on the hyperplane.
- The distance of the hyperplane from the origin is $|w_0| / \|\mathbf{w}\|$,
where $\|\mathbf{w}\| = \sqrt{\sum_j w_j^2}$ denotes the Euclidean norm

2 Linear classifiers

We subsume term w_0 into the weight vector for convenient

$$\mathbf{w} \Leftarrow \begin{bmatrix} \mathbf{w} \\ w_0 \end{bmatrix}$$

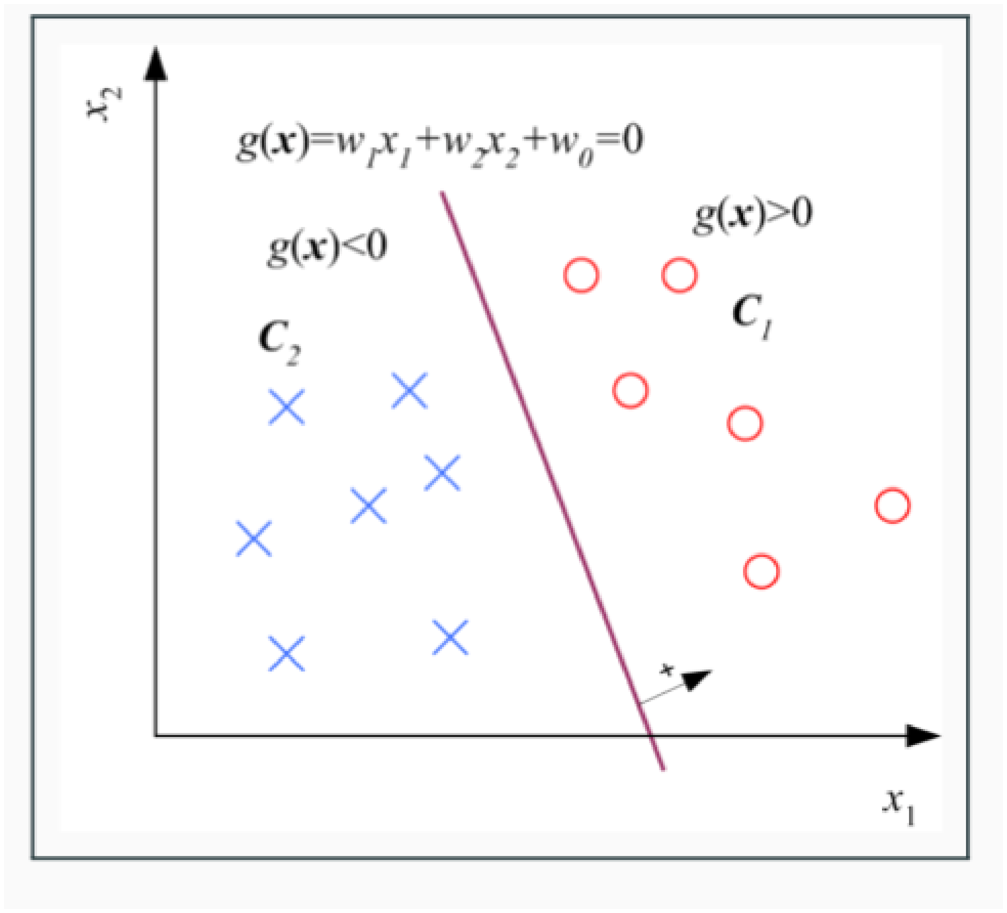


Figure 1: hyperplane split the space

and augment all inputs with a constant 1:

$$x \Leftarrow \begin{bmatrix} x \\ 1 \end{bmatrix}$$

The model give the same value for x:

$$\begin{bmatrix} \mathbf{w} \\ w_0 \end{bmatrix}^T \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \mathbf{w}^T\mathbf{x} + w_0$$

