# 2. Batch Learning

## Notation

- $\mathcal{X}$ is the input space, and $\mathcal{Y}$ is the output space
- In classification problems, output space $\mathcal{Y}$ comprised of $K$ possible classes (or categories)
- For simplicity, we'll just call them $\mathcal{Y} := \{1, 2, \cdots, K\}$
  (When $K = 2$, we typically use $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{-1, +1\}$)
- Labeled example: $(\boldsymbol{x}, y) \in \mathcal{X} \times \mathcal{Y}$. Interpretation:
  - $\boldsymbol{x}$ represents the description or measurements of an object
  - $y$ is the category to which that object belongs

## Loss

Assume there's a distribution *D* over space of labeled example X x Y. You have access only to the sample that you are observing. Your data are drawn from distribution D that you don't actually know.

Assume also that the labels are produced by a correct labeling function

$$f : \mathcal{X} \to \mathcal{Y} \text{ and that } Y = f(X)$$

We define the **loss** of the classifier h as the function that measures the error between the assigned prediction and the prediction of the correct labeling function.

$$L(h(X), f(X)) = L(h(X), Y) = \begin{cases} 1, & \text{if } h(X) \neq Y \\ 0, & \text{if } h(X) = Y \end{cases}$$

## True Risk

We define the **loss** L(h(X), Y) of the classifier h as the function that measures the error between the assigned prediction and the prediction of the correct labeling function.

We need to define the ***true risk*** → is the probability that it does not predict the correct label on a random data point drawn from *D*

$$R_{\mathcal{D}}(h) := \mathbb{P}[h(X) \neq f(X)] = \mathbb{P}[h(X) \neq Y]$$

In other words, the true risk of the learner h is the expected value of the loss

$$R_{\mathcal{D}}(h) := \mathbb{E}_{X \sim \mathcal{D}}\{L(h(X), Y)\}$$

## Bayes Classifier

- Assume $\mathcal{X}$ is discrete for now
- Suppose $(x, y) \sim \mathcal{D}$
- Want to find $h : \mathcal{X} \to \mathcal{Y}$ that minimizes $\mathbb{P}[h(x) \neq y] = \sum_{x \in \mathcal{X}} \mathbb{P}[h(X) \neq y | X = x] \cdot \mathbb{P}[X = x]$
- For every $x \in \mathcal{X}$ , set $h(x)$ to minimize $\mathbb{P}[h(X) \neq Y | X = x]$
- $\mathbb{P}[h(X) \neq Y | X = x] = 1 - \mathbb{P}[h(X) = Y | X = x]$
- So, to minimize $\mathbb{P}[h(X) \neq Y | X = x]$, maximize $\mathbb{P}[h(X) = Y | X = x]$
- i.e. choose

$$h(x) = \arg\max_{y \in \mathcal{Y}} \mathbb{P}[Y = y | X = x]$$

As we already know, this is the **Bayes classifier**

The goal of the **Bayes Classifier** is to *minimize classification errors* by always choosing the class with the highest probability, given the observed data. The Bayes Classifier is considered optimal because it minimize the probability of misclassification when the true probability distribution are known.

## Bayes Risk

- Denote by $h^*$ the Bayes classifier
- For the **Bayes Risk** $R_{\mathcal{D}}(h^*)$ it holds $R_{\mathcal{D}}(h^*) \leq R_{\mathcal{D}}(h)$ for any $\mathcal{H}$ and $h \in \mathcal{H}$
  - You cannot improve over the Bayes Risk, regardless of what ML algorithm you use!

The **Bayes Risk** *quantifies the expected cost of classification errors*. It's a way to measure how well a classifier performs under uncertainty, taking into account the true class probabilities and any associated costs or losses.

The Bayes Classifier is the theoretical classifier that minimizes the Bayes Risk.

The Bayes Risk represent the cost or error when using the Bayes Classifier. It tells us how much error is unavoidable even with an optimal classifier, given the inherent uncertainty in the data and overlapping distributions of classes.

