Logistic Regression Classifiers

Francesca Naretto
Computer Science Department

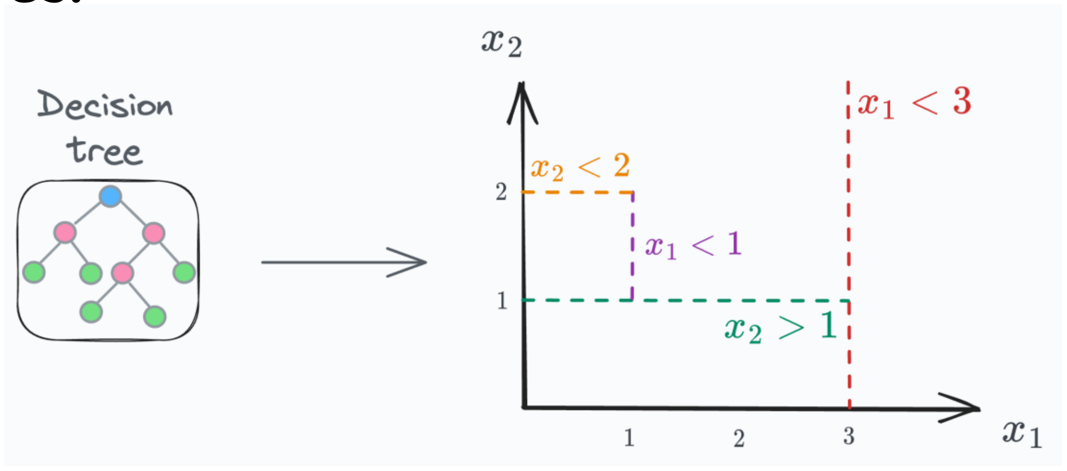
Introduction to Data Mining, 2nd Edition
Chapter 5.2

Logistic Regression Classifiers

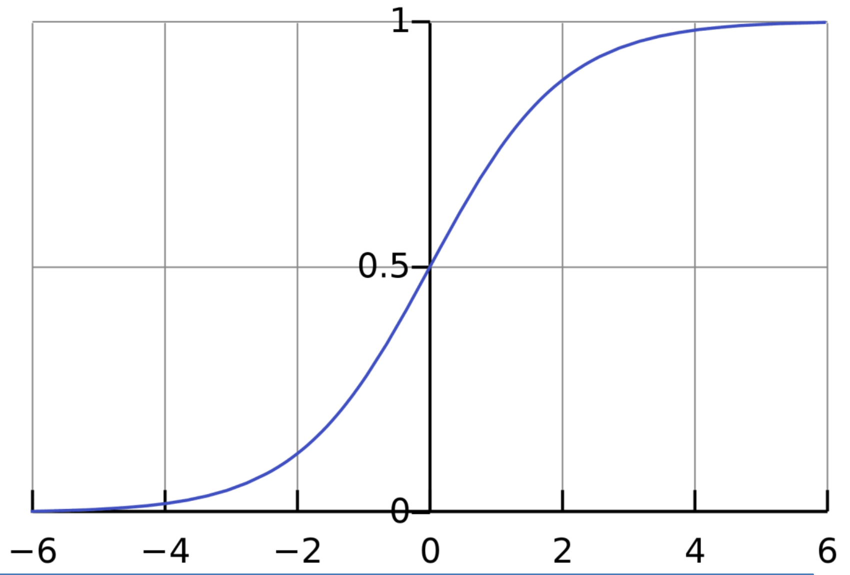
- A classifier, supervised learning
- A statistical approach and a Machine Learning algorithm, based on the concept of probabilities.
- It exploits a sigmoid function

Logistic Regression Classifiers

Decision tree boundaries:

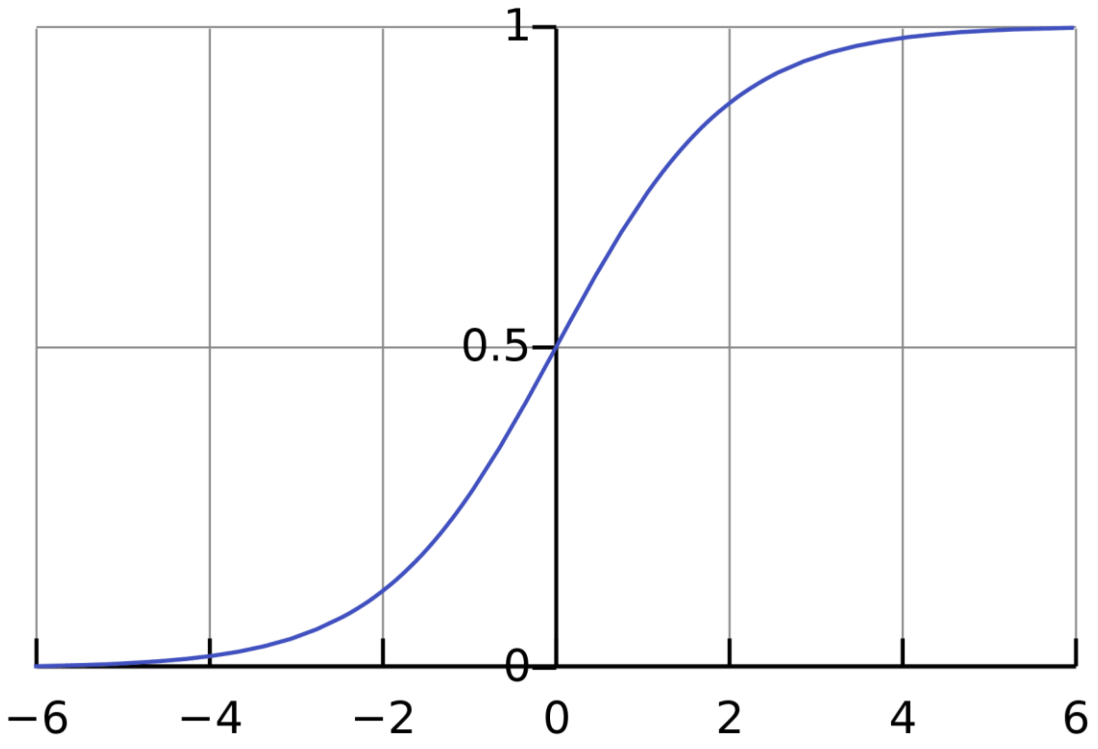


Logistic regression:



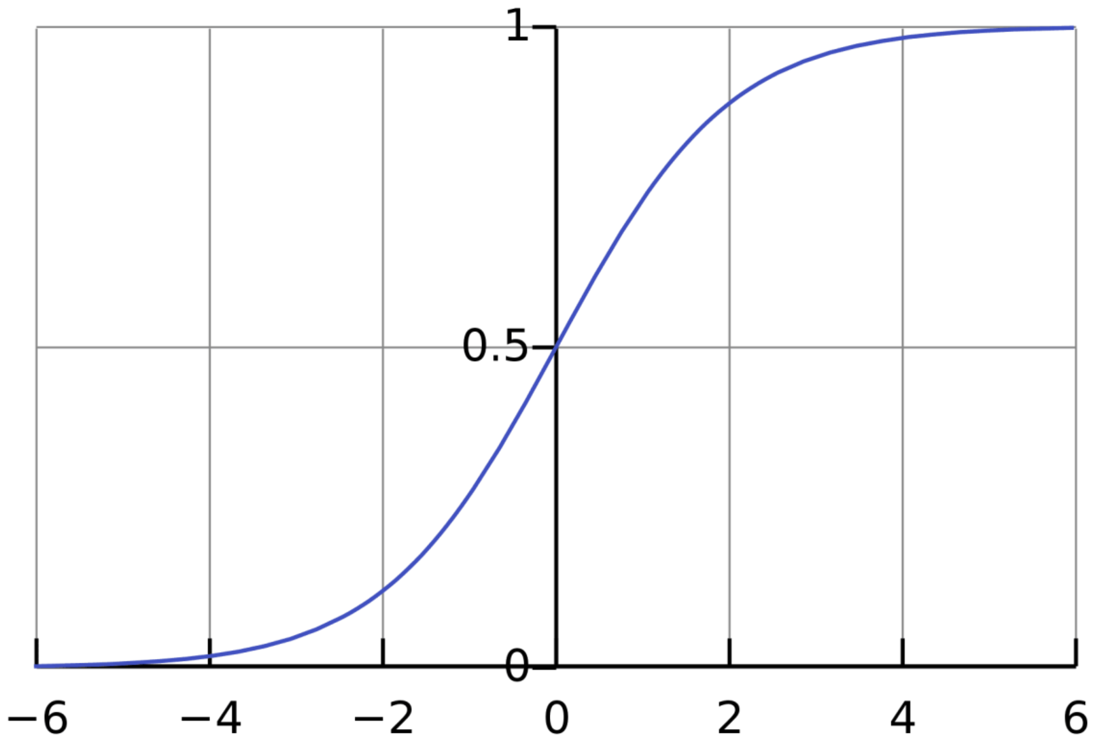
Logistic Regression Classifiers

Sigmoid Function is a mathematical function used to map the predicted values to probabilities. The function has the ability to map any real value into another value within a range of 0 and 1.



Logistic Regression Classifiers

Given a **binary** classification task and a record x to classy: we want to know the probability that the record x is a member of class 1. $P(y = 1|x)$



Logistic Regression Classifiers

Given a **binary** classification task and a record x to classy: we want to know the probability that the record x is a member of class 1. $P(y = 1|x)$

To solve this task, Logistic Regression learns weights and bias terms.

The record x is composed by a set of features, and the logistic regression models tries to find a weight for each of the features. The weight can be positive (towards the positive class) or negative (towards the negative class). The bias, instead, is a number that is added to the total.

Logistic Regression Classifiers

Let’s assume that the LG is already been trained.

To make a decision, we first need to:

$$z = \left(\sum_{i=1}^{n} w_i + x_i \right) + b$$

Where w are the weights associated with the variables, and b is the bias.

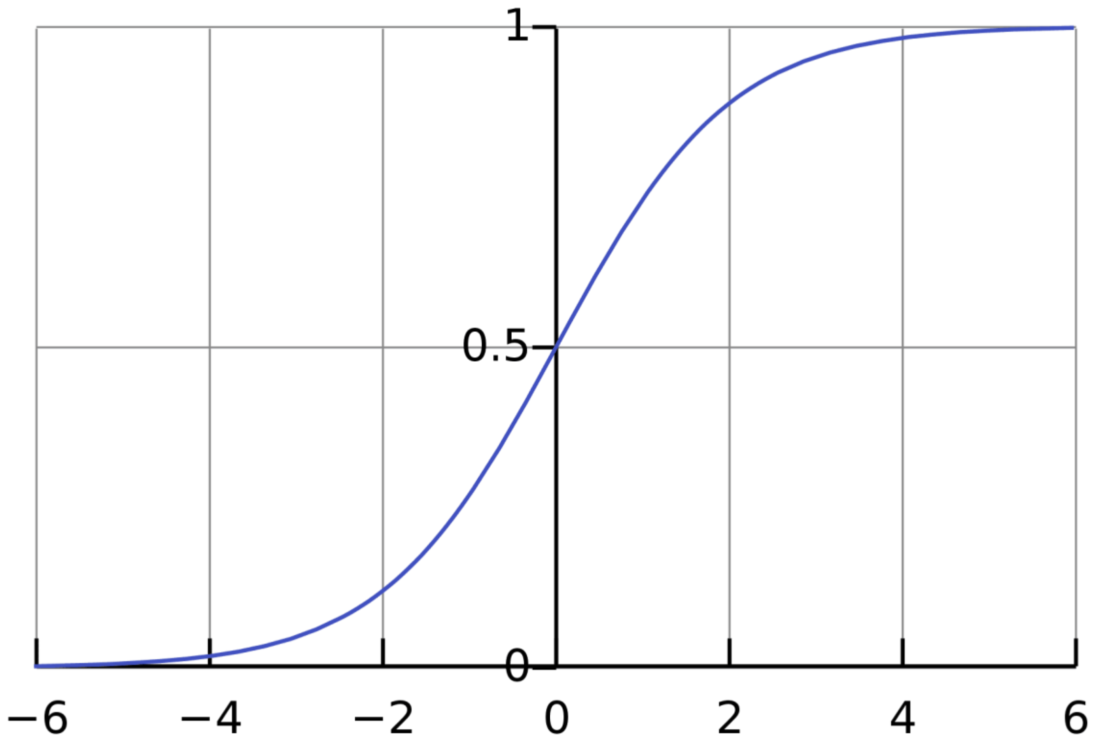
The value of z ranges from $(-\infty, +\infty)$ since the weights and bias are real values, hence they are not limited.

