CS446: Machine Learning	**Fall 2014**
Problem Set 4	
Handed Out: October 8^{th}*,* 2014	*Due: October* 17^{th}*,* 2014

1. **[VC Dimension - 30 points]**

   (a) For concept space $\mathbf{H}_3$ consisting of circles that need not be centered at the origin, the VC dimension is 3.

   First let us show that there exist three points, $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3 \in \mathbb{R}^2$ that can be shattered. Let us pick $(0,0)$, $(1,0)$, and $(0,1)$. Consider the following sub-cases:

   - None of the points are positive : Choose any circle that does not include any of the three points.
   - All points are positive: Choose a center of radius 2 that is centered at the origin.
   - Only one point out of the three is positive: Choose a circle centered at the positive point with radius 0.5.
   - Two points are positive: Convince yourself that we can always choose a circle that includes the two positive points and excludes the negative point.

   We now need to show that no set of four points can be shattered by concepts in $\mathbf{H}_3$, and this is somewhat harder. We will base our proof on the convexity property of circles. Consider the convex hull of the four points (the smallest convex set that contains all four points).

   - If one of the four points is inside the convex hull of the other three points, or on the triangle formed by them, it is also in the circumscribing circle of those three points. So if that point is negative while the other three points are positive, there is no concept in $\mathbf{H}_3$ that is consistent with such a labeling.
   - Otherwise, all four points are the vertices of the convex hull. Consider the two diagonals – and note that these must intersect. Without loss of generality, let us call the end points of the first diagonal $\mathbf{x}_1$ and $\mathbf{x}_2$ and the end-points of the other diagonal $\mathbf{x}_3$ and $\mathbf{x}_4$. Consider two circles, C_1 that covers points $\mathbf{x}_1$ and $\mathbf{x}_2$, and C_2 that covers points $\mathbf{x}_3$ and $\mathbf{x}_4$. Since the diagonals must intersect, the intersection point lies within the intersection of circles C_1 and C_2 (using convexity of circles). But then at least one of the four points, the one closest to the intersection point, must also lie within the intersection of circles C_1 and C_2.

   Therefore, we proved that for any set of four points, the points cannot be shattered using concepts in $\mathbf{H}_3$. Hence, VC dimension of $\mathbf{H}_3$ is 3.

   (b) First, we prove the VC dimension is at most $2k$. Consider a sequence of $2k+1$ distinct points on a line. Note that the points must be distinct since otherwise, giving different labels to the points at the same position will make the data not separable. If the points are distinct, we label them alternatively: $+-+-\ldots-+$.

This pattern cannot be realized with our hypothesis space. Therefore the VC dimension is at most $2k$. Next, we prove that the VC dimension is at least $2k$. Consider a set of $2k$ distinct samples $x_1 < x_2 < \ldots < x_{2k}$ ordered on the line. We can divide the points into k chunks, such that the i-th chunk contains only two points x_i, x_{i+1}. We can shatter the two points in each chunk with an interval. Therefore, the union of k intervals can shatter these $2k$ points. Thus, VC dimension of the class of union of k intervals on the real line is $2k$.

2. [**Decision Lists - 40 points**]

(a) Given a k-decision list $c = \{(c_1, b_1), \ldots, (c_\ell, b_\ell), b\}$, its complement is given by $\neg c = \{(c_1, \neg b_1), \ldots, (c_\ell, \neg b_\ell), \neg b\}$, which is also a k-decision list.

(b) Given a k-DNF $f = d_1 \vee d_2 \vee \ldots \vee d_m$, wherein each d_i is a k-conjunction, we can define the following decision list: $c = \{(d_1, 1), (d_2, 1), \ldots, (d_m, 1), 0\}$. It is clear that $c \equiv f$, since both functions are satisfied by an assignment $\mathbf{x} \in \{0,1\}^n$ iff at least one of the d_i is satisfied by it. Hence, we showed that every k-DNF can be represented by a k-DL, which proves that k-DNF $\subseteq$ k-DL.

To show that every k-CNF can be represented as a k-DL, we use the fact that if $g = t_1 \wedge t_2 \wedge \ldots \wedge t_m$ is a k-CNF, then $\neg g = \neg t_1 \vee \neg t_2 \vee \ldots \vee \neg t_m$ is a k-DNF. Note that $\neg t_i$ is a k-conjunction, similar to d_i above. Since k-DNF $\subseteq$ k-DL, $\neg g \in k$-DL. But from part (a), if $\neg g \in k$-DL, then $\neg(\neg g) = g \in k$-DL. Hence, we showed that every k-CNF can be represented by a k-DL, which proves that k-CNF $\subseteq$ k-DL.

Putting the two together, k-DNF $\cup$ k-CNF $\subseteq$ k-DL.

(c) Let S be a non-empty set of examples consistent with the target k-DL c. Denote by C_k the collection of all conjunctions of size at most k: $|C_k| = O(n^k)$. Note that the empty conjunction is also included in C_k.

