Linear classification

- **Loss function:**

  $$\ell(f(x), y) = \mathbf{1}\{f(x) \neq y\}$$

> **Learn**: f:**X —>**Y
  - **X** – features
  - **Y – target classes**

    $$Y \in \{-1, 1\}$$

> **Expected loss of f:**
>

$$\mathbb{E}_{XY}[\mathbf{1}\{f(X) \neq Y\}] = \mathbb{E}_X[\mathbb{E}_{Y|X}[\mathbf{1}\{f(x) \neq Y\}|X = x]]$$

$$\mathbb{E}_{Y|X}[\mathbf{1}\{f(x) \neq Y\}|X = x] = 1 - P(Y = f(x)|X = x)$$

> **Bayes optimal classifier:**

$$f(x) = \arg\max_y \mathbb{P}(Y = y|X = x)$$

> **Model of logistic regression:**

$$P(Y = y|x, w) = \frac{1}{1 + \exp(-y\, w^T x)}$$

What if the model is wrong?

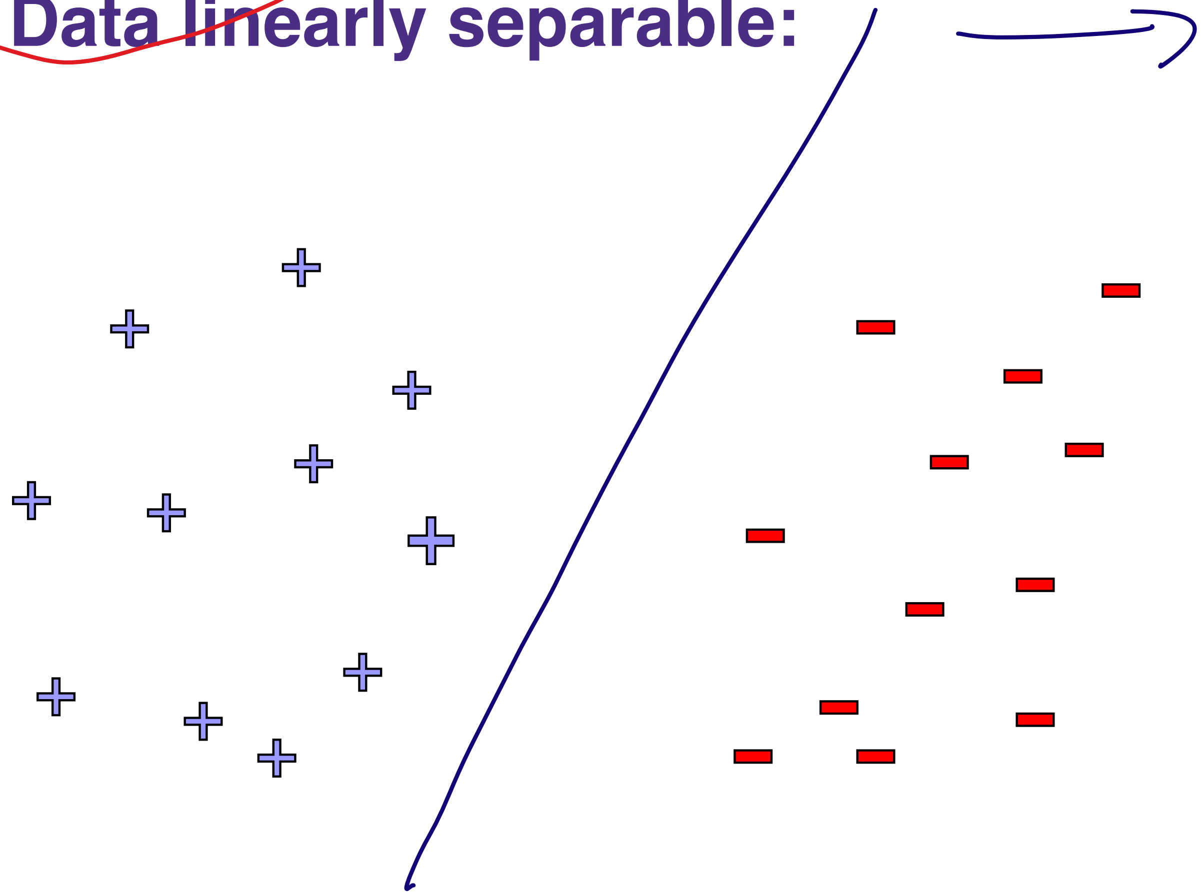
©Kevin Jamieson 2018

Binary Classification

> **Perceptron guaranteed to converge if**
  - **Data linearly separable:**

→ alg

∃ a hyperplane which perfectly classifies all points



Can we do classification without a model of $\mathbb{P}(Y = y|X = x)$?