To obtain a probability, we pass z through the sigmoid function. Hence, we calculate $P(y = 1|x)$.

We obtain a number between 0 and 1.

Logistic Regression Classifiers

Logistic regression requires the concept of threshold. The threshold value defines the class 0 or 1.



Logistic Regression

The threshold is also called decision boundary, here it is at 0.5.

$$\text{decision}(x) = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Logistic Regression: training

For training, we need two components:

1. A cost function (or loss function) which evaluates the distance between the prediction and its correct label.
2. An optimization algorithm that iteratively updates the weights to minimize the loss function (minimize the distance)

Logistic Regression: cost function

Our objective is to learn the weights that maximize the probability of the correct label $p(y|x), y \in [0,1]$.

Since we can only have 2 outcomes, this can be represented by a Bernoulli distribution.

$$p(y|x) = \bar{\bar{y}}^{y}(1-\bar{\bar{y}})^{1-y}$$

Let's apply the log:

$$\log[p(y|x)] = \log[\bar{\bar{y}}^{y}(1-\bar{\bar{y}})^{1-y}]$$
$$ylog\bar{\bar{y}} + (1-y)\log(1-\bar{\bar{y}})$$

We obtain the log likelihood that we want to maximize.

For the cost function, we need something to minimize, hence we flip the sign:

$$L_{CE}(\bar{\bar{y}}, y) = -\log\big(p(y|x)\big) = -[ylog\bar{\bar{y}} + (1-y)\log(1-\bar{\bar{y}})]$$

Logistic Regression: minimum

Our goal is to:

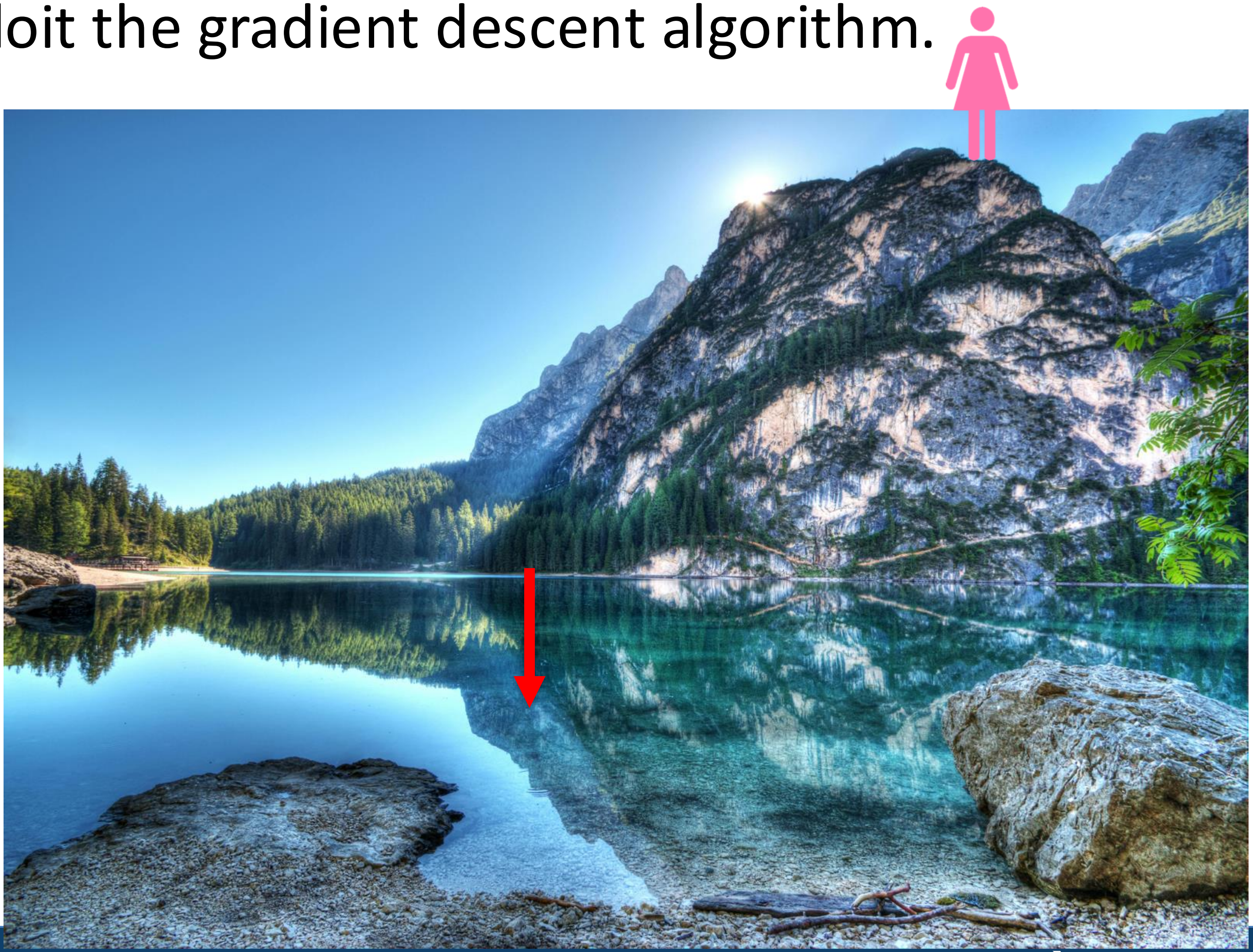
$$\theta = argmin_{\theta}(\frac{1}{m}\sum_{i=1}^{m} L_{CE}(f\left(x^{i};\theta\right), y^{i}))$$

Which, put simply, means finding the minimum of our cost function.

How to find the minimum?

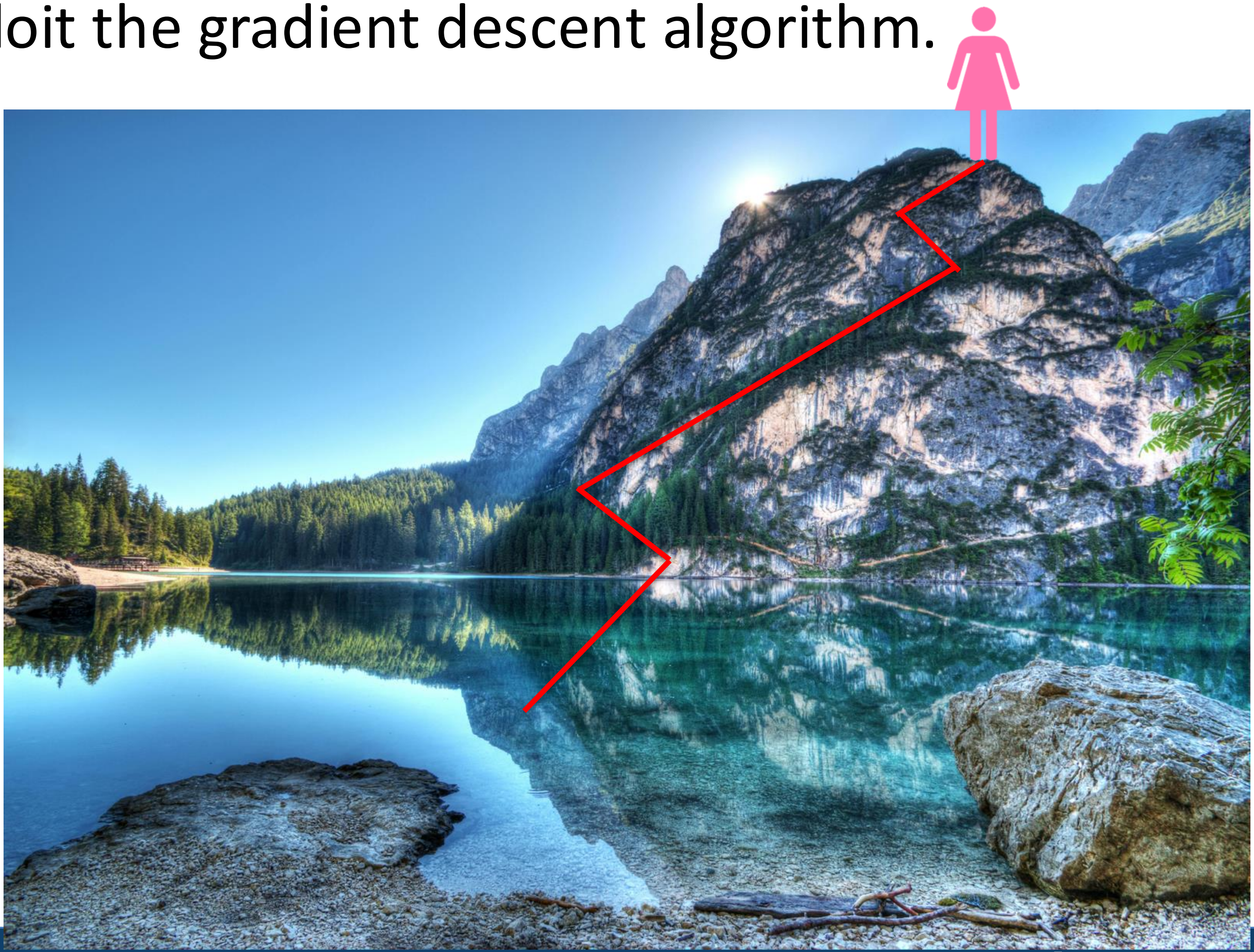
Logistic Regression: gradient descent

To find the minimum of the cost function, we can exploit the gradient descent algorithm.



Logistic Regression: gradient descent

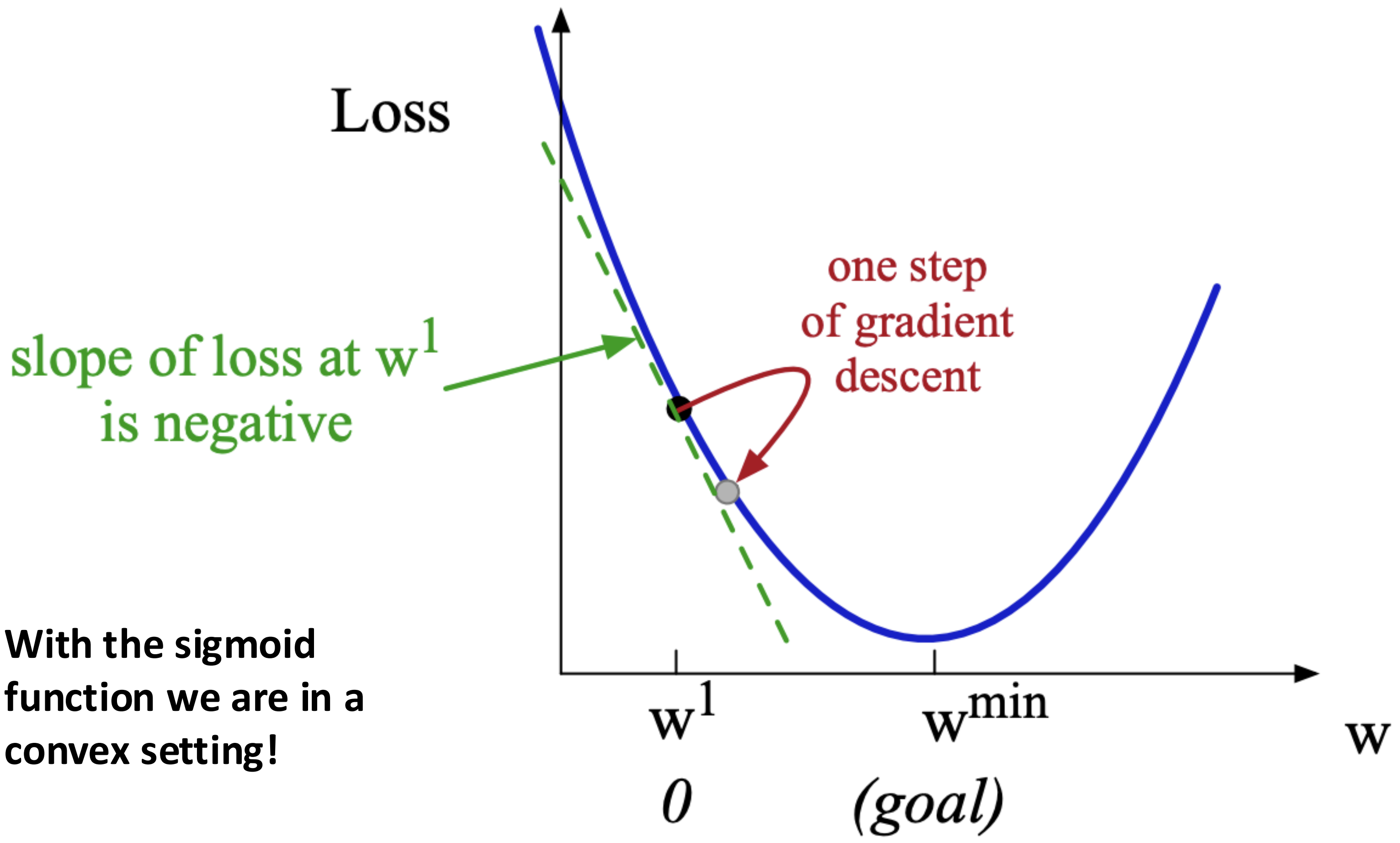
To find the minimum of the cost function, we can exploit the gradient descent algorithm.



Logistic Regression: gradient descent

1. We find the gradient of the cost function of the point in which we are
2. We move in the opposite direction of the gradient
3. We do this until we reach the minimum**

Logistic Regression: gradient descent


Loss
slope of loss at w^1 is negative
one step of gradient descent
w^1
w^{min}
w
0
(goal)

With the sigmoid function we are in a convex setting!

Logistic Regression: gradient descent

The gradient descent gives us the direction of the minimum, but how to understand how big the step should be?

Learning rate

1. Big steps: you find faster the minimum, if you only have one minimum (what about the local minimum?)
2. Small steps: slow learner, but you don’t miss anything

Logistic Regression: gradient descent

How to compute the gradient descent?

$$w = w - \alpha\left(\frac{\partial L_{CE}}{\partial w_j}\right)$$

$$\begin{aligned}
\frac{\partial L_{\text{CE}}}{\partial w_j} &= \frac{\partial}{\partial w_j} - [y \log \sigma(\mathbf{w}\cdot\mathbf{x}+b) + (1-y)\log(1-\sigma(\mathbf{w}\cdot\mathbf{x}+b))] \\
&= -\left[\frac{\partial}{\partial w_j} y \log \sigma(\mathbf{w}\cdot\mathbf{x}+b) + \frac{\partial}{\partial w_j}(1-y)\log[1-\sigma(\mathbf{w}\cdot\mathbf{x}+b)]\right]
\end{aligned}$$

Logistic Regression: gradient descent

How to compute the gradient descent?

$$w = w - \alpha \left(\frac{\partial L_{CE}}{\partial w_j} \right)$$

Applying the chain rule and the derivative of the log:

$$\frac{\partial L_{\text{CE}}}{\partial w_j} = -\frac{y}{\sigma(\mathbf{w}\cdot\mathbf{x}+b)}\frac{\partial}{\partial w_j}\sigma(\mathbf{w}\cdot\mathbf{x}+b) - \frac{1-y}{1-\sigma(\mathbf{w}\cdot\mathbf{x}+b)}\frac{\partial}{\partial w_j}1-\sigma(\mathbf{w}\cdot\mathbf{x}+b)$$

$$\frac{\partial L_{\text{CE}}}{\partial w_j} = -\left[\frac{y}{\sigma(\mathbf{w}\cdot\mathbf{x}+b)} - \frac{1-y}{1-\sigma(\mathbf{w}\cdot\mathbf{x}+b)}\right]\frac{\partial}{\partial w_j}\sigma(\mathbf{w}\cdot\mathbf{x}+b)$$

Logistic Regression: gradient descent

How to compute the gradient descent?

$$w = w - \alpha\left(\frac{\partial L_{CE}}{\partial w_j}\right)$$

Applying the chain rule and the derivative of the sigmoid:

$$\begin{aligned}
\frac{\partial L_{\mathrm{CE}}}{\partial w_j} &= -\left[\frac{y-\sigma(\mathbf{w}\cdot\mathbf{x}+b)}{\sigma(\mathbf{w}\cdot\mathbf{x}+b)[1-\sigma(\mathbf{w}\cdot\mathbf{x}+b)]}\right]\sigma(\mathbf{w}\cdot\mathbf{x}+b)[1-\sigma(\mathbf{w}\cdot\mathbf{x}+b)]\frac{\partial(\mathbf{w}\cdot\mathbf{x}+b)}{\partial w_j} \\
&= -\left[\frac{y-\sigma(\mathbf{w}\cdot\mathbf{x}+b)}{\sigma(\mathbf{w}\cdot\mathbf{x}+b)[1-\sigma(\mathbf{w}\cdot\mathbf{x}+b)]}\right]\sigma(\mathbf{w}\cdot\mathbf{x}+b)[1-\sigma(\mathbf{w}\cdot\mathbf{x}+b)]x_j \\
&= -[y-\sigma(\mathbf{w}\cdot\mathbf{x}+b)]x_j \\
&= [\sigma(\mathbf{w}\cdot\mathbf{x}+b)-y]x_j
\end{aligned}$$

Stochastic gradient descent

It is an online algorithm that minimizes the loss function by computing the gradient after each training example.

Batch gradient descent

Since computing the gradient for each record may results in strange movements, not always ‘correct’.

A solution may be to update the gradient after all the training records, or after small parts of it.

Underfitting, overfitting

As for the decision trees, we want a model able to generalize well (hence, avoid under/overfitting).

In this case, to avoid overfitting, we can introduce a **regularization** term to the loss function, able to penalize large weights.

If a set of weights fits perfectly the training, they are going be high. Hence, the regularization term penalizes this setting.

Regularization

L2 regularization (Ridge regression) is a quadratic function of the weight values:

$$R(\theta) = ||\theta||^2 = \sum_{i=1}^{n} \theta_i^2$$

L1 (Lasso Regression) is a linear function of the weight values (Manhattan distance):

$$R(\theta) = ||\theta|| = \sum_{i=1}^{n} |\theta_i|$$

Regularization

The regularization term is added to the loss function, hence for the gradient descent we need to compute its gradient (its derivative).

1. L2 is easier to optimize since its derivative is simple
2. L1 has a more complex derivative
3. L2 tends to favor solutions with smaller weights for all of the features
4. L1 tends to favor solutions sparse, where there are few high weights and a lot of smaller/zero ones

What about more than 2 classes?

In this case we talk about multinomial logistic regression. Our loss function:

$$L_{CE}(\bar{\bar{y}}, y) = -[y log g\bar{\bar{y}} + (1-y)\log(1-\bar{\bar{y}})]$$

Is limited to 2 classes. To extend it for multiple classes, we need to consider vectors:

True label vector:

.0	.0	.0	1	.0

Prediction probabilities vector:

0.2
0.6
0.1
0.1

What about more than 2 classes?

What about the novel version of the loss?

$$L_{CE}(\overline{y}, y) = -\sum_{k=1}^{K} y_k \log(\overline{y}_k)$$

$$L_{CE(\overline{y},y)} = -\log(\overline{y}_c)$$

Where c is the correct class (only one at a time).

$$L_{CE(\overline{y},y)} = -\log(\overline{p})(y_c = 1|x)$$