MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.036—Introduction to Machine Learning
Spring Semester 2016

Assignment 1: SOLUTIONS

Issued: Friday, Feb. 2nd, Due: Friday, Feb. 12th

Classification and the Perceptron Algorithm

The Perceptron Algorithm (with offset):

$\theta^{(0)} = 0$
$\theta_0^{(0)} = 0$
for $i = 1$ to n **do**
  if $y^{(i)} \cdot \text{sgn}(\theta^{(k)} \cdot x^{(i)} + \theta_0^{(k)}) \leq 0$ **then**
    # kth update on ith data point
    $\theta^{(k+1)} = \theta^{(k)} + y^{(i)} x^{(i)}$
    $\theta_0^{(k+1)} = \theta_0^{(k)} + y^{(i)} 1$
  end if
end for

1. Perceptron Mistakes

(a) Let $S_n = \{(x^{(i)}, y^{(i)}) : i = 1 \dots n\}$ be our training set. Consider a training set

$$S_3 = \{([-1, 1]^T, +1), ([0, -1]^T, -1), ([1.5, 1]^T, +1)\}$$

Starting with $x^{(1)} = [-1, 1]^T, y^{(1)} = +1$, let's compute the number of updates k required until convergence.

i=0: $\theta^{(0)} = 0$
i=1: $y^{(1)} \cdot \text{sgn}(\theta^{(0)} \cdot x^{(1)}) = (+1) \cdot \text{sgn}(0 \cdot [-1, 1]^T) = 0$
$\theta^{(1)} = \theta^{(0)} + y^{(1)} x^{(1)} = 0 + (+1) x^{(1)} = [-1, 1]^T$
i=2: $y^{(2)} \cdot \text{sgn}(\theta^{(1)} \cdot x^{(2)}) = (-1) \cdot \text{sgn}([-1, 1]^T \cdot [0, -1]^T) = 1$
i=3: $y^{(3)} \cdot \text{sgn}([-1, 1]^T \cdot [1.5, 1]^T) = (+1) \cdot \text{sgn}([-1, 1]^T \cdot [1.5, 1]^T) = -0.5 < 0$
$\theta^{(2)} = \theta^{(1)} + y^{(3)} x^{(3)} = [-1, 1]^T + (+1)[1.5, 1]^T = [0.5, 2]$

At the end of $n = 3$ iterations, the classifier parameters are $\theta^{(2)} = [0.5, 2]^T$. To check that all points are classified correctly, we want to make sure that $y^{(i)} \cdot h(x^{(i)}; \theta^{(k)}) > 0$ for all $i = 1 \dots n$, where $h(x^{(i)}; \theta^{(k)}) = \theta^{(k)} \cdot x^{(i)}$ for a linear classifier. This is true in our example, and therefore it takes $k = 2$ updates to classify all points correctly.

Suppose we started the algorithm with a different order of data points. Starting with $x^{(2)} = [0, -1]^T, y^{(1)} = -1$, let's compute the number of updates k required until convergence.

i=0: $\theta^{(0)} = 0$
i=1: $y^{(2)} \cdot \text{sgn}(\theta^{(0)} \cdot x^{(2)}) = (-1) \cdot \text{sgn}(0 \cdot [0, -1]^T) = 0$
$\theta^{(1)} = \theta^{(0)} + y^{(2)} x^{(2)} = 0 + (-1) x^{(2)} = -x^{(2)} = [0, 1]^T$

After the first update, computing $y^{(i)} \cdot h(x^{(i)}; \theta^{(k)})$ for all $i = 1 \dots n$, we find that the answer is positive, and therefore the algorithms converges in $k = 1$ update. The order in which the data and the labels are presented impacts the number of updates it takes for the perceptron algorithm to converge.

(b) Let $x^{(3)} = [10, 1]^T$ so that our training set is

$$S_3 = \{([-1, 1]^T, +1), ([0, -1]^T, -1), ([10, 1]^T, +1)\}$$

We start with $x^{(1)} = [-1, 1]^T, y^{(1)} = +1$, and proceed as in part (a). On $x^{(3)}$, the algorithm makes a mistake and updates:

i=3: $\theta^{(2)} = x^{(1)} + x^{(3)} = [9, 2]^T$

However, we find that $x^{(1)}$ is now misclassified given $\theta^{(2)}$. Geometrically, we can see that $x^{(3)}$ is an outlier. During the update on $x^{(3)}$, the parameter vector $\theta^{(3)}$ overshoots such that $x^{(1)}$ is no longer correctly classified. Since the data is linearly separable, we iterate the perceptron algorithm until convergence:

i=4: $\theta^{(3)} = \theta^{(2)} + x^{(1)} = [8, 3]^T$
i=7: $\theta^{(4)} = \theta^{(3)} + x^{(1)} = [7, 4]^T$
i=10: $\theta^{(5)} = \theta^{(4)} + x^{(1)} = [6, 5]^T$
i=13: $\theta^{(6)} = \theta^{(5)} + x^{(1)} = [5, 6]^T$

We can now verify that all points are correctly classified. The following table summarizes the number of parameter updates for each data point:

i	$x^{(i)}$	$y^{(i)}$	updates
1	[-1, 1]	+1	5
2	[10, 1]	+1	1
3	[0,-1]	-1	0

Thus, we can see that a total of $k = 6$ updates were needed for the algorithm to converge.

Consider starting with $x^{(2)} = [0, -1]^T$, then the first update results in $\theta^{(1)} = \theta^{(0)} - x^{(2)} = [0, 1]^T$ that correctly classifies the remaining data points. This emphasizes again the importance of order in the input vector to the perceptron algorithm.

(c) In part (a) the algorithm converged in one pass through all data points. However, in part (b), the parameter vector was perturbed by an outlier that resulted in an overshoot and misclassification of earlier correctly classified points. As a result, the parameter vector was adjusted in the direction of error over two iterations until convergence.

(d) Consider setting $x^{(1)} = \max_i\{\|x_i\|, i = 1, \ldots, n\}$, i.e. initializing the algorithm with a data point of greatest magnitude. As a result, the magnitude of theta after the first update will also equal to max magnitude among all training examples: $\|\theta^{(1)}\| = \|y^{(1)}x^{(1)}\| = \|x^{(1)}\|$, where $(y^{(i)})^2 = 1$ for all i. Therefore, subsequent updates will not be able change the direction of θ to a significant extent and result in an increased number of classification errors.

Alternatively (for perceptron with offset), notice that the first iteration always results in an update of $\theta^{(1)} = y^{(1)}x^{(1)}$ and $\theta_0^{(1)} = y^{(1)}$. In subsequent iterations, an update occurs if $y^{(i)} \cdot \text{sgn}(\theta \cdot x^{(i)} + \theta_0) \leq 0$ evaluates to 1. One way to maximize the number of updates is to satisfy the update condition with equality:

$$y^{(i)}\text{sgn}(\theta^{(k)} \cdot x^{(i)} + \theta_0^{(k)}) = 0 \tag{1}$$

Choose $x^{(i+1)}$ such that:

$$\begin{aligned} \theta^{(k)} \cdot x^{(i+1)} &= -\theta_0^{(k)} \\ \sum_{k=1}^{i} y^{(k)}x^{(k)} \cdot x^{(i+1)} &= -\sum_{k=1}^{i} y^{(i)} \end{aligned} \tag{2}$$

By using a construction for $x^{(i+1)}$ above, the perceptron algorithm will make an update on every data point.

2. **Perceptron Performance**

(a) In the reading material, we studied the relationship between θ^k and θ^*. We examined the definition of $\frac{\theta^k \cdot \theta^*}{\|\theta^k\| \|\theta^*\|}$. We showed in lecture that $\frac{\theta^k \cdot \theta^*}{\|\theta^*\|} \geq k\gamma$ if θ was initialized to 0. However, if we initialize θ to some nonzero value, our recursion changes the bound to a function on our initial value plus our old bound so that $\frac{\theta^k \cdot \theta^*}{\|\theta^*\|} \geq \frac{\theta^{(0)} \cdot \theta^*}{\|\theta^*\|} + k\gamma = a + k\gamma$, where we substituted $\frac{\theta^{(0)} \cdot \theta^*}{\|\theta^*\|} = a$ for convenience.

Similarly, the derivation for $\|\theta^k\|^2 \leq kR^2$ becomes $\|\theta^k\|^2 \leq \|\theta^{(0)}\|^2 + kR^2$ and so $\|\theta^k\| \leq \sqrt{\|\theta^{(0)}\|^2 + kR^2} \leq \sqrt{\|\theta^{(0)}\|^2} + \sqrt{kR^2} = c + \sqrt{k}R$ where the last inequality comes from the concavity of the square-root function, and $c = \|\theta^{(0)}\|$ is a substitution for convenience.

Finally we have $1 \geq \frac{\theta^k \cdot \theta^*}{\|\theta^k\| \|\theta^*\|} \geq \frac{a+k\gamma}{c+\sqrt{k}R}$. Rearranging this gives us $a + k\gamma - c \leq \sqrt{k}R$, and then square both sides to get $a^2 + k^2\gamma^2 + c^2 + 2ak\gamma - 2ac - 2k\gamma c \leq kR^2$. Solving this quadratic inequality for the larger root gives us the upper bound $k \leq \frac{R\sqrt{R^2+(4c-4a)\gamma}+R^2+(2c-2a)\gamma}{2\gamma^2}$.

(b) (For Perceptron with offset) Consider a counterexample with a training set as in 1 (a):

$$S_3 = \{([-1, 1]^T, +1), ([0, -1]^T, -1), ([1.5, 1]^T, +1)\}$$

Let $(\theta_1^{(0)}, \theta_{1,0}^{(0)})$ and $(\theta_2^{(0)}, \theta_{2,0}^{(0)})$ be two different initializations of the parameter vectors. Let $\theta_1^{(0)} = 0$ and $\theta_{1,0}^{(0)} = 0$. From 1 (a), we know that the perceptron algorithm converges to $\theta_1^{(2)} = [-1, 2]^T$ and $\theta_{1,0}^{(2)} = 0$. Consider the following initialization for $\theta_2^{(0)} = -x^{(1)}$ and $\theta_{2,0}^{(0)} = -y^{(1)}$. There exists $x^{(1)}$ such that

$$y^{(i)} \text{sgn}(\theta_2^{(0)} x^{(1)} + \theta_{2,0}^{(0)}) \leq 0 \tag{3}$$

In particular, for our counterexample $x^{(1)} = [-1, 1]^T$:

$$+1 \cdot (-[0, -1][-1, 1]^T - 1) = 0$$

As a result, the first update is:

$$\begin{aligned} \theta_2^{(1)} &= \theta_2^{(0)} + x^{(1)} = -x^{(1)} + x^{(1)} = 0 \\ \theta_{2,0}^{(1)} &= \theta_2^{(0)} + y^{(1)} = -y^{(1)} + y^{(1)} = 0 \end{aligned}$$

After the first update, the parameter vector $(\theta_2^{(1)}, \theta_{2,0}^{(1)})$ is equal to $(\theta_1^{(0)}, \theta_{1,0}^{(0)})$. However, $(\theta_2^{(1)}, \theta_{2,0}^{(1)})$ acts on the second datapoint $x^{(2)} = [0, -1]^T$ and we know from 1 (a) that this results in convergence to $\theta_2^{(2)} = [0, 1]^T$ and $\theta_{2,0}^{(2)} = -1$ in the next iteration. Therefore, the two initialization methods converge to different parameter vectors.

(c) Since the parameter vectors are different for each classifier, they generalize differently on test data, and therefore test data performance will not necessarily be the same.

(d) For linearly separable data, the perceptron algorithm is iterated until all data points are classified correctly. As a result, if we combine the update equations, then for all updates k assuming

$\theta^{(0)} = 0$ and $\theta_0^{(0)}$:

$$\theta^{(4)} = \sum_{k=1}^{4} \alpha_i y^{(i)} x^{(i)} \tag{4}$$

$$\theta_0^{(4)} = \sum_{k=1}^{4} \alpha_i y^{(i)} 1 \tag{5}$$

Numerically, $\theta^{(4)} = [-3, 2]^T - 2[-1, -1]^T - 1[2, 2]^T = [-3, 2]^T$ and $\theta_0^{(4)} = 1(+1) + 2(-1) + 1(-1) = -2$.

3. **Decision Boundaries**

(a) A binary $x_i \in \{0, 1\}$ and a vector $x = [x_1, \ldots, x_n]^T$ define a cube in $\mathbb{R}^n$ with 2^n vertices and $2n$ faces. For $n = 3$, we have a unit cube centered at $(1/2, 1/2, 1/2)$. A function $f(x_1, x_2, x_3) = (x_1 \wedge x_2 \wedge x_3)$ is defined for each of the $2^3 = 8$ corners of the cube in $\mathbb{R}^3$. The point $(1, 1, 1)$ is the only corner for which $f(x_1, x_2, x_3) = 1$.

i. If $\theta_0 = 0$, our decision surface described by $\theta \cdot x = 0$ contains the origin. Since we want $\theta \cdot x > 0$ when $f(x_1, x_2, x_3) = 1$, we must satisfy $\theta_1 + \theta_2 + \theta_3 > 0$. However, vectors $[1, 0, 0]$, $[0, 1, 0]$, $[0, 0, 1]$ all must evaluate to $f(x_1, x_2, x_3) = 0$ and therefore the following three inequalities must hold: $\theta_1 < 0$, $\theta_2 < 0$, $\theta_3 < 0$. It is easy to see that no set of θ will satisfy our constraints, and we will not be able to learn a parameter that will classify all points correctly.

ii. We can translate our decision surface by θ_0 and still classify $f(x_1, x_2, x_3)$ correctly. For example, we can assign an arbitrary value to τ such that $\tau = \theta_1 = \theta_2 = \theta_3 > 0$. We can choose θ_0 from the interval $(-2\tau, -3\tau)$.

(b) We are given the following training set:

$$S_4 = \{([1, 1]^T, -1), ([2, 2]^T, -1), ([-1, 1]^T, +1), ([1, -1]^T, +1)\}$$

Consider the following classifiers:

i. A circle centered at the origin with radius r. The resulting non-linear decision boundary is defined by $\{x : \|x\| = r\}$. We can define the classifier $h(x; r)$ as:

$$h(x; r) = \begin{cases} +1, & \text{if } \|x\| < r. \\ -1, & \text{otherwise.} \end{cases} \tag{6}$$

Since $x^{(1)} = [1, 1]^T$ and $x^{(3)} = [-1, 1]^T$ are equidistant from the origin but have opposite labels, $h(x; r)$ will not be able to classify the training examples correctly.

ii. A circle centered at $x_0 \in \mathbb{R}^2$ with radius r. The resulting non-linear decision boundary is defined by $\{x : \|x - x_0\| = r\}$. We can define the classifier $h(x; r, x_0)$ as:

$$h(x; r, x_0) = \begin{cases} +1, & \text{if } \|x - x_0\| > r. \\ -1, & \text{otherwise.} \end{cases} \tag{7}$$

Consider $x_0 = [1.5, 1.5]^T$. The classifier $h(x; r, x_0)$ with $r = 1$ is now able to distinguish between all positively and negatively labeled training examples.

iii. A line through the origin with normal θ. The resulting linear decision boundary is defined by $\{x : \theta \cdot x = 0\}$. We can define the classifier $h(x; \theta)$ as:

$$h(x;\theta) = \begin{cases} +1, & \text{if } \theta \cdot x > 0. \\ -1, & \text{otherwise.} \end{cases} \tag{8}$$

Note that choosing any θ different than $[1, 1]^T$ or $[-1, -1]^T$ will cause the two positive examples to be on opposite sides of the classifier. Among these two options, $\theta = [1, 1]^T$ causes $[1, 1]^T$ and $[2, 2]^T$ to be classified as positive, so we have to discard it too. Taking $\theta = [-1, -1]^T$, on the other hand, causes $[-1, 1]$ and $[1, -1]$ to be fall on the decision boundary, and therefore classifies them as negative. We conclude that no line going through the origin will be able to correctly separate all training examples.

iv. A line through the origin with normal θ and offset θ_0. The resulting linear decision boundary is defined by $\{x : \theta \cdot x + \theta_0 = 0\}$. We can define the classifier $h(x; \theta, \theta_0)$ as:

$$h(x;\theta,\theta_0) = \begin{cases} +1, & \text{if } \theta \cdot x + \theta_0 > 0. \\ -1, & \text{otherwise.} \end{cases} \tag{9}$$

Taking $\theta = [-1, -1]^T$ and $\theta_0 = \varepsilon$ with $\varepsilon \in (0, 2)$, all training examples can be classified correctly by $h(x; \theta, \theta_0)$.

(c) The first two classifiers (i and ii) in 8 (b) are non-linear, while the last two (iii and iv) are linear.

4. **Feature Vectors**

(a) The feature vectors $z^{(i)}$ in $\mathbb{R}^m$ are related to data points $x^{(i)}$ in $\mathbb{R}^n$ via a linear transformation $z^{(i)} = Ax^{(i)}$. For $m = 2$ and $n = 6$, the linear map is defined as follows:

$$A = \begin{bmatrix} +1/6 & +1/6 & +1/6 & +1/6 & +1/6 & +1/6 \\ +1/3 & +1/3 & +1/3 & -1/3 & -1/3 & -1/3 \end{bmatrix}$$

(b) Given that θ_z correctly classifies the data in feature space of dimension $m < n$, there exists θ_x in data space that will identically classify the associated x_i's: $\theta_z \cdot z = \theta_z^T Ax = \theta_x \cdot x$, where $\theta_x = A^T \theta_z$.

(c) Given that we can find a classifier in higher dimension, we cannot always find a classifier in lower dimension. These additional dimensions can support finding a linear separator (more dimensions to find a separator through the features) while collapsing into a lower dimension, we can lose these necessary dimensions.

Optional: Formally, suppose we are given a θ_x that correctly classifies the points in data space of dimension $m < n$. We are looking for θ_z such that $\theta_x^T x = \theta_z^T Ax$ for all x. Finding such θ_z is equivalent to solving the overdetermined linear system $A^T \theta_z = \theta_x$, which can be done only if the system is *consistent*, i.e. if it has solution. This will happen if and only if θ_x is in the span of the columns of A^T.[1]

In that case, by looking at the equivalent system $AA^T \theta_z = A\theta_x$ we can identify two cases:

i. A has linearly independent rows. In this case AA^T is invertible, so there is a unique solution given by $\theta_z = (AA^T)^{-1} A\theta_x$.

[1] Equivalently, $A^T \theta_z = \theta_x$ has a solution if and only if $(A^T (A^T)^\dagger)\theta_x = \theta_x$, where $\dagger$ denotes the pseudo-inverse, as described above.

ii. A has linearly dependent rows. In this case, the system is indeterminate and has an infinite number of solutions.

The matrix $(AA^T)^{-1}A$ of part (i) is known as the Moore-Penrose pseudo-inverse of A^T, and it is denoted by $(A^T)^\dagger$.

(d) If $m < n$, the perceptron algorithm can converge faster when training in z-space. Let $x_i = e_i$ where e_i is the i^{th} basis vector for $\mathbb{R}^n$. Then, the perceptron algorithm makes n updates in data space in comparison to only m updates in feature space. For example, consider the case where $m = 1$, $n = 2$, $x^{(1)} = [1, 0]^T$, $y^{(1)} = -1$, $x^{(2)} = [0, 1]^T$, $y^{(2)} = +1$ and $A = [-1 \quad 1]$. The perceptron algorithm will take two steps to converge training in x-space, but only one in z-space.

(e) Suppose $m < n$ and we find a classifier in z-space given by θ_z. By part (b), we can find a classifier in x-space that will identically classify the training points. Therefore, there is always a classifier in x-space that is as least as good in the training set as the best classifier in z-space. However, training on a lower-dimensional space can yield a classifier that performs better un unseen data (i.e. has better generalization). By having more parameters to determine, the higher-dimensional classifier is more prone to overfitting the training data. Another advantage of training in z-space is that the lower computational complexity of the feature space can yield solutions of high accuracy (low distortion) and provide a significant computational advantage.