5.

Let be the set of all undirected graphs over n vertices numbered 1,…,n with degree at most 7.

as the described function

(a)

--TODO-- what do you mean in “smallest dependence on plz respond

(b)

Let us calculate the size of the hypothesis class:

Any labeled undirected graph size n can have different edges as the way to pick 2 vertices

and for any graph we can choose between assign a specific edge