2.1 Checking for prediction errors

When the labels are $\mathcal{Y} = \{-1, +1\}$ for a training example $(\mathbf{x}, y)$ we have for $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

$$yg(\mathbf{x}) = \begin{cases} \geq 0 & \text{if } \mathbf{x} \text{ is correctly classified} \\ < 0 & \text{if } \mathbf{x} \text{ is incorrectly classified} \end{cases}$$

The geometric margin of a labeled example $(\mathbf{x}, \boldsymbol{y})$ is given by $\gamma(\mathbf{x}) = yg(\mathbf{x})/\|\mathbf{w}\|$
It takes into account both the distance $\left|\mathbf{w}^T\mathbf{x}\right|/\|\mathbf{w}\|$ from the hyperplane, and whether $\mathbf{x}$ is on the correct side of the hyperplane

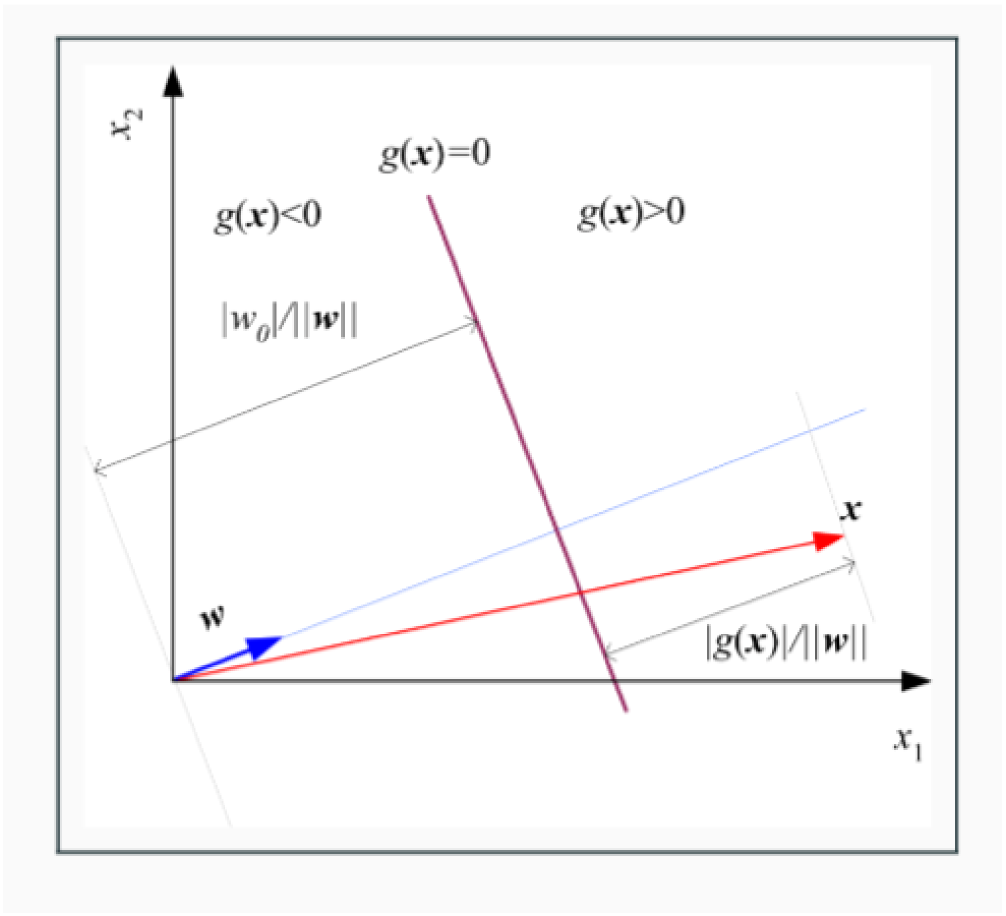
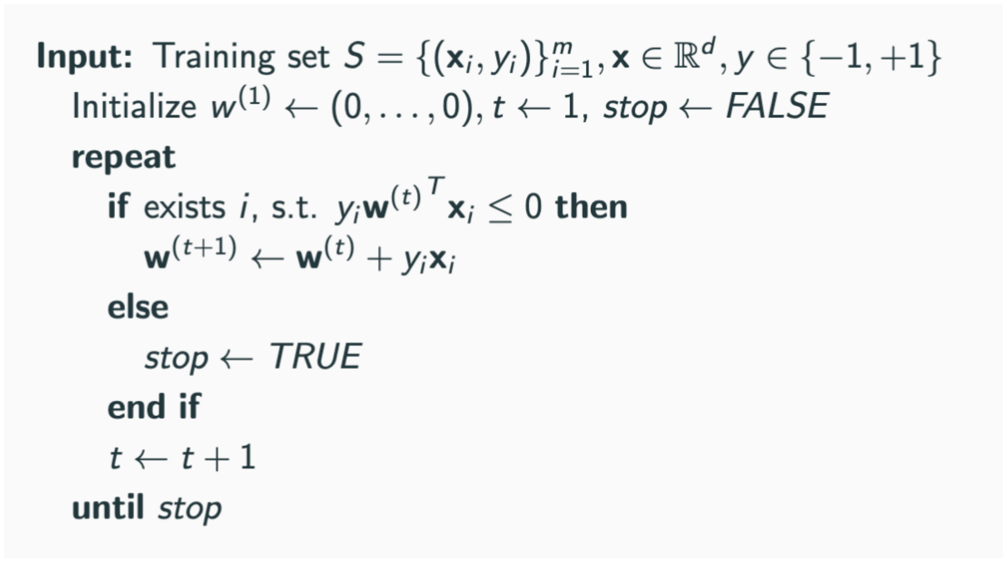


