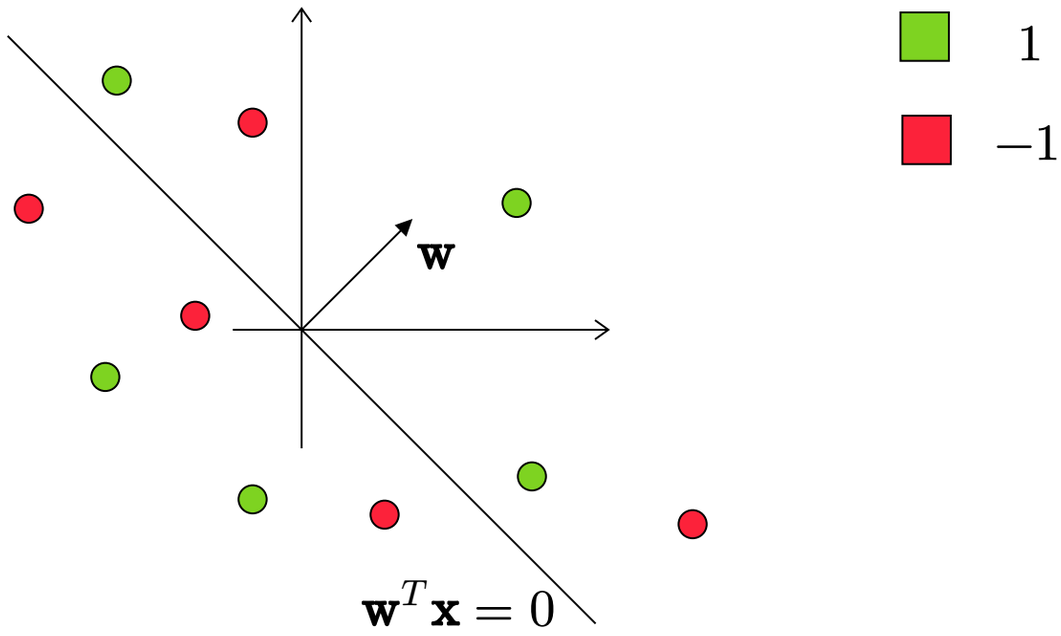
Week-9

Karthik Thiagarajan

1. Linear classifier

Recall that a linear classifier is $h : \mathbb{R}^d \to \{1, -1\}$ with $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, where $\mathbf{w} \in \mathbb{R}^d$.



We wish to minimize the $0-1$ loss:

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \mathbb{I}\Big[\mathbf{w}^T\mathbf{x}_i \neq y_i\Big]$$

This problem is NP-hard in general. We shall now impose some constraints on the data and see if it simplifies the problem. One such assumption is linear separability.

Remark: For the rest of the document, we will assume that the labels lie in the set $\{1, -1\}$. Green data-points belong to class 1 and red data-points belong to class -1.

2. Linear Separability

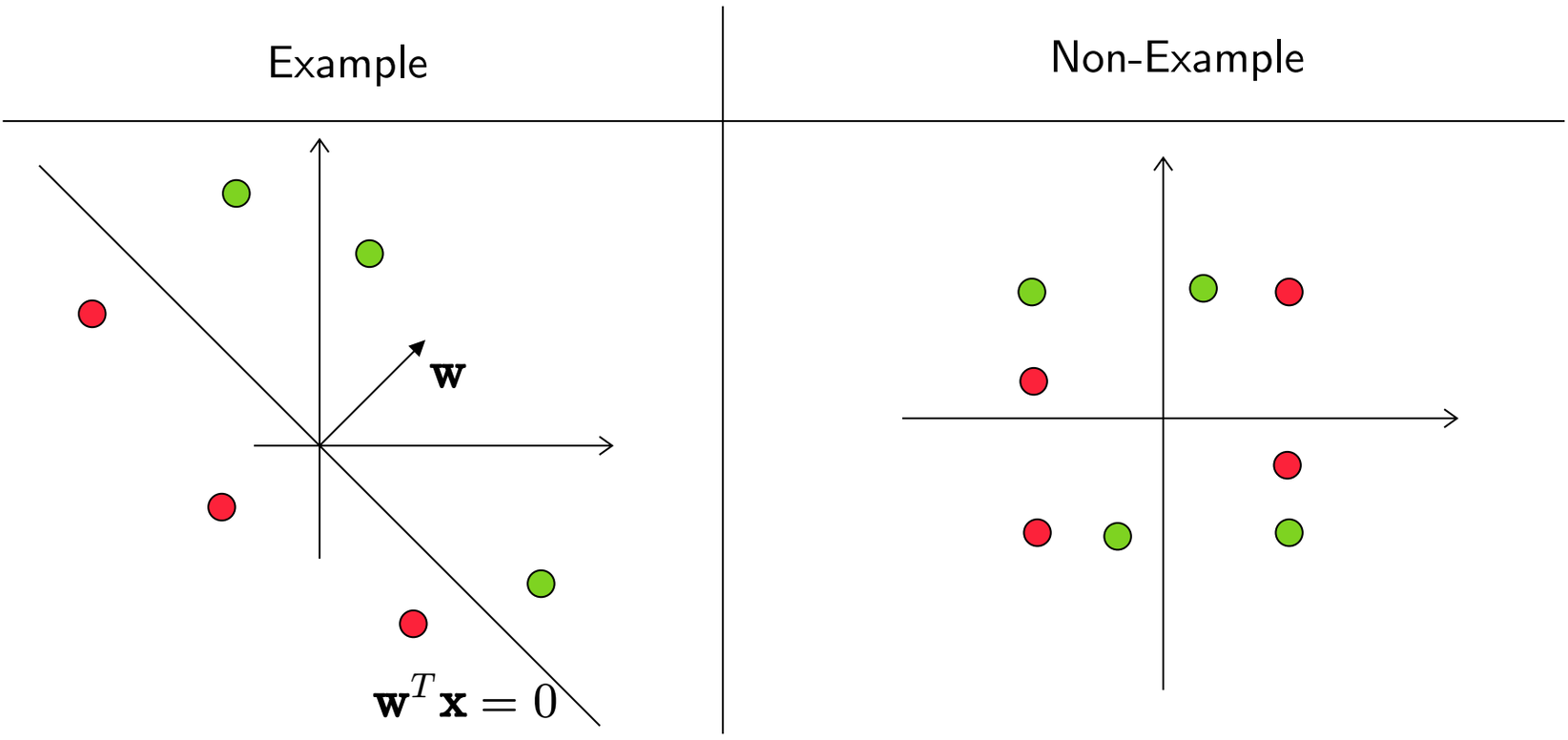
Statement

A dataset $D = \{(\mathbf{x}_1, y_1),\ \cdots, (\mathbf{x}_n, y_n)\}$ is linearly separable if there exist a $\mathbf{w} \in \mathbb{R}^d$ such that:

$$\left(\mathbf{w}^T \mathbf{x}_i\right) y_i \geqslant 0,\ 1 \leqslant i \leqslant n$$

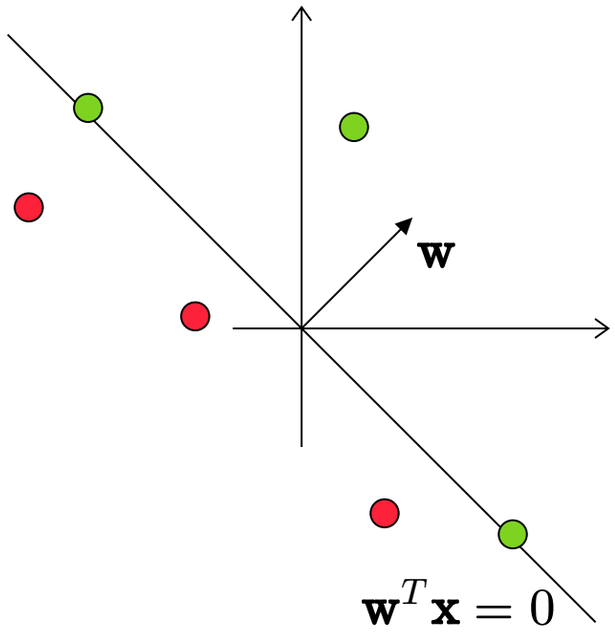
where $y_i \in \{1, -1\}$.

