The Perceptron algorithm

Input: A sequence of training examples $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots$ where all $x_i \in \Re^n$, $y_i \in \{-1,1\}$

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example $(\boldsymbol{x}_i, y_i)$:
  - Predict $y' = \text{sgn}(\mathbf{w}_t^T \boldsymbol{x}_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\, \boldsymbol{x}_i)$
- Return final weight vector

The Perceptron algorithm

Input: A sequence of training examples $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots$ where all $x_i \in \Re^n$, $y_i \in \{-1,1\}$

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example $(\boldsymbol{x}_i, y_i)$:
  - Predict $y' = \mathrm{sgn}(\mathbf{w}_t^T \boldsymbol{x}_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\, \boldsymbol{x}_i)$
- Return final weight vector

Remember:
Prediction = $\mathrm{sgn}(\mathbf{w}^T\mathbf{x})$

There is typically a bias term also ($\mathbf{w}^T\mathbf{x}$ + b), but the bias may be treated as a constant feature and folded into **w**

Footnote: For some algorithms it is mathematically easier to represent False as -1, and at other times, as 0. For the Perceptron algorithm, treat -1 as false and +1 as true.

The Perceptron algorithm

Input: A sequence of training examples $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots$ where all $x_i \in \Re^n$, $y_i \in$ {-1,1}

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\, \boldsymbol{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\, \boldsymbol{x}_i$

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example $(\boldsymbol{x}_i, y_i)$:
  - Predict $y' = \mathrm{sgn}(\mathbf{w}_t^T \boldsymbol{x}_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\, \boldsymbol{x}_i)$
- Return final weight vector

The Perceptron algorithm

Input: A sequence of training examples $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots$ where all $x_i \in \Re^n$, $y_i \in \{-1,1\}$

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\, \boldsymbol{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\, \boldsymbol{x}_i$

r is the learning rate, a small positive number less than 1

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example $(\boldsymbol{x}_i, y_i)$:
  - Predict $y' = \mathrm{sgn}(\mathbf{w}_t^T \boldsymbol{x}_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\, \boldsymbol{x}_i)$
- Return final weight vector

The Perceptron algorithm

Input: A sequence of training examples $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots$ where all $x_i \in \Re^n$, $y_i \in \{-1,1\}$

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example $(\boldsymbol{x}_i, y_i)$:
  - Predict $y' = \text{sgn}(\mathbf{w}_t^T \boldsymbol{x}_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\, \boldsymbol{x}_i)$
- Return final weight vector

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,\boldsymbol{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\,\boldsymbol{x}_i$

r is the learning rate, a small positive number less than 1

Update only on error. A mistake-driven algorithm

The Perceptron algorithm

Input: A sequence of training examples $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots$
where all $x_i \in \Re^n$, $y_i \in \{-1,1\}$

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example $(\boldsymbol{x}_i, y_i)$:
  - Predict $y' = \text{sgn}(\mathbf{w}_t^T \boldsymbol{x}_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\, \boldsymbol{x}_i)$
- Return final weight vector

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\, \boldsymbol{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\, \boldsymbol{x}_i$

r is the learning rate, a small positive number less than 1

Update only on error. A mistake-driven algorithm

This is the simplest version. We will see more robust versions at the end

The Perceptron algorithm

Input: A sequence of training examples $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots$ where all $x_i \in \Re^n$, $y_i \in \{-1,1\}$

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example $(\boldsymbol{x}_i, y_i)$:
  - Predict $y' = \text{sgn}(\mathbf{w}_t^T \boldsymbol{x}_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\, \boldsymbol{x}_i)$
- Return final weight vector

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\, \boldsymbol{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\, \boldsymbol{x}_i$

r is the learning rate, a small positive number less than 1

Update only on error. A mistake-driven algorithm

This is the simplest version. We will see more robust versions at the end

Mistake can be written as $y_i \mathbf{w}_t^T \boldsymbol{x}_i \leq 0$

Intuition behind the update

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\, \boldsymbol{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\, \boldsymbol{x}_i$

Suppose we have made a mistake on a positive example

That is, y = +1 and $\mathbf{w}_t^{\mathsf{T}}\boldsymbol{x} < 0$

Call the new weight vector $\mathbf{w}_{t+1}$.

The update means $\mathbf{w}_{t+1} = \mathbf{w}_t + \boldsymbol{x}$ (say r = 1)

The new dot product will be $\mathbf{w}_{t+1}^{\mathsf{T}}\boldsymbol{x} = (\mathbf{w}_t + \boldsymbol{x})^{\mathsf{T}}\boldsymbol{x} = \mathbf{w}_t^{\mathsf{T}}\boldsymbol{x} + \boldsymbol{x}^{\mathsf{T}}\boldsymbol{x} \geq \mathbf{w}_t^{\mathsf{T}}\boldsymbol{x}$

For a positive example, the Perceptron update will increase the score assigned to the same input

Similar reasoning for negative examples

1. The "standard" algorithm

Given a training set D = $\{(\mathbf{x}_i, y_i)\}$, $\mathbf{x}_i \in \Re^n$, $y_i \in \{-1,1\}$

1. Initialize **w** = $0 \in \Re^n$
2. For epoch = 1 … T:
    1. Shuffle the data
    2. For each training example $(\mathbf{x}_i, y_i) \in$ D:
        - If $y_i\, \mathbf{w}^T\mathbf{x}_i \leq 0$, update $\mathbf{w} \leftarrow \mathbf{w} + \underline{r}\, y_i\, \mathbf{x}_i$
3. Return **w**

Prediction: sgn($\mathbf{w}^T\mathbf{x}$)