Figure 2: distance in hyperplane

The unnormalized version of the margin is sometimes called the functional margin $\gamma(\mathbf{x}) = yg(\mathbf{x})$
If $y \cdot g(\mathbf{x})$ is positive and large, it indicates that the point $\mathbf{x}$ is correctly classified with high confidence.
If it's negative or close to zero, it indicates a misclassification or a point very close to the decision boundary.
In SVMs, the objective is to find the hyperplane that maximizes the geometric margin. This is equivalent to finding the hyperplane that has the largest minimum distance to the nearest training data point of any class.

3 Perceptron

Our goal is to find the $\mathbf{w}$ vector that can perfectly classify positive inputs and negative inputs in our data. (maximize the geometric margin)
Perceptron algorithm (only for linear-separable training set)
The update rule:



```
Input: Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}, \mathbf{x} \in \mathbb{R}^d, y \in \{-1, +1\}$
  Initialize $w^{(1)} \leftarrow (0, \dots, 0), t \leftarrow 1$, $stop \leftarrow FALSE$
  repeat
    if exists $i$, s.t. $y_i {\mathbf{w}^{(t)}}^T \mathbf{x}_i \leq 0$ then
      $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y_i \mathbf{x}_i$
    else
      $stop \leftarrow TRUE$
    end if
    $t \leftarrow t + 1$
  until $stop$
```

Figure 3: Perceptron algorithm

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

We can see that the margin of the example $(\mathbf{x}_i, y_i)$ increases after the update

$$\begin{aligned}
& y_i g^{(t+1)}(\mathbf{x}_i) \\
&= y_i {\mathbf{w}^{(t+1)}}^T \mathbf{x}_i \\
&= y_i \left(\mathbf{w}^{(t)} + y_i \mathbf{x}_i\right)^T \mathbf{x}_i \\
&= y_i {\mathbf{w}^{(t)}}^T \mathbf{x}_i + y_i^2 \mathbf{x}_i^T \mathbf{x}_i \\
&= y_i g^{(t)}(\mathbf{x}_i) + \|\mathbf{x}_i\|^2 \\
&\geq y_i g^{(t)}(\mathbf{x}_i)
\end{aligned}$$

Note that this does not guarantee that $y_i g^{(t+1)}(\mathbf{x}_i) > 0$
we can proof that the perceptron algorithm will stop after at most $t \leq \left(\frac{2R}{\gamma}\right)^2$ iterations.
γ : The largest achievable geometric margin so that all training examples have at least that margin
R: The smallest radius of the d-dimensional ball that encloses the training data
Intuitively: how large the margin in is relative to the distances of the training points

