TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG
Die Ressourcenuniversität. Seit 1765.

Prof. Dr. Björn Sprungk
Faculty of Mathematics and Computer Science
Institute of Stochastics

Mathematics of machine learning

2. Linear predictors

Winter term 2024/25

Chapter 2: Linear predictors

Contents

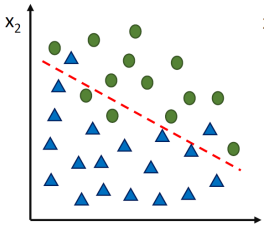
Chapter 2: Linear predictors

What it's about?

1. Get to know the basic representatives of linear classification methods:
   - the Perceptron algorithm (1958) – the first appearance of (simplest) neural networks for classification,
   - logistic regression (1940s) – one of the most widely used classification techniques for data analysis (according to a survey conducted by the KAGGLE platform in 2017),
2. Understand the corresponding approaches, assumptions and advantages.
3. Get to know about the building blocks and structure of neural networks (analysis comes later!)

Linear hypotheses

- Linear predictors (or hypotheses) form an important class for the following reasons:
  - They are often used in applications and often describe the data in a good way.
  - They are usually efficiently computable and learnable.
  - They are easy to interpret and accessible to intuition.
- Linear hypotheses require a feature space with inner product.



Lineare Klassifikation

Source: data-science-blog

- Therefore, in this chapter let $\mathcal{X} \subseteq \mathbb{R}^d$ equipped with the Euclidean inner product

$$\mathbf{w} \cdot \mathbf{x} := \mathbf{w}^\top \mathbf{x} \in \mathbb{R}, \qquad \mathbf{w}, \mathbf{x} \in \mathbb{R}^d.$$

- As label space we choose here $\mathcal{Y} = \{-1, 1\}$ for convenience.

Definition 2.1: Linear hypotheses

We define the set of all affine-linear functions $f\colon \mathbb{R}^d \to \mathbb{R}$ by

$$\mathrm{L}_d := \left\{ f_{\mathbf{w},b}(\mathbf{x}) := \mathbf{w} \cdot \mathbf{x} + b \mid \mathbf{w} \in \mathbb{R}^d,\ b \in \mathbb{R} \right\}$$

where we call $\mathbf{w}$ the weight vector and b the bias of the function $f_{\mathbf{w},b}$.

A hypothesis $h\colon \mathcal{X} \to \mathcal{Y}$, $\mathcal{X} \subseteq \mathbb{R}^d$, is a linear hypothesis if there exists an affine-linear function $f_{\mathbf{w},b} \in \mathrm{L}_d$ and an activation function $\phi\colon \mathbb{R} \to \mathcal{Y}$ such that

$$h(\mathbf{x}) = \phi \circ f_{\mathbf{w},b}(\mathbf{x}) = \phi\left(\mathbf{w} \cdot \mathbf{x} + b\right), \qquad \mathbf{x} \in \mathcal{X}.$$

We denote the set of all linear hypotheses with activation function ϕ by

$$\mathcal{L}_{d,\phi} := \left\{ h_{\mathbf{w},b}(\mathbf{x}) := \phi(\mathbf{w} \cdot \mathbf{x} + b) \mid \mathbf{w} \in \mathbb{R}^d,\ b \in \mathbb{R} \right\}$$

For the sign activation function function $\phi = \mathrm{sgn}$, we simply write $\mathcal{L}_d$.

Halfspaces

A linear hypothesis

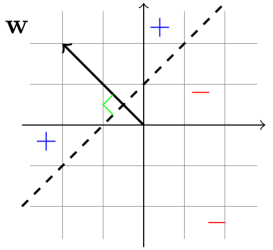
$$h_{\mathbf{w},b}(\mathbf{x}) = \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x} + b), \qquad \mathbf{x} \in \mathcal{X},$$

divides the features space $\mathcal{X} = \mathbb{R}^d$ in **two halfspaces**

$$H^+_{\mathbf{w},b} := \{\mathbf{x} \in \mathbb{R}^d : \mathbf{w} \cdot \mathbf{x} + b \geq 0\},$$
$$H^-_{\mathbf{w},b} := \{\mathbf{x} \in \mathbb{R}^d : \mathbf{w} \cdot \mathbf{x} + b < 0\},$$

by a separating hyperplane

$$H_{\mathbf{w},b} := \{\mathbf{x} \in \mathbb{R}^d : \mathbf{w} \cdot \mathbf{x} + b = 0\}.$$

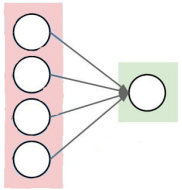


Source: "Understanding Machine Learning" (2014)

The **interpretability of the linear hypothesis is provided by the halfspaces and we can identify each $h_{\mathbf{w},b}$ with the halfspaces $H^+_{\mathbf{w},b}$, $H^-_{\mathbf{w},b}$.**

