CS446: Machine Learning	**Fall 2014**
Problem Set HW 4	
Siting Chang	*Handed In: 10/17/2014*

1 VC Dimesion

1.1 part a

VC dimension of H is 3.

It is trivial to show that there exists one or two points on the plane that can be shattered by H. We now show that H is able to shatter 3 points which implies that the VC dimension of H is at least 3.

Select three points with coordinates A(1,1), B(-1,1) and C(-1, -1). There are 8 possible combination of labels which H are able to classify all. The orgin and radius choices are listed as follows for each combination. The order is: label of point A, label of point B, label of point C, origin, radius.

- +, +, +, (0,0), 2
- -, -, -, (0,0), 0.5
- +, -, -, (1,0), 1
- +, -, +, (1,-1), 2
- -, +, -, (-1,1), 1
- -, +, +, (-1,0), 1
- -, -, +, (-1,-1), 0.5
- +, +, -, (0,1), 1

Next, we show that H is not able to shatter any four points on the plane which means that the VC dimension of H is less than 4.

There are two possible situations when randomly choose four points:

- four points form a convex hull. This situation cannot be classified by any hypothese in H when the opposing points with the largest distance both have positive lables and the other two have negative labels.
- three points form a convex hull and one point is internal. This situation cannot be classified when the first three points (on the convex hull) have positive label and the fourth point has negative label.

Therefore, we proved that the VC dimension of H is 3.

1.2 part b

VC dimension of H is 2k.

We start with the base case with one point x_1 on the real line. It is easy to see that we are able to shatter this point no matter it has a positive label (choose $a_1 < x_1 < b_1$ or a negative label (choose $b_1 < x_1 < a_2$).

The most complicated which is also the hardest case to classify is when neighboring points have opposite labels. For example, the most complicated case of four points is when x_1 and x_3 have positive labels and x_2 and x_4 have negative labels. The reasont that it is the hardest case to classify is because each of these four points need to be placed or assigned to a disjoint interval. This leads to the requirement of four intervals. As long as we have enough intervals, at least four intervals, to cope this situation, the points are shattered.

Given 2k points on the real line which has k disjoint intervals within which points are labeled as potive and k more disjoint intervals within which points are negative. The largest number of points that can be shattered is 2k since we are able to assign 2k points with neighboring points have opposite labels into these 2k intervals.

However, H cannot shatter 2k+1 points. Again, consider the most complicated situation with the leftmost point with a positive label. Since the leftmost point must fall in the region with $x_{leftmost} < a_1$, there is no way any hypothesis from H can classify these points, especially the leftmost point.

Therefore, we proved that the VC dimension of H is 2k.

2 Decision Lists

2.1 part a

$\neg c =< (c_1, \neg b_1), ..., (c_l, \neg b_l), \neg b >$

2.2 part b

First, show k-DNF$\subseteq$k-DL. Since each term of k-DNF can be transformed into an item of a decision list with value 1, then clearly k-DNF$\subseteq$k-DL.

Next, according to DeMorgan's Rules, we can always find some k-DNF that complements any k-CNF. Along with the fact that k-DL is closed under complementation (shown in part a), we say that k-CNF$\subseteq$k-DL.

With each component a subset of k-DL, we say there union is also a subset of k-DL denoted as k-DNF $\cup$ k-CNF$\subseteq$k-DL.

2.3 part c

Input of the algorithm: sample space S over samples $\mathbf{x}$ which have n dimensions. Output: decision list DL that is consistent with all samples.

Notations: L_n denotes the set of 2n literals $(x_n, \neg x_n)$ associated with samples. C_k^n denotes the set of all conjunctions of size at most k with literals drawn from L_n.

1. Check if sample space S is empty. If so, stop the algorithm. If not, continue.
2. Set the default output which is denoted by $b = 0$.
3. Check each item in C_k^n in turn until found an item e that all samples $\mathbf{x}$ in S outputs the same label l, either positive or negative, when e is true.
4. Move the samples from S into T if it makes e true.
5. Put e in decision list DL along with its output label l.
6. For the rest samples in S, repeat step 1 to 4 until S is empty.

Note that in this algorithm, we didn't double check the existence of an item e or a decision list that is consistent with all samples since the question statement claims that the samples are consistent with some k-decision list.

The intuition of this algorithm is that if a conjunction/item e that is consistent with the given samples, then no matter in which order it presents or being added to the decision list, it will always be consistent with any subset of the samples. Therefore, the order we examine items from C_k^n does not kill the algorithm.

In summary, the algorithm starts by checking items in C_k^n and finding the first item that is consistent with the samples in S. As soon as it finds such an item, the algorithm puts it into the decision list and delete the samples from the sample space S. The algorithm continues finding item that is consistent with the updated S until S is empty which means all samples are being classified or explained.

2.4 part d

According to Occam's Razor: $m > \frac{1}{\epsilon}(ln(|H|) + ln(\frac{1}{\delta}))$. In order to show that the class of k-decision lists is efficiently PAC-learnable, we need to study the size of k-decision list is linear in n. Also, we need to study the computational complexity of the class.

Again, we adopt the notations L_n denotes the set of 2n literals $(x_n, \neg x_n)$ associated with samples and C_k^n denotes the set of all conjunctions of size at most k with literals drawn from L_n.

Since all samples have n dimensions, we know that for all items in decision list there are $3^{|C_k^n|}$ combinations given each term has three options of missing, negative and positive. Another thing to notice is that the order of terms appeared in each item does not matter which gives us $(C_k^n)!$ combinations.

Therefore, the size of a k-decision list is $\mathcal{O}(3^{|C_k^n|}(C_k^n)!)$. Then we have $lg(|k - DL(n)|) = \mathcal{O}(n^t)$ for some constant t.We conclude that k-decision list has size which is polynomial in n.

Next, we study the computational complexity of the algorithm proposed in part c to find a k-decision list. The computation complexity is also polynomial in time since the critical component C_k^n, in the algorithm, is polynomial in n for any fixed k.

Therefore, we conclude that with polynomial sample size and polynomial computation time, the k-decision list class is efficiently PAC-learnable.

3 Constructing Kernels

3.1 part a

Give function $c(\cdot)$ represents conjunctions containing up to k different literals, what it means is that $c(x_1)$ is checking if all literals in $c(\cdot)$ are active literals in example x_1. If yes, it has output of positive 1, otherwise it ouputs 0. The same for $c(x_2)$. Then we know, $c(x_1)c(x_2)$ checks if all literals in $c(\cdot)$ are active literals both in example x_1 and x_2. If yes, outputs positive 1 otherwise 0.

The kernel $K(x_1, x_2)$ as stated in the problem is summing over all possible conjunctions $c(\cdot)$ evaluated on x_1 and x_2. If to compute the kernel function follows the function defined in the problem, the computational time required is not linear in n. So we need to propose a new approach to compute the kernel value.

We propose the following approach:

$$K(x_1, x_2) = \sum_{j=0}^{min(k,\text{numCommon}(x_1,\, x_2))} \binom{\text{numCommon}(x_1,\, x_2)}{j} \tag{1}$$

in which numCommon(x_1, x_2) returns the number of common literals shared by x_1 and x_2.

The idea of this approach is to first calculate the number of common literals shared by x_1 and x_2, and then formulate the class C by choosing 1, 2 to up to k literals from the common literals. Computing the total number of such c gives us the exactly the same kernel value as defined in the problem statement. And the above equation helps us to compute this total number by summing over all possible $j's$. The number j is upbounded by the min(k, numCommon(x_1, x_2)) to take care of the case where the number of common literals is bigger than k.

Now, we prove that our proposed apporach gives the same result as the function defined in the problem statement. Observe the kernel function defined in the problem, the only conjunctions c that will contriubte to the kernel value are those returning value 1 given x_1 and x_2. And these conjunctions are exactly the ones we selected using the proposed approach since each c from C is guaranteeed to return 1 evaluating on either x_1 or x_2. So it is sufficient only to consider these c.

Next, we prove the computational complexity of our proposed approach is linear in n. Given any two n-dimensional example x_1 and x_2, we are able to compute the number of their common literals in $\mathcal{O}(n)$. Then we compute $1!, 2!, ..., \text{numCommon}(x_1,\, x_2)!$ in $\mathcal{O}(n)$. The final step is to perform summation of all chooses which could be finished in $\mathcal{O}(1)$. Therefore, the proposed approach could be computed in $\mathcal{O}(n)$.

3.2 part b

Data: Training samples denoted by x_t.
Result: A hypothesis h

set $h = x_1 \vee \neg x_1 \vee x_2 \vee \neg x_2 ... x_n \vee \neg x_n$;
for *every example* (x_t, y_t) **do**
  for *entry* x_i *in* s **do**
    if $x_i = 1$ **then**
      remove x_i from h;
    else
      remove $\neg x_i$ from h;
    end
  end
end
for *sample* $s \in Tr_s$ *with label* 1 **do**
  if *all entry(entries) in s can not be found in h* **then**
    return: Cannot find a hypothesis h consistent with all training samples;
  end
end
return: h;

Algorithm 1: Kerlenl Perceptron Algorithm