The Perceptron

CMSC 422

MARINE CARPUAT
marine@cs.umd.edu

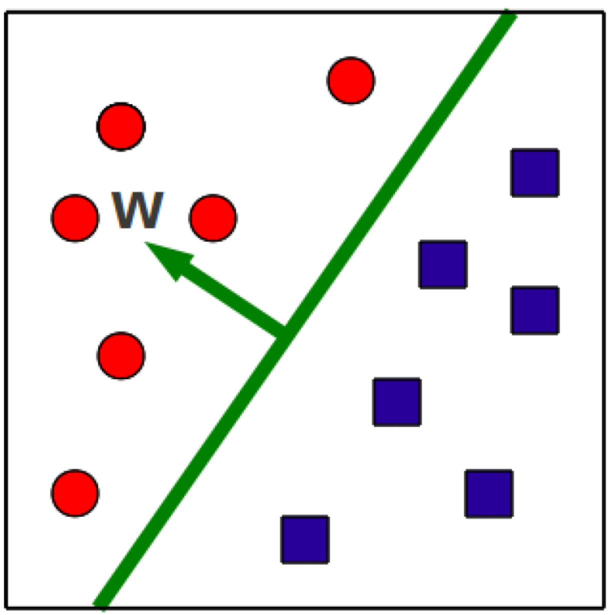
Credit: figures by Piyush Rai and Hal Daume III

This week

- The perception: a new model/algorithm
  - **its variants: voted, averaged**
  - **convergence proof**
- Fundamental Machine Learning Concepts
  - Online vs. batch learning
  - Error-driven learning
  - **Linear separability and margin of a dataset**

- Project 1 published today

Recap: Perceptron for binary classification



- Classifier = hyperplane that separates positive from negative examples

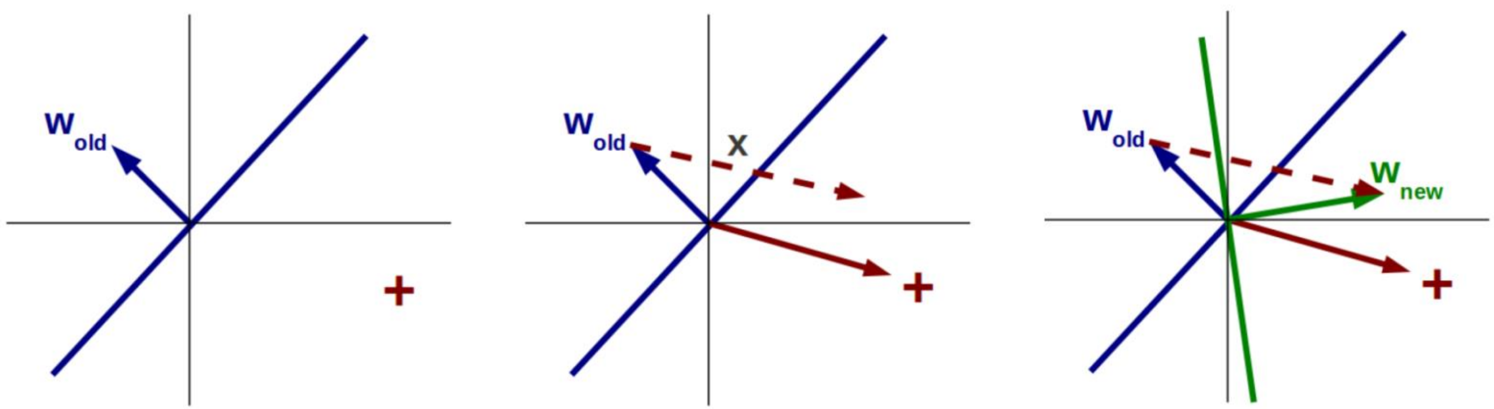
$$\hat{y} = sign(w^T x + b)$$

- Perceptron training
  - Finds such a hyperplane
  - Online & error-driven

Recap: Perceptron updates

Update for a misclassified positive example:

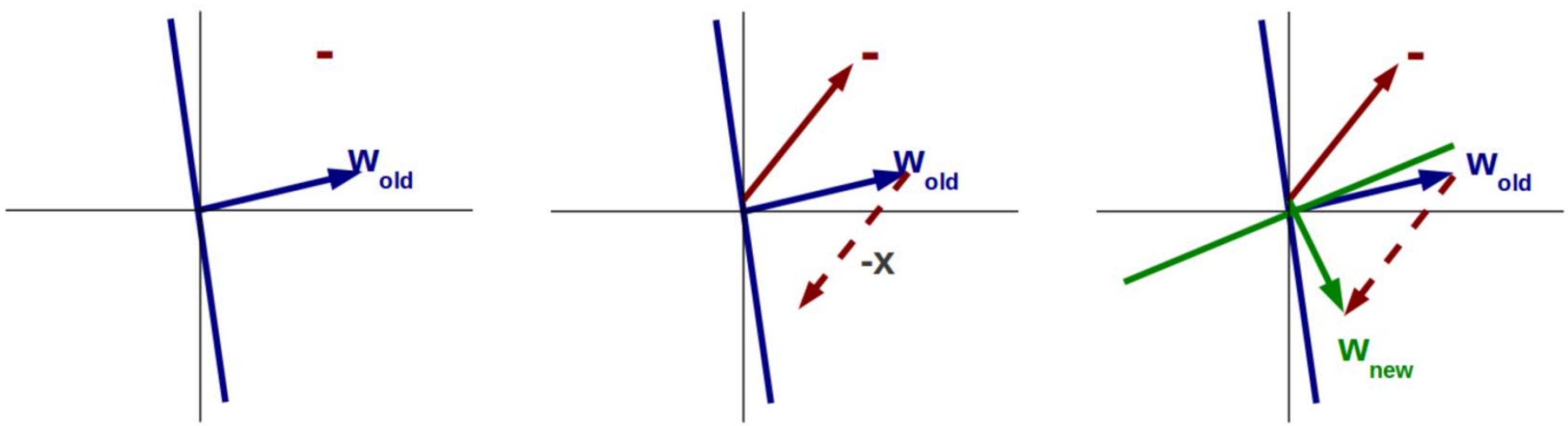
$$\mathbf{w}_{new} = \mathbf{w}_{old} + \mathbf{x}$$



Recap: Perceptron updates

Update for a misclassified negative example:

$$\mathbf{w}_{new} = \mathbf{w}_{old} - \mathbf{x}$$



Standard Perceptron: predict based on final parameters

Algorithm 5 $\textsc{PerceptronTrain}(\mathbf{D}, \mathit{MaxIter})$

$w_d \leftarrow 0$, for all $d = 1 \dots D$ // initialize weights

$b \leftarrow 0$ // initialize bias

for $\mathit{iter} = 1 \dots \mathit{MaxIter}$ **do**

  for all $(x,y) \in \mathbf{D}$ **do**

    $a \leftarrow \sum_{d=1}^{D} w_d\, x_d + b$ // compute activation for this example

    if $ya \leq 0$ **then**

      $w_d \leftarrow w_d + y x_d$, for all $d = 1 \dots D$ // update weights

      $b \leftarrow b + y$ // update bias

    end if

  end for

end for

return $w_0, w_1, \dots, w_D, b$

Predict based on final + intermediate parameters

- The voted perceptron

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(\mathsf{k})}\text{sign}\left(\boldsymbol{w}^{(\mathsf{k})} \cdot \hat{\boldsymbol{x}} + b^{(\mathsf{k})}\right)\right)$$

- The averaged perceptron

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(\mathsf{k})}\left(\boldsymbol{w}^{(\mathsf{k})} \cdot \hat{\boldsymbol{x}} + b^{(\mathsf{k})}\right)\right)$$

- Require keeping track of "survival time" of weight vectors $c^{(1)}, \ldots, c^{(\mathsf{K})}$

How would you modify this algorithm for voted perceptron?

Algorithm 5 PerceptronTrain($\mathbf{D}$, *MaxIter*)

```
w_d ← 0, for all d = 1 ... D                          // initialize weights
b ← 0                                                 // initialize bias
for iter = 1 ... MaxIter do
    for all (x,y) ∈ D do
        a ← Σ_{d=1}^{D} w_d x_d + b                   // compute activation for this example
        if ya ≤ 0 then
            w_d ← w_d + y x_d, for all d = 1 ... D    // update weights
            b ← b + y                                 // update bias
        end if
    end for
end for
return w_0, w_1, ..., w_D, b
```

How would you modify this algorithm for averaged perceptron?

Algorithm 5 PERCEPTRONTRAIN(**D**, *MaxIter*)

$w_d \leftarrow 0$, for all $d = 1 \dots D$ // initialize weights
$b \leftarrow 0$ // initialize bias
for $iter = 1 \dots MaxIter$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $a \leftarrow \sum_{d=1}^{D} w_d \, x_d + b$ // compute activation for this example
    if $ya \leq 0$ **then**
      $w_d \leftarrow w_d + yx_d$, for all $d = 1 \dots D$ // update weights
      $b \leftarrow b + y$ // update bias
    end if
  end for
end for
return $w_0, w_1, \dots, w_D, b$

Averaged perceptron decision rule

$$\hat{y} = \operatorname{sign}\left(\sum_{k=1}^{K} c^{(\mathrm{k})}\left(\boldsymbol{w}^{(\mathrm{k})} \cdot \hat{\boldsymbol{x}} + b^{(\mathrm{k})}\right)\right)$$

can be rewritten as

$$\hat{y} = \operatorname{sign}\left(\left(\sum_{k=1}^{K} c^{(\mathrm{k})}\boldsymbol{w}^{(\mathrm{k})}\right) \cdot \hat{\boldsymbol{x}} + \sum_{k=1}^{K} c^{(\mathrm{k})} b^{(\mathrm{k})}\right)$$

Averaged Perceptron Training

Algorithm 7 $\textsc{AveragedPerceptronTrain}(\mathbf{D}, \mathit{MaxIter})$

$\boldsymbol{w} \leftarrow \langle 0, 0, \ldots 0 \rangle$, $b \leftarrow 0$ // initialize weights and bias
$\boldsymbol{u} \leftarrow \langle 0, 0, \ldots 0 \rangle$, $\beta \leftarrow 0$ // initialize cached weights and bias
$c \leftarrow 1$ // initialize example counter to one
for $\mathit{iter} = 1 \ldots \mathit{MaxIter}$ **do**
    for all $(\boldsymbol{x}, y) \in \mathbf{D}$ **do**
        if $y(\boldsymbol{w} \cdot \boldsymbol{x} + b) \leq 0$ **then**
            $\boldsymbol{w} \leftarrow \boldsymbol{w} + y\,\boldsymbol{x}$ // update weights
            $b \leftarrow b + y$ // update bias
            $\boldsymbol{u} \leftarrow \boldsymbol{u} + y\,c\,\boldsymbol{x}$ // update cached weights
            $\beta \leftarrow \beta + y\,c$ // update cached bias
        end if
        $c \leftarrow c + 1$ // increment counter regardless of update
    end for
end for
return $\boldsymbol{w} - \frac{1}{c}\,\boldsymbol{u},\ b - \frac{1}{c}\,\beta$ // return averaged weights and bias

Can the perceptron always find a hyperplane to separate positive from negative examples?

Convergence of Perceptron

- The perceptron has converged if it can classify every training example correctly
  - i.e. if it has found a hyperplane that correctly separates positive and negative examples

- Under which conditions does the perceptron converge and how long does it take?

Convergence of Perceptron

Theorem (Block & Novikoff, 1962)

If the training data $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ is **linearly separable** with margin γ by a unit norm hyperplane w_* ($||w_*|| = 1$) with $b = 0$,

Then **perceptron training converges after $\frac{R^2}{\gamma^2}$ errors** during training

(assuming ($||x|| < R$) for all x).

Margin of a data set *D*

$$margin(\mathbf{D}, \boldsymbol{w}, b) = \begin{cases} \min_{(\boldsymbol{x},y)\in\mathbf{D}} y(\boldsymbol{w}\cdot\boldsymbol{x}+b) & \text{if } \boldsymbol{w} \text{ separates } \mathbf{D} \\ -\infty & \text{otherwise} \end{cases} \tag{4.8}$$

Distance between the hyperplane (w,b) and the nearest point in **D**

$$margin(\mathbf{D}) = \sup_{\boldsymbol{w},b} margin(\mathbf{D}, \boldsymbol{w}, b) \tag{4.9}$$

Largest attainable margin on **D**

Theorem (Block & Novikoff, 1962)

If the training data $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ is **linearly separable** with margin γ by a unit norm hyperplane w_* ($||w_*|| = 1$) with $b = 0$, then **perceptron training converges after $\frac{R^2}{\gamma^2}$ errors** during training (assuming ($||x|| < R$) for all x).

Proof:

- Margin of $\mathbf{w}_*$ on any *arbitrary example* $(\mathbf{x}_n, y_n)$: $\frac{y_n \mathbf{w}_*^T \mathbf{x}_n}{||\mathbf{w}_*||} = y_n \mathbf{w}_*^T \mathbf{x}_n \geq \gamma$
- Consider the $(k+1)^{th}$ mistake: $y_n \mathbf{w}_k^T \mathbf{x}_n \leq 0$, and update $\mathbf{w}_{k+1} = \mathbf{w}_k + y_n \mathbf{x}_n$
- $\mathbf{w}_{k+1}^T \mathbf{w}_* = \mathbf{w}_k^T \mathbf{w}_* + y_n \mathbf{w}_*^T \mathbf{x}_n \geq \mathbf{w}_k^T \mathbf{w}_* + \gamma$
- Repeating iteratively k times, we get $\mathbf{w}_{k+1}^T \mathbf{w}_* > k\gamma$ (1)
- $||\mathbf{w}_{k+1}||^2 = ||\mathbf{w}_k||^2 + 2 y_n \mathbf{w}_k^T \mathbf{x}_n + ||\mathbf{x}||^2 \leq ||\mathbf{w}_k||^2 + R^2$ (since $y_n \mathbf{w}_k^T \mathbf{x}_n \leq 0$)
- Repeating iteratively k times, we get $||\mathbf{w}_{k+1}||^2 \leq kR^2$ (2)

Theorem (Block & Novikoff, 1962)

If the training data $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ is **linearly separable** with margin γ by a unit norm hyperplane w_* ($||w_*|| = 1$) with $b = 0$, then **perceptron training converges after $\frac{R^2}{\gamma^2}$ errors** during training (assuming ($||x|| < R$) for all x).

What does this mean?

- Perceptron converges quickly when margin is large, slowly when it is small
- Bound does not depend on number of training examples N, nor on number of features
- Proof guarantees that perceptron converges, but not necessarily to the max margin separator

What you should know

- Perceptron concepts
  - training/prediction algorithms (standard, voting, averaged)
  - convergence theorem and what practical guarantees it gives us
  - how to draw/describe the decision boundary of a perceptron classifier
- Fundamental ML concepts
  - Determine whether a data set is linearly separable and define its margin
  - Error driven algorithms, online vs. batch algorithms