2.1 The Perceptron algorithm

- In 1943 Warren McCulloch and Walter Pitts developed a computational model for the activity of (artificial) neurons which still forms the basis of modern neural networks

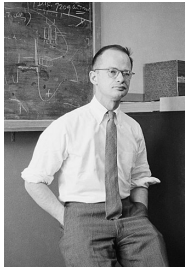


- If the weighted sum of the input signals x_i exceeds the bias b, then the neuron "fires" and outputs the signal $y = 1$, otherwise it does not, i.e. $y = 0$

$$y := \begin{cases} 1, & \text{if } \sum_{i=1}^{d} w_i \, x_i + b \geq 0, \\ 0, & \text{else} \end{cases}$$

- We now know, that is a linear hypothesis with $\phi = \mathbf{1}_{[0,\infty)}$

- Based on their idea Frank Rosenblatt developed and built the Mark I Perceptron in 1958. The first machine which learned to distinguish images.



Walter Pitts
(1923-1969)

The perceptron uses a step function to determine the output. If the weighted sum exceeds a certain threshold, it outputs one class (e.g., 1); otherwise, it outputs another class (e.g., 0)

Setting

Task

- **Given:** Sample s with m data pairs $(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \{-1, +1\}$.
- **Goal:** Calculate weight vector $\mathbf{w}_s \in \mathbb{R}^d$ and bias $b_s \in \mathbb{R}$ such that

$$h_{\mathbf{w}_s, b_s}(\mathbf{x}) = \operatorname{sgn}(\mathbf{w}_s \cdot \mathbf{x} + b_s), \qquad \mathbf{x} \in \mathbb{R}^d,$$

minimizes the empirical risk w.r.t. the 0-1 loss:

$$h_{\mathbf{w}_s, b_s} = \operatorname*{argmin}_{h_{\mathbf{w},b} \in \mathcal{L}_d} \mathcal{R}_s(h_{\mathbf{w},b}) = \operatorname*{argmin}_{h_{\mathbf{w},b} \in \mathcal{L}_d} \frac{1}{2m} \sum_{i=1}^{m} |h_{\mathbf{w},b}(\mathbf{x}_i) - y_i|.$$

Weights determine the importance of each input feature. A higher weight means that the corresponding input feature has a greater influence on the output

Note that since $y_i, h_{\mathbf{w},b}(\mathbf{x}_i) \in \{-1, +1\}$, the following holds true for the 0-1 loss ℓ:

$$\frac{1}{2} |h_{\mathbf{w},b}(\mathbf{x}_i) - y_i| = \ell\left(h_{\mathbf{w},b}(\mathbf{x}_i), y_i\right) = \begin{cases} 0, & h_{\mathbf{w},b}(\mathbf{x}_i) = y_i \\ 1, & h_{\mathbf{w},b}(\mathbf{x}_i) \neq y_i. \end{cases}$$

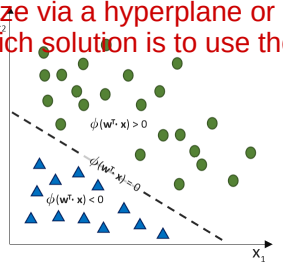
Prerequisites

therefore we have to cases in order to apply the ERM:
1. if the data is linearly SEPRABLE, it could divide the and categorize via a hyperplane or a line.
2. if the data is linearly INSEPRABLE, it could get to overfitting (which solution is to use the ReLU)

Assumption

The sample s is linearly separable, i. e., there exists a $\mathbf{w}_s$ and b_s with

$$\mathcal{R}_s(h_{\mathbf{w}_s,b_s}) = 0$$



This assumption allows rewriting the ERM rule to

$$\text{Find} \quad (\mathbf{w}_s, b_s) \in \mathbb{R}^{d+1} \quad \text{s. t.} \qquad y_i \ (\mathbf{w}_s \cdot \mathbf{x}_i + b_s) > 0 \quad \forall i = 1, \ldots, m.$$

Because $\mathcal{R}_s(h_{\mathbf{w}_s,b_s}) = 0$ means that for all i

$$h_{\mathbf{w},b}(\mathbf{x}_i) = \operatorname{sgn}(\mathbf{w}_s \cdot \mathbf{x}_i + b_s) = y_i$$

and thus y_i and $\mathbf{w}_s \cdot \mathbf{x}_i + b_s$ always have the same sign, i.e.,

$$y_i \ (\mathbf{w}_s \cdot \mathbf{x}_i + b_s) > 0.$$

The perceptron only works well with linearly separable data. If the classes cannot be separated by a straight line (or hyperplane in higher dimensions), it will not converge.
for convergence the Novikoff theorm is required

The Perceptron Algorithm

The following algorithm is an iterative method for finding

$$(\mathbf{w}_s, b_s) \in \mathbb{R}^{d+1} \qquad \text{s. t.} \qquad y_i \ (\mathbf{w}_s \cdot \mathbf{x}_i + b_s) > 0 \quad \forall i = 1, \dots, m.$$

To make calculations simpler, we add a "1" to each data point xi, making it augmented point xí = (xi, 1) which allows us handle bias.

