Notes from *Learning From Data*

David Lago

July 20, 2014

1 The learning problem

Components of learning:

1. The **input** $\mathbf{x} \in \mathcal{X}$
2. The **output** $y \in \mathcal{Y}$
3. The *unknown* **target function** $f\colon \mathcal{X} \longrightarrow \mathcal{Y}$
4. The **training dataset** $\mathcal{D} = \{(\mathbf{x}_1, y_1), \cdots 1, (\mathbf{x}_N, y_N)\}$, where $y_i = f(\mathbf{x}_i)\ \forall i = 1, \cdots, N$
5. The **hypothesis set** $\mathcal{H}$, which is a set of functions $h\colon \mathcal{X} \longrightarrow \mathcal{Y}$
6. The **learning algorithm** $\mathcal{A}$, which based on the training dataset $\mathcal{D}$ chooses from among all $h \in \mathcal{H}$ the one that best approximates f
7. The **final hypothesis** $g \approx f$, as chosen by $\mathcal{A}$

We are going to call $\mathcal{A}$ and $\mathcal{H}$ our **learning model**, since they are the only two components we have control over.

One simple learning model is the **perceptron model**, whose hyphothesis functions are of the form:

$$h(\mathbf{x}) = \mathrm{sign}((\sum_{i=1}^{d} w_i x_i) + b), \tag{1}$$

where

- $x_1, \cdots, x_d$ are the components of vector $\mathbf{x}$
- $y \in \mathcal{Y} = \{+1, -1\}$
- b is the threshold ($y = +1 \Rightarrow \sum_{i=1}^{d} w_i x_i > -b$)

The **perceptron learning algorithm**, or PLA, will look for different h's by varying the weights vector $\mathbf{w}$ and the bias b. To simplify equation 1, we can add b to the weights vector as w_0, so that $\mathbf{w} = [w_0, w_1, \cdots, w_d]^T$, and add $x_0 = 1$ to $\mathbf{x}$ so that now $\mathbf{x} = [1, x_1, \cdots, x_d]^T$ (x and w are *column vectors*). With these changes, the perceptron hypothesis equation can be reduced to:

$$h(\mathbf{x}) = \mathrm{sign}(\mathbf{w}^T \mathbf{x}). \tag{2}$$

PLA is an iterative algorithm. Its steps are:

1. Initialize $\mathbf{w}$ (any value, $\mathbf{w} = [0, \cdots, 0]^T$ for example)
2. Calculate $h(\mathbf{x})$ with current $\mathbf{w}$
3. Compare $h(\mathbf{x_i})$ with y_i for all $i = 1, \cdots, N$
4. If no mismatches, return current h as our g
5. Otherwise, pick one of the misclassified $\mathbf{x}$'s and update the weights vector:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + y(t)\mathbf{x}(t).$$

Exercise 1.3

Show that $y(t)\mathbf{w}^T(t)\mathbf{x}(t) < 0$ for a misclassified point:

$$\begin{aligned} y(t)\mathbf{w}^T(t)\mathbf{x}(t) &= -\operatorname{sign}(\overbrace{\mathbf{w}^T(t)\mathbf{x}(t)}^{a})\,\overbrace{\mathbf{w}^T(t)\mathbf{x}(t)}^{a} \\ &= -\operatorname{sign}(a)a \\ &< 0. \end{aligned}$$

Show that $y(t)\mathbf{w}^T(t+1)\mathbf{x}(t) > y(t)\mathbf{w}^T(t)\mathbf{x}(t)$:

$$\begin{aligned} y(t)\mathbf{w}^T(t+1)\mathbf{x}(t) &> y(t)\mathbf{w}^T(t)\mathbf{x}(t) \\ y(t)(\mathbf{w}^T(t) + y(t)\mathbf{x}^T)(t))\mathbf{x}(t) &> y(t)\mathbf{w}^T(t)\mathbf{x}(t) \\ y(t)\mathbf{w}^T(t)\mathbf{x}(t) + y^2(t)\mathbf{x}^T(t)\mathbf{x}(t) &> y(t)\mathbf{w}^T(t)\mathbf{x}(t) \\ \overbrace{y^2(t)}^{>0}\,\overbrace{\mathbf{x}^T(t)\mathbf{x}(t)}^{>0} &> 0. \end{aligned}$$

Show that $\mathbf{w}(t+1)$ is an improvement over $\mathbf{w}(t)$:

In order to show this, we have to prove that $h_{\mathbf{w}(t+1)}(\mathbf{x}_i) = y_i = -h_{\mathbf{w}(t)}(\mathbf{x}_i)$ for the misclassified $\mathbf{x}_i$:

$$\begin{aligned} h_{\mathbf{w}(t+1)}(\mathbf{x}_i) &= \operatorname{sign}(\mathbf{w}^T(t+1)\mathbf{x}(t)) \\ &= \operatorname{sign}((\mathbf{w}^T(t) + y(t)\mathbf{x}^T(t))\mathbf{x}(t)) \\ &= \operatorname{sign}(\overbrace{\mathbf{w}^T(t)\mathbf{x}(t)}^{y(t)} + y(t)\mathbf{x}^T(t)\mathbf{x}(t)) \\ &= \operatorname{sign}(y(t)(\overbrace{1 + \mathbf{x}^T(t)\mathbf{x}(t)}^{>0})) \\ &= \operatorname{sign}(y(t)) \\ &= y(t). \end{aligned}$$

Exercise 1.4

The solution to this exercise has been implemented in Octave. Figures 1, 2 and 3 contain the source code:

Figure 1: ex_1_4.m

```
% This exercise creates a random target function f. Generates 20 samples, and then
    applies
% the perceptron algorithm to predict

% Generate 20 random points between -10 and 10
x=[ones(20,1) 20.*rand(20,2)-10];

% Initial w for f
w0 = [1; 2; 4];

% Obtain their f(x)
for i = 1:20
    y(i) = f(x(i,:)',w0);
end
y = y';

% Plot our sample
plot(x(:,2)(y>0),x(:,3)(y>0),'ro');
hold on;
plot(x(:,2)(y<0),x(:,3)(y<0),'bx');

% Plot target f in black
axis=-10:0.1:10;
fline=(-w0(1)/w0(3))-(w0(2)/w0(3))*axis;
plot(axis,fline,'k');


% Use perceptron to calculate g
w = perceptron(x,y);

% Plot g in green
gline=(-w(1)/w(3))-(w(2)/w(3))*axis;
plot(axis,gline,'g');
legend('+1', '-1', 'f', 'g','location','north');
legend('boxoff');
hold off;
```

Figure 2: `perceptron.m`

```
function w = perceptron(x,y)

    % Initialize w to all zeros
    m = size(x)(1);
    n = size(x)(2);
    w=zeros(n,1);

    found = true;
    iter = 0;
    maxiter = 1000;

    while (found) % Repeat while misclassified points exist (or maxiter reached)

        found = false;
        iter++;
        if (iter == maxiter)
            break;
        end

        % Calculate output with current w
        y2 = sign(w'*x')';

        % See if there is any misclassified point
        for i=1:m
            if (y(i)!=y2(i))
                found = true;

                % Adjust weights
                w=w+(y(i)*x(i,:))';

                break;
            end
        end

    end
```

Figure 3: `f.m`

```
function y = f(x,w)

    y = sign(w'*x);
```

If we have a probability distribution whose mean μ we are trying to know by means of extracting random samples and observing their proportion ν, the **Hoeffding inequality** bounds the difference between μ and ν as follows:

$$P[|\nu - \mu| > \epsilon] \leq 2e^{-2\epsilon^2 N} \text{ for any } \epsilon > 0. \tag{3}$$

As the sample size N increases, it becomes exponentially less likely that ν will deviate from μ by more than ϵ. Only N affects the bound, not the size of the (unknown) population.

Translating this into our learning model, we can see our unknown distribution as our input space $\mathcal{X}$, with μ being the fraction of inputs $\mathbf{x}$ for which $h(\mathbf{x}) \neq f(\mathbf{x})$ (the hypothesis incorrectly predicts the value for that input). In the same way, our sample would be our dataset $\mathcal{D}$, and ν the proportion of $\mathcal{X}$ for which h incorrectly fits the sample. If we define $E_{in}(h)$ (*in error sample*) as the fraction of $\mathcal{D}$ where f and h disagree, and E_{out} (*out of error sample*) as the fraction of $\mathcal{X}$ where f and h disagree, equation 3 can be rewritten as follows:

$$E_{in}(h) = \frac{1}{N}\sum_{n=1}^{N}[[h(\mathbf{x}_n) \neq f(\mathbf{x}_n)]],\ E_{out}(h) = \mathbb{P}[h(\mathbf{x}) \neq f(\mathbf{x})]$$

$$P[|E_{in}(h) - E_{out}(h)| > \epsilon] \leq 2e^{-2\epsilon^2 N} \text{ for any } \epsilon > 0, \tag{4}$$

where operator [[statement]] is equal to 1 if statement is true, 0 otherwise. Extending equation 4 for M possible h's in $\mathcal{H}$:

$$P[|E_{in}(h) - E_{out}(h)| > \epsilon] \leq 2Me^{-2\epsilon^2 N} \text{ for any } \epsilon > 0, \tag{5}$$

It is because of Hoeffding's that we can use E_{in} as a proxy for E_{out}. We'll try to find an h that makes $E_{in} \approx 0$. There is a tradeoff: we want to make $\mathcal{H}$ as simple as possible so that M is small and the bounds dictated by equation 5 are tighter, but at the same time we want to make $\mathcal{H}$ complex enough so that it gives us more flexibility to fit the data well.