## Batch Learning

In practical cases we do not know *f (target function)* and we need to estimate *h* from the data. In **batch learning,**this process involves:

The learner's input:

$$\text{Training data, } S = \{(\boldsymbol{x}_1, y_1) \cdots (\boldsymbol{x}_m, y_m)\} \in (\mathcal{X} \times \mathcal{Y})^m$$

**The learner has access to this data all at once during training**

The learner's output:

$$\text{A prediction rule, } h : \mathcal{X} \to \mathcal{Y}$$

In *Batch Learning*, the model is trained using all the available data at once.

The dataset is typically divided into multiple passes through the model (epochs), and during each pass, the model's parameters are adjusted to minimize the error or loss function.

The model parameters are updated after processing the entire batch, so the gradient is computed across the full dataset at once.

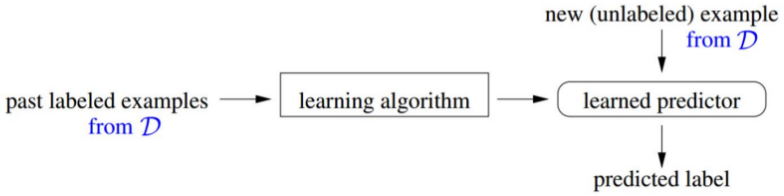
What should be the goal of the learner? *h performs* well not just on the training data but also on future examples → h should be correct on future examples

## Batch Learning - IID condition

In
**batch learning**, during the model's training process, it is assumed that the training data is **independent and identically distributed.**

Also achieving a classifier with high accuracy depend on the IID assumption.

> Key assumption: Data $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_n, y_n)$ are *identically and independentely distributed (i.i.d.)* random labeled examples with distribution $\mathcal{D}$, i.e., an i.i.d. sample from $\mathcal{D}$.




This assumption is the connection between what we've seen in the past (training set) to what we expect to see in the future data.

What does an accurate classifier look like?

## Empirical Risk

The empirical Risk measures how often the predictor h makes incorrect prediction on the training data

$$R_S(h) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}[h(\boldsymbol{x}_i) \neq y_i]$$

In batch learning, this metric is used during training to evaluate the model's performance on the training set (minimize the error of the prediction). The goal is to minimize Rs(h) in the hope that this will lead to a low error on unseen data.

💡 Minimizing the **empirical risk** (i.e., the error on the training set) is not enough to guarantee good performance on future data. You also need to ensure that the **IID condition** (independence and identical distribution) is satisfied.

### Can only be Probably Correct

- Claim: Even when $R_{\mathcal{D}}(h^*) = 0$, we can't hope to find $h$ s.t. $R_{\mathcal{D}}(h) = 0$
- Proof: for every $\epsilon \in (0, 1)$ take $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2\}$ and $\mathbb{P}_{\mathcal{D}}(\{\boldsymbol{x}_1\}) = 1 - \epsilon$, $\mathbb{P}_{\mathcal{D}}(\{\boldsymbol{x}_2\}) = \epsilon$
- The probability not to see $\boldsymbol{x}_2$ at all among $m$ i.i.d. examples is $(1 - \epsilon)^m \approx e^{-\epsilon m}$
- So, if $\epsilon \ll 1/m$ we're likely not to see $\boldsymbol{x}_2$ at all, and we will not know its label!
- Relaxation: We'd be happy with $R_{\mathcal{D}}(h) \leq \epsilon$, where $\epsilon$ is user-specified

### Can only be Approximately Correct

- Recall that the input to the learner is randomly generated
- There's always a (very small) chance to see the same example again and again
- Claim: No algorithm can guarantee $R_{\mathcal{D}}(h) \leq \epsilon$ for *sure*
- Relaxation: We'd allow the algorithm to fail with probability $\delta$, where $\delta \in (0, 1)$ is user-specified
  - The probability is over the random choice of examples

## Probably Approximately Correct (PAC) learning

The learner does not know the underlying data distribution $D$ or the Bayes predictor. The goal of the learner is to approximate the best possible predictor without direct access to this information.