In simple terms, a dataset is linearly separable if there exists a linear classifier that perfectly classifies (separates) all the data-points thereby producing zero training error.



We could also have a situation where a dataset is just about linearly separable. That is, there is only one linear classifier that perfectly separates the data (after

normalizing the weights).



For the algorithms that we plan to discuss, it would be convenient to get rid of this limitation. So we go for a stronger assumption.

3. Linear Separability with γ margin

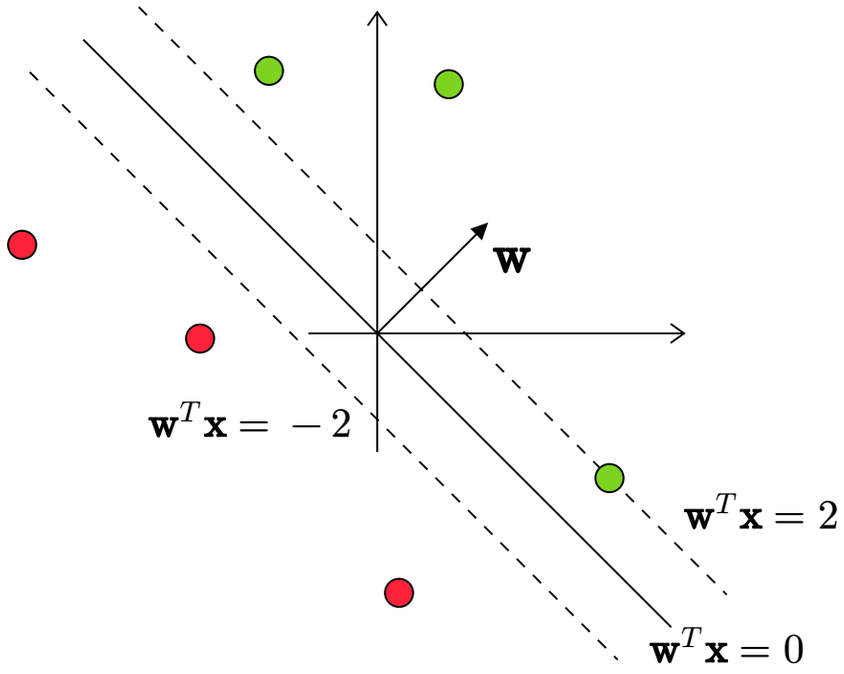
Statement

A dataset $D = \{(\mathbf{x}_1, y_1),\ \cdots, (\mathbf{x}_n, y_n)\}$ is linearly separable with γ margin if there exist a $\mathbf{w} \in \mathbb{R}^d$ and a $\gamma > 0$ such that:

$$\left(\mathbf{w}^T\mathbf{x}_i\right)y_i \geqslant \gamma,\ 1 \leqslant i \leqslant n$$

where $y_i \in \{1, -1\}$.

As an example:



For linearly separable datasets with a positive margin, we have algorithms that can return a valid classifier that achieves zero training error.

4. Perceptron Learning Algorithm

4.1. Algorithm

The prediction rule for the perceptron is given as:

$$\widehat{y} = \begin{cases} 1, & \mathbf{w}^T\mathbf{x} \geqslant 0 \\ -1, & \mathbf{w}^T\mathbf{x} < 0 \end{cases}$$

Note that in the situation where $\mathbf{w}^T\mathbf{x} = 0$, the predicted label is 1 as per our convention.

1) Initialize $\mathbf{w}^{(0)} = \mathbf{0}$

2) while sign$\left({\mathbf{w}^{(t)}}^T \mathbf{x}_i\right) \neq y_i$ for some i :

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

In more detail:

- $\mathbf{w}^{(0)} = \mathbf{0}$
- Cycle through the data-points in a fixed order, say:
  - $(\mathbf{x}_1, y_1) \rightarrow \cdots \rightarrow (\mathbf{x}_n, y_n)$
- If (x_i, y_i) is a mistake w.r.t $\mathbf{w}^{(t)}$:
  - $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \mathbf{x}_i y_i$
- If there are no mistakes for n consecutive iterations
  - terminate the algorithm
  - return the weight vector $\mathbf{w}^{(T)}$
  - here T is the number of updates to $\mathbf{w}$

Some observations:

- In this course, we always begin with $\mathbf{w}^{(0)} = \mathbf{0}$
- In the course of the algorithm, the number of mistakes made is equal to the number of updates to the weight vector.
- The number of iterations is greater than or equal to the number of mistakes as there might be iterations in which there are no mistake.
- For a linearly separable data with positive margin, the perceptron learning algorithm converges.

4.2. Proof of Convergence

4.2.1. Assumptions

Assumption-1: Linear Separability with γ margin

There exists some $\gamma > 0$ and $\mathbf{w^*}$ such that for every point $(\mathbf{x}_i, y_i)$ in the dataset, we have:

$$(\mathbf{w^*}^T \mathbf{x}_i) y_i \geqslant \gamma$$

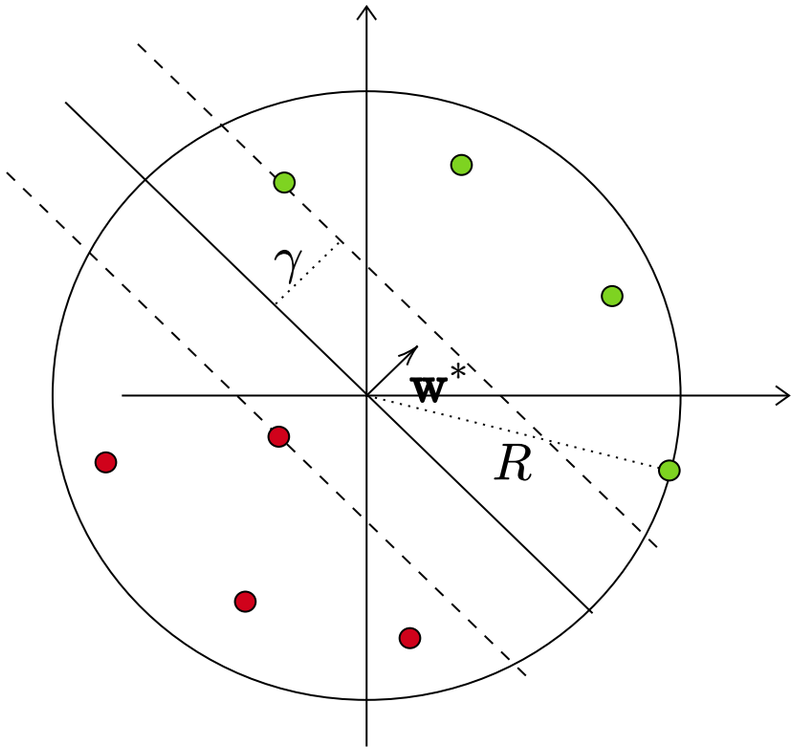
with $||\mathbf{w}^*|| = 1$. This assumption encodes the fact that the data is linearly separable with a margin of γ. Since $||\mathbf{w}^*|| = 1$, the margin is the distance between the decision boundary and the closest point in the dataset.