©Kevin Jamieson 2016

The Perceptron Algorithm [Rosenblatt '58, '62]

- Classification setting: y in {-1,+1}
- Linear model
  - Prediction:

- Training:
  - Initialize weight vector:
  - At each time step:
    - Observe features:
    - Make prediction:
    - Observe true class:

    - Update model:
      - If prediction is not equal to truth

©Kevin Jamieson 2016

The Perceptron Algorithm [Rosenblatt '58, '62]

- Classification setting: y in {-1,+1}
- Linear model
  - Prediction:

$$\text{sign}(w^T x_i + b)$$

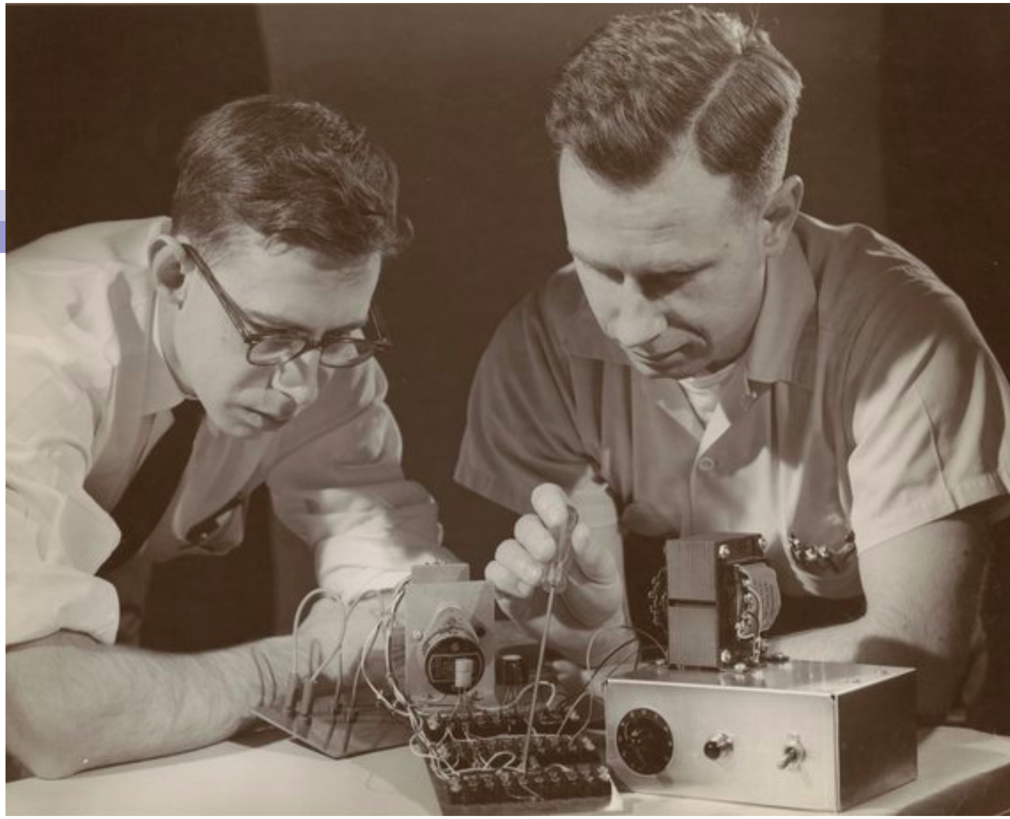
→ linear in features

- Training:
  - Initialize weight vector: $w_0 = 0, b_0 = 0$
  - At each time step:
    - Observe features: x_k → $\in \mathbb{R}^d$
    - Make prediction: $\hat{y}_k = \text{sign}(x_k^T w_k + b_k)$ → current model
    - Observe true class: y_k → may or may not be $\hat{y}_k$
    - Update model:
      - If prediction is not equal to truth

$$\begin{bmatrix} w_{k+1} \\ b_{k+1} \end{bmatrix} = \begin{bmatrix} w_k \\ b_k \end{bmatrix} + y_k \begin{bmatrix} x_k \\ 1 \end{bmatrix}$$