The learner can ask for training data $S$ containing $m(\epsilon, \delta)$ examples (that is, the number of examples can depend on the value of $\epsilon$ and $\delta$ but it can't depend on $\mathcal{D}$ or the Bayes function)

Learner should output a hypothesis $h$ s.t. $\mathbb{P}[R_{\mathcal{D}}(h) \leq \epsilon] \geq 1 - \delta$

**The probability of *h* is below e would be higher than 1 - delta**

The learner is given two parameters:

- $\varepsilon$ → represents the desired level of accuracy.
- $\delta$ → represents the desired level of confidence.

Together, these parameters quantify what it means for the learner to perform "probably (with probability at least 1 - $\delta$) approximately (within accuracy $\varepsilon$) correctly," which defines **PAC Learning** (Probably Approximately Correct Learning).

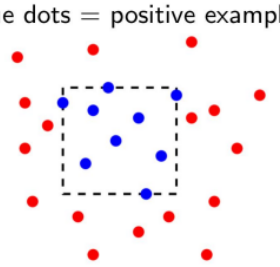
## Realizability Assumption

The **realizability assumption** is a key concept in machine learning, particularly in the context of **PAC Learning** (Probably Approximately Correct Learning). It refers to the idea that there exists a true hypothesis (or model) in the hypothesis class that perfectly matches the true underlying distribution of the data.

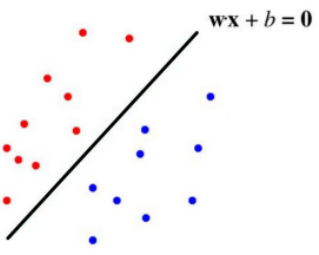
- Consider binary classification with $\mathcal{Y} = \{0, 1\}$.
- **Realizability assumption:** Assume that, for a given class $\mathcal{H}$ of functions from $\mathcal{X} \to \{0, 1\}$, there exists $h^* \in \mathcal{H}$ such that $R_{\mathcal{D}}(h^*) = 0$
  - This implies that $h^*$ is the Bayes classifier
- The learner knows $\mathcal{H}$

Examples of function classes $\mathcal{H}$:

Rectangles in $\mathbb{R}^2$
(Blue dots = positive examples)



Linear classifiers in $\mathcal{X} = \mathbb{R}^d$



**The learner knows *H* →** when we create a model we know which king of shape our model should have.

## Learning in the Realizable Setting

What is a sensible learning algorithm for the realizable setting?

- Given training data $S$ with $m$ samples, return any $h \in \mathcal{H}$ such that $R_S(h) = 0$.
- This is a special case of **Empirical Risk Minimization** (ERM), that is

$$h_S = \arg\min_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}[h(\boldsymbol{x}_i) \neq y_i]$$

The realizable setting assumes that there exists a perfect hypothesis that can correctly classify all the samples in the data with zero error.

In this ideal setting, a sensible learning algorithm would be one that finds a hypothesis h form the hypothesis class H such that the true rist R(h) equals zero.

In the realizable setting, since the true risk can be zero for some hypothesis h, ERM is a reasonable approach because it can find a hypothesis that has zero empirical risk on the training data, and since the data is realizable, this will also correspond to zero true risk

# PAC Learning

**Definition (PAC learnability)**

A hypothesis class $\mathcal{H}$ is PAC learnable if there exists a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm with the following property:

- for every $\epsilon, \delta \in (0,1)$
- for every distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$ such that the realizability assumption holds in $\mathcal{H}$

when running the learning algorithm on $m \geq m_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. examples generated by $\mathcal{D}$, the algorithm returns a hypothesis $h$ such that, with probability of at least $1 - \delta$ (over the choice of the examples), $R_{\mathcal{D}}(h) \leq \epsilon$

$m_{\mathcal{H}}$ is called the **sample complexity** of learning $\mathcal{H}$ PAC learnability formalizes the question: Can I learn anything with this hypothesis class (under the realizability assumption)? And how many samples I need?

