Supervised Learning Review

DS4440

Assume access to training data (X, Y)

$$X = \{x_1, x_2, \dots, x_n\}$$

$$= \begin{array}{cc} & d \\ n & \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ \vdots & & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix} \end{array} \qquad X \in \mathbb{R}^{n \times d}$$

$$Y = \{y_1, y_2, \dots, y_n\} \qquad Y \in \mathbb{R}^{n \times c}$$

In the simplest case, $c = 1$.

$$Y = \begin{array}{cc} & c \\ n & \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \end{array} \quad \text{e.g.} \quad \begin{bmatrix} -1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

X are our instances

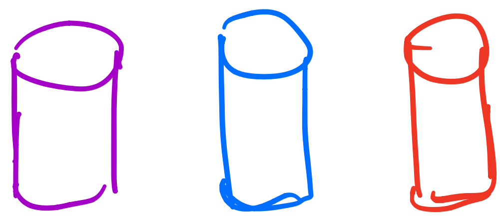
Y are our labels

The goal is to find a model with parameters θ, f_θ, s.t. $f_\theta(x_i) = y_i$.

The real hope is that f_θ generalizes to unseen or held-out data.

We estimate this via a test set (x', y') which might contain ℓ examples.

In practice we usually use three datasets



!!! Do not touch your test data !!!

Train — estimate θ

dev — explore models & hyperparams

Test — eval. final performance.

The idea behind learning is to find $\hat{\theta}$ s.t. the loss on the train data is small.

Formally

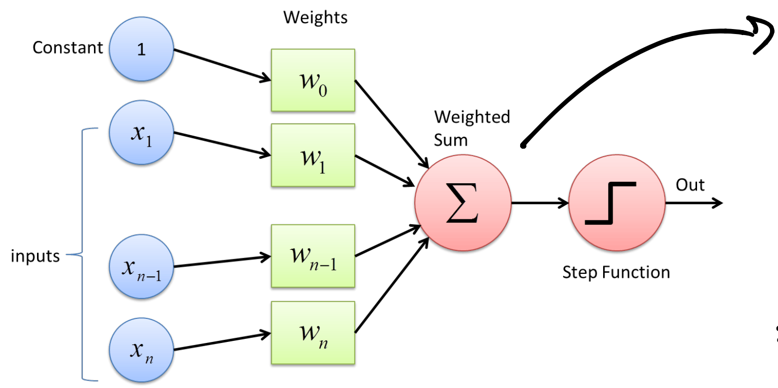
Training objective

$$\boxed{\arg\min_{\hat{\theta}} \sum_{i=1}^{N} \mathcal{L}_{\hat{\theta}}(x_i, y_i)}$$

$$= \mathcal{L}(\overbrace{f_{\hat{\theta}}(x_i)}^{\hat{y}_i}, y_i)$$

This requires (at very least) Specifying f_θ

A classic model is the Perceptron



$$W_0 + W_1 \cdot x_{i1} \cdots + W_n \cdot x_{in}$$

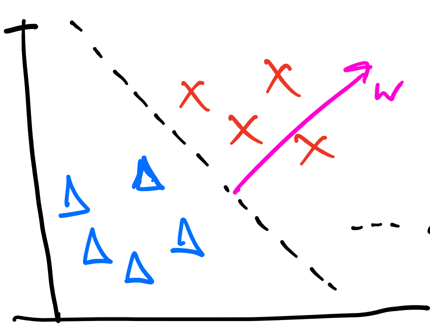
$$= \sum_{j}^{n} W_j x_{ij}$$

$$= w \cdot x_i$$

Here $\theta = W$ and a prediction is made:

$$\hat{y}_i = \begin{cases} 1 & \text{if } w \cdot x_i > \tau \\ -1 & \text{otherwise} \end{cases}$$

W defines a decision boundary (line or plane) that delineates examples from $y_i = 1$ / $y_i = -1$



--- decision boundary

This is perpendicular to W; $u \cdot v = 0$ iff u is perp. to v.

So here the natural loss is zero/one loss

$$\mathcal{L}(\hat{y}_i, y_i) = \begin{cases} 0 & \text{if } y_i == \hat{y}_i \\ 1 & \text{otherwise} \end{cases}$$

We need an estimation procedure to find a good $\hat{\theta}$.

```
fitPerceptron(X, y, max_iters)
    w ← 0⃗        // init weights to zero-vector
    for max_iters
      for (x_i, y_i) ∈ X, Y
        a ← w · x_i        // activation for x_i
        if y_i · a ≤ 0      // we made a mistake
          w ← w + y_i · x_i
    return w
```

Let's see this in colab.