CS446: Machine Learning **Fall 2014**

Problem Set HW 4

Siting Chang *Handed In: 10/17/2014*

1 VC Dimension

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 combinations in terms of labels. We claim that H is able to classify all the combinations.

The origin and radius choices are listed as follows to classify each combination. The order of the terms in each line 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

So we proved that the VC dimension of H is at least 3.

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 hypotheses in H when the opposing points with the largest distance both have positive labels 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 $b_1 < x_1 < a_2$).

After observing the cases with two points and three points, we see that the most complicated case, 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 reason 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 disjoint intervals. As long as we have enough intervals, at least four intervals, to cope this situation, the points are shattered. Figure 1 gives an illustration of this situation.

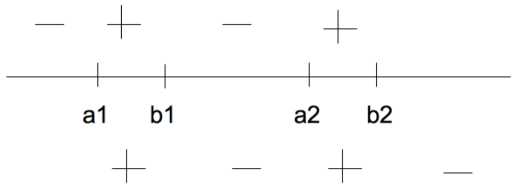

Figure 1: Four points with neghboring potins have opposite labels.

Note that with 4 parameters we have 5 intervals on the real line, however, the maximum number of points that could be shattered with these 4 parameters is only 4 which is described in Figure 1.

Now, we generalize the situation to 2k points on the real line. There are k disjoint intervals within which points are labeled as positive and another k+1 disjoint intervals within which points are negative. The largest number of points that can be shattered is 2k since we are able to assign at most 2k points with neighboring points have opposite labels into 2k intervals.

So far, we showed that the VC dimension of H is at least 2k.

Next, we show that 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 where $x_{leftmost} > a_1$, there is no way any hypothesis from H can classify the right most point which has a positive label.

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 condition 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 given 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 conclude their 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. Initialize the sample space S as a set of all the training data, and decision list DL as empty.

2. Set the default output which is denoted by b as 0.

3. Check if sample space S is empty. If so, stop the algorithm. If not, continue.

4. 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.

5. Move the samples from S into DL if it makes e true.

6. Put e in decision list DL as a condition along with its output label l.

7. For the rest samples in S, repeat step 2 to 6 until S is empty.

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

The intuition of this algorithm is that if a conjunction item e (later added as a condition to the decision list) 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 items that are 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 both the size of the k-decision list and its computational complexity.

Again, we adopt the notations L_n to represent 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.

We claim there are $3^{|C_k^n|}$ possible combinations since each term in C_k^n has three options of missing, label as negative and label as positive. Another thing to notice is that the order of conditions appeared in decision list is arbitrary which means number of $(C_k^n)!$ possible orders.

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 complexity 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 outputs 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 up-bounded 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 approach 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 contribute 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 guaranteed 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 $1!, 2!, ...,$ 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 examples denoted by $x_t$ with their corresponding label $y_t$.
Result: Mistake counter $\vec{\alpha}$ with $m$ dimension.
1 set $\vec{\alpha} = 0$;
2 for every training example $(x_t, y_t)$ do
3     $y_{pred} = sign(\sum_i^m \alpha_i y_i K(x_i, x_t))$
4     if $y_{pred} \neq y_t$ then
5         update $\alpha_t$: $\alpha_t = \alpha_t + 1$;
6     end
7 end
8 return: $\alpha$;
```

Algorithm 1: Kernel Perceptron Algorithm

where m denotes the number of training examples available, α_i denotes the number of mistakes made on example i.

After obtaining the classifier, in order to predict label for future examples, such as x_{new}, we follow:

$$y_{pred} = sign(\sum_i^m \alpha_i y_i K(x_i, x_{new})) \tag{2}$$

where x_i, y_i are provided by training data while α_i is returned by the training algorithm.

3.3 part c

One key point to notice is that running kernel Perceptron in the original sample space is the same as running original (or unmodified) Perceptron in the blown-up space. In our case, the blown-up space has m dimensions. m is the number of training samples available. Each training example x is mapped to the new space P according to the kernel function

$K(x_i, x)$ where $i \in 1, 2, ...m$. To be more clear, training example x_i is mapped to point $p_i =< K(x_1, x_i), K(x_2, x_i), ..., K(x_m, x_i) >$.

First, we try to find R. According to the Novikoff's theorem, for the original Perceptron, $R \geq \|x_i\|$ for any i. Inspired by this, in order to find the R for the new space we need to find the maximum p_i in the space P. Based on the kernel function specified in part a, we see that for any pair of examples, $K(\cdot, \cdot) \leq \sum_{j=0}^{n} \binom{n}{j}$ since the maximum number of common literals between any pair of examples is n. Therefore, we claim that $\|p_i\| \leq \sqrt{m(\sum_{j=0}^{n} \binom{n}{j})^2}$. We define $R = \sqrt{m(\sum_{j=0}^{n} \binom{n}{j})^2}$

Next, we try to find γ. Given by the problem statement, we know for sure we could learn a classifier that is consistent with all the training data. Therefore, in the new space P, we could compute the minimum distance between any two points with one point has positive label and the other point has negative label. The distance could be described as follows:

$$d = min\|p_1 - p_2\| \tag{3}$$

where p_1 is any point from P that has positive label while p_2 has negative label.

With the returned vector $\vec{\alpha}$, we know it satisfies the equation

$$y_i \alpha p_i \geq \frac{1}{2} d \tag{4}$$

Normalizing α by dividing both side with $\|\alpha\|$ we have $y_i \alpha p_i \frac{1}{\|\alpha\|} \geq \frac{1}{2} d \frac{1}{\|\alpha\|}$.

Suppose each training example is being trained k times, which means $\|\alpha\| \leq \sqrt{km}$ in which km describes the worst possible case that all examples were misclassified k times. Now, we substitute $\|\alpha\|$ back in the right hand side of the above equation and obtain

$$y_i \alpha p_i \frac{1}{\|\alpha\|} \geq \frac{1}{2} d \frac{1}{\sqrt{km}} \tag{5}$$

in which $\alpha \frac{1}{\|\alpha\|}$ can be seen as the u in the Novikoff's theorem. Therefore, we define

$$\gamma = \frac{1}{2} d \frac{1}{\sqrt{km}} \tag{6}$$

At last, we conclude the maximum number of mistakes the kernel Perceptron is bounded by $\frac{R^2}{\gamma^2}$ in which R and γ are defined as above.