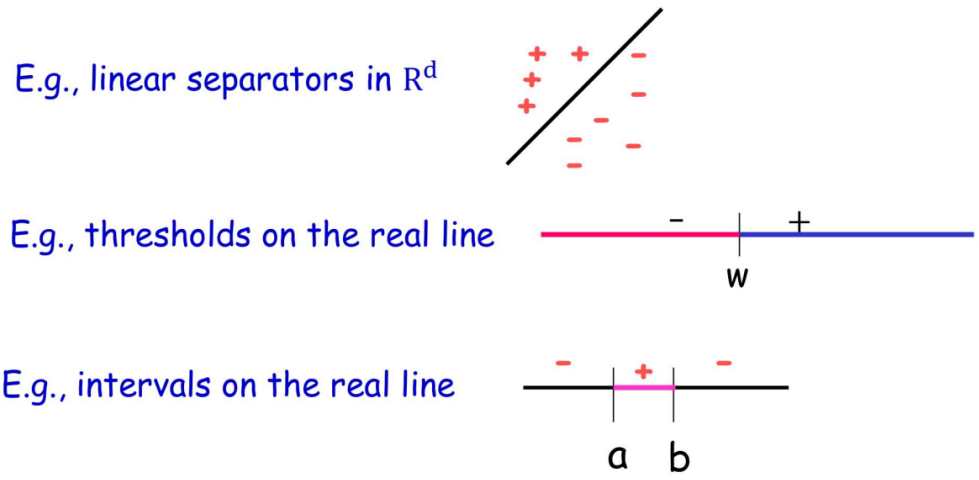
## What is Learnable and how to learn?

- Let $\delta \in (0,1)$, $\epsilon > 0$, and $m \geq \frac{\ln(|\mathcal{H}|/\delta)}{\epsilon}$. Then, with probability $1-\delta$, we have $R_{\mathcal{D}}(h) \leq \epsilon$

Clearly these bounds are useful if $\ln|\mathcal{H}|$ is finite and not too large w.r.t. $|S|$

*H* represent all possible lines that identifies our classifier and should be infinite

## Infinite Hypothesis Classes?




How to solve this problem? **Discretization trick**

Any learning algorithm we use will be implemented on a computer. Any parametrization of the algorithm, are store in a floating point representation → the class is actually finite.

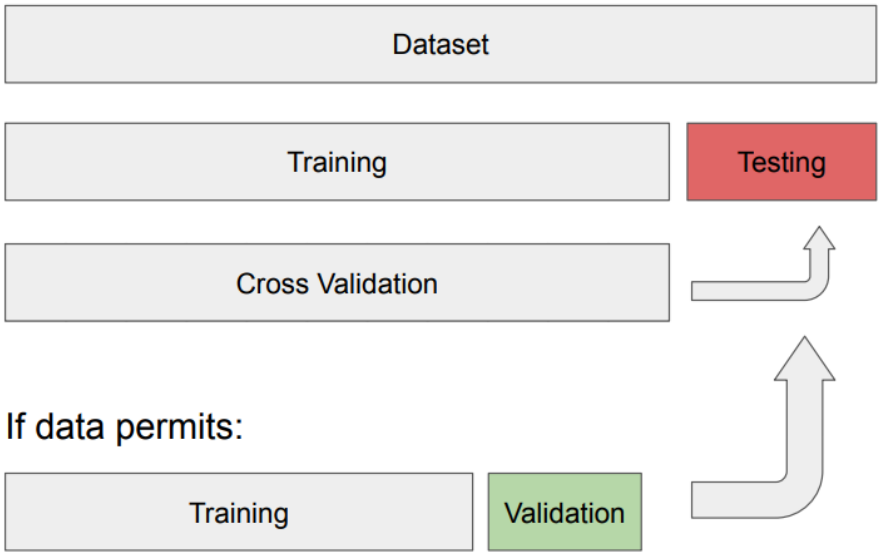
# Overfitting

**Definition →** This occurs when a predictor performs excellently on the training set but performs poorly on the real-world data. You are minimizing the empirical risk, not the true risk.