For every $c_i \in C_k$, let $T_i(S) = \{x \in S | c_i(x) = 1\}$, i.e. $T_i(S)$ is the set of all examples in S that satisfy c_i. For $b \in \{0,1\}$ we say that c_i is b-consistent with the sample S if $\forall x \in T_i(S)$, we have that $c(x) = b$, i.e. the examples in $T_i(S)$ (i.e. the ones that satisfy c_i) are either all positive or all negative.

Algorithm

i. Start with an empty decision list.

ii. For $b \in \{0,1\}$, search for a $c_i \in C_k$ that is b-consistent with S . To do this, iterate through all $c_i \in C_k$, evaluating c_i on all the examples in $T_i(S)$. If you find that c_i is b-consistent, stop; otherwise, move on to c_{i+1}.

iii. Add (c_i, b) to the decision list.

iv. Update $S \leftarrow S \setminus T_i(S)$, i.e. remove elements of $T_i(S)$ from S.

v. If S is not empty, go to step (ii). Otherwise, return the current decision list.

Correctness If the algorithm terminates then (1) we have constructed a k-decision list and (2) it is consistent with the sample S. To see (2) is true, take any example e from S and let its label be b. We need to show that the final decision list will assign label b to the example e. To show this, we will start with the observation that at the end of the algorithm, S was empty. So, e must have been in some $T_i(S)$ corresponding to a $c_i \in C_k$ for it to be removed from S. At that time, c_i was added to the decision list along with a label b_i. Further, it is also true that no c_j placed in the decision list before c_i is true for e, for otherwise e would have been removed from S then. Now, the $T_i(S)$ to which e belongs is also b_i-consistent (only then is the pair (c_i, b_i) added to the decision list). This means, all elements in $T_i(S)$ have the label b_i. But we know that e belongs to $T_i(S)$ and e has a label b. So, it must be the case that $b_i = b$. Since c_i is the first

conjunction in the list that satisfies e, the constructed decision list will assign it the label b, its true label. So the decision list is consistent for e. Since this is true for all $e \in S$, the decision list is consistent with the sample S.

The algorithm always terminates, because we can always find a b-consistent conjunction in C_k. The reason is that there exists a k-DL list which is consistent with S. Thus, as long as the first conjunction, c_p, in the true DL has not been selected, even in the case that other conjunctions are selected before it, c_p is b-consistent with S, for some $b \in \{0, 1\}$. Once it has been selected, and we have updated the sample $S \leftarrow S \setminus T_p(S)$, the second conjunction in the true DL is consistent with the new S.

Continuing inductively in this way, as long as S is not empty, there is a conjunction that is part of the true decision list, has not been selected yet, and is consistent with the current S. Hence, it is always possible to continue until S is empty, guaranteeing termination of the algorithm. The algorithm is efficient, given that $|C_k| = O(n^k)$ is polynomial in the number of variables, and each iteration of the algorithm examines each of the (at most) m examples once per conjunction in C_k. Since the worst case complexity of the number of iterations is $O(n^k)$, the total complexity of learning the decision list is $O(mn^k)$.

(d) To use Occam's razor, given that we have a consistent algorithm, all we need is to bound the size of the class k-DL. Because $|C_k| = O(n^k)$, the number of k-DL functions is bounded by $2^{|C_k|} \cdot \ell! \cdot 2^\ell$, where ℓ is the size of the k-DL. The $\ell! \cdot 2^\ell$ factor comes from the ordering of the ℓ nodes and then assigning a bit $b \in \{0, 1\}$ to each node. Since, the maximum value of ℓ is $|C_k|$, the overall size of the class k-DL is $O(2^{|C_k|}(C_k)!)$. Therefore the number of examples we need in the sample S , which is logarithmic in the size of the class, is polynomial in n ($\log |k\text{-DL}| = n^k \log 2 + kn^k \log n$), and hence, k-DL is PAC-learnable.

3. **[Constructing Kernels - 30 points]**

(a) $K(\mathbf{x}_1, \mathbf{x}_2) = \sum_{j=1}^{k} \binom{same(\mathbf{x}_1, \mathbf{x}_2)}{j}$
where $same(\mathbf{x}_1, \mathbf{x}_2)$ is the number of features set to the same value in the two input vectors. Computing $same(\mathbf{x}_1, \mathbf{x}_2)$ takes time linear in the number of features.

(b) The kernel Perceptron algorithm is executed similar to Perceptron with a few modifications. First, instead of using the function

$$f(\mathbf{x}) = sgn\,(\mathbf{w} \cdot \mathbf{x} + \theta)$$

to classify examples, we use the following function, which transforms the examples into a higher dimensional space and reorganizes the computation of the high dimensional dot product:

$$f(\mathbf{x}) = sgn\left(\left(\sum_{\mathbf{x}_m \in M} S(\mathbf{x}_m) K(\mathbf{x}, \mathbf{x}_m)\right) + \theta\right)$$

where M is the set of training examples that the algorithm has previously made a mistake on, and $S(\mathbf{x}_m) \in \{-1, 1\}$ is the label of example $\mathbf{x}_m$. We make these ideas explicit in the KernelPerceptron algorithm.

