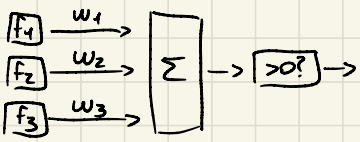
Linear classifiers

$x \Rightarrow \underset{\text{feature vector}}{f(x)} \rightarrow y$

loosely inspired by biology

- inputs: feature values
- each feature has weight
- sum is the activation

$$\text{activation}_w(x) = \sum_i w_i f_i(x) = w \cdot f(x)$$

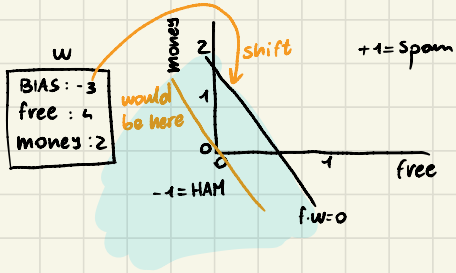


activation $\overset{+1}{\text{positive}}$ or $\overset{-1}{\text{negative}}$

weights

Binary case: compare features to a weight vector

learning: figure out weight vector from examples



dot product $w \cdot f$ either positive or negative

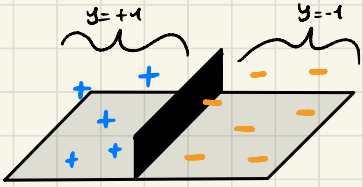
Decision rules

Binary decision rules

space of feature vectors

examples are points

any weight vector is a hyperplane

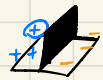


learning : binary perceptron

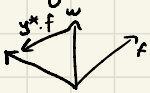
start with weights = 0

for each training instance:

- classify with current weights



$$y = \begin{cases} +1 & \text{if } w \cdot f(x) \geq 0 \\ -1 & \text{if } w \cdot f(x) < 0 \end{cases}$$

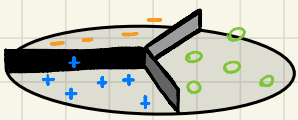


- if correct (i.e. $y = y^*$) no change

- if wrong adjust the weight vector

$$w = w + y^* \cdot f$$

Multiclass decision rule



- must have a weight vector w_j for each class
- score (activation) of a class y

$$w_y \cdot f(x)$$

- prediction highest score wins

$$y = \arg\max_y w_y \cdot f(x)$$

Learning

- start with all weights = 0
- pickup training examples one by one
- predict with current weights

$$y = \arg\max_y w_y \cdot f(x)$$

- if correct no change
- if wrong lower score of wrong answer, rise of right answer

$$w_y = w_y - f(x)$$

$$w_{y^*} = w_{y^*} + f(x)$$

Properties of perceptrons

- separability: true if some parameters get the training set perfectly correct
- convergence: if separable, perceptron will eventually converge (binary case)
- Mistake bound: the max number of mistakes (binary case) related to the margin or degree of separability

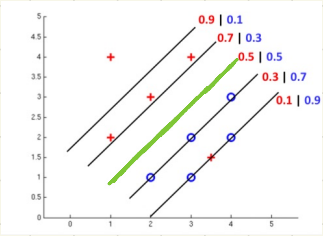
$$\text{mistakes} < \frac{k}{\delta^2}$$

Problems with perceptron

- Noise: if data not separable, weights might trash
  - averaging vectors over time can help
- Mediocre generalization: finds a "barely" separating solution
- Overtraining: test/held-out accuracy usually rises then falls
  - overtraining is a kind of overfitting

Improvements

Non separable case: probabilistic decision



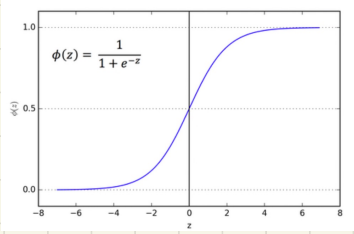
how to get the decision?

Perceptor scoring $z = w \cdot f(x)$

if z very positive → want probability going to 1

very negative → " " to 0

Sigmoid function $\phi(z) = \frac{1}{1+e^{-z}}$



what is the best w?