3.1 Loss function

It can be shown that the Perceptron algorithm is using the following loss:
$L_{\text{Perceptron}}\left(y, \mathbf{w}^T \mathbf{x}\right) = \max\left(0, -y \mathbf{w}^T \mathbf{x}\right)$
$y \mathbf{w}^T \mathbf{x}$ is the margin
if $y \mathbf{w}^T \mathbf{x} < 0$, a loss of $-y \mathbf{w}^T \mathbf{x}$ is incurred
otherwise no loss is incurred

3.2 Convexity

A function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is convex if for all x, y, and $0 \leq \theta \leq 1$, we have

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y).$$

Convexity has an important consequence: every local minimum is also the global minimum
In principle we can minimize it with incremental updates that gradually decrease the loss
Perceptron loss is convexity

4 Logistic regression

Logistic regression is a classification technique

4.1 Logistic function

The logit function is a transformation that takes a probability p (the probability of an event occurring) and converts it into the logarithm of the odds of that event.
The odds of an event is the ratio of the probability of the event occurring to the probability of it not occurring, which is $\frac{p}{1-p}$.
The logit function is then defined as

$$\operatorname{logit}(p) = \log\left(\frac{p}{1-p}\right)$$

The logistic function is the inverse of the logit function. It takes a real-valued input (often a linear combination of predictors) and transforms it into a value between 0 and 1 , which can be interpreted as a probability.
Mathematically, the logistic function is defined as

$$\phi_{\text{logistic }}(z) = \frac{1}{1+\exp(-z)}$$

Given a value z (which can be thought of as the log odds), the logistic function returns the probability p that corresponds to these log odds.

4.2 Logistic regression

Logistic regression model assumes a underlying conditional probability:

$$\operatorname{Pr}(y \mid \mathbf{x}) = \frac{\exp\left(+\frac{1}{2} y \mathbf{w}^T \mathbf{x}\right)}{\exp\left(+\frac{1}{2} y \mathbf{w}^T \mathbf{x}\right) + \exp\left(-\frac{1}{2} y \mathbf{w}^T \mathbf{x}\right)}$$

Dividing the numerator and denominator by $\exp\left(+\frac{1}{2} y \mathbf{w}^T \mathbf{x}\right)$ reveals the logistic function

$$\operatorname{Pr}(y \mid \mathbf{x}) = \phi_{\text{logistic }}\left(y \mathbf{w}^T \mathbf{x}\right) = \frac{1}{1+\exp\left(-y \mathbf{w}^T \mathbf{x}\right)}$$

The margin $z = y\mathbf{w}^T\mathbf{x}$ is thus interpreted as the log odds ratio of label y vs. label $-y$ given input $\mathbf{x}$:

$$y\mathbf{w}^T\mathbf{x} = \log \frac{\Pr(y \mid \mathbf{x})}{\Pr(-y \mid \mathbf{x})}$$

4.3 Logistic loss

Recall:
The likelihood function measures how probable the observed outputs are, given the inputs and a specific set of weights $\mathbf{w}$.
Maximizing this likelihood means finding the weight vector $\mathbf{w}^*$ that makes the observed data most probable under the model.
In logistics regression

$$\mathbf{w}^* = \operatorname{argmax}_{\mathbf{w}} \prod_{i=1}^{m} P\left(y_i \mid \mathbf{x}_i\right) = \operatorname{argmax}_{\mathbf{w}} \prod_{i=1}^{m} \frac{1}{1+\exp\left(-y\mathbf{w}^T\mathbf{x}\right)}$$

Since the logarithm is monotonically increasing function, we can take the logarithm to obtain an equivalent objective:

$$\sum_{i=1}^{m} \log \Pr\left(y_i \mid \mathbf{x}_i\right) = -\sum_{i=1}^{m} \log\left(1+\exp\left(-y_i\mathbf{w}^T\mathbf{x}_i\right)\right)$$