Assumption-2: Radius bound

There exists some $R > 0$ such that for every point $(\mathbf{x}_i, y_i)$ in the dataset, we have:

$$||\mathbf{x}_i||^2 \leqslant R^2$$

This assumption encodes the fact that all data-points are within a ball of radius R centered at the origin. These two assumptions can be visualized using this image:



We start with $\mathbf{w}^{(0)} = \mathbf{0}$. Since each update to the weight vector happens after a mistake is observed, we will use the term mistakes and updates interchangeably. Formally, let the weight vector **after** t updates be denoted by $\mathbf{w}^{(t)}$, where $t \geqslant 0$. The case of $t = 0$ corresponds to zero updates and the corresponding weight vector will be $\mathbf{w}^{(0)}$. After t rounds, the perceptron has seen t mistakes and the weight vector has been updated t times.

4.2.2. Upper bound

Assume that the perceptron has gone through $t-1$ updates. Let $(\mathbf{x}, y)$ be some point that is misclassified by the perceptron at this stage. This is the t^{th} mistake that it is seeing. As the current weight vector of the perceptron is $\mathbf{w}^{t-1}$, we have:

$$\left(\mathbf{w}^{(t-1)^T}\mathbf{x}\right)y \leqslant 0$$

This necessitates one more round of weight update. The update rule for this step is as follows:

$$\mathbf{w}^{(t)} := \mathbf{w}^{(t-1)} + \mathbf{x}y$$

Let us now look at the norm of the weight vector and see how it changes across iterations:

$$\begin{aligned}
||\mathbf{w}^{(t)}||^2 &= (\mathbf{w}^{(t-1)} + \mathbf{x}y)^T(\mathbf{w}^{(t-1)} + \mathbf{x}y) \\
&= ||\mathbf{w}^{(t-1)}||^2 + y^2 \cdot ||\mathbf{x}||^2 + 2 \cdot y \cdot \left({\mathbf{w}^{(t-1)}}^T \mathbf{x}\right) \\
&\leqslant ||\mathbf{w}^{t-1}||^2 + R^2
\end{aligned}$$

We have used three facts to get the inequality:

- $y^2 = 1$
- $||\mathbf{x}||^2 \leqslant R^2$
- $\left({\mathbf{w}^{(t-1)}}^T \mathbf{x}\right) y \leqslant 0$

We can now apply this inequality recursively on the earlier rounds:

$$\begin{aligned}
||\mathbf{w}^{(t)}||^2 &\leqslant ||\mathbf{w}^{(t-1)}||^2 + R^2 \\
&\leqslant ||\mathbf{w}^{(t-2)}||^2 + R^2 + R^2 \\
&\leqslant ||\mathbf{w}^{(t-3)}||^2 + R^2 + R^2 + R^2 \\
&\leqslant \quad \vdots \\
&\leqslant ||\mathbf{w}^{(t-t)}||^2 + tR^2 \\
&= ||\mathbf{w}^{(0)}||^2 + tR^2 \\
&= t \cdot R^2
\end{aligned}$$

We have used the fact that $\mathbf{w}^{(0)} = \mathbf{0}$. Therefore, the upper bound is:

$$\boxed{||\mathbf{w}^{(t)}||^2 \leqslant t \cdot R^2}$$

4.2.3. Lower bound

Now, we look at the relationship between the optimal weight vector $\mathbf{w}^*$ and the weight vector $\mathbf{w}^{(t)}$. The dot product is a measure of this relationship:

$$\begin{aligned}
\mathbf{w^*}^T \mathbf{w}^{(t)} &= \mathbf{w^*}^T (\mathbf{w}^{(t-1)} + \mathbf{x}y) \\
&= \mathbf{w^*}^T \mathbf{w}^{(t-1)} + \left(\mathbf{w^*}^T \mathbf{x}\right) y \\
&\geqslant \mathbf{w^*}^T \mathbf{w}^{(t-1)} + \gamma
\end{aligned}$$

We have used the fact that $\left(\mathbf{w^*}^T \mathbf{x}\right) y \geqslant \gamma$ to get the above inequality. Recursively applying this for earlier rounds and using the fact that $\mathbf{w}^{(0)} = \mathbf{0}$, we get:

$$\mathbf{w^*}^T \mathbf{w}^t \geqslant t \cdot \gamma$$

Now, we use the Cauchy-Schwartz inequality and the assumption that $||\mathbf{w}^*|| = 1$ to get:

$$\begin{aligned}
||\mathbf{w^*}|| \cdot ||\mathbf{w}^{(t)}|| &\geqslant \mathbf{w^*}^T \mathbf{w}^{(t)} \\
||\mathbf{w}^{(t)}|| &\geqslant \mathbf{w^*}^T \mathbf{w}^{(t)}
\end{aligned}$$

Clubbing this result with the previous one, we get:

$$||\mathbf{w}^{(t)}|| \geqslant \mathbf{w^*}^T \mathbf{w}^{(t)} \geqslant t \cdot \gamma$$

Squaring the first and last terms:

$$\boxed{||\mathbf{w}^{(t)}||^2 \geqslant t^2 \cdot \gamma^2}$$

4.2.4. Combining the bounds

We now have a lower bound and an upper bound for $||\mathbf{w}^t||^2$:

$$\boxed{t^2 \cdot \gamma^2 \leqslant ||\mathbf{w}^t||^2 \leqslant t \cdot R^2}$$

Using these two bounds, we have:

$$\boxed{t \leqslant \frac{R^2}{\gamma^2}}$$

This is called the radius-margin bound.

4.2.5. Interpreting the bound

$\mathbf{w}^{(t)}$ is the weight vector after t updates. Another way of seeing this is $\mathbf{w}^{(t)}$ is the weight vector after the perceptron has made t mistakes. Each update is mapped to one mistake. Therefore, the total number of updates to the weight vector is bounded above $\frac{R^2}{\gamma^2}$. Or, we could also say, the perceptron has to

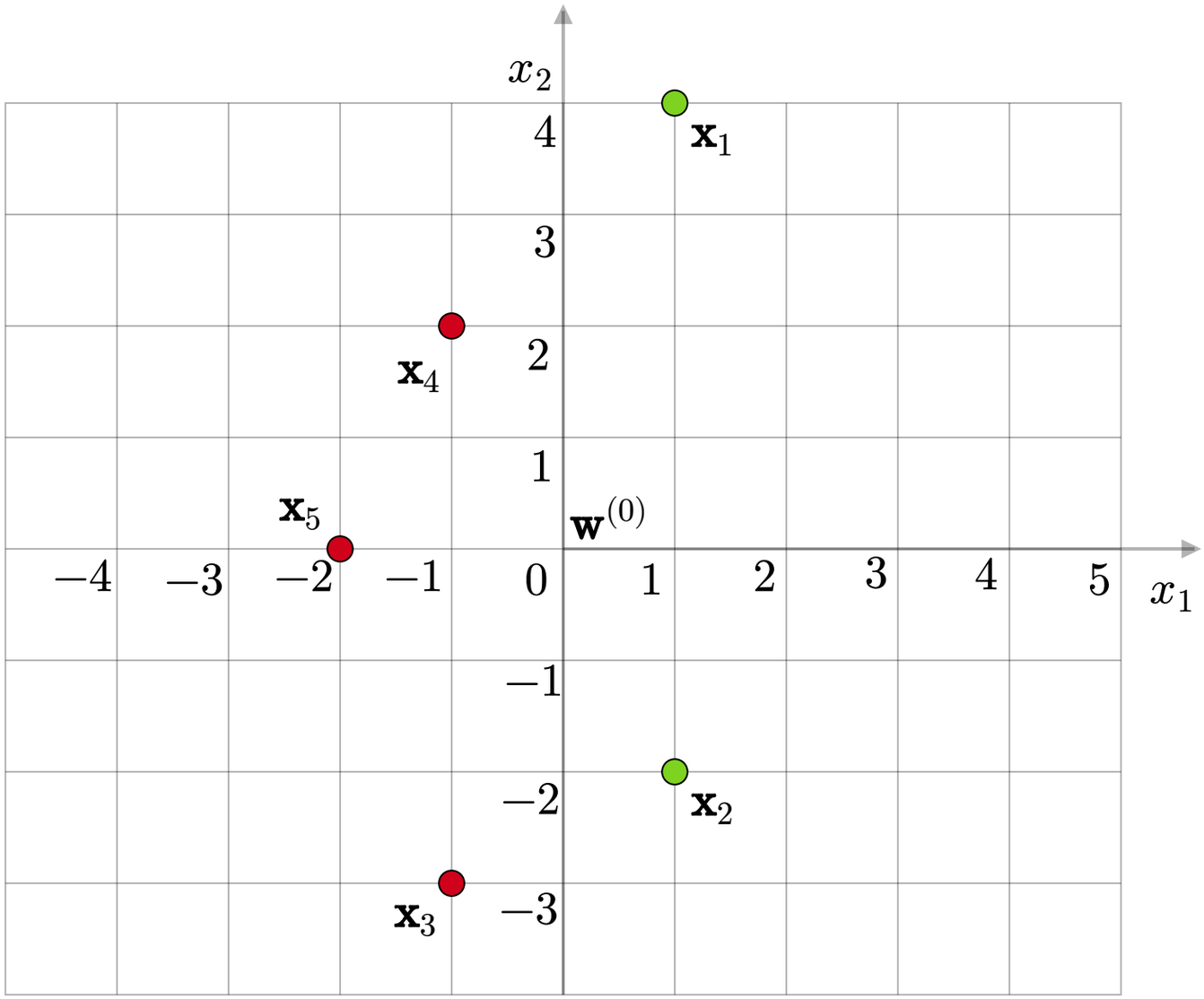
correct **at most** $\frac{R^2}{\gamma^2}$ mistakes. Since this is a finite quantity, the perceptron algorithm converges. Note that $\mathbf{w}^{(0)}$ doesn't count as an update.

