Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

# 6.036 Introduction to Machine Learning

**Midterm exam (March 19, 2015)**

---

**Your name & ID:** Solutions

- This is a closed book exam
- You do not need nor are permitted to use calculators
- The value of each question – number of points awarded for full credit – is shown in parenthesis
- The problems are not necessarily in any order of difficulty. We recommend that you read through all the problems first, then do the problems in whatever order suits you best.
- Record all your answers in the places provided

| Problem 1 | Problem 2 | Problem 3 | Problem 4 | Problem 5 | Total |
|---|---|---|---|---|---|
| 20 | 10 | 4 | 10 | 16 | 60 |
| 20 | 10 | 4 | 10 | 16 | 60 |

**Problem 1** The simple perceptron algorithm for estimating a linear classifier cycles through training examples $(x^{(i)}, y^{(i)})$, $i = 1, \dots, n$, and performs an update of the parameters in response to each mistake. If we omit the offset parameter, then $\theta \leftarrow \theta + y^{(i)}x^{(i)}$ whenever $(x^{(i)}, y^{(i)})$ is misclassified with the current setting of the parameters.

(1.1) **(4 points) (T/F)** Please mark the statements as **T** or **F**.

( F ) Suppose we normalize each training example such that $\|x^{(i)}\| = 1$. If there exists any $\theta^*$ such that $y^{(i)}(\theta^* \cdot x^{(i)}) \geq 1$, $i = 1, \dots, n$, then the perceptron algorithm converges after only a single mistake. may need several updates

( F ) Whenever the perceptron algorithm converges, the parameters of the averaged perceptron algorithm also linearly separate the training examples. averaged perceptron takes into account all thetas, not just final one

We can turn the linear perceptron into a non-linear classifier by mapping each example to a feature vector $\phi(x)$. We can also rewrite the algorithm such that it uses only inner products between examples (feature vectors). In other words, only values of the kernel function $K(x, x') = \phi(x) \cdot \phi(x')$ are needed. The kernel perceptron algorithm cycles through the training examples as before, updating mistake counts $\alpha_i$ whenever a mistake occurs.

(1.2) **(2 points) (T/F)** ( F ): The kernel perceptron without offset classifies any new example $x$ according to the sign of $\sum_{j=1}^{n} \alpha_j y^{(j)} K(x^{(j)}, x)$ where $\boxed{\sum_{j=1}^{n} \alpha_j \leq n.}$ sum of alphas depends on *iterations too

Here we use a kernel perceptron to classify nodes in an undirected graph. You can think of the graph as a social network representing friendship relations. Each person is a node in the graph and there's an edge between two nodes whenever people are friends. Our goal is to learn to predict a positive/negative label for each node. We were thinking of using a particular type of kernel function based on how many neighbors any two nodes have in common. In other words, if $N(i)$ is the set of neighbors of node $i$, then

$$K(i, j) = |N(i) \cap N(j)| \quad (\text{\# of common neighbors}) \tag{1}$$

(1.3) **(4 points)** Which of the following properties of $K(i, j)$ are *also* properties of any valid kernel function over the nodes. Check all that apply.

( T ) $K(i, j) = K(j, i)$ for all $i, j$ symmetry

( F ) $K(i, j) \geq 0$ for all $i, j$ not true for a <u>general</u> kernel

( T ) $K(i,i) \geq 0$ for all $i$ $K(i,i) = \phi(i)\cdot\phi(i) = \|\phi(i)\|^2 \geq 0$ always true

( F ) $K(i,j) \leq K(i,i)$ for all $i, j$

(1.4) **(4 points)** Write down the feature representation $\phi(i)$ corresponding to our chosen kernel $K(i,j) = \phi(i) \cdot \phi(j)$. Specifically, define the $k^{th}$ coordinate of $\phi(i)$ as

$$\phi(i)_k = \begin{cases} 1, & \text{if } \underline{\text{node } k \text{ is a neighbor of node } i} \\ 0, & \text{otherwise} \end{cases} \qquad (2)$$

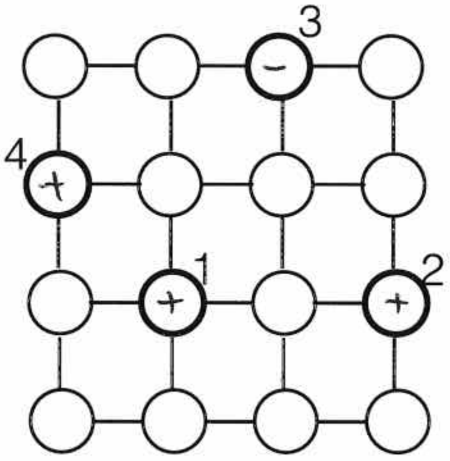
(k belongs to $N(i)$)

(1.5) **(6 points)** Consider the following simple grid graph with four nodes numbered and highlighted. These are the training nodes (training examples) we have labels for. Once labels are made explicit, we can run the kernel perceptron algorithm with our kernel on these nodes, in the order given, i.e., cycling through 1,2,3, and 4.

(a) Is it possible to label the highlighted nodes such that our kernel perceptron algorithm would make only 1 mistake in total. Please answer **Y** or **N** ( N ) would misclassify the second point

(b) Please label each of the highlighted nodes in the graph with a "+" or "-" such that the kernel perceptron algorithm, if run with these labels, would make a mistake only on nodes 1 and 3, converging after the first round.

larification: t least one oint of each class this was specified during the exam on the board]



- points that share a neighbor get the same class label

**Problem 2** The passive-aggressive algorithm differs from the perceptron algorithm in that it updates its parameters in response to each training example $(x, y)$ by minimizing

$$\frac{\lambda}{2}\|\theta - \theta^{(k)}\|^2 + \text{Loss}(y\,\theta \cdot x) \tag{3}$$

with respect to $\theta$ where $\theta^{(k)}$ is the current setting of the parameters. Here $\lambda > 0$ is a parameter we have to set and $\text{Loss}(z)$ defines how we measure errors. As in lectures, we will adopt the Hinge loss $\text{Loss}(z) = \max\{0, 1 - z\}$ throughout this problem. We have seen that passive-aggressive updates have the stereotypical form

$$\theta^{(k+1)} = \theta^{(k)} + \eta_k\, y\, x \tag{4}$$

where $\eta_k$ depends on $\lambda$, the loss function, as well as the example.

(2.1) **(2 points)** Suppose we have parameters $\theta^{(k)}$ and see a training example $(x, y)$. If $y\theta^{(k)} \cdot x < 1$ before the update, is it possible that $y\theta^{(k+1)} \cdot x > 1$ after the update? Please answer **Y** or **N** **(N)** $y\theta^{(k+1)} \cdot x \leq 1$ because Loss = 0 after update & no need to overshoot ✓

(2.2) **(2 points)** If $y\theta^{(k)} \cdot x < 1$ before the update, we can always just solve for $\theta^{(k+1)}$ by minimizing

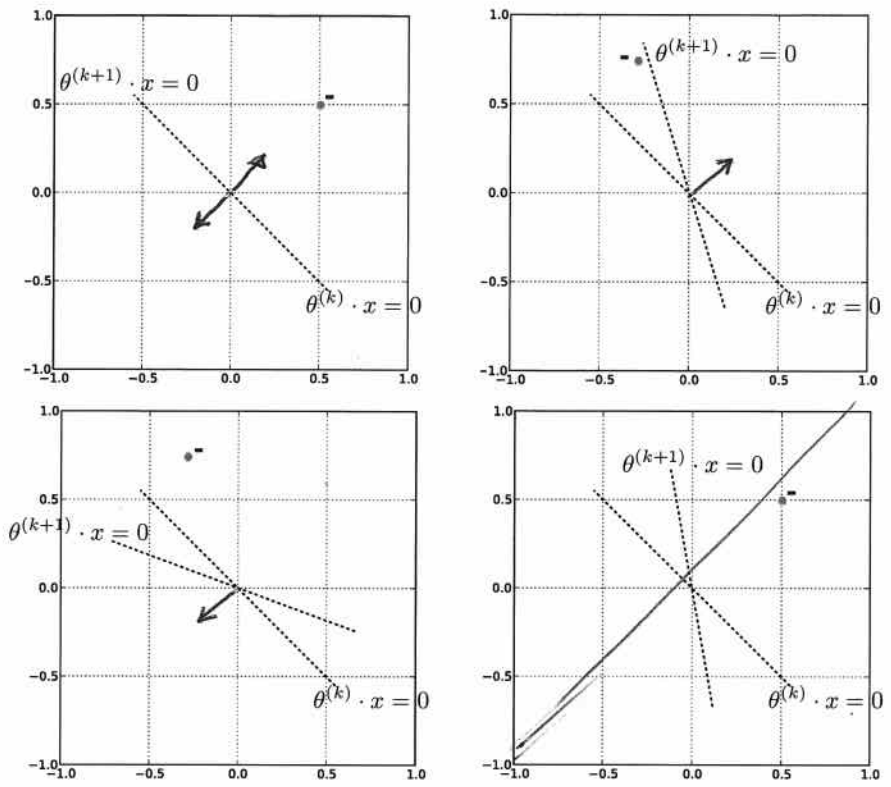
$$\frac{\lambda}{2}\|\theta - \theta^{(k)}\|^2 + \underbrace{(1 - y\theta \cdot x)} \tag{5}$$

Please answer **Y** or **N** **(N)** missing constraint of being bounded below by 0

(2.3) **(6 points)** The passive-aggressive algorithm will result in slightly different updates depending on how we set $\lambda$. In Figure 1, you see four plots representing decision boundaries before and after the update obtained with some value of $\lambda$ in response to a negatively labeled point also shown in the Figures. Note that the negative point is NOT in the same location in all the plots. The tricky part is that some of the plots may not correspond to a passive-aggressive update for any value of $\lambda > 0$. Please

(a) draw the orientation of $\theta^{(k)}$ (parameters before the update) in each *possible* figure and

(b) *cross out* all the plots which are **NOT** possible with any value $\lambda > 0$.



not possible

Figure 1: Possible updates resulting from the passive-aggressive algorithm

**Problem 3** Recommender problems are everywhere, from Netflix, Amazon, to Google News. Here we aim to reconstruct a matrix of user-item preferences using matrix factorization learned in class. Since each user is likely to provide ratings for only a small subset of possible items, we must heavily constrain the models so as not to overfit. In fact, our goal here is to understand how constrained they really are.

(3.1) **(4 points)** Suppose we try to estimate a simple rank-1 model $X_{ai} = u_a v_i$ where $u_1, \ldots, u_n$ and $v_1, \ldots, v_m$ are just scalar parameters associated with users $u_a$ and items $v_i$, respectively. Please select which (if any) of the following 2x2 matrices **cannot** be reproduced by the rank-1 model.

$$\begin{array}{c|c|c|} & v_1 & v_2 \\ \hline u_a & +1 & +1 \\ \hline u_b & +1 & -1 \\ \hline \end{array} \qquad \begin{array}{c|c|c|} & v_1 & v_2 \\ \hline u_a & -1 & +1 \\ \hline u_b & +1 & -1 \\ \hline \end{array} = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & -1 \end{pmatrix}$$

(X) ( )

has rank 2

**Problem 4** In this question, we will consider a kernel-based K-means clustering algorithm. The distance measure for this algorithm is defined exactly as before but now evaluated in terms of the feature vectors, i.e., $\|\phi(x^{(i)}) - \phi(x^{(j)})\|^2$.

(4.1) **(2 points)** Suppose we have a cluster of points $\{x^{(i)}, i \in C\}$. Provide an expression for the corresponding centroid $z$ we would obtain in feature coordinates.

$$z = \frac{1}{|C|} \sum_{i \in C} \phi(x^{(i)})$$

(4.2) **(6 points)** Show that when we reassign points to clusters, we don't have to explicitly compute the centroid $z^{(j)}$ or feature vectors $\phi(x^{(i)})$, because the cost of assigning a point $x^{(i)}$ to a cluster represented by $z^{(j)}$, i.e., $||\phi(x^{(i)}) - z^{(j)}||^2$, can be expressed entirely in terms of the kernel function $K(x, x') = \phi(x) \cdot \phi(x')$.

$$\|\phi(x^{(i)}) - z^{(j)}\|^2$$

$$= \|\phi(x^{(i)})\|^2 + \|z^{(j)}\|^2 - 2\langle \phi(x^{(i)}), z^{(j)} \rangle$$

$$= K(x^{(i)}, x^{(i)}) + \left\| \frac{1}{|C_j|} \sum_{l \in C_j} \phi(x^{(l)}) \right\|^2 - 2 \left\langle \phi(x^{(i)}), \frac{1}{|C_j|} \sum_{l \in C_j} \phi(x^{(l)}) \right\rangle$$

$$= K(x^{(i)}, x^{(i)}) + \frac{1}{|C_j|^2} \sum_{l \in C_j} \sum_{m \in C_j} \langle \phi(x^{(l)}), \phi(x^{(m)}) \rangle - \frac{2}{|C_j|} \sum_{l \in C_j} \langle \phi(x^{(i)}), \phi(x^{(l)})$$

$$= K(x^{(i)}, x^{(i)}) + \frac{1}{|C_j|^2} \sum_{l \in C_j} \sum_{m \in C_j} K(x^{(l)}, x^{(m)}) - \frac{2}{|C_j|} \sum_{l \in C_j} K(x^{(i)}, x^{(l)})$$

(4.3) **(2 points)** Suppose we are given the kernel matrix $K_{ij} = K(x^{(i)}, x^{(j)})$ evaluated between any pair of training examples. Which of the following expressions best capture the complexity (number of operations) involved in a single iteration of the kernel k-means algorithm. Here $n$ is the number of points, $d$ is the dimension of $x$, and $k$ is the number of clusters.

( ) $kdn^2$
(✓) $kn^2$
( ) $kdn$
( ) $kn$

for each point, to find the closest cluster ~~center~~ centroid, the point is compared to all the other points in each cluster (recall: we do not compute the centroid explicitly)

**Problem 5**

(5.1) **(4 points)** Consider weighted training points shown in Figure 2. The figure also shows two possible decision stumps to consider at this round of boosting: A and B. Which one of the stumps will be selected? Briefly justify your answer.

weighted error for B = 0.1 + 0.2 = 0.3
for A = 0.3 + 0.1 = 0.4

⇒ B will be selected because it has the lower weighted error

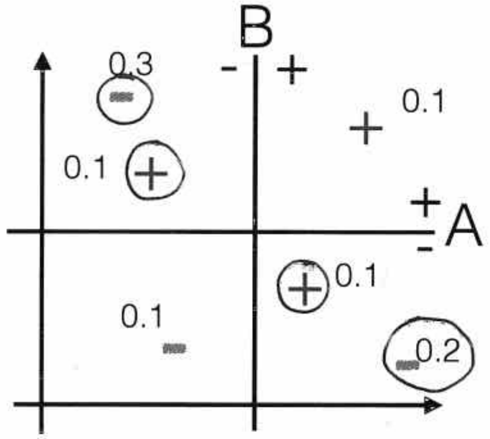


Figure 2: Training set and two candidate decision stumps

(5.2) **(6 points)** Figure 3 shows a set of four labeled points. Construct an ensemble of decision stumps that correctly classifies these points. Use the smallest number of decision stumps. Clearly indicate the decision boundary and the positive/negative direction of the stumps, including their votes (if not uniform).

(5.3) **(6 points)** Consider training set in Figure 4. Assume that the points are equally weighted, i.e., each have weight 1/4.

(a) What is the weighted error of the stump shown in the Figure? What is the corresponding votes $\alpha$ that this stump receives?

$$\varepsilon_m = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$$

$$\alpha = \frac{1}{2}\log\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right) = \frac{1}{2}\log\left(\frac{1/2}{1/2}\right) = \frac{1}{2}\overset{0}{\cancel{\log(1)}} = 0$$

(b) Will AdaBoost succeed in finding an ensemble that correctly classifies these points? Briefly justify your answer.

No it is not possible to classify the points correctly with any number of stumps. Points of opposite classes are axis-aligned so any stump will incorrectly classify half the points (we can't make progress). By part (a), $\alpha = 0$ for all new stumps, so ensemble does not change.

↗ lots of possible symmetries

multiple solutions are possible (all correct solutions have 3 stumps)

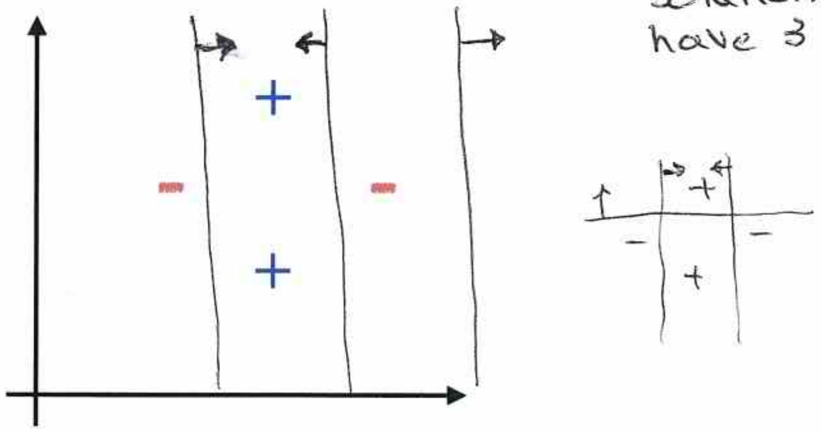


Figure 3: Training set of four points

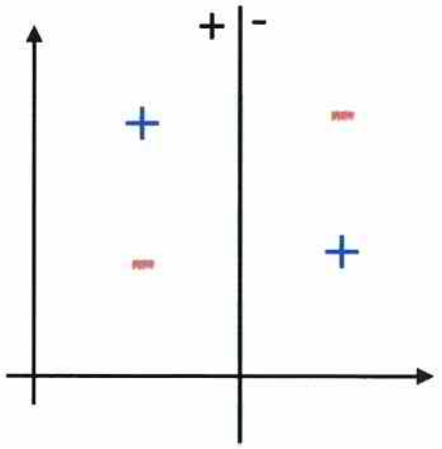


Figure 4: Training set (#2) involving four points