To avoid overfitting, we can use techniques such as **Cross Validation** and **Regularization**.

## Cross Validation

*Cross Validation* helps in understanding how well a model will perform on unseen data by using different subsets of the data for training and testing.

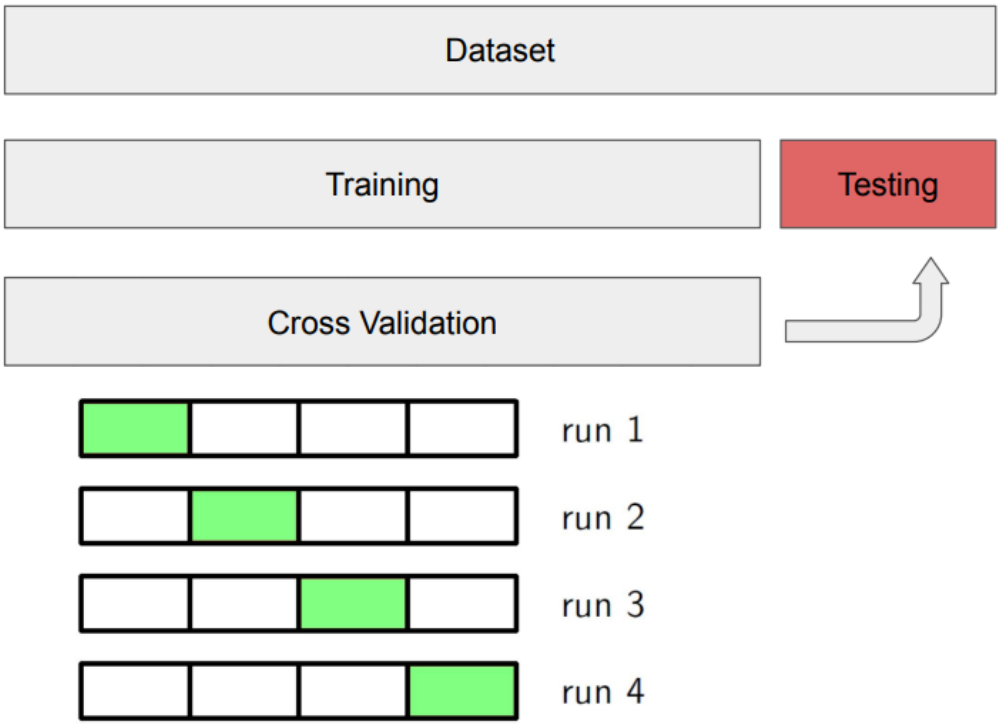



Cross validation is applied only on the test part

***How perform Cross Validation?***

### k-fold Cross Validation

The dataset is divided into K equally sized subset. For each fold the model is trained on K-1 folds and tested on the remaining fold. The process is repeated K times, each time using a different fold as the test set.




### Leave One Out Cross Validation

If I have m samples I just leave one sample out. For each fold, the model is trained on N-1 samples (N is the total number of samples) and tested on the one left-out sample.

## Train - Validation - Test

In practice, we usually have one pool of examples and we split them into 3 sets:

- Training set: apply the learning algorithm with different parameters on the training set to produce H = h1...hr
- Validation set: choose h* from H based on the validation set
- Test set: estimate the true error of h* using the test set

> ⚠ You cannot avoid the use of a test set, the error on the validation set is negatively biased

if the number of samples in the test set is too small, I cannot reliably estimate the true error.

## Regularization

We introduce an extra term in the loss

- Given training data $S$ with $m$ samples, return any $h \in \mathcal{H}$ such that $R_S(h) = 0$.
- This is a special case of **Empirical Risk Minimization** (ERM), that is

$$h_S = \arg\min_{h \in \mathcal{H}} \left[ \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}[h(x_i) \neq y_i] \quad + \lambda \text{ Complexity(h)} \right]$$

- λ is a parameter, a positive number that serves as a conversion rate between the loss and the hypothesis complexity (they might not have the same scale)
- the form of the complexity /regularization function depends on the hypothesis space
- we still need cross validation and in this case we need to include the parameter λ - select the one that gives the best validation score

is an hyperparameter

# Linear Regression

With **linear regression** we are trying to seach for a line that well describe a trend of a certain data and we need to measure the number of mistakes

- So far, mostly focused on classification problems, where the label space $\mathcal{Y} = \{1, 2, \cdots, K\}$ is discrete.
- **Regression** analysis considers prediction problems where label space $\mathcal{Y}$ is continuous (e.g. $[0, 1]$, $\mathbb{R}^d$, etc.)
- Measuring quality of a predictor $h : \mathcal{X} \to \mathcal{Y}$ using prediction error, $\mathbb{P}[h(X) \neq Y]$ doesn't make much sense in the regression context (why?)
- **Goal:** find $h : \mathcal{X} \to \mathcal{Y}$, from some function class $\mathcal{H}$, minimizing error measured using a loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, i.e.

$$\mathbb{E}[L(Y, h(X))]$$

## Flashback: Loss Function

- A loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ maps decisions to costs: $L(\hat{y}, y)$ defines the penalty paid for predicting $\hat{y}$ when the true value is $y$.
- Standard choice for classification: $0/1$ loss

$$L_{0/1}(\hat{y}, y) = \begin{cases} 0 & \hat{y} = y \\ 1 & otherwise \end{cases}$$

This is the 0/1 loss, which is the case for classification. In this case, you count the errors. This kind of loss doesn't work well for regression because it is continuous

Standard choice for regression: squared loss $L(\hat{y}, y) = (\hat{y} - y)^2$
- Not the only possible choice

With this loss we measure the distance between the prediction and the ground truth. For regression this is not the only possible choice, but is the easiest one.

## Empirical Loss

- We consider a parametric function $h_w(x)$
  E.g., linear function: $h_w(x) = w^\top x + b$ we need to search w and b that minimize the loss
- The empirical loss of function $y = h_w(x)$ on a set $S$:

This is what we need to do

$$L_S(w) = \frac{1}{N} \sum_{i=1}^{N} L(h_w(x_i), y_i)$$

we have to minimize this and in this way we have to perform ERM

- We will find an ERM over $\mathbb{R}^d$ w.r.t. $L$
  - E.g., Least Squares minimizes the empirical loss for the squared loss.

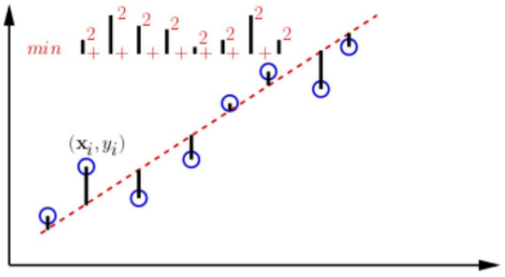
## Linear predictors in 1D

- Our features are vectors in $\mathbb{R}$, i.e. $\mathcal{X} = \mathbb{R}$
- The labels are real numbers, i.e. $\mathcal{Y} = \mathbb{R}$
- Consider predictors of the form

$$\hat{y} = h_{(w,b)}(x) = w\,x + b$$

- Hypothesis class: $\mathcal{H} = \{h_{(w,b)} : w, b \in \mathbb{R}\}$
  - Infinite, but remember the discretization trick!