4.3. Example

Consider the following dataset:

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & -1 & -1 & -2 \\ 4 & -2 & -3 & 2 & 0 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

Cycling through the data-points in the order $1 \to 2 \to \cdots \to 5$:



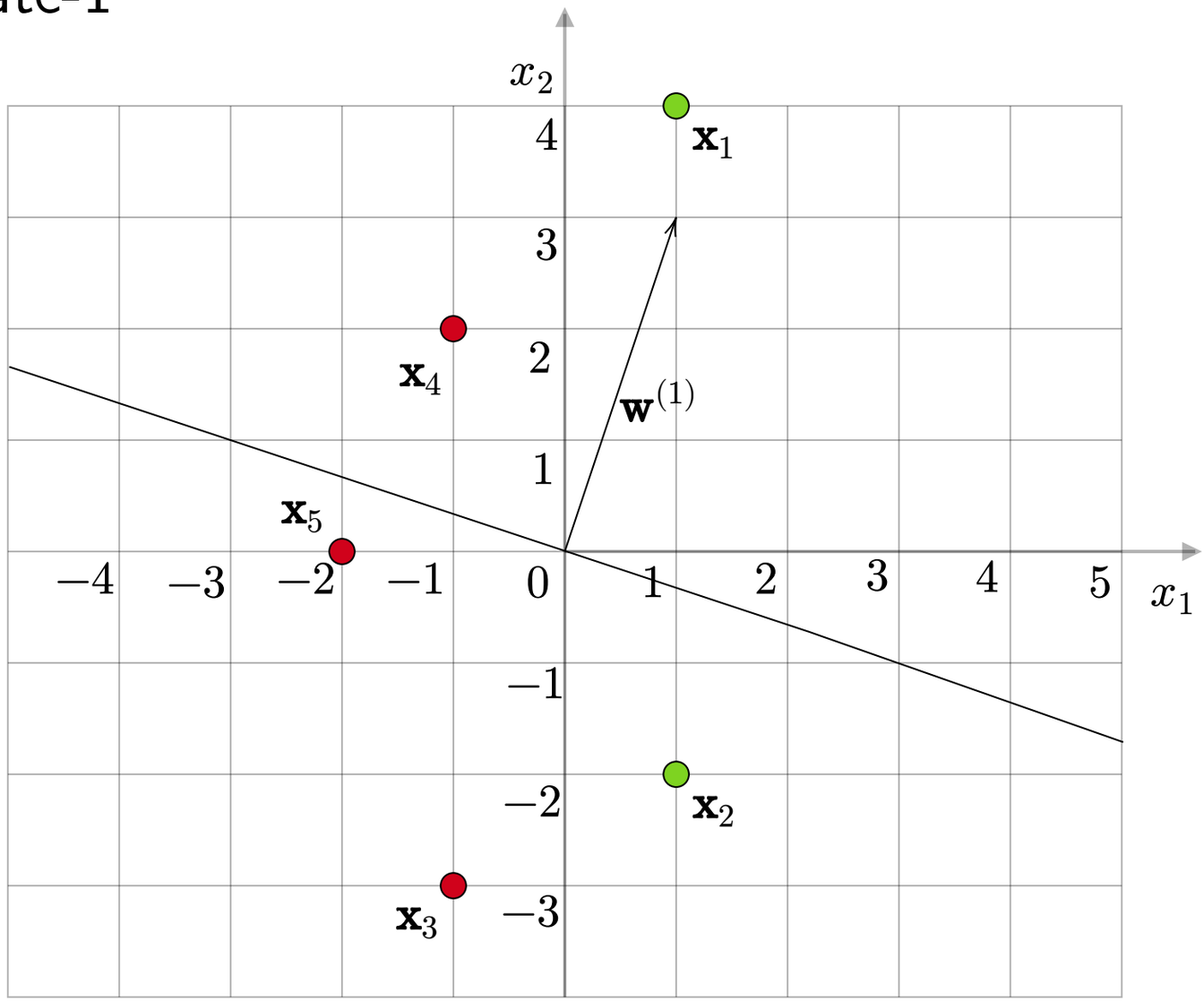
Since $\mathbf{w}^{(0)} = 0$, all points are going to be predicted as 1. The first mistake is $(\mathbf{x}_3, y_3)$.

$$\mathbf{w}^{(1)} = \mathbf{w}^{(0)} + \mathbf{x}_3 y_3$$

$$= \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} -1 \\ -3 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