Perceptron Algorithm (F. Rosenblatt, 1958)

- **Input:** Sample s with m data points $(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \{-1, +1\}$.
- **Start:** Set $t = 0$ and $\mathbf{w}'_t = (\mathbf{w}_t, b_t) = \mathbf{0}$.
- **Iteration:** For $t = 0, 1, 2, \dots,$ do
  - If there exists a $i = 1, \dots, m$ with

$$y_i \ (\mathbf{w}'_t \cdot \mathbf{x}'_i) \leq 0,$$

checking for misclassified points

$$\mathbf{x}'_i := (\mathbf{x}_i, 1),$$

this includes a bias term

  - then set

$$\mathbf{w}'_{t+1} := \mathbf{w}'_t + y_i \mathbf{x}'_i,$$

the weight updator formula

  - else stop and output $\mathbf{w}'_t$.

Now the algorithm goes through iteration and each data to check if it is misclassified(if the line is on the wrong side which has been declared by weights) if the results are less or equal that 0. then it is misclassified.
in this case we shall update the weights to correct the model by shifting the line closer to the classifying

The perceptron algorithm suits for linearly separable data. In this manner, we do have some steps and iterations to make the data labels of more converge and separated via a hyper plane. Note that the iterations are finite in this point. in this matter, we would have an upper bound on number of mistakes and updates for each iterations. (if the data are misclassified which means that the separating hyperplane is on the wrong side, making the weight vectors wrongly configured, then we have and update formula for the iteration to fix the weight vectors to implement better.) then we would have a finitely number of iterations.

if the data was non-linearly separable (like Logistic regression, Soft SVM) it would not work properly because there would be infinite iterations which may lead to over-fitting due to updating lot.

The T is the update times iterations for the upper bound of changes or mistakes.

we have 2 variables as R and B.

1. R means that how spread out the data are. we need this as max.
2. B means that how well separated classes are. it is a distance of the points from hyperplane margin of that. the larger the B the better which means . but we need the min of this so we could determine that the closes point to the hyperplane, is how much far of it which makes the scenario worse.
3. A large B means the perceptron can find a solution because the classes are far apart. But a large B also means more updates are needed in this case Perceptron works converges well.
4. A small B means fewer updates are needed, so the perceptron learns faster. But if B is too small, the perceptron might struggle to find a separating boundary.

   SO, WE NEED A SUFICIENT LARGE B AND ALSO SUFFICENT SMALL BE TO BOTH WORK WELL AND FAST TO UPDATE THE ITERATIONS.

Convergence of the Perceptron algorithm

If the points are spread out and well-separated, it's easier for the Perceptron to find a solution. This is because there is more room to find a decision boundary that clearly separates the classes.

R tells you how far the farthest data point is from the origin (size of the data).
B tells you how thick the separating line can be (how well-separated the classes are).

- The recursive rule

$$\mathbf{w}'_{t+1} := \mathbf{w}'_t + y_i \mathbf{x}'_i \qquad \text{for } i \text{ with} \qquad y_i \left(\mathbf{w}'_t \cdot \mathbf{x}'_i\right) \leq 0$$

in the formla we have a min of weights, B is the smallest (minimum) length of the weight vector that can separate all data points correctly while ensuring they meet the classification condition.

points the vector $\mathbf{w}'_{t+1}$ in the right direction, since

$$y_i \left(\mathbf{w}'_{t+1} \cdot \mathbf{x}'_i\right) \;=\; y_i \left(\mathbf{w}'_t \cdot \mathbf{x}'_i\right) \;+\; y_i^2 \left(\mathbf{x}'_i \cdot \mathbf{x}'_i\right) \;>\; y_i \left(\mathbf{w}'_t \cdot \mathbf{x}'_i\right).$$

Their multiplication gives an idea of how difficult it will be for the Perceptron to find a separating line. The closer the points are to each other, the harder it is to find a solution, resulting in more iterations of the algorithm.

- In particular, the algorithm converges in finitely many steps:

Theorem 2.2: (Novikoff, 1962)

Let the sample $s = ((\mathbf{x}_i, y_i))_{i=1,\ldots,m}$ be linearly separable. Then the Perceptron algorithm stops after $T \leq R^2 B^2$ steps with a solution $\mathbf{w}'_T$ which satisfies $y_i \left(\mathbf{w}'_T \cdot \mathbf{x}'_i\right) > 0$ for all i. Here

$$R := \max_{i=1,\ldots,m} \|\mathbf{x}'_i\|, \qquad B := \min\{\|\mathbf{w}'\| \colon y_i(\mathbf{w}' \cdot \mathbf{x}'_i) \geq 1 \quad \forall i = 1, \ldots, m\}.$$

When the points of different classes are close to each other (which makes B smaller), it becomes harder to separate them. This leads to more mistakes by the Perceptron algorithm.