($w_{k+1} \in \mathbb{R}^d$, $w_k \in \mathbb{R}^d$, $x_k \in \mathbb{R}^d$)

©Kevin Jamieson 2016



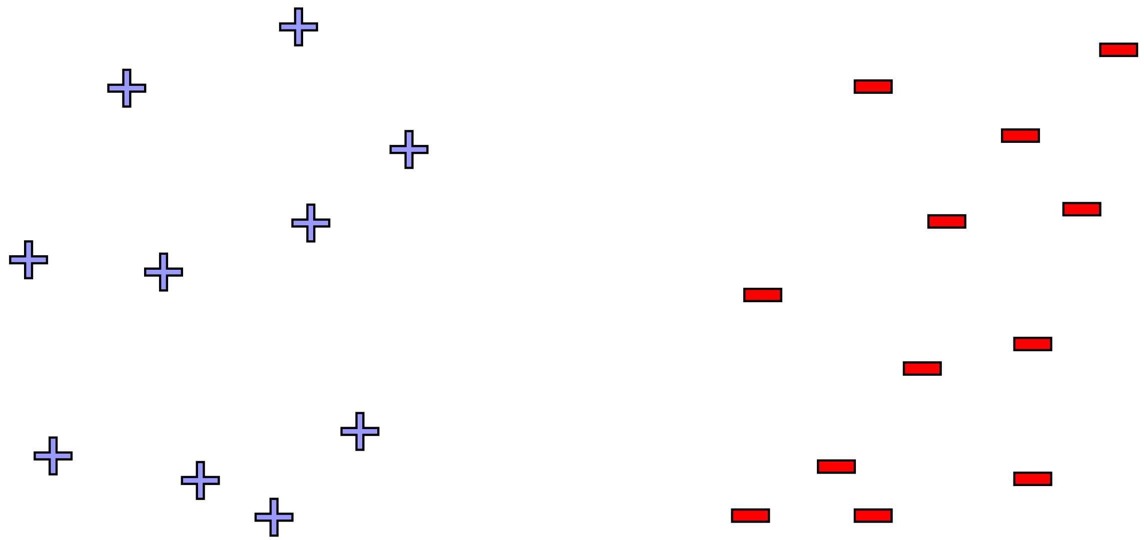
Rosenblatt 1957

"the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

The New York Times, 1958

©Kevin Jamieson 2016

Linear Separability



- Perceptron guaranteed to converge if
  - Data linearly separable:

©Kevin Jamieson 2016

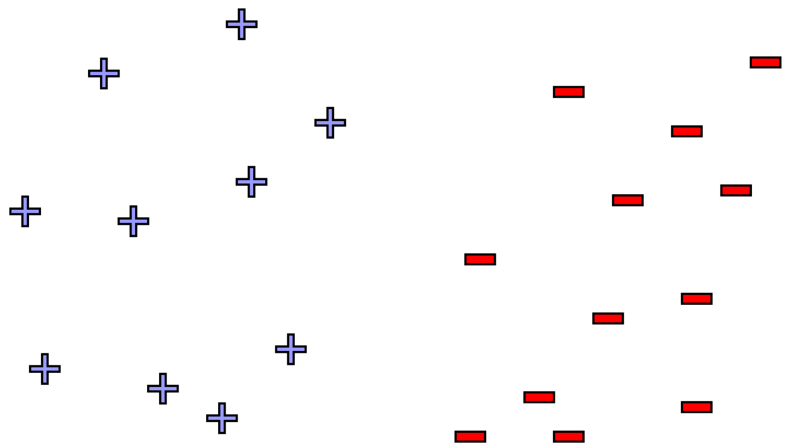
Perceptron Analysis: Linearly Separable Case