KernelPerceptron($S \in (X \times Y)^m$)
  $M \leftarrow \emptyset$
  foreach $(\mathbf{x}, y) \in S$
    if $y \neq sgn\left(\left(\sum_{(\mathbf{x}_m, y_m) \in M} y_m K(\mathbf{x}, \mathbf{x}_m)\right) + \theta\right)$
      $M \leftarrow M \cup (\mathbf{x}, y), \quad \theta \leftarrow \theta + y$
  return $f(x) = sgn\left(\sum_{(\mathbf{x}_m, y_m) \in M} y_m K(\mathbf{x}, \mathbf{x}_m) + \theta\right)$

(c) In class, we proved a theorem by Novikoff showing that Perceptron will make

$$M \leq \frac{R^2}{\gamma^2}$$

mistakes on any training set known to be linearly separable. This analysis was done under the assumption that Perceptron makes classifications with the function $f(\mathbf{x}) = sgn(\mathbf{w} \cdot \mathbf{x})$, which implies that the threshold is being learned. (Why?) All we need to do is compute the values of R and γ.

Every example in the original space translates to an example containing $\sum_{j=1}^{k} \binom{n}{k}$ active features in the blown-up space, plus the "threshold feature" that is always active. Therefore,

$$R = \sqrt{\sum_{j=1}^{k} \binom{n}{j} + 1}$$

γ is more complicated. It is defined in terms of a correct weight vector u such that $||u|| = 1$. Given a data set, we would like to maximize γ to make the bound as tight as possible. But we are interested in learning any k-DNF concept, so we must find the concept that yields the smallest maximum γ. In the blown-up feature space, we are learning a simple disjunction of $r \leq n'$ variables, where n' is the dimensionality of the blown-up feature space. Without loss of generality, we can assume the u maximizing γ for such a function has the following form (why?):

$$u = \frac{(u_1 = 1, u_2 = 1, ..., u_r = 1, u_{r+1} = 0, ..., u_n = 0, u_{n+1} = \theta)}{\sqrt{r + \theta^2}}$$

where $-1 < \theta < 0$. The positive example x^+ closest to the hyperplane induced by u contains a single active feature out of the first r, in addition to the last feature which must be active. Therefore,

$$u \cdot x^+ = \frac{1 + \theta}{\sqrt{r + \theta^2}}$$

For every negative example x^-, we have

$$u \cdot x^- = \frac{\theta}{\sqrt{r + \theta^2}}$$

And since

$$\gamma = \min(|u \cdot x^+|, |u \cdot x^-|)$$

the best choice for θ is $-\frac{1}{2}$. Substituting, we get

$$\gamma = \frac{1}{\sqrt{4r + 1}}$$

Remember that we are trying to find the worst case that leads to a smallest maximum γ. From the above analysis, we see the worst case happens when r is maximized. That is every possible k-conjunction is a term in the k-DNF we're learning:

$$r = \sum_{j=1}^{k} \binom{n}{j} 2^j$$

Therefore,

$$M \leq \frac{\sum_{j=1}^{k} \binom{n}{j} + 1}{\frac{1}{4\sum_{j=1}^{k} \binom{n}{j} 2^j + 1}} = \left(\sum_{j=1}^{k} \binom{n}{j} + 1 \right) \left(4 \sum_{j=1}^{k} \binom{n}{j} 2^j + 1 \right) = O\left(\sum_{j=1}^{k} \binom{n}{j} 2^j \sum_{j=1}^{k} \binom{n}{j} \right)$$

It is possible to further simplify the bound using the following inequations:

$$\begin{aligned}
\sum_{j=0}^{k} \binom{n}{j} &\leq \left(\frac{n}{k}\right)^k \sum_{j=0}^{k} \binom{n}{j} \left(\frac{k}{n}\right)^j \\
&\leq \left(\frac{n}{k}\right)^k \sum_{j=0}^{n} \binom{n}{j} \left(\frac{k}{n}\right)^j \\
&\leq \left(\frac{n}{k}\right)^k \left(1 + \frac{k}{n}\right)^n \\
&\leq \left(\frac{n}{k}\right)^k e^k
\end{aligned}$$

$$\begin{aligned}
\sum_{j=0}^{k} \binom{n}{j} 2^j &\leq \left(\frac{n}{k}\right)^k \sum_{j=0}^{k} \binom{n}{j} \left(\frac{k}{n}\right)^j 2^j \\
&\leq \left(\frac{n}{k}\right)^k \sum_{j=0}^{n} \binom{n}{j} \left(\frac{2k}{n}\right)^j \\
&\leq \left(\frac{n}{k}\right)^k \left(1 + \frac{2k}{n}\right)^n \\
&\leq \left(\frac{n}{k}\right)^k e^{2k}
\end{aligned}$$

Therefore

$$M = O\left(\left(\frac{n}{k}\right)^{2k} e^{3k}\right).$$