Remarks

- The bound B^2R^2 is sharp and can become arbitrarily large: If one tries to separate in $\mathcal{X} = \mathbb{R}^2$ the points $\mathbf{x}_1 = (0, 1)$ with $y_1 = -1$ and $\mathbf{x}_2 = (1/n, 1)$, $n \in \mathbb{N}$, with $y_2 = +1$, this leads to $B^2R^2 \geq 4n^2 \to \infty$ for $n \to \infty$.

- I.e., the closer points with different labels, the more steps the Perceptron algorithm needs to find a solution

- **Reason:** The set of all weight vectors $\mathbf{w}_s$ and biases b_s with $\mathcal{R}_s(h_{\mathbf{w}_s, b_s}) = 0$ forms a cone in the parameter space $\mathbb{R}^{d+1}$, since

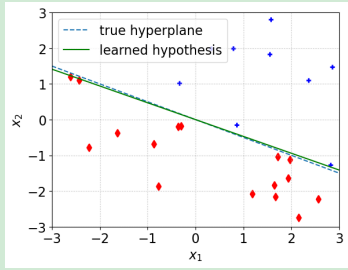
$$y_i \ (\mathbf{w}_s \cdot \mathbf{x}_i + b_s) > 0 \qquad \Longleftrightarrow \qquad y_i \ (\lambda \mathbf{w}_S \cdot \mathbf{x}_i + \lambda b_S) > 0$$

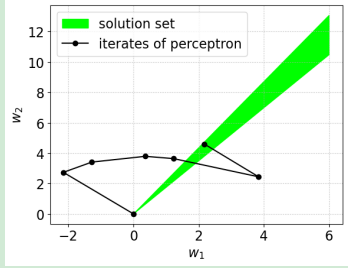
  for all $\lambda > 0$, hence we have $\mathcal{R}_s(h_{\lambda \mathbf{w}_s, \lambda b_s}) = 0$.

- The closer points of different labels, the less room there is for separating hyperplanes and the narrower cone of solutions is (corresponds to large value of B).

Example: Synthetic dataset

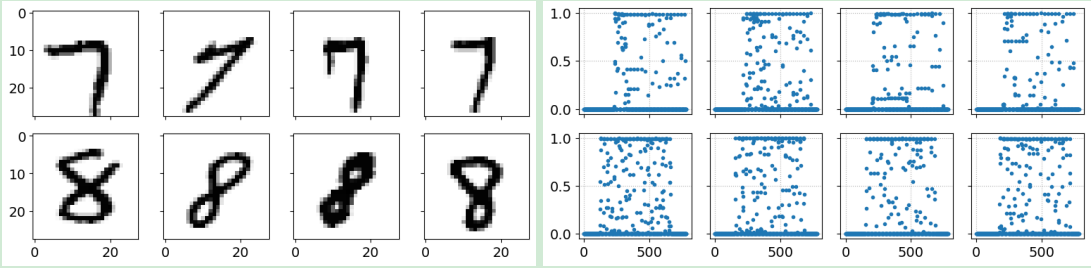
- In $\mathcal{X} = \mathbb{R}^2$ we want to learn a separating hyperplane between the training data.
- We assume that the hyperplane passes through the origin, so $b = 0$ and only $\mathbf{w} = (w_1, w_2)$ is to be learned.
- We generate $m = 25$ training data using $\mathbf{X} \sim \mathrm{U}[-3, 3]^2$ and a true separating hyperplane with $\mathbf{w}^\dagger = (1, 2)^\top$ and $b^\dagger = 0$.
- Our perceptron algorithm randomly selects one i with $y_i(\mathbf{w}'_t \cdot \mathbf{x}'_i) \leq 0$ per step and requires a total of $T = 6$ steps (here $R^2 B^2 \approx 938.15$).
- To reproduce the example, a Jupyter notebook is provided – you just need to implement the Perceptron algorithm (**Exercise**).





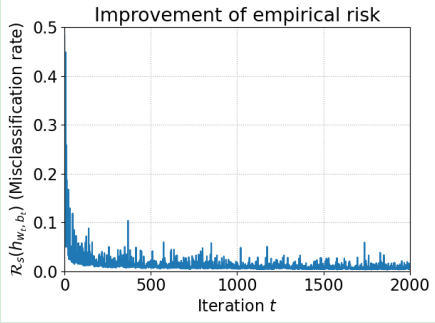
Example: MNIST data set

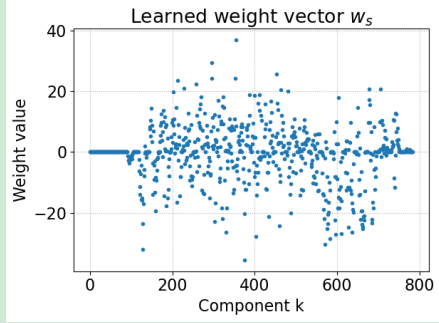
- Recall the MNIST dataset of 70,000 grayscale images with handwritten digits
- Each of these images consists of 28x28 grayscale pixels and can be stored as a vector $\mathbf{x} \in [0,1]^{784}$
- We want to use the 6265 as "7" labelled images and 5851 as "8" labelled images (i.e., $m = 12116$) to learn to distinguish between the number 7 and the number 8:

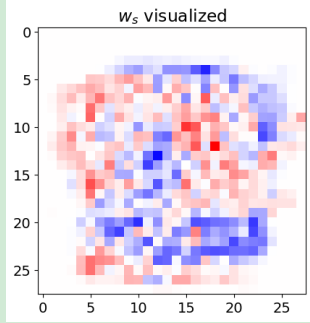


- Thus, we want to find a separating hyperplane between the images with label 7 and 8 in the space of 28x28 grayscale images $\mathcal{X} = [0,1]^{784}$. For this we use the perceptron algorithm!

- We run the perceptron algorithm for $T = 2000$ iterations and obtain:



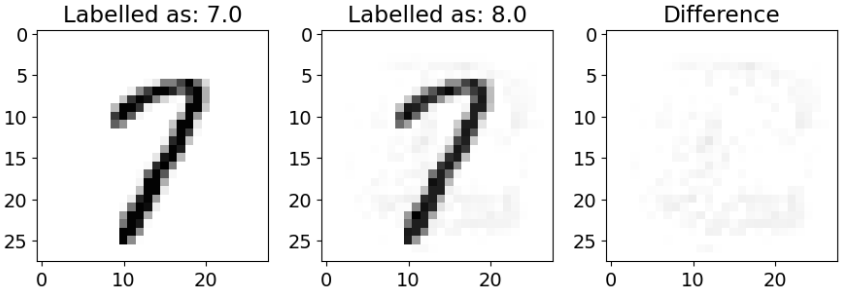




- The learned linear hyothesis $h_{\mathbf{w}_s,b_s}$ or separating hyperplane $H_{\mathbf{w}_s,b_s}$ assigns the correct label "7" or "8" to 99.6% of the training data
- The learned bias is $b_s = 82$ and the learned weight vector $\mathbf{w}_s \in \mathbb{R}^{784}$ is shown in the mid and left panel.

Fooling the Perceptron

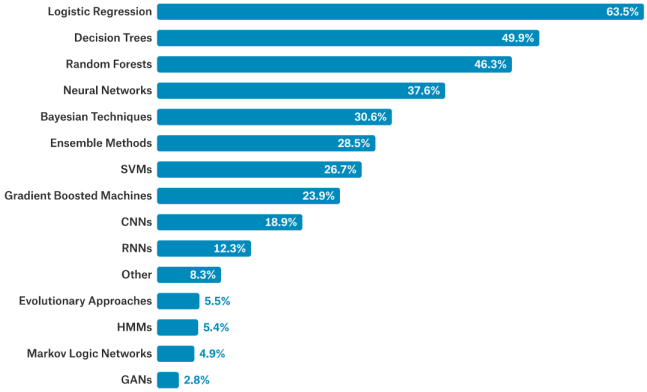
- We can easily interpret the learned hypothesis $h_{\mathbf{w}_s, b_s}$.
- E.g., the zero values of $\mathbf{w}_s$ in the first 93 components tell us that the pixel values at the left edge have no influence on the classification (they are all always white).
- Furthermore, we can specifically fool the learned classifier $h_{\mathbf{w}_s, b_s}$:



- You will learn how to calculate such a perturbation in the **Exercise**.

3.2 Logistic Regression

- This is a classical method from statistics, dating back to the 1940's
- However, it is still a very common tool in data science and machine learning
- In 2017 the data science platform Kaggle conducted a industry-wide survey on the state of data science and machine learning asking more than 16,000 data scientists which methods they use at work (among other things):



Setting

- We take the classical statistical approach to logistic regression and later relate it to our ERM setting
- Instead of predicting a label -1 or $+1$ with $h(\mathbf{x})$, logistic regression deals with hypotheses that predict the probability for $Y = \pm 1$:

$$h(\mathbf{x}) \approx \mathbb{P}(\text{object with feature } \mathbf{x} \in \mathbb{R}^d \text{ has label } +1).$$

- More formally:

$$h(\mathbf{x}) \approx \mathbb{P}_\mu(Y = +1 \mid X = \mathbf{x}).$$

- Again, we consider linear hypotheses of the form

$$\mathbb{P}(Y = +1 \mid X = \mathbf{x}) = \phi(\mathbf{w} \cdot \mathbf{x} + b) = h_{\mathbf{w},b}.$$

- **Thus,** we need now activation functions $\phi \colon \mathbb{R} \to [0, 1]$ taking values for probabilities!