The right-hand side is the logistic loss:

$$L_{\text{logistic}}\left(y, \mathbf{w}^T\mathbf{x}\right) = \log\left(1+\exp\left(-y\mathbf{w}^T\mathbf{x}\right)\right)$$

So minimizing the logistic loss correspond maximizing the likelihood of the training data
Logistic loss is convex and differentiable
It is a monotonically decreasing function of the margin $y\mathbf{w}^T\mathbf{x}$

4.4 Optimization

To train a logistic regression model, we need to find the $\mathbf{w}$ that minimizes the average logistic loss

$$J(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} L_{\text{logistic}}\left(y_i, \mathbf{w}^T\mathbf{x}_i\right)$$

over the training set:

$$\min J(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} \log\left(1+\exp\left(-y_i\mathbf{w}^T\mathbf{x}_i\right)\right)$$
$$\text{w.r.t parameters } \mathbf{w} \in \mathbb{R}^d$$

The function to be minimized is continuous and differentiable However, it is a non-linear function so it is not easy to find the optimum directly (e.g. unlike in linear regression)

We will use **stochastic gradient descent** to incrementally step towards the direction where the objective decreases fastest, the negative gradient

4.4.1 Gradient Descend

The gradient is the vector of partial derivatives of the objective function $J(\mathbf{w})$ with respect to all parameters w_j

$$\nabla J(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} \nabla J_i(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m}\left[\frac{\partial}{\partial w_1} J_i(\mathbf{w}), \ldots, \frac{\partial}{\partial w_d} J_i(\mathbf{w})\right]^T$$

Compute the gradient by using the regular rules for differentiation. For the logistic loss we have

$$\begin{aligned}\frac{\partial}{\partial w_j} J_i(\mathbf{w}) &= \frac{\partial}{\partial w_j}\log\left(1+\exp\left(-y_i\mathbf{w}^T x_i\right)\right) = \frac{\exp\left(-y_i\mathbf{w}^T x_i\right)}{1+\exp\left(-y_i\mathbf{w}^T x_i\right)}\cdot\left(-y_i x_{ij}\right)\\ &= -\frac{1}{1+\exp\left(y_i\mathbf{w}^T x_i\right)} y_i x_{ij} = -\phi_{\text{logistic}}\left(-y_i\mathbf{w}^T x_i\right) y_i x_{ij}\end{aligned}$$

We collect the partial derivatives with respect to a single training example into a vector:

$$\nabla J_i(\mathbf{w}) = \begin{bmatrix} -\left(\phi_{\text{logistic}}\left(-y_i\mathbf{w}^T\mathbf{x}_i\right) y_i\right)\cdot x_{i1} \\ \vdots \\ -\left(\phi_{\text{logistic}}\left(-y_i\mathbf{w}^T\mathbf{x}_i\right) y_i\right)\cdot x_{ij} \\ \vdots \\ -\left(\phi_{\text{logistic}}\left(-y_i\mathbf{w}^T\mathbf{x}_i\right) y_i\right)\cdot x_{id}\end{bmatrix} = -\phi_{\text{logistic}}\left(-y_i\mathbf{w}^T\mathbf{x}_i\right) y_i\cdot\mathbf{x}_i$$

The vector $-\nabla J_i(\mathbf{w})$ gives the update direction that fastest decreases the loss on training example $(\mathbf{x}_i, y_i)$
Evaluating the full gradient

$$\nabla J(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m}\nabla J_i(\mathbf{w}) = -\frac{1}{m}\sum_{i=1}^{m}\phi_{\text{logistic}}\left(-y_i\mathbf{w}^T\mathbf{x}_i\right) y_i\cdot\mathbf{x}_i$$

is costly since we need to process all training examples Stochastic gradient descent instead uses a series of smaller updates that depend on single randomly drawn training example $(\mathbf{x}_i, y_i)$ at a time
The update direction is taken as $-\nabla J_i(\mathbf{w})$ Its expectation is the full negative gradient:

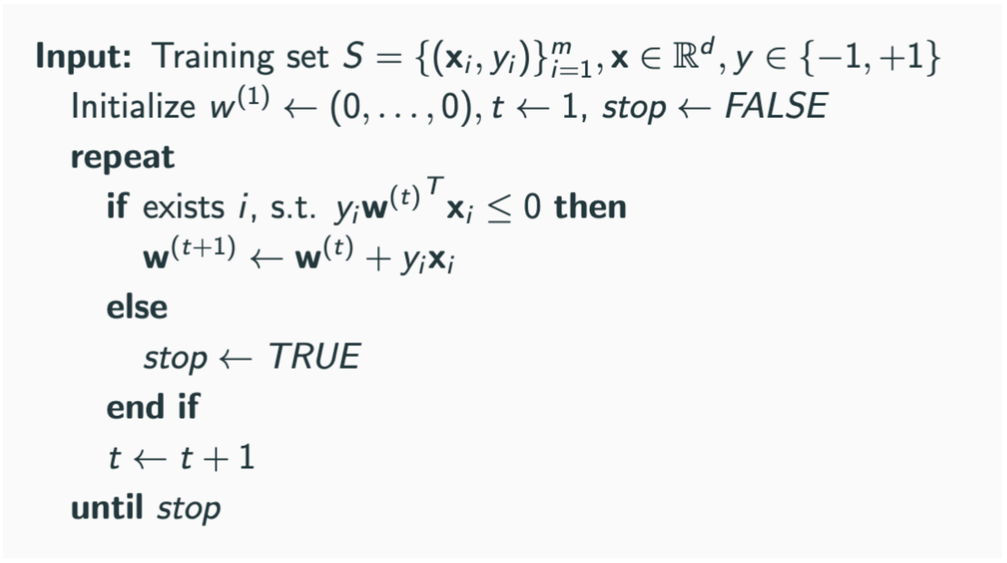
$$-\mathbb{E}_{i=1\ldots,m}\left[\nabla J_i(\mathbf{w})\right] = -\nabla J(\mathbf{w})$$

Thus on average, the updates match that of using the full gradient

4.4.2 Algorithm

4.4.3 Stepsize

Consider the SGD update: $\mathbf{w} = \mathbf{w} - \eta\nabla J_i(\mathbf{w})$



```
Input: Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m, \mathbf{x} \in \mathbb{R}^d, y \in \{-1, +1\}$
  Initialize $w^{(1)} \leftarrow (0, \ldots, 0), t \leftarrow 1$, $stop \leftarrow FALSE$
  repeat
    if exists $i$, s.t. $y_i {\mathbf{w}^{(t)}}^T \mathbf{x}_i \leq 0$ then
      $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y_i \mathbf{x}_i$
    else
      $stop \leftarrow TRUE$
    end if
    $t \leftarrow t + 1$
  until $stop$
```

Figure 4: Algorithm for GD

The stepsize parameter η, also called the learning rate is a critical one for convergence to the optimum value
One uses small constant stepsize, the initial convergence may be unnecessarily slow
Too large stepsize may cause the method to continually overshoot the optimum.
Alternatively, we can use diminishing stepsize
Initially larger but diminishing stepsize is one option:

$$\eta^{(t)} = \frac{1}{\alpha t}$$

for some $\alpha > 0$, where t is the iteration counter Caution: In practice, finding a good value for parameter α requires experimenting with several values

4.4.4 Stopping criterion

When should we stop the algorithm? Some possible choices:
1. Set a maximum number of iterations, after which the algorithm terminates
- This needs to be separately calibrated for each dataset to avoid premature termination
2. Gradient of the objective: If we are at a optimum point $\mathbf{w}^*$ of $J(\mathbf{w})$, the gradient vanishes $\nabla J(\mathbf{w}^*) = 0$, so we can stop $\|J(\mathbf{w})\| < \gamma$ where γ is some user-defined parameter 3. It is usually sufficient to train until the zero-one error on training data does not change anymore
- This usually happens before the logistic loss converges