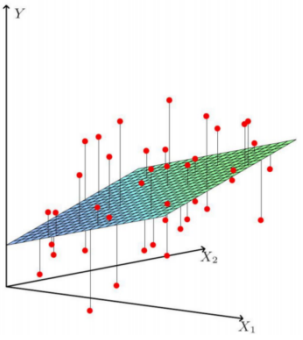
- We want to fit a linear function to an observed set of points $\mathcal{X} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N]$ with associated labels $\mathcal{Y} = [y_1, ..., y_N]$.
  - Once we fit the function, we want to use it to *predict* the $y$ for new $\boldsymbol{x}$.
- Least squares (LSQ) fitting criterion: find the function that minimizes sum (or average) of square distances between actual $y_i$ in the training set and predicted ones, that is ERM




Whenever we have a new sample, we are able to assign it to the correct position by taking our predicted line and calculating the distance between the point and the line. We then sum all these distances together and minimize this sum.

## Linear functions in 2D

- General form: $\hat{y} = h_{(b,\boldsymbol{w})}(\boldsymbol{x}) = b + w_1 x_1 + \cdots + w_d x_d = b + \langle \boldsymbol{w}, \boldsymbol{x} \rangle$
- 1D case ($\mathcal{X} = \mathbb{R}$): a line
- $\mathcal{X} = \mathbb{R}^2$: a plane
- Hyperplane in general, $d$-dimensional case
- Hypothesis class: $\mathcal{H} = \{h_{(b,\boldsymbol{w})} : b \in \mathbb{R}, \boldsymbol{w} \in \mathbb{R}^d\}$




--

## Least Square Criterion

- Given pairs of input/output values, find $(b, \boldsymbol{w})$ to minimize the prediction errors (on average)
- I.e. using square loss on $\mathcal{H} = \{h_{(b,\boldsymbol{w})} : b \in \mathbb{R}, \boldsymbol{w} \in \mathbb{R}^d\}$: squared error on $(\boldsymbol{x}, y)$ is $(y - (b + \langle \boldsymbol{w}, \boldsymbol{x} \rangle))^2$

We have a model that is described by b and w and this can be written also in matricial form.

Given training data

$$\boldsymbol{X} = \begin{bmatrix} — & \boldsymbol{x}_1^\top & — \\ — & \boldsymbol{x}_2^\top & — \\ & \vdots & \\ — & \boldsymbol{x}_N^\top & — \end{bmatrix} \in \mathbb{R}^{N \times d} \qquad \boldsymbol{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^N,$$

find $b \in \mathbb{R}$ and $\boldsymbol{w} \in \mathbb{R}^d$ to minimize

$$\sum_{i=1}^{N} (y_i - (b + \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle))^2 = \left\| \boldsymbol{y} - \begin{bmatrix} \mathbf{1} & \boldsymbol{X} \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{w} \end{bmatrix} \right\|_2^2$$

**Simplification:** Replace $\boldsymbol{X}$ with $\begin{bmatrix} \mathbf{1} & \boldsymbol{X} \end{bmatrix}$ and $\boldsymbol{w}$ with $\begin{bmatrix} b \\ \boldsymbol{w} \end{bmatrix}$, to have the simpler expression

$$\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2$$

# Ridge Regression

$$\frac{1}{2}\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w}\|_2^2+\frac{\alpha}{2}\|\boldsymbol{w}\|_2^2$$

I want to minimize errors, but at the same time the model should not be too complex. We add a regularization term. It is a standard way to control overfitting and becomes particularly important for small number of samples (N) and a large vector dimensionality (d)

$$\nabla_{\boldsymbol{w}}\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w}\|_2^2+\frac{\alpha}{2}\|\boldsymbol{w}\|_2^2$$
$$=\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y})+\alpha\boldsymbol{w}=0$$

We can write down the matricial form.

We can take the derivative and put derivative equal to zero

The result is:

$$\Rightarrow \boldsymbol{w}=(\boldsymbol{X}^\top\boldsymbol{X}+\alpha\boldsymbol{I})^{-1}\boldsymbol{X}^\top\boldsymbol{y}$$

This is called **Ridge Regression**