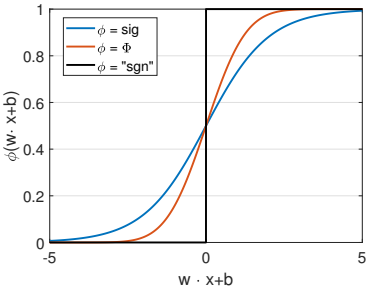
Activation functions

Examples for ϕ:

1. $\phi = \Phi$: Standard normal cumulative distribution function (Probit regression)

2. $\phi = \mathrm{sig}$: Logistic function (Logit regression)

$$\mathrm{sig}(t) := \frac{1}{1+\exp(-t)}, \qquad t \in \mathbb{R}.$$



- The logistic function is also called sigmoid function because of its sigmoidal or S-shaped graph.
- It can also be understood as a smooth approxiamation to the indicator function $\mathbb{1}_{[0,\infty)}$
- In particular, $\lim_{t\to-\infty} \mathrm{sig}(t) = 0$ and $\lim_{t\to+\infty} \mathrm{sig}(t) = 1$.

The Likelihood

- For a fixed hypothesis we obtain by using the logistic activation function that

$$\mathbb{P}(Y = +1 \mid X = \mathbf{x}) = h_{\mathbf{w},b}(\mathbf{x}) = \operatorname{sig}(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + \exp(-\mathbf{w} \cdot \mathbf{x} - b)}.$$

In logistic regression, likelihood helps us find the best weights for predictions.

- Due to

$$1 - \frac{1}{1 + \mathrm{e}^{-t}} = \frac{1 + \mathrm{e}^{-t} - 1}{1 + \mathrm{e}^{-t}} = \frac{\mathrm{e}^{-t}}{1 + \mathrm{e}^{-t}} = \frac{1}{1 + \mathrm{e}^{t}}$$

this is a mapper, maps the linear combination of w.x + b into a range of 0 and 1 probability

we get

$$\mathbb{P}(Y = -1 \mid X = \mathbf{x}) = 1 - h_{\mathbf{w},b}(\mathbf{x}) = \frac{1}{1 + \exp(\mathbf{w} \cdot \mathbf{x} + b)}$$

- Summarizing, for the hypothesis $h_{\mathbf{w},b}(\mathbf{x}) = \operatorname{sig}(\mathbf{w} \cdot \mathbf{x} + b)$ we thus get as predicted probabilities

$$\mathbb{P}_{\mathbf{w},b}(Y = y \mid X = \mathbf{x}) := \frac{1}{1 + \exp\left(-y\ [\mathbf{w} \cdot \mathbf{x} + b]\right)}$$

Maximum Likelihood Estimation of $\mathbf{w}$ and b

is about finding the values of the parameters (weights w and bias b) that make our model's predictions match the observed data as closely as possible. In simpler terms, it's a way of "tuning" the model so it's most likely to have produced the actual labels

- **Given:** Sample s with m data pairs $(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \{-1, +1\}$
- **Goal:** Find weight vector $\mathbf{w}_s$ and bias b_s that best explain the data.
- **In other words:** Find the pair $(\mathbf{w}_s, b_s)$ which yields the largest likelihood for observing the data $(\mathbf{x}_i, y_i)$ (maximum likelihood principle).
- **Assuming:** Data pairs $(\mathbf{x}_i, y_i)$ are drawn independently which leads to

$$L_{\mathbf{w},b}(s) := \mathbb{P}_{\mathbf{w},b}(Y_1 = y_1, \ldots, Y_m = y_m \mid X_1 = \mathbf{x}_1, \ldots, X_m = \mathbf{x}_m)$$
$$= \prod_{i=1}^{m} \mathbb{P}_{\mathbf{w},b}(Y = y_i \mid X = \mathbf{x}_i) = \prod_{i=1}^{m} \frac{1}{1 + \exp\left(-y\ [\mathbf{w} \cdot \mathbf{x} + b]\right)}$$

- **Maximum Likelihood Estimate**:

We want the likelihood to be as high as possible, meaning the model's predictions should align well with the true labels across all data points

$$(\mathbf{w}_s, b_s) = \underset{\mathbf{w},b}{\operatorname{argmax}}\, L_{\mathbf{w},b}(s)$$

Obviously

$$\operatorname*{argmax}_{\mathbf{w},b} L_{\mathbf{w},b}(s) = \operatorname*{argmin}_{\mathbf{w},b} -\ln\left(L_{\mathbf{w},b}(s)\right)$$

However, the latter is easier to calculate:

Maximum Likelihood Estimation for logistic regression

$$(\mathbf{w}_s, b_s) = \operatorname*{argmin}_{\mathbf{w},b} \sum_{i=1}^{m} \ln\left(1 + \exp\left(-y\ [\mathbf{w}\cdot\mathbf{x} + b]\right)\right)$$