Maximum likelihood estimation

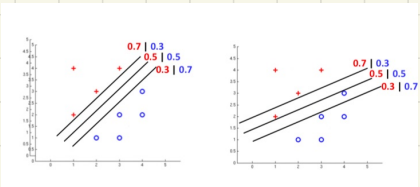
$$\max_w ll(w) = \max_w \sum_i \log P(y^i | x^i; w)$$

⇒ logistic regression

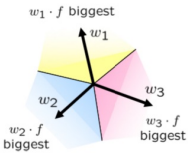
with: $P(y^i = +1 | x^i; w) = \frac{1}{1+e^{-wf(x^i)}}$

$$P(y^i = -1 | x^i; w) = 1 - \frac{1}{1+e^{-wf(x^i)}}$$

Separable case: Probabistic decision - clear preference



- Recall Perceptron:
  - A weight vector for each class: w_y
  - Score (activation) of a class y: $w_y \cdot f(x)$
  - Prediction highest score wins $y = \arg\max_y w_y \cdot f(x)$



how to make the scores into probabilities

$$\underbrace{z_1, z_2, z_3}_{\text{original activations}} \rightarrow \underbrace{\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}, \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}, \frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}}_{\text{softmax activations}}$$

best w?

$$\max_w \quad ll(w) = \max_w \sum_i \log P(y^i|x^i, w)$$

=> multiclass logistic regression

with $P(y^i|x^i, w) = \frac{e^{w_{y^i} \cdot f(x^i)}}{\sum_y e^{w_y \cdot f(x^i)}}$

Optimisation

How to solve $\max_w ll(w) \equiv \max_w \sum_i \log P(y^i|x^i; w)$

⇓

Hill climbing

start wherever, repeat: move to best neighbouring state, if no better neighbors quit

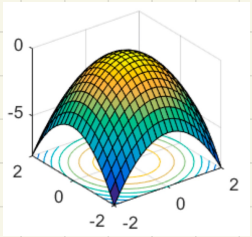
what's tricky about multiclass logistic regression?

- optimisation over continous space -> infinite neighbors

1D optimisation

- could evaluate $g(w_0+h)$ and $g(w_0-h)$, then step in the best direction
- or evaluate derivative $\frac{\partial g(w_0)}{\partial w} = \lim_{h \to 0} \frac{g(w_0+h)-g(w_0+h)}{2h}$

2D optimisation - Gradient ascend



perform update in uphill direction for each coordinate

the steeper the slope (der.↑) the bigger the step for that coordinate

consider $g(w_1, w_2)$

updates:

$$w_1 \leftarrow w_1 + \alpha \cdot \frac{\partial g}{\partial w_1}(w_1, w_2)$$

$$w_2 \leftarrow w_2 + \alpha \cdot \frac{\partial g}{\partial w_2}(w_1, w_2)$$

in vector notation

$$w \leftarrow w + \alpha \cdot \nabla_w g(w)$$

with: $\nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(w) \\ \frac{\partial g}{\partial w_2}(w) \end{bmatrix}$ = gradient

idea: start somewhere, take a step in steepest direction

what is the steepest direction

$$\max_{\Delta : \Delta_1^2 + \Delta_2^2 \leq \epsilon} g(w + \Delta)$$

first order taylor expansion: $g(w+\Delta) \approx g(w) + \frac{\partial g}{\partial w_1}\Delta_1 + \frac{\partial g}{\partial w_2}\Delta_2$

steepest descend direction: $\max_{\Delta : \Delta_1^2 + \Delta_2^2 \leq \epsilon} g(w) + \frac{\partial g}{\partial w_1}\Delta_1 + \frac{\partial g}{\partial w_2}\Delta_2$

recall: $\max_{\Delta : \|\Delta\| \leq \epsilon} \Delta^T a \rightarrow \Delta = \epsilon \frac{a}{\|a\|}$

hence, solution: $\Delta = \epsilon \frac{\nabla g}{\|\nabla g\|}$ gradient direction = steepest direction $\nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \end{bmatrix}$

Gradient in n dimensions

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \\ \dots \\ \frac{\partial g}{\partial w_n} \end{bmatrix}$$

init w

For (i=0 to ...)

$w = w + \alpha \cdot \nabla g(w)$ ← learning rate (to be chosen carefully)

we wanted to solve $\max_w l(w) = \max_w \underbrace{\sum_i \log P(y^i | x^i; w)}_{g(w)}$

observation: once gradient on one training example is done, might as well incorporate before next

```
init w
for (i=0 ...)
    j = random()
    w = w + α · ∇ log P(y^j | x^j ; w)
```

observation: gradient over small set of training examples (= mini batch) can be computed in parallel, might do it instead of a single one

```
init w
for (i=0 ...)
    J = random()
    w = w + α · Σ_{j∈J} ∇ log P(y^j | x^j ; w)
```