CS446: Machine Learning	**Fall 2015**
Problem Set 4	
Dominik Durner	*Handed In: October 22, 2015*

Question 1

1.a

Lower Bound: We can show that the lower bound of the triangles is at least 7. For all possible combination I just used the idea of having symmetries in the graph. Therefore, I am just showing each shape at least once since every shaped triangle can be positioned differently and in all possible angles. As seen for the rectangles in the lectures, all points that lie in the triangle are marked positive all others negative. Therefore, I am going to showcase different graphs for each of the number of positive labeled points.

If we consider the points being drawn as the heptagon (see Figure 1), we can always shatter the points regardless the labeling. This can be seen, easily that 1 and 2 points can always be positive if we draw a thin triangle around them. 6 points can always be positive since I just use a big triangle which excludes the one point by being parallel to this point (see the orange triangle in Figure 1).

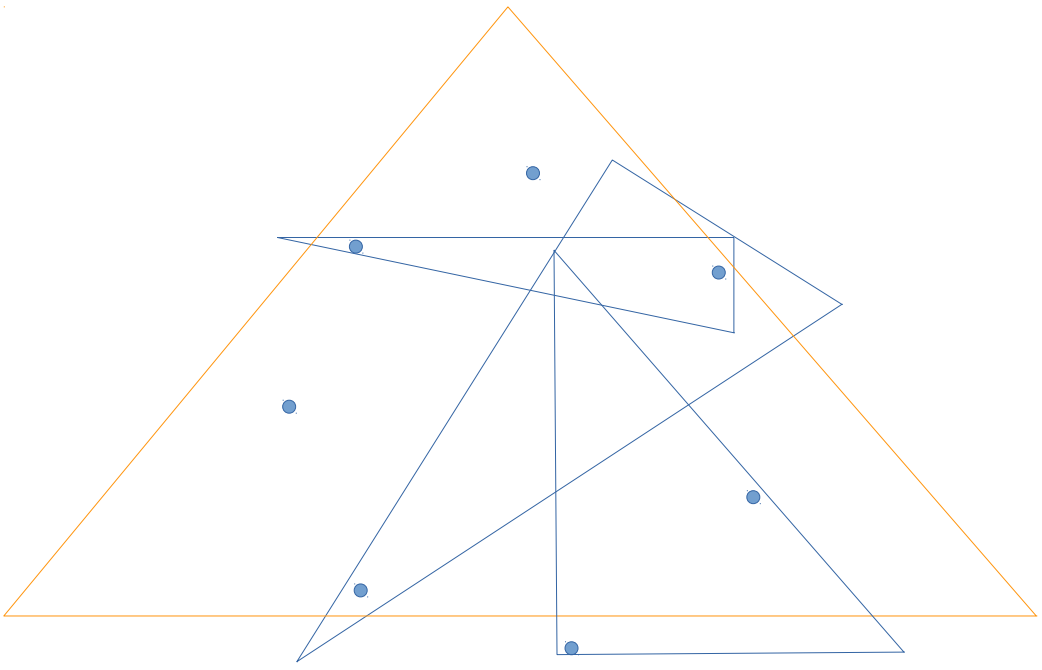


Figure 1: Heptagon with triangles for 2 and 6 positive

If we have three positive points those three can always lie together in one of the shapes (or repositioned and turned triangles) of Figure 2.

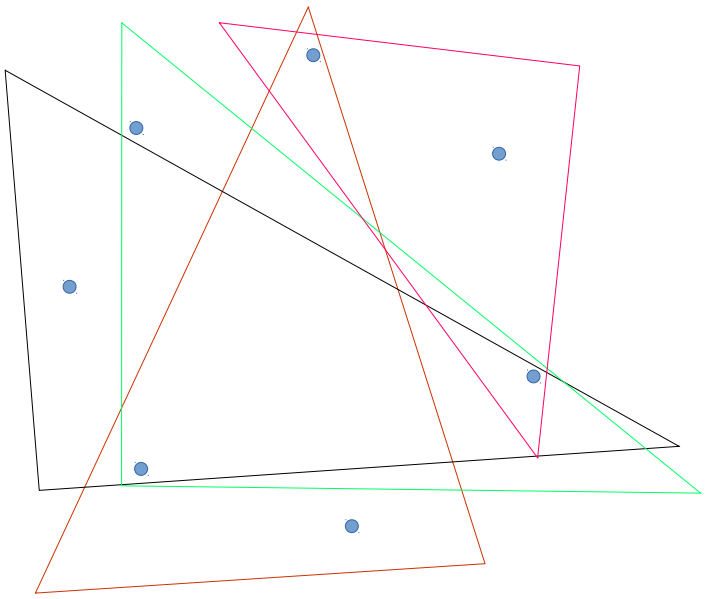


Figure 2: Heptagon with triangles for 3 positive

The same holds for all possible combinations if we have a labeling with 4 positive points. This can be found in Figure 3.

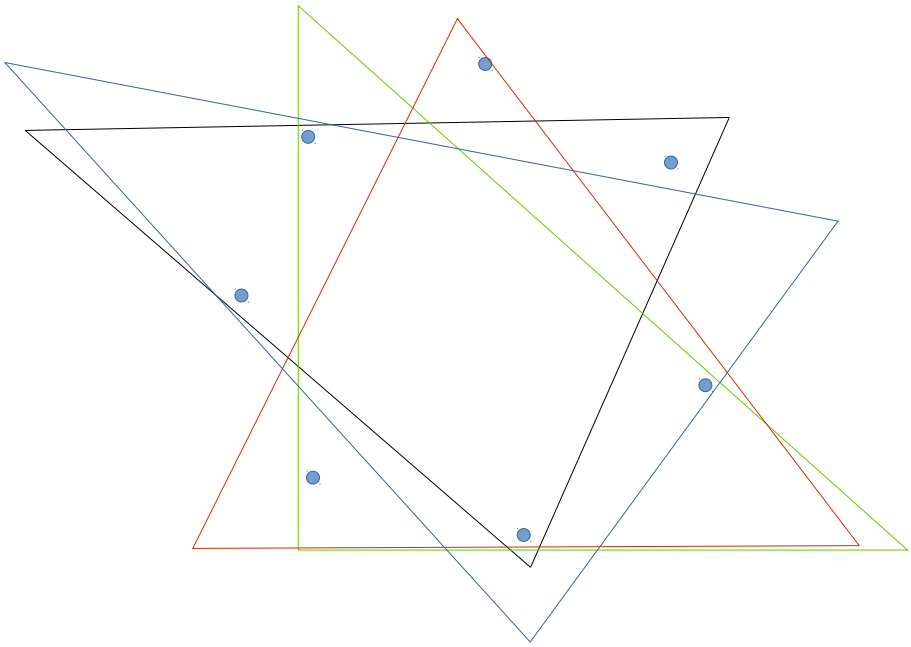


Figure 3: Heptagon with triangles for 4 positive

In Figure 4 we can see all shapes that are needed to shatter 5 positive points in a heptagon.

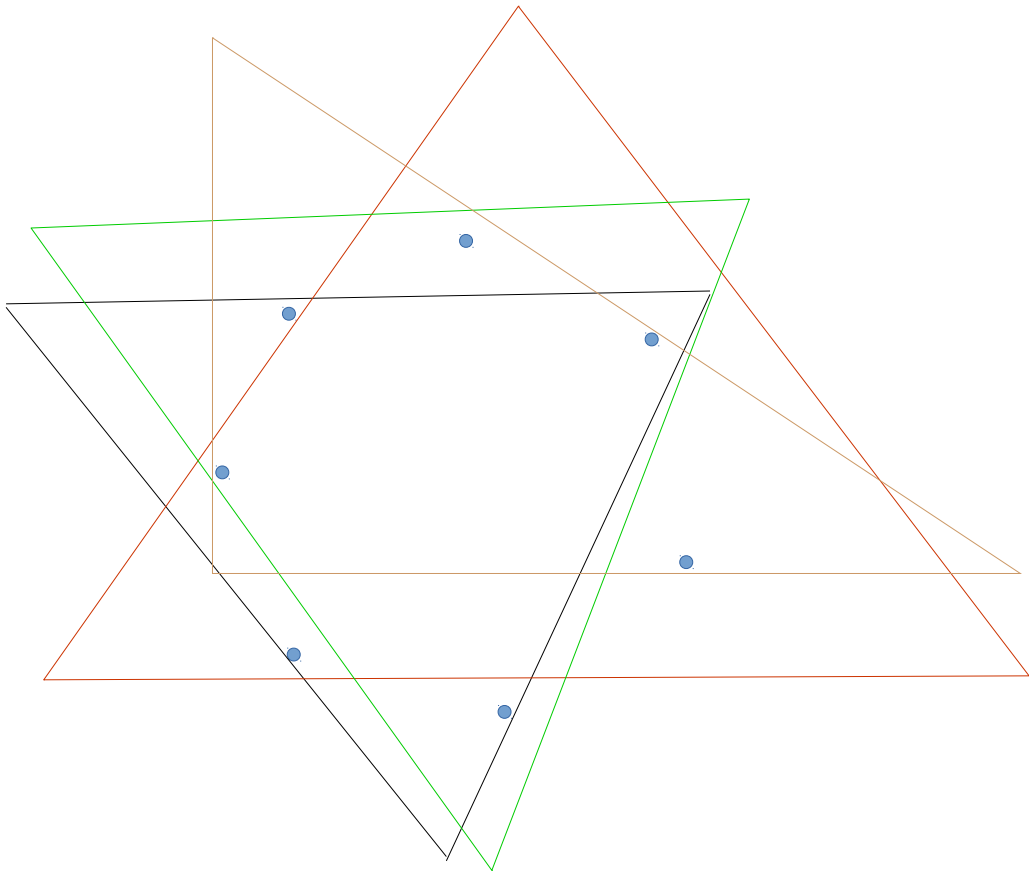


Figure 4: Heptagon with triangles for 5 positive

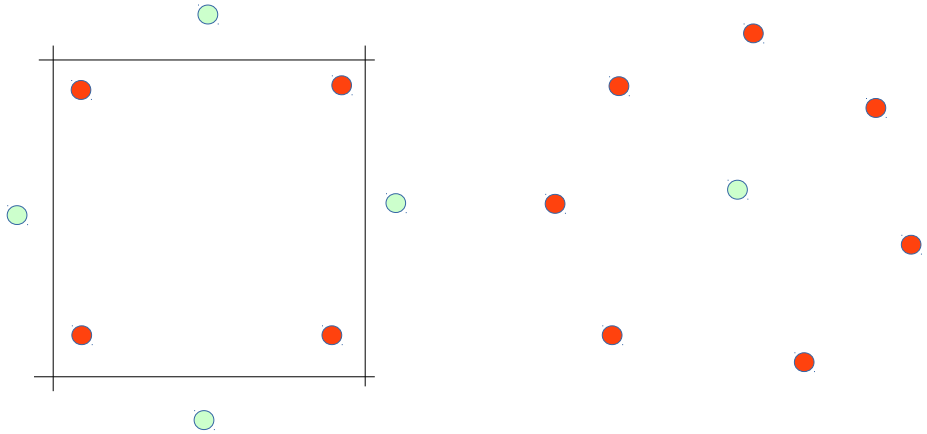
Furthermore, the base case of all 7 and 0 points positive are trivial by just having a big triangle around all points or a small one which doesn't include any.

Hence, we showed that for every number of positive points we can shatter all possible point labeling combinations with 7 points in the heptagon configuration. Hence the VC dimension of the problem 1.a is at least 7.

Upper Bound: If we consider the two possible options of either having a convex hull that is either on all nodes or spans only on less than all nodes (therefore at least one node is included into the convex hull), we can see that the dimension can not be 8. Since without loss of generality we can think about all possible configurations with the idea of those two classes (and all points on one line, which is not possible to shatter if the labeling is alternating) of having a fully convex setting or not. It is easy to see if the configuration is not fully convex, we can always set the outside points to be positive in the inner one negative. This can be easily seen in Figure 5b.

If the configuration is convex I can find an alternating labeling scheme with 4 positive and 4 negative points. The problem is that I can not find a triangle including those 4 points without having a negative one included. This can be seen in Figure 5a if we think about those black lines as infinite lines. I can only move those lines on the connection points of each of the other line but I am never able to bring two connection points together to one

position without cutting a negative one. For example if the left-lower three positive ones are included the upper right positive one is not. To include it we need to stretch both sides, that are not the hypotenuse (the black lines of the triangle). If we do so, we can include the upper right positive one but we will never be able to not include one of the negative ones in the upper right quarter. Since this is generally true for all convex settings, since I always can find the alternating labeling such that I get to those 4 black connected lines with a negative point on the outside of the line. If I don't find the setting of having 4 lines separating the positive ones from the negative ones, the configuration was not convex in the first place. Hence, I am running in the previously described problem that I cannot include all points without including a negative one. Therefore, there is no setting with 8 points in which an arbitrary labeling can be shattered.



(a) Convex hull is over all nodes (b) With nodes inside

Figure 5: Hexagon is not possible

After we had the lower bound of 7 and we saw that we can not find a shattering with 8 points, we know that the VC dimension is 7. Hence, VC(problem 1.a) = 7.

1.b

If we divide the problem into sub problems of 2 points we can separate all possible labelings of those two points with one starting and one end bound. On the other hand it is not possible to shatter 3 points with the labeling order of +,-,+. Hence, the VC dimension of the sub problem is 2.

Lower Bound: If we now want to glue those subproblems together we can have ++, -+, +- or – as possible labelings for the right point of the left interval and the left point of the right interval. We always need a separator if the labeling is different of those two points, otherwise we can just increase the current negative / positive interval by moving one separator. Since we have to look for the worst case labeling we need to consider a labeling for that needs 4 separators. By further induction we can find that we can shatter at least 2k points with 2k separators.

Upper Bound: Let's think about the alternating labeling of 2k+1 points starting with a + label. Therefore, we get the +,-,+...,-,+ label order. One important observation is that this labeling starts and ends with a + label. Since $[a_i, b_i]$ is the interval for positive labeling we need at least 2k+1 of those positive interval boxes since we always alternate between + and - labels. Hence, we can only put one + label in one of the positive interval boxes. Accordingly, we would need 2k+1 boxes. This is impossible with having only 2k separators and therefore we showed that the upper bound is 2k.

Hence, VC(problem 1.b) = 2k.

Question 2

2.a

For finding the negation of a decision list, we can just consider the not negated decision list of this problem and negate all the outcomes b_i and b. Therefore, we just get the following decision list.

$$\overline{c} = \{(c_1, \overline{b_1}), ..., (c_k, \overline{b_k}), \overline{b}\}$$

$\overline{c}$ still satisfies all the conditions for a k-term decision list, since we didn't change the term rules c_i, nor the structure of the decision list.

2.b

K-DNF: We know that a K-DNF consists of a conjuncted disjunctions.

$$f = d_1 \vee ... \vee d_l \text{ , with } d_i = v_1 \wedge ... \wedge v_m$$

To represent the up to k conjunctions each variable v_i can be either a real variable representing our input or 1 (to represent non consisting variable that do not change the result). We can now build a k-DL with the fact that if one term d_i evaluates to true the whole function f is true. Therefore, we get the following decision list.

$$DL = \{(d_1, 1), ..., (d_l, 1), 0\}$$

The default b has to be 0, since we just opt out of the default value if one of our d_is is true. Since each d_i has at maximum k variables, it is consistent with the definition of a k-DL.

K-CNF: We can build a K-DNF from a K-CNF. To achieve this, we need to negate the whole K-CNF including the the overall outcome.

$$g = c_1 \wedge ... \wedge c_l \text{ , with } c_i = v_1 \vee ... \vee v_m$$

Be g our K-DNF and be each variable either a real variable or 0 (to represent non consisting variable that do not change the result) we can rewrite this K-CNF as a negated K-DNF.

$$\begin{aligned}
&\overline{g} = \overline{c_1 \wedge ... \wedge c_l} \text{ , with } c_i = v_1 \vee ... \vee v_m \\
\Leftrightarrow\ &\overline{g} = \overline{c_1} \vee ... \vee \overline{c_l} \text{ , with } c_i = v_1 \vee ... \vee v_m \\
\Leftrightarrow\ &\overline{g} = c_1 \vee ... \vee c_l \text{ , with } c_i = \overline{v_1 \vee ... \vee v_m} \\
\Leftrightarrow\ &\overline{g} = c_1 \vee ... \vee c_l \text{ , with } c_i = \overline{v_1} \wedge ... \wedge \overline{v_m}
\end{aligned}$$

With the fact of 2.a that every negation of K-DL is also a K-DL and we transformed a K-CNF to a K-DNF, we showed that also K-CNF can be rewritten to K-DL.

Hence, we showed K-CNF $\cup$ K-DNF $\subseteq$ K-DL ■

2.c

1) Let's be K-Term the set of all possible functions with up to k variables
   and b(x) the function which return the b value of the input x
2) While $S \neq \emptyset$
   2.1) Find a concunction c $\in$ K–Term and a subset $S_s \subseteq S$ such that
   $(\forall x \in S_s : c(x) = 1 \wedge b(x) = 1) \vee (\forall x \in S_s : c(x) = 1 \wedge b(x) = 0)$
   2.2) add $(c, b(x))$ to the Decision List
   2.3) $S = S \setminus S_S$
3) Add 0 as the default b value (doesn't matter whether 0 or 1, but needed to have a complete DL)

Proof of Correctness: For the first step we need to be able to compute the possible conjunctions. As we see in question 2.d., this is bound polynomial in $\mathcal{O}(n^k)$ with k as a constant. Hence, we can compute the set of all possible decision list rules c_i. b(x) is just the function which returns the corresponding y value to a x vector, so therefore b(x) should be the b result for vector x.

Importantly, we need to operate over all our input vectors until we all consider them for a rule. For the first step in the while loop, we know that we are always able to find a concunction such that there is a subset that is having the same result for the whole subset. This comes directly from the properties of the question that the problem is K-DL learnable and therefore there can not be two vectors which would be equal for all up to k conjunctions and have different b values.

If we come to the point where S is empty we know that every vector has been assigned to a b value after applying a rule c. Let's consider x_l with value b_l and say the rule which removed x_l from the set is given with c_m. Since the set was not removed from the set for all variables c_i with $i = [0...m-1]$, we know that the vector can not get one of the b values before the conjunction c_m. If the value of the vector would now be calculated again it would first skip all c_i rules until it reaches c_m. Since c_m is matching and the corresponding b_m value of c_m is given by b_l, we know that the decision list would now directly return b_m. Hence, we found the right b value for the vector.

We can also be sure to find a rule for each of the vectors, since we know that we S is k-DL consistent. Therefore, we can always find a conjunction for each of the variables such as we can find a match of a vector to a b value. Furthermore, we can just assign 0 or 1 as default value since we don't need that to be k-DL consistent. One of the rules that would match all the vectors is already included and can always be applied instead of using the default b value. This conjunction is the plain True conjunction with 0 variables.

Hence, we saw that the algorithm would find for each vector x a matching conjunction such that the value of the decision list initialized with the vector x would return the b-value of x.

2.d

To apply Occams Razor we need to bound the Hypothesis space. As we already know the number of up to k-Term conjunctions is bound by $\mathcal{O}(n^k)$. Let's state that p is the cardinality of the possible conjunctions $p = |\text{k-Term}|$. Since we can have either 0 or 1 as result for b we get, 2^p possible decision list pieces (on rule aka (c_i, b_i)). Furthermore, a decision list takes also the ordering into account. Hence we have maximal $p!$ different orderings of those pieces.

Hence we get the overall size s.

$$s = 2^p * p!, \text{ with } p = |\text{k-Term}|$$

Now we want to show that we are in polynomial time of $|H|$ in Occams Razor. First, let us remember the formula for the number of samples we need to see.

$$m > \frac{1}{\epsilon}(ln(|H|) + ln(\frac{1}{\delta}))$$

The only important factor for our calculations is $ln(|H|)$. Therefore, we will just look at this constraint in the following.

$$ln(|H|) = ln(2^p * p!) = p * ln(2) + ln(p!) \leq p * ln(2) + p * ln(p)$$

We can now resubstitute $p = |\text{k-Term}| = \mathcal{O}(n^k)$.

$$\mathcal{O}(ln(|H|)) = \mathcal{O}(n^k * ln(2) + n^k * ln(n^k))$$

Hence, the logarithm of the hypothesis space is bound polynomial in n for Occams Razor since k is a constant ($\forall k > 0$). Therefore, K-DL is PAC learnable!

Question 3

3.a

All possible conjunctions are bound by $\mathcal{O}(n^k)$. We know that the product $c(x)c(z) = 1$ if and only if both terms are evaluated to 1. Since we use conjunctions we know that we just need to consider the variables for the conjunction that has 1 values in both x and z. Otherwise if the conjunction would include a variable which is either 0 in x or z, the whole term $c(x)c(z)$ would be 0. Therefore, we get b as the number of the variables that are on in both x and z. Since our kernel space is bound by k variables, we know that there can not be more than k variables in a conjunction. Hence, we get following number of possible conjunctions.

$$\text{Possible Conjunctions} = \sum_{i=1}^{k} C_i^b$$

Since the evaluation of function c is bounded by $\mathcal{O}(n)$, all evaluations of the kernel can be bounded by the costs for the evaluation of c multiplied by all possible conjunctions. Those conjunctions are bounded by $\sum_{i=1}^{k} C_i^b$ and therefore we know that the whole calculation is bounded in poly(n).

3.b

```
1  PerceptronWithKernel(XYVectorSet):
2  M = ∅
3  w = (0...0)^T
4  Θ = 0
5  foreach (x, y) ∈ XYVectorSet
6        ŷ = sign(Σ_{(z_x,z_y)∈M}(z_y * K(x, z_x)) + Θ)
7        if y ≠ ŷ
8              M = M ∪ (x, y)
9              Θ = Θ + y
10       endif
11 endforeach
12
13 h(x) = sign(Σ_{(z_x,z_y)∈M}(z_y * K(x, z_x)) + Θ)
14 return h(x)
```

3.c

The Novikoff theorem is predicting the mistake bound on an Perceptron algorithm with the evaluation function $wx \geq 0$. Hence, the θ must already be included in the dimension (see lectures that we can always go to the n+1 dim room instead of saving Θ!). The theorem needs t (x_i, y_i) pairs with $x_i \in R^n$ and $||x_i|| \leq R$. It further requires a $w \in R^n$, and a $\gamma > 0$ such that w is a unit vector and $y_i * w^T * x_i \geq \gamma$. Then the Perceptron algorithm does make at most $\frac{R^2}{\gamma^2}$ mistakes on the t given test pairs.

Let's first think about what $||x_i||$ is in the kernel space. Therefore, we need to find out the maximum number of possible conjunctions to describe all features.

$$\text{Possible Conjunctions} = \sum_{i=1}^{k} C_i^n$$

Including the additional dimension for Θ we get the following $||x_i||$.

$$||x_i|| = \sqrt{x_i^T * x_i} \leq \sqrt{\sum_{i=1}^{k} C_i^n + 1} \text{ (equality if all features are on, } x_{ij} \in \{0|1\})$$

Therefore, we can bound $R^2 = \sum_{i=1}^{k} C_i^n + 1$.

Next we need to consider γ which defines our Perceptron margin. Therefore, we want to maximize our margin without loosing the possibility to learn k-DNFs in the new feature space. As an intermediate step we need to calculate the unit vector $\hat{w}$ since our Perceptron doesn't need to give us a unit vector w .

$$\hat{w} = \frac{w}{|w|}$$

Hence, we get the following equation.

$$\forall x_i, y_i : \gamma \leq y_i * \hat{w}^T * x_i$$

Following we can found the smallest inner product and bound γ

$$\gamma = min\{\forall x_i : |\hat{w}^T * x_i|\}$$

Thus, the mistakes are bound by M.

$$M \leq \frac{R^2}{\gamma^2} = \frac{\sum_{i=1}^{k} C_i^b + 1}{(min\{\forall x_i : |\hat{w}^T * x_i|\})^2}$$