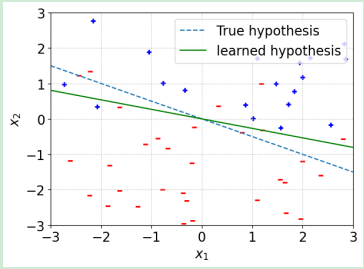
this is minimizing the Negative liklihood. which is the maximizaing the Positive one

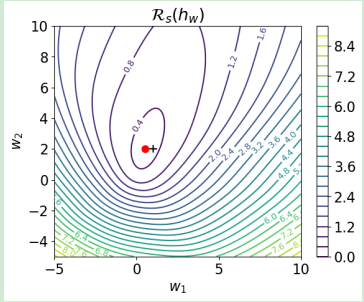
Remarks:

- The minimization problem can be solved efficiently with convex optimization (see later chapter).
- Logistic regression assumes no linear separability of the sample s – unlike the perceptron algorithm.
- **On the contrary:** If the sample s is linearly separable, then no minimizer $(\mathbf{w}_s, b_s)$ of $-\ln(L_{\mathbf{w},b}(S))$ exists. (**Exercise**)

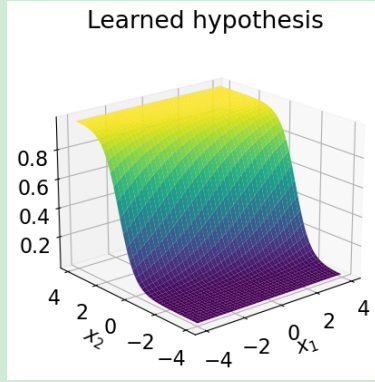
Example: Synthetic data

- In $\mathcal{X} = \mathbb{R}^2$, we want to learn a hypothesis $h_{\mathbf{w},0}$ from non-linearly separable training data.
- The $m = 50$ data points were randomly generated, the labels y follow a true Bernoulli distribution $\mathbb{P}_{\mathbf{w}^\dagger,0}$ based on a true hypothesis $h_{\mathbf{w}^\dagger,0}$ with $\mathbf{w}^\dagger = (1,2)^\top$.
- The maximum likelihood estimate was calculated by numerical optimization (using `scipy.optimize`).
- Contour lines of the convex function $\mathbf{w} \mapsto -\ln(L_{\mathbf{w},0}(s))$ are shown on the right including the numerically calculated $\mathbf{w}_s \approx (0.54, 2.00)$ (red) and the true $\mathbf{w}^\dagger$ (black).





- Let us visualize the learned hypothesis $h_{\mathbf{w}_s,0}$ in another way:
- The example can be reproduced by a Jupyter notebook provided on OPAL – you just have to complete it accordingly.



Notes

- The value $h_{\mathbf{w},b}(\mathbf{x})$ is very close to 1 or 0 if $|\mathbf{w} \cdot \mathbf{x} + b|$ is very large.
- Consequently, the predictions by $\mathrm{sig}(\mathbf{w} \cdot \mathbf{x} + b)$ and $\mathrm{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$ practically equal if $|\mathbf{w} \cdot \mathbf{x} + b|$ is very large.
- However, if $|\mathbf{w} \cdot \mathbf{x} + b|$ is close to 0, then $\mathrm{sig}(\mathbf{w} \cdot \mathbf{x} + b) \approx \frac{1}{2}$. The hypothesis $h_{\mathbf{w},b}$ is not sure which label it should predict for $\mathbf{x}$.
- This can be used for a more sophisticated prediction, e.g., "sure +1", "sure -1", "uncertain".
- Of course, if desired, a fixed label can also be predicted by, e.g.,

$$y = +1 \quad \Longleftrightarrow \quad h_{\mathbf{w},b}(\mathbf{x}) \geq \frac{1}{2} \quad \Leftrightarrow \quad \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x} + b) \geq 0.$$

View from the Machine Learning Perspective

- In logistic regression, we consider linear hypotheses from the class

$$\mathcal{L}_{d,\mathrm{sig}} := \{h_{\mathbf{w},b}(\mathbf{x}) = \mathrm{sig}(\mathbf{w} \cdot \mathbf{x} + b) \colon \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$$

- The maximum likelihood estimation can be understood as an ERM rule.

- To this end, we define a suitable loss function

Log loss measures the difference between the actual labels and the predicted probabilities produced by the model.