Update-1



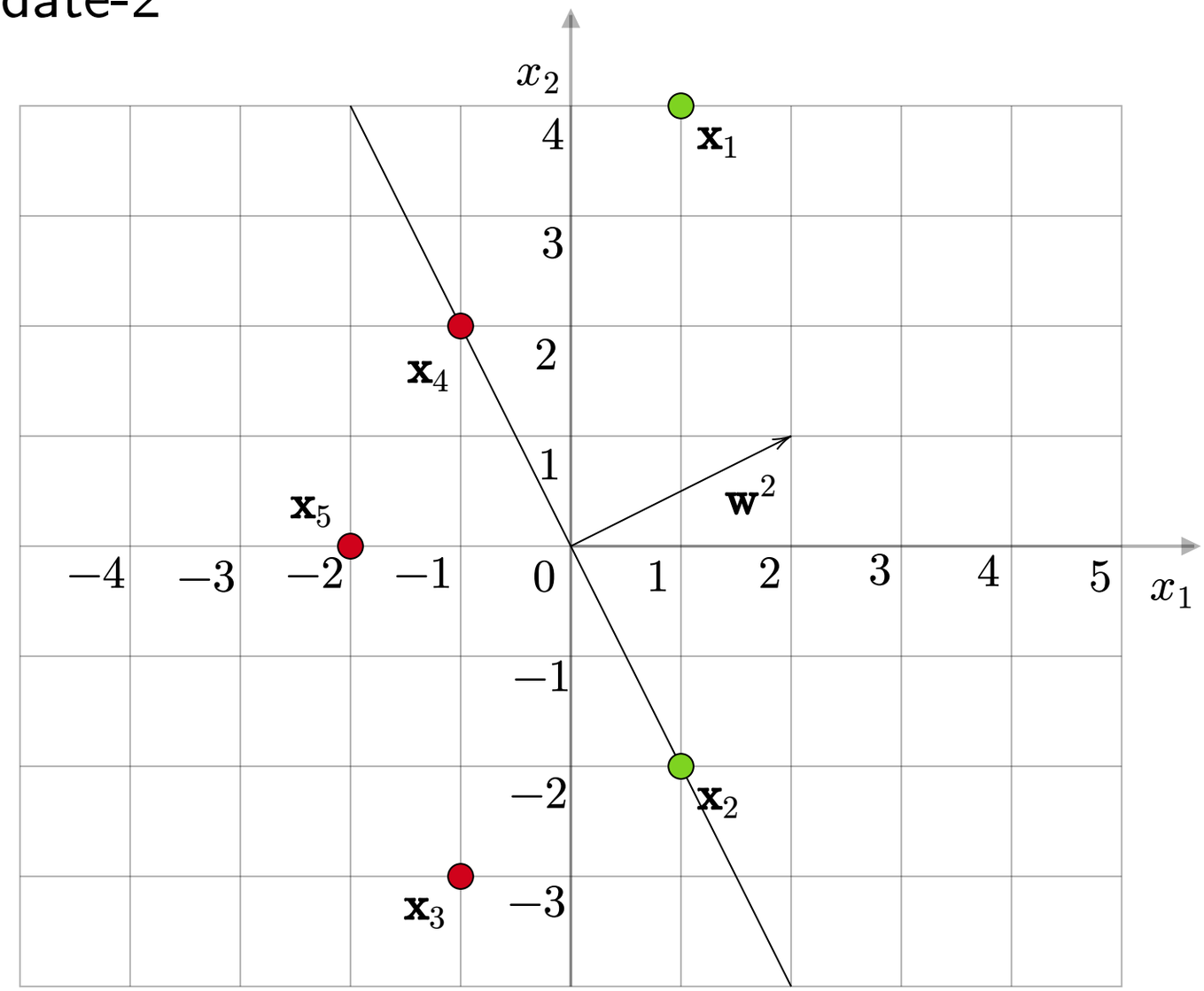
We now have to resume the cycle from data-point $(\mathbf{x}_4, y_4)$.

$$\mathbf{w}^{(2)} = \mathbf{w}^{(1)} + \mathbf{x}_4 y_4$$

$$= \begin{bmatrix} 1 \\ 3 \end{bmatrix} - \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

Update-2



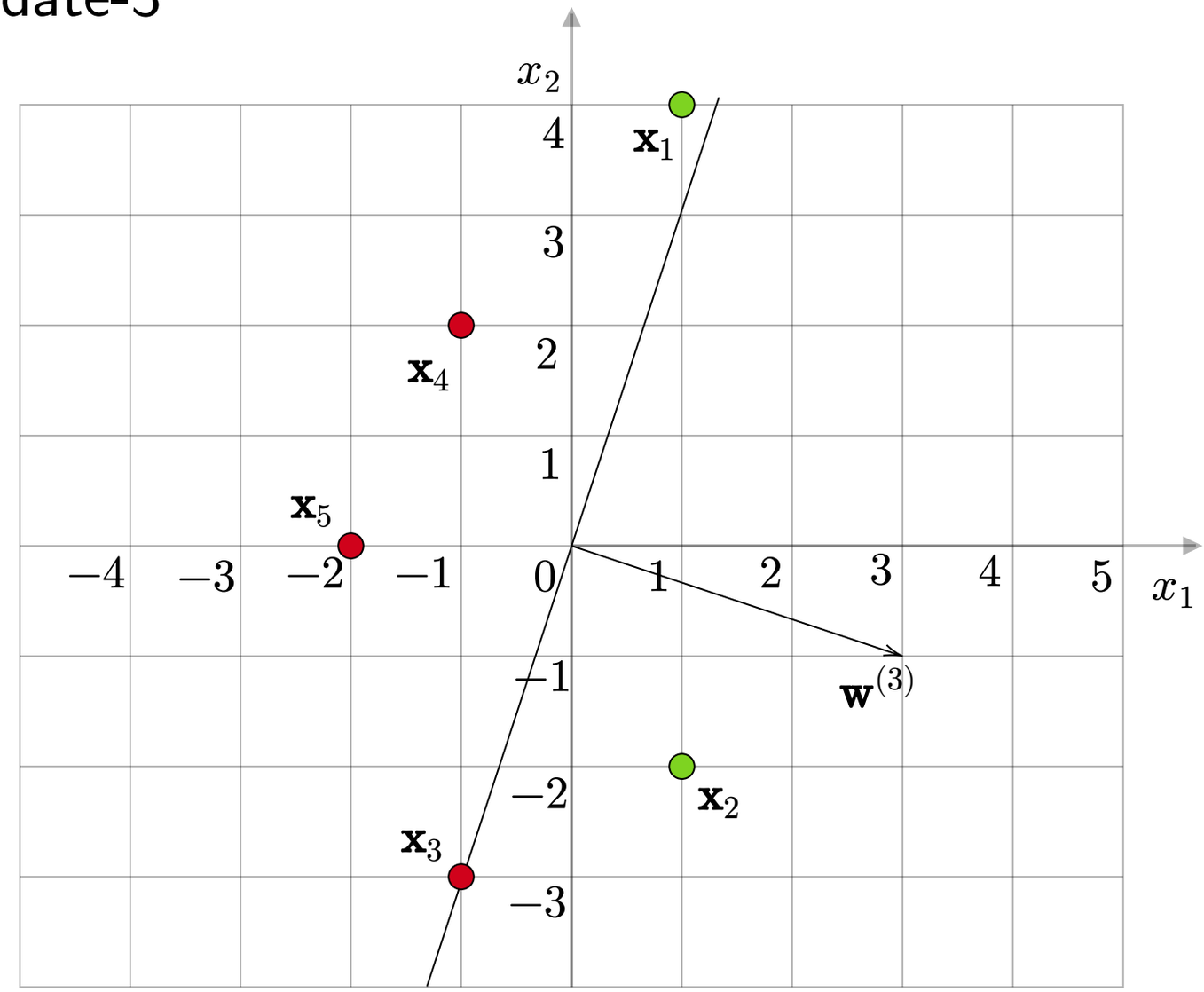
We now have to resume the cycle from data-point $(\mathbf{x}_5, y_5)$.

$$\mathbf{w}^{(3)} = \mathbf{w}^{(2)} + \mathbf{x}_4 y_4$$

$$= \begin{bmatrix} 2 \\ 1 \end{bmatrix} - \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

Update-3



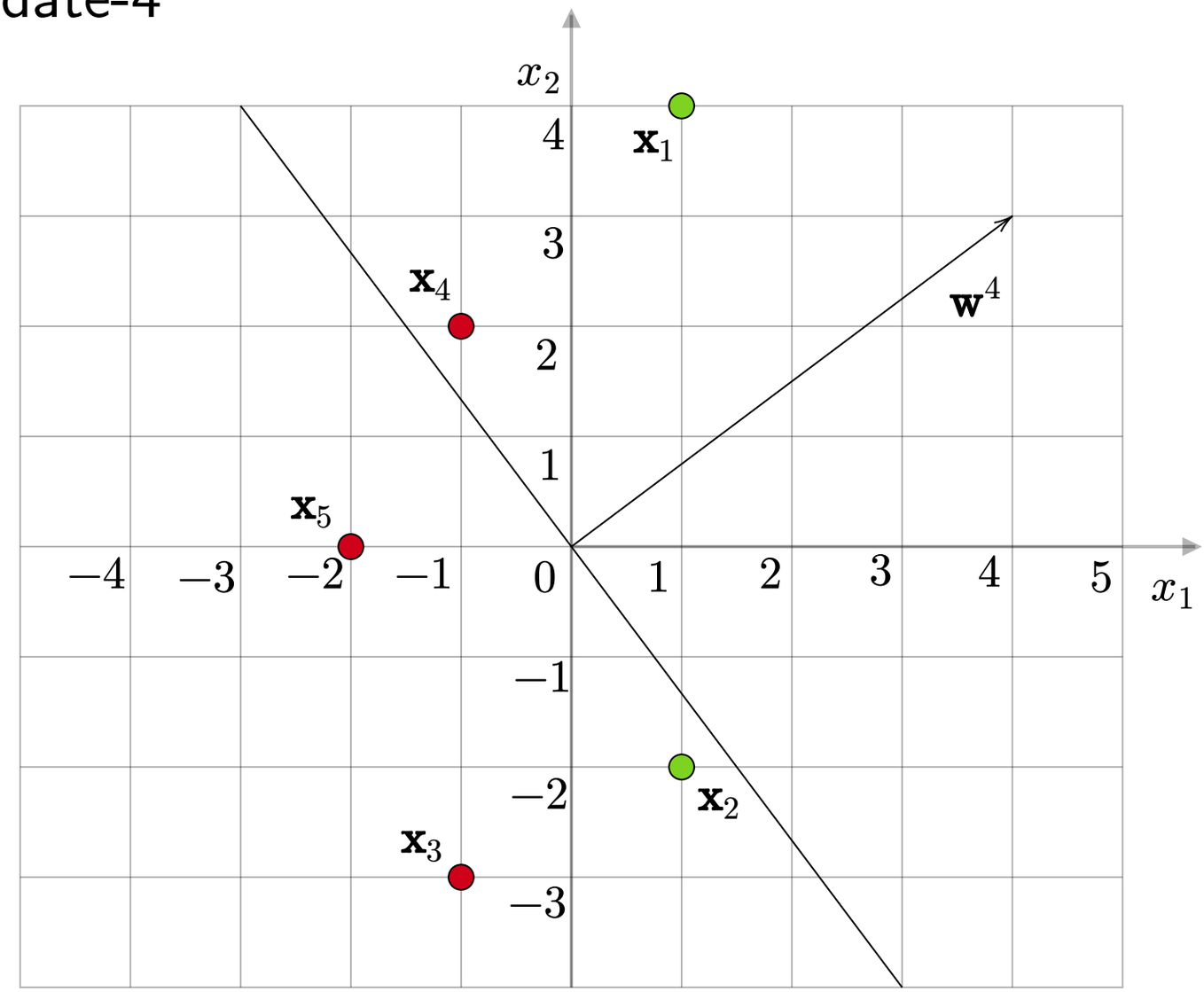
We now have to resume the cycle from data-point $(\mathbf{x}_5, y_5)$.

$$\mathbf{w}^{(4)} = \mathbf{w}^{(3)} + \mathbf{x}_1 y_1$$

$$= \begin{bmatrix} 3 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

$$= \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

Update-4



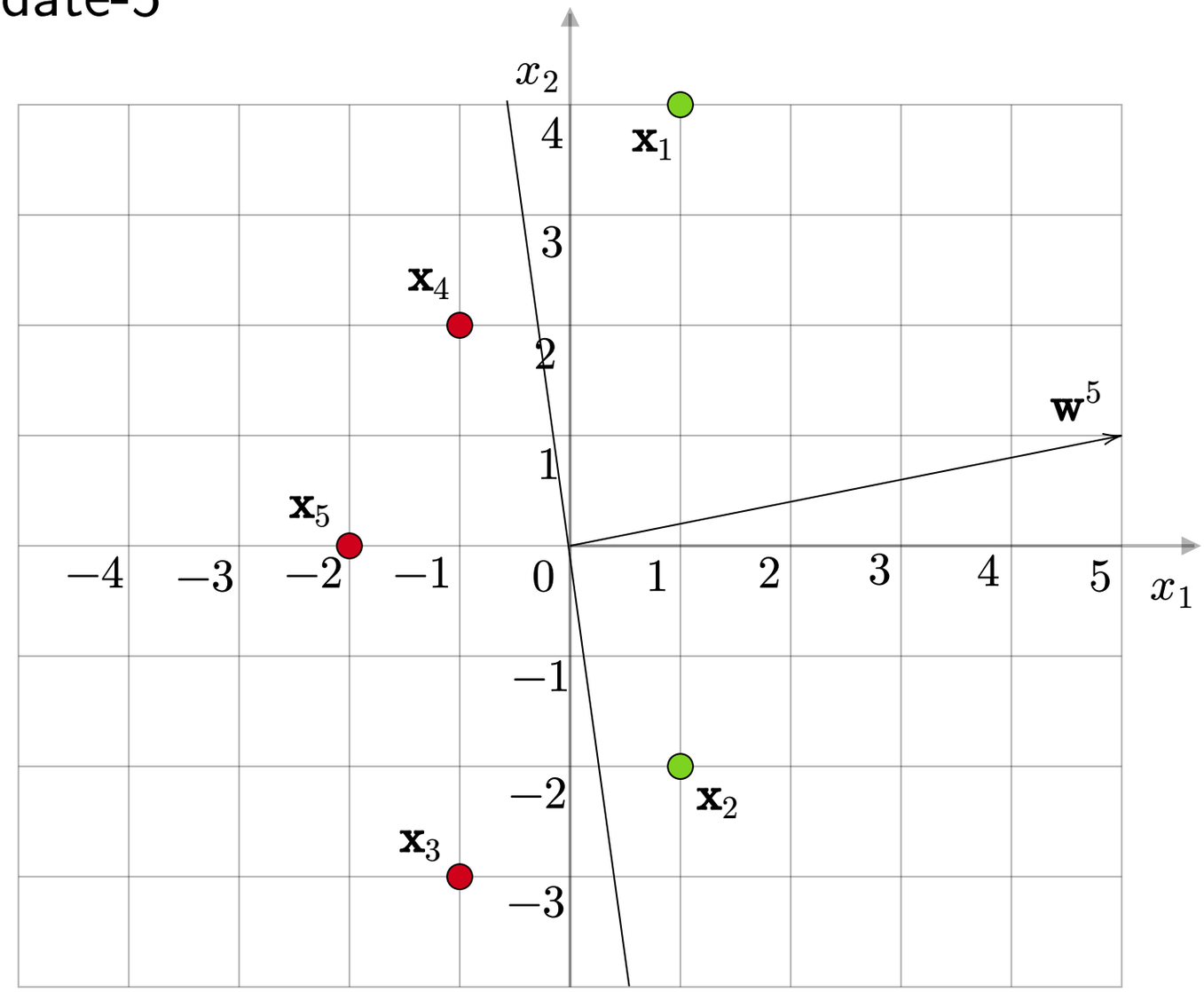
We now have to resume the cycle from data-point $(\mathbf{x}_2, y_2)$.

$$\mathbf{w}^{(5)} = \mathbf{w}^{(4)} + \mathbf{x}_2 y_2$$

$$= \begin{bmatrix} 4 \\ 3 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$= \begin{bmatrix} 5 \\ 1 \end{bmatrix}$$

Update-5



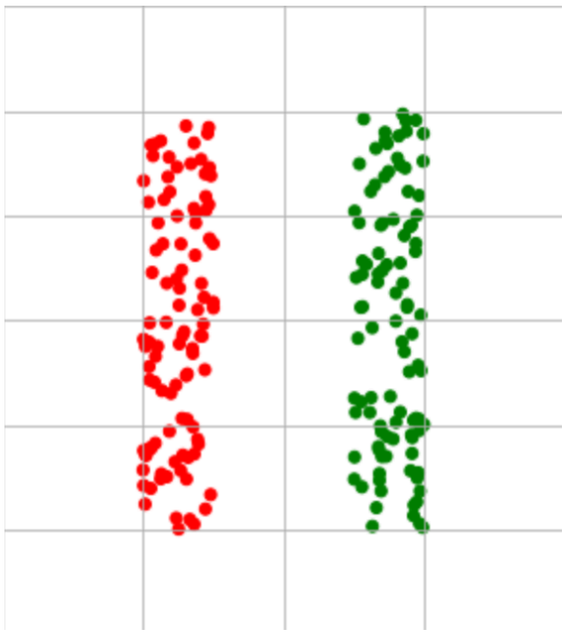
We now have to resume the cycle from data-point $(\mathbf{x}_3, y_3)$. Now, we have five consecutive iterations without any mistake. Therefore, we stop and claim that perceptron has converged.

Remark: There may be multiple valid weight vectors. Perceptron returns one of them. Depending on how you cycle through the data-points, it may return a different weight vector.

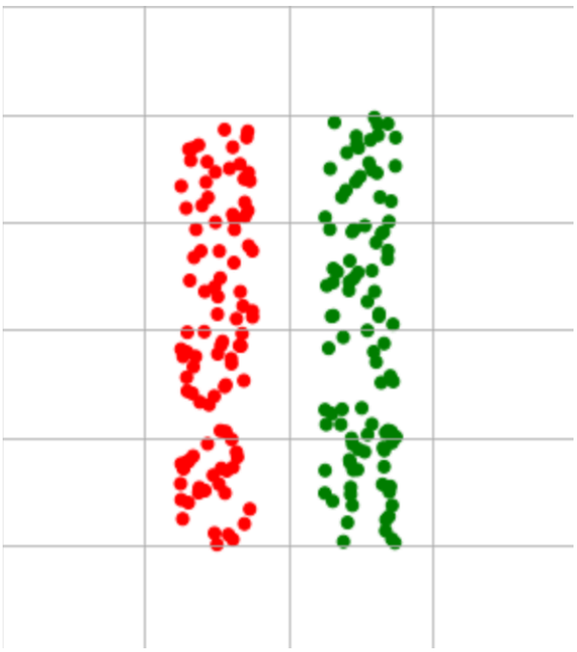
4.4. Effect of Margin

This is an experiment that demonstrates the effect of the margin on the number

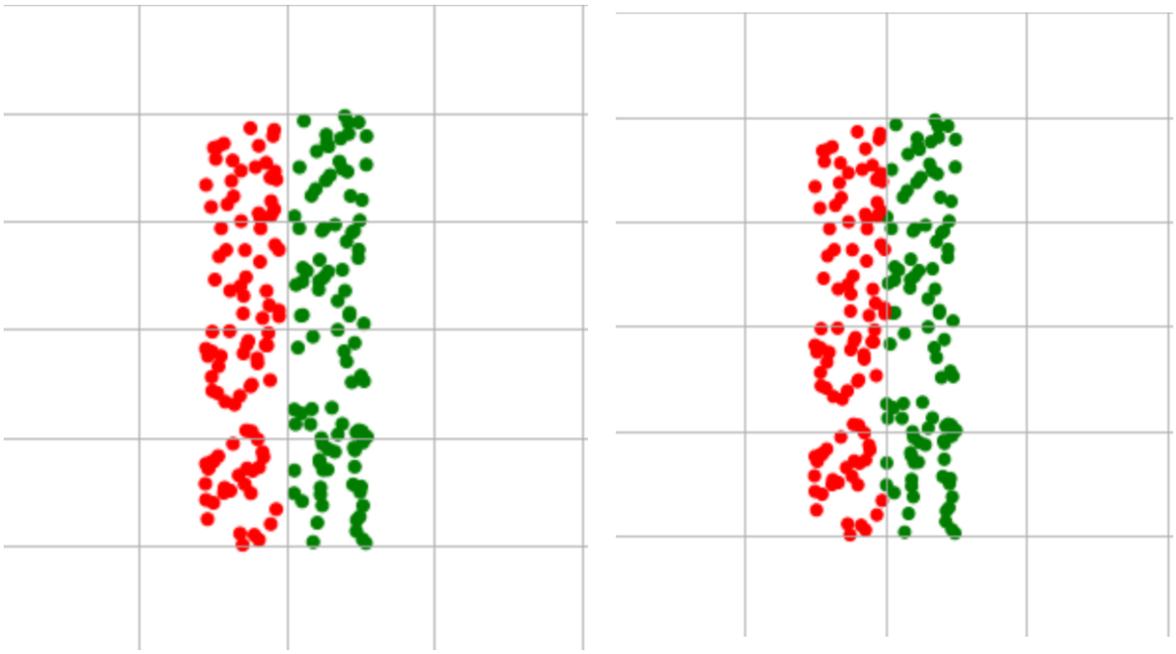
of iterations needed to converge. Note that t refers to the number of updates to the weight vector. It is not the upper bound but the actual number of updates while training a perceptron.



$d = 1$
$t = 1$



$d = 0.5$
$t = 2$



$d = 0.1$
$t = 7$

$d = 0.01$
$t = 44$

As the margin decreases, the number of updates increases.

5. Logistic Regression

The assumption of linear separability is too strict. We now relax it and turn to a discriminative model that outputs non-trivial probabilities. Logistic regression is a popular classifier that models the conditional probability of the label given the data-point, $P(y \mid \mathbf{x})$, in the following manner:

$$P(y = 1 \mid \mathbf{x}) = \sigma\big(\mathbf{w}^T \mathbf{x}\big)$$

where, $\sigma(z) = \dfrac{1}{1 + e^{-z}}$ is called the sigmoid or the logistic function, and $\mathbf{w}$ is the parameter vector or weight vector ($\mathbf{w} \in \mathbb{R}^d$). The sigmoid function lies between 0 and 1. More about this function in the next section. To predict the label for a data-point using this model, the probability is converted to a binary

outcome using a threshold. A typical threshold of 0.5 is used:

- if $P(y = 1 \mid \mathbf{x}\ ) \geqslant 0.5$, the model outputs class-1
- if $P(y = 1 \mid \mathbf{x}\ ) < 0.5$, the model outputs class-0

More formally, the classifier $h : \mathbb{R}^d \to \mathbb{R}$ is given as:

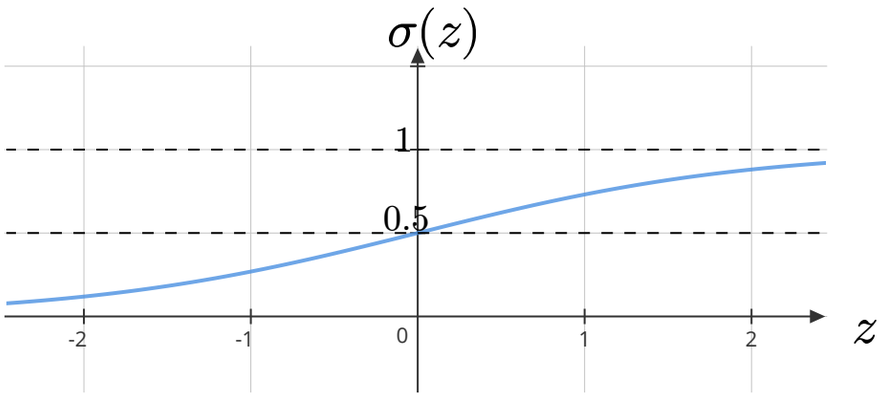
$$h(\mathbf{x}) = \begin{cases} 1, & \sigma\big(\mathbf{w}^T\mathbf{x}\big) \geqslant 0.5 \\ 0, & \sigma\big(\mathbf{w}^T\mathbf{x}\big) < 0.5 \end{cases} = \mathbb{I}\Big[\sigma\big(\mathbf{w}^T\mathbf{x}\big) \geqslant 0.5\Big]$$

We need a way to learn the weights of this model. As we have begun with a probabilistic interpretation, we will turn to the method of parameter estimation using MLE. But before that, let us look at some properties of the sigmoid function.