- Theorem [Block, Novikoff]:
  - Given a sequence of labeled examples:
  - Each feature vector has bounded norm:
  - If dataset is linearly separable:
- Then the number of mistakes made by the online perceptron on any such sequence is bounded by

©Kevin Jamieson 2016

Beyond Linearly Separable Case

- Perceptron algorithm is super cool!
  - No assumption about data distribution!
    - Could be generated by an oblivious adversary, no need to be iid
  - Makes a fixed number of mistakes, and it's done for ever!
    - Even if you see infinite data

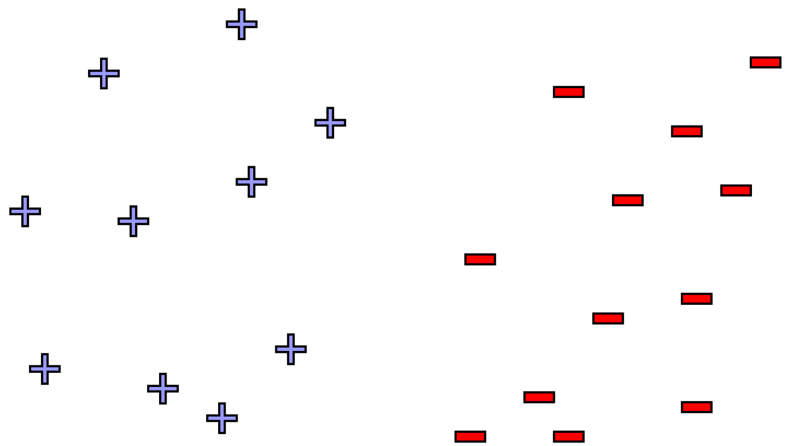


©Kevin Jamieson 2016

Beyond Linearly Separable Case

- Perceptron algorithm is super cool!
  - No assumption about data distribution!
    - Could be generated by an oblivious adversary, no need to be iid
  - Makes a fixed number of mistakes, and it's done for ever!
    - Even if you see infinite data

- Perceptron is useless in practice!
  - Real world not linearly separable
  - If data not separable, cycles forever and hard to detect
  - Even if separable may not give good generalization accuracy (small margin)



What is the Perceptron Doing???

- When we discussed logistic regression:
  - Started from maximizing conditional log-likelihood
- When we discussed the Perceptron:
  - Started from description of an algorithm
- What is the Perceptron optimizing????

©Kevin Jamieson 2016

Support Vector Machines

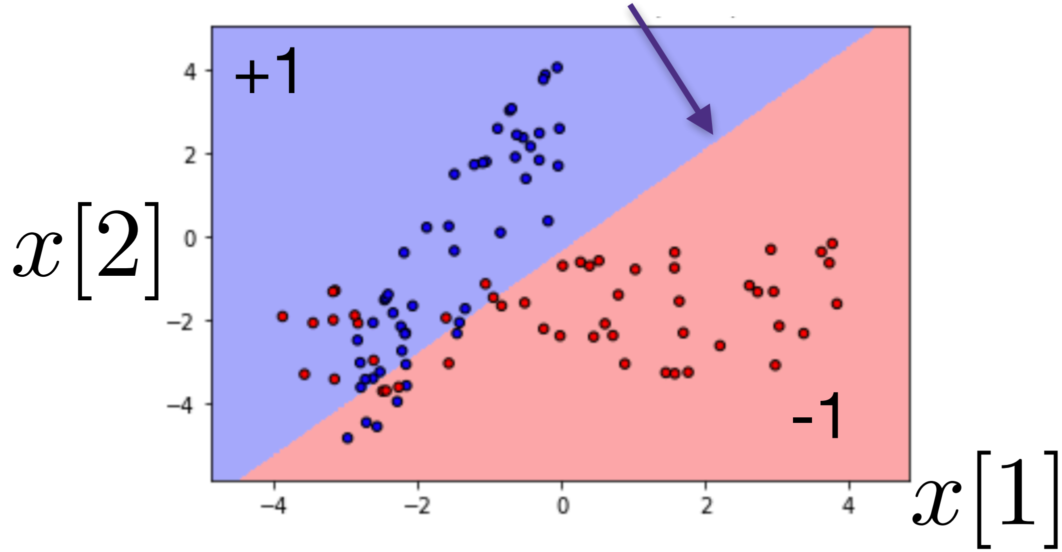
Logistic regression for binary classification