Log loss

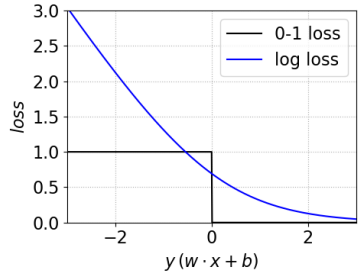
$$\ell(h_{\mathbf{w},b}, (\mathbf{x}, y)) := \ln\left(1 + \mathrm{e}^{-y(\mathbf{w}\cdot\mathbf{x}+b)}\right),$$

- Thus, the log loss ℓ penalizes small values of the probability

$$\frac{1}{1+\mathrm{e}^{-y(\mathbf{w}\cdot\mathbf{x}+b)}} = \mathbb{P}_{\mathbf{w},b}(Y = y_i \mid X = \mathbf{x}_i).$$

- and can be seen as a smooth version of the 0-1 loss used for the perceptron

$$\ell(h_{\mathbf{w},b}, (\mathbf{x}, y)) = \mathbf{1}_{(-\infty,0]}(y_i[\mathbf{w}\cdot\mathbf{x}_i + b])$$



ERM rule for logistic regression

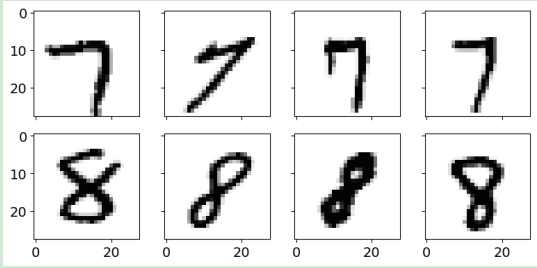
Given: Sample s with m data pairs $(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \{-1, +1\}$.

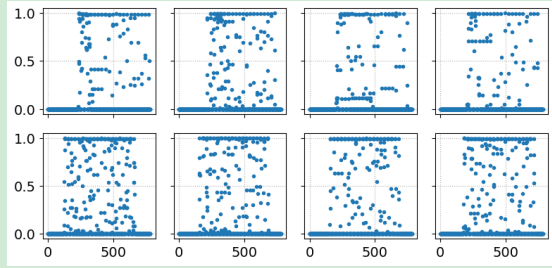
Compute: $h_{\mathbf{w}_s,b_s} \in \mathcal{L}_{d,\mathrm{sig}}$ where $(\mathbf{w}_s, b_s) \in \operatorname{argmin}_{\mathbf{w}'} \mathcal{R}_s(h_{\mathbf{w},b})$ with

$$\mathcal{R}_s(h_{\mathbf{w},b}) := \frac{1}{m}\sum_{i=1}^{m} \ln\left(1 + \exp(-y_i[\mathbf{w}\cdot\mathbf{x}_i + b])\right).$$

Example: Logistic regression for MNIST

- Again, we consider the MNIST dataset and the task of distinguishing handwritten "7" from "8" based on their greyscale images $\mathbf{x} \in [0,1]^{784}$

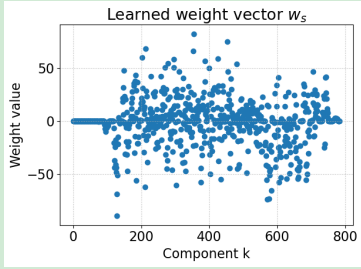


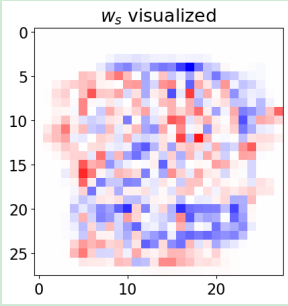


- However, since we do not know if the images are linearly separable we apply logistic regression to learn a classification rule
- To this end, we can use the routine `LogisticRegression` in the `scikit-learn` package in `Python`

- In Python:

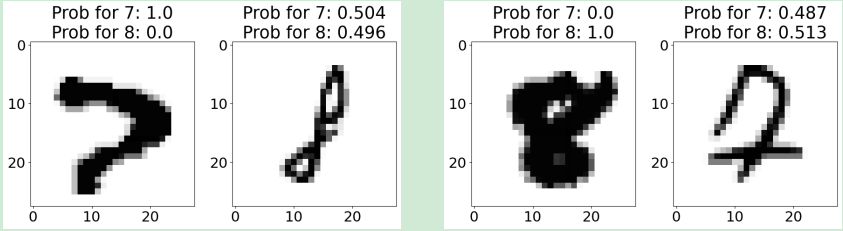
```
from sklearn.linear_model import LogisticRegression

# Define the logistic model, i.e., no penalty term (default is "l2
LogReg = LogisticRegression(random_state=0, penalty = None)

# Fitting the model, i.e., minimizing empirical risk
LogReg = LogReg.fit(x.T, y)

# Output number of required itersions
# (default setting for maximal is 100)
print(LogReg.n_iter_)

# Output mean accuracy on training data
print(LogReg.score(x.T, y))
```





- It took 68 iterations to learn a weight vector $\mathbf{w}_s$ and bias $b_s \approx 141.53$ such that the training data is linearly separated (mean accuracy $= 1 - \mathcal{R}_s^{0\text{-}1}(h_s) = 1$).
- On the next slide we show results for different $\mathbf{w}_s$ and b_s obtained after only `max_iter = 25` iterations

- We can use the additional information of the logistic regression and look for the most "certain" and "uncertain" detected 7s and 8s:



- Analogously for the subset of misclassified 7 and 8: