Introduction to Machine Learning

CSE474/574: Lecture 6

Varun Chandola `<chandola@buffalo.edu>`

6 Feb 2015

Outline

Contents

1 Analyzing Online Learning Algorithms

1.1 Halving Algorithm

- $\xi_0(\mathcal{C}, x) = \{c \in \mathcal{C} : c(x) = 0\}$
- $\xi_1(\mathcal{C}, x) = \{c \in \mathcal{C} : c(x) = 1\}$

```
1: $V_0 \leftarrow \mathcal{C}$
2: for $i = 1, 2, \ldots$ do
3:   if $|\xi_1(V_{i-1}, x^{(i)})| \geq |\xi_0(V_{i-1}, x^{(i)})|$ then
4:     predict $c(x^{(i)}) = 1$
5:   else
6:     predict $c(x^{(i)}) = 0$
7:   end if
8:   if $c(x^{(i)}) \neq c_*(x^{(i)})$ then
9:     $V_i = V_{i-1} - \xi_{c(x^{(i)})}(\mathcal{C}, x^{(i)})$
10:  end if
11: end for
```

- Every mistake results in halving of the version space
- Not computationally feasible
  - Need to store and access the version space

- Are there any *efficient implementable* learning algorithms
  - With comparable mistake bounds

2 Learning Monotone Disjunctions

- A restricted concept class: **monotone disjunctions** of at most k variables
  - $\mathcal{C} = \{x_{i_1} \vee x_{i_2} \vee \ldots x_{i_k}\}$
  - $|\mathcal{C}|$?
  - Mistake bound $= \log_2 |\mathcal{C}|$
- An efficient algorithm - *Winnow*

What are monotone disjunctions? These are disjunctive expressions in which no entry appears negated.

The size of the concept space can be determined by counting the number possible concepts with $0, 1, \ldots, k$ variables:

$$|\mathcal{C}| = \binom{d}{k} + \binom{d}{k-1} + \ldots + \binom{d}{0}$$

Note that $\log_2 |\mathcal{C}| = \Theta(k \log_2 d)$

2.1 Linearly Separable Concepts

- Concept c is linearly separable if $\exists w \in \Re^d, \Theta \in \Re$ such that:

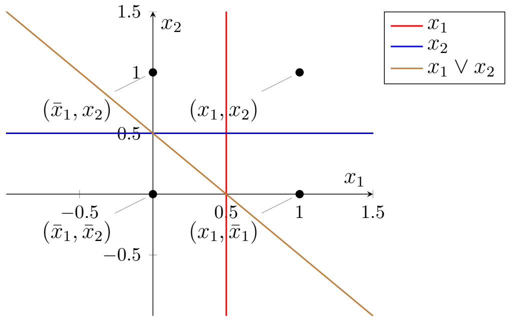
$$\forall x, c(x) = 1 \Leftrightarrow w^\top x \geq \Theta$$

$w^\top x$ denotes the inner product between two vector. Since $x \in \{0,1\}^d$, the inner product is essentially the sum of the weights corresponding to attributes which are 1 for x.

- Monotone disjunctions **are linearly separable**
- For a disjunction $x_{i_1} \vee x_{i_2} \vee \ldots x_{i_k}$

$$x_{i_1} + x_{i_2} + \ldots + x_{i_k} = \frac{1}{2}$$

separates the points labeled 1 and 0 by the disjunctive concept.



2.2 Winnow Algorithm

The name *winnow* comes from the fact that the algorithm finds the k attributes out of a large number of attributes, most of which ($d-k$ to be exact) are not useful.

```
1: $\Theta \leftarrow \frac{d}{2}$
2: $w \leftarrow (1, 1, \ldots, 1)$
3: for $i = 1, 2, \ldots$ do
4:   if $w^\top x^{(i)} \geq \Theta$ then
5:     $c(x^{(i)}) = 1$
6:   else
7:     $c(x^{(i)}) = 0$
8:   end if
9:   if $c(x^{(i)}) \neq c_*(x^{(i)})$ then
10:    if $c_*(x^{(i)}) = 1$ then
11:      $\forall j : x_j^{(i)} = 1, w_j \leftarrow \alpha w_j$
12:    else
13:      $\forall j : x_j^{(i)} = 1, w_j \leftarrow 0$
14:    end if
15:  end if
16: end for
```

- *Move* the hyperplane when a mistake is made
- $\alpha > 1$, typically set to 2
- Θ is often set to $\frac{d}{2}$
- *Promotions* and *eliminations*

2.3 Analyzing Winnow

- Winnow1 makes $O(k \log_\alpha d)$ mistakes
- Optimal mistake bound
- One can use different values for α and Θ
- Other variants exist
  - *Arbitrary* disjunctions
  - k-DNF (disjunctive normal forms)
    - $(x_1 \wedge x_2) \vee (x_4) \vee (x_7 \wedge \neg x_3)$

The Winnow1 algorithm is a type of a linear threshold classifier which divides the input space into two regions using a hyperplane.

3 Perceptrons

- Human brain has 10^{11} neurons
- Each connected to 10^4 neighbors

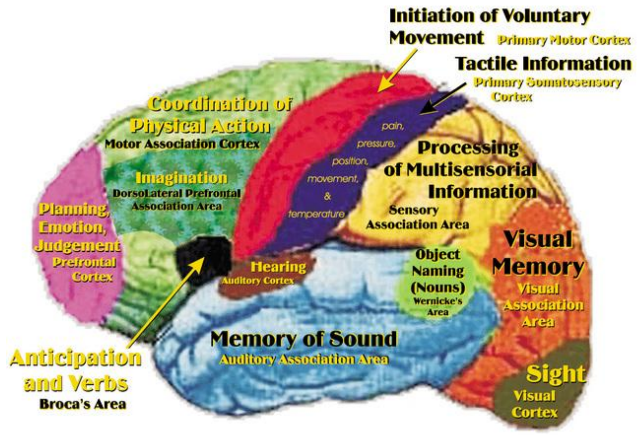


Figure 1: *Src: http://brainjackimage.blogspot.com/*

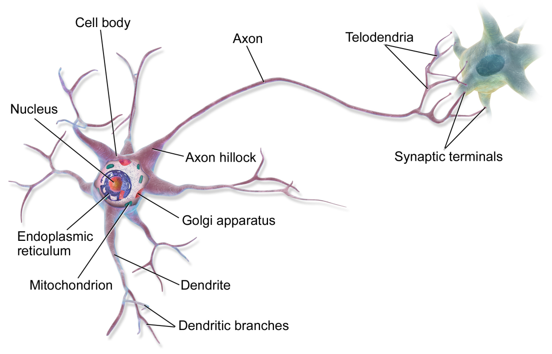
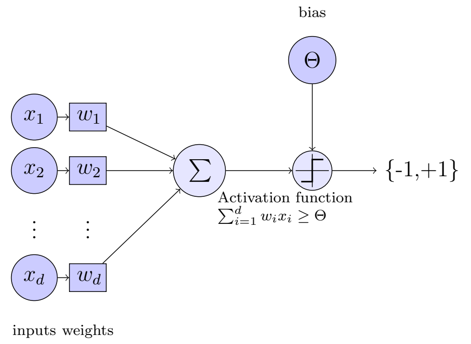


Figure 2: *Src: Wikipedia*

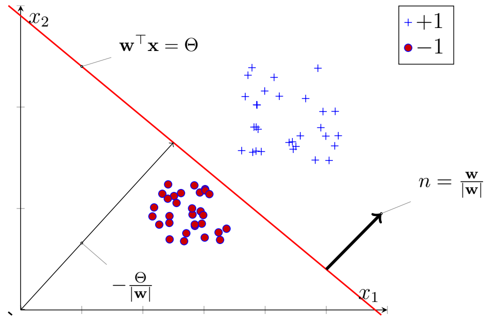
So far we have focused on learning concepts defined over d binary attributes. Now we generalize this to the case when the attributes are numeric, i.e., they can take infinite values. Thus the concept space becomes infinite. How does one search for the target concept in this space and how is their performance analyzed. We begin with the simplest, but very effective, algorithm for learning in such a setting, called **perceptrons**. Perceptron algorithm was developed by Frank Rosenblatt [2] at the Calspan company in Cheetowaga, NY. In fact, it is not only the first machine learning algorithm, but also the first to be implemented in hardware, by Rosenblatt. The motivation for perceptrons comes directly from the model by McCulloch-Pitts [1] for the artificial neurons. The neuron model states that multiple inputs enter the main neuron (*soma*) through multiple *dendrites* and the output is sent out through a single *axon*.



Simply put, a perceptron computes a weighted sum of the input attributes and adds a bias term. The sum is compared to a threshold to classify the input as $+1$ or -1.

The bias term Θ signifies that the weighted sum of the actual attributes should be greater than Θ to become greater than 0.

3.1 Geometric Interpretation



1. The hyperplane, characterized by $\mathbf{w}$ is also known as the decision boundary (or surface).
2. The decision boundary divides the input space into two *half-spaces*.
3. Changing $\mathbf{w}$ *rotates* the hyperplane
4. Changing Θ *translates* the hyperplane
5. Hyperplane passes through the origin if $\Theta = 0$

3.2 Perceptron Training

- Add another attribute $x_{d+1} = 1$.
- w_{d+1} is $-\Theta$
- Desired hyperplane goes through origin in $(d+1)$ space
- **Assumption**: $\exists \mathbf{w} \in \Re^{d+1}$ such that $\mathbf{w}$ can *strictly* classify all examples correctly.

- *Hypothesis space*: Set of all hyperplanes defined in the $(d+1)$-dimensional space passing through origin
  - The target hypothesis is also called **decision surface** or **decision boundary**.

```
1: $w \leftarrow (0, 0, \ldots, 0)_{d+1}$
2: for $i$=1, 2, ... do
3:   if $w^\top x^{(i)} > 0$ then
4:     $c(x^{(i)}) = 1$
5:   else
6:     $c(x^{(i)}) = 0$
7:   end if
8:   if $c(x^{(i)}) \neq c_*(x^{(i)})$ then
9:     $w \leftarrow w + c_*(x^{(i)})x^{(i)}$
10:  end if
11: end for
```

- Every mistake *tweaks* the hyperplane
  - Rotation in $(d+1)$ space
  - Accomodate the offending point
- Stopping Criterion:
  - Exhaust all training example, or
  - No further updates

To demonstrate the working of perceptron training algorithm, let us consider a simple 2D example in which all data points are located on a unit circle.

To get an intuitive sense of why this training procedure should work, we should note the following: Every mistake makes $\mathbf{w}^\top \mathbf{x}$ become more positive (if $c_*(x) = 1$) or more negative (if $c_*(x) = -1$).

Let $c_*(x) = 1$. The new $\mathbf{w}' = \mathbf{w} + \mathbf{x}$. Hence, by substitution, $\mathbf{w}'^\top \mathbf{x} = (\mathbf{w} + \mathbf{x})^\top \mathbf{x} = \mathbf{w}^\top \mathbf{x} + \mathbf{x}^\top \mathbf{x}$. The latter quantity is always positive, thereby making $\mathbf{w}^\top \mathbf{x}$ more positive.

Thus whenever a mistake is made, the surface is "moved" to accomodate that mistake by increasing or decreasing $\mathbf{w}^\top \mathbf{x}$.

Sometimes a "learning rate" (η) is used to update the weights.

References

[1] W. Mcculloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

[2] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.