- Data $\mathcal{D} = \{(x_i \in \mathbb{R}^d, y_i \in \{-1, +1\})\}_{i=1}^n$
- Model: $\hat{y} = x^T w + b$
- Loss function: logistic loss $\ell(\hat{y}, y) = \log(1 + e^{-y\hat{y}})$
- Optimization: solve for

$$(\widehat{b}, \widehat{w}) = \arg\min_{b,w} \sum_{i=1}^{n} \log(1 + e^{-y_i(b + x_i^T w)})$$

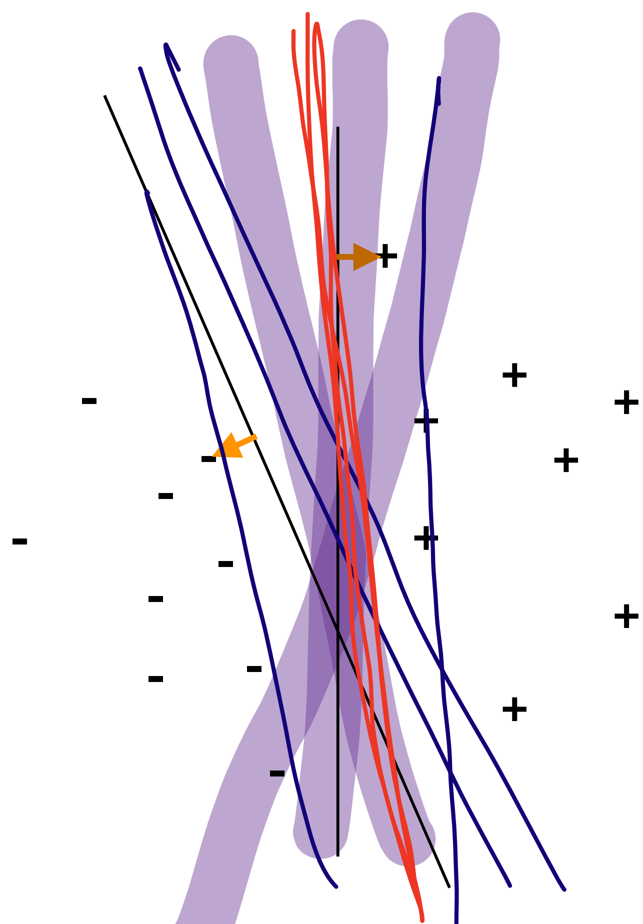
- As this is a **smooth convex** optimization, it can be solved efficiently using gradient descent
- Prediction: $\text{sign}(b + x^T w)$

decision boundary at $w^T x + b = 0$



How do we choose the best linear classifier?

- Informally, **margin** of a set of examples to a decision boundary is the distance to the closest point to the decision boundary
- For linearly separable datasets, **maximum margin** classifier is a natural choice
- Large margin implies that the decision boundary can change without losing accuracy, so the learned model is more robust against new data points



Geometric margin

- Given a set of training examples $\{(x_i, y_i)\}_{i=1}^n$, with $y_i \in \{-1, +1\}$
- and a linear classifier $(w, b) \in \mathbb{R}^d \times \mathbb{R}$
- such that the decision boundary is
  a separating hyperplane $\{x \mid \underbrace{b + w_1 x[1] + w_2 x[2] + \cdots + w_d x[d]}_{w^T x + b} = 0\}$,
  which is the hyperplane orthogonal to w with a shift of b
- we define **margin** of (b, w)
  with respect to a training example (x_i, y_i) as
  the distance from the point (x_i, y_i) to the
  decision boundary, which is

$$\gamma_i = y_i \frac{(w^T x_i + b)}{\|w\|_2}$$

(The proof is on the next slide)