5.1. Sigmoid Function

The graph of the sigmoid function looks as follows:

$$\sigma : \mathbb{R} \to (0, 1)$$
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



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

Useful properties:

- As $z \to \infty$, $\sigma(z) \to 1$
- As $z \to -\infty$, $\sigma(z) \to 0$.
- We have $\sigma(0) = 0.5$.
- If $z > 0$, then $\sigma(z) > 0.5$ and if $z < 0$, $\sigma(z) < 0.5$.
- Next, $\sigma(-z) = 1 - \sigma(z)$:

$$\begin{aligned}
\sigma(-z) &= \frac{1}{1+e^{z}} \\
&= \frac{e^{-z}}{1+e^{-z}} \\
&= 1 - \frac{1}{1+e^{-z}} \\
&= 1 - \sigma(z)
\end{aligned}$$

- Finally, $\sigma'(z) = \sigma(z) \cdot [1 - \sigma(z)]$:

$$\begin{aligned}
\sigma'(z) &= \frac{-1}{\left(1+e^{-z}\right)^2} \cdot e^{-z} \cdot (-1) \\
&= \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}} \\
&= \sigma(z) \cdot [1 - \sigma(z)]
\end{aligned}$$

This is a very important property and will be used quite a lot in the upcoming sections.

5.2. MLE: Computing the Log-Likelihood

We will assume that the labels are drawn independently. In such a case, the likelihood function is:

$$\begin{aligned} L(\mathbf{w}; D) &= P(D \mid \mathbf{w}) \\ &= \prod_{i=1}^{n} P(y_i \mid \mathbf{x}_i; \mathbf{w}) \end{aligned}$$

Recall that $P(y = 1 \mid \mathbf{x}) = \sigma\big(\mathbf{w}^T \mathbf{x}\big)$. For a data-point $\mathbf{x}_i$, let us call this probability σ_i:

$$\sigma_i = P(y = 1 \mid \mathbf{x}_i) = \sigma\big(\mathbf{w}^T \mathbf{x}_i\big)$$

For example, if $\sigma_i = 0.75$, the model believes that there is a 75% chance that $\mathbf{x}_i$ belongs to class-1. The probability that this point belongs to class-0 is:

$$P(y = 0 \mid \mathbf{x}_i) = 1 - \sigma_i$$

The actual label for this data-point is y_i, which is either 1 or 0, but we don't exactly know what it is. We therefore define it in a piece-wise manner:

$$P(y_i \mid \mathbf{x}_i) = \begin{cases} \sigma_i, & y_i = 1 \\ 1 - \sigma_i, & y_i = 0 \end{cases}$$

However, we can't plug something like this into the likelihood function. We need a form that can be easily manipulated, something like this:

$$P(y_i \mid \mathbf{x}_i) = \sigma_i^{y_i} \cdot (1 - \sigma_i)^{1 - y_i}$$

Notice how this is the same as the piece-wise function defined earlier, but much more suited for algebraic operations. Try out $y_i = 1$ and $y_i = 0$ to see the likeness. Armed with this expression, we go back to the likelihood:

$$\begin{aligned} L(\mathbf{w}; D) &= P(D \mid \mathbf{w}) \\ &= \prod_{i=1}^{n} P(y_i \mid \mathbf{x}_i; \mathbf{w}) \\ &= \prod_{i=1}^{n} \sigma_i^{y_i} \cdot (1 - \sigma_i)^{1 - y_i} \end{aligned}$$

As it is easier to work with $\log$, let us compute the log-likelihood:

$$\begin{aligned} l(\mathbf{w}; D) &= \sum_{i=1}^{n} y_i \log \sigma_i + (1 - y_i) \log(1 - \sigma_i) \\ &= \sum_{i=1}^{n} y_i \log\Big[\sigma\big(\mathbf{w}^T \mathbf{x}_i\big)\Big] + (1 - y_i) \log\Big[1 - \sigma\big(\mathbf{w}^T \mathbf{x}_i\big)\Big] \end{aligned}$$

Cross Entropy Loss

The negative log-likelihood function can be viewed as a loss function for the classifier. It is often called the cross entropy loss.

$$CE(\mathbf{w}, D) = \sum_{i=1}^{n} -y_i \log\Big[\sigma\Big(\mathbf{w}^T\mathbf{x}_i\Big)\Big] - (1-y_i)\log\Big[1-\sigma\Big(\mathbf{w}^T\mathbf{x}_i\Big)\Big]$$

The negative log-likelihood is a convex function. As a result of this convexity, the logistic regression model will have an optimal weight vector for any given dataset. This is independent of the data being linearly separable or not.

5.3. MLE: Computing the Gradient

Let us now turn to the gradient of the log-likelihood. We will make use of two facts:

- $\sigma'(z) = \sigma(z) \cdot [1 - \sigma(z)]$
- $\nabla\Big(\mathbf{w}^T\mathbf{x}\Big) = \mathbf{x}$

This derivation is all about a careful application of the chain-rule. We will split the sum into two parts and compute the gradient for each one of them.

$$
\begin{aligned}
\nabla l(\mathbf{w}) &= \sum_{i=1}^{n} y_i \cdot \frac{1}{\sigma\left(\mathbf{w}^T \mathbf{x}_i\right)} \cdot \sigma\left(\mathbf{w}^T \mathbf{x}_i\right)\left[1 - \sigma\left(\mathbf{w}^T \mathbf{x}_i\right)\right] \cdot \mathbf{x}_i \\
&\quad + \sum_{i=1}^{n} (1 - y_i) \cdot \frac{1}{1 - \sigma\left(\mathbf{w}^T x_i\right)} \cdot \left\{-\sigma\left(\mathbf{w}^T \mathbf{x}_i\right)\left[1 - \sigma\left(\mathbf{w}^T \mathbf{x}_i\right)\right]\right\} \cdot \mathbf{x}_i \\
&= \sum_{i=1}^{n} y_i \cdot \left[1 - \sigma\left(\mathbf{w}^T \mathbf{x}_i\right)\right] \cdot \mathbf{x}_i - \sum_{i=1}^{n} (1 - y_i) \cdot \sigma\left(\mathbf{w}^T \mathbf{x}_i\right) \cdot \mathbf{x}_i \\
&= \sum_{i=1}^{n} \left[y_i - \sigma\left(\mathbf{w}^T \mathbf{x}_i\right)\right] \mathbf{x}_i \\
&= \sum_{i=1}^{n} (y_i - \sigma_i) \mathbf{x}_i
\end{aligned}
$$

We can treat $y_i - \sigma_i$ as some kind of an error in prediction. For example, if $y_i = 1$ and the model's probability is $\sigma_i = 0.7$, then the error in prediction is 0.3. With this perspective, the gradient for a single data-point takes the following form:

$$\text{gradient} = \text{error} \times \text{data-point}$$

Larger the error, more is the contribution of a data-point towards the gradient. If the model is already doing a good job of classifying a data-point, then the error is going to be small, and such a point won't disturb the gradient much.

5.4. Gradient Ascent

Though we have computed the gradient of the log-likelihood, we can't obtain a closed-form expression for $\mathbf{w}^*$, the optimal weight vector, by setting the gradient to zero. As a result, we have to resort to numerical methods for optimization. One obvious choice is gradient ascent (not descent, as we are maximizing the log-likelihood):

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \eta \cdot \nabla l(\mathbf{w})|_{\mathbf{w}_t}$$

Plugging in the gradient, we get:

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \eta \cdot \mathbf{X}(\mathbf{y} - \boldsymbol{\sigma})$$

Since the log-likelihood is concave, gradient ascent will converge to the optimal weight vector for suitable choices of η. In practice, gradient ascent is **not** used to learn the weights of a logistic regression model. Gradient ascent is a first order method, meaning, it uses the first derivatives or gradients. There are a class of second order methods that make use of the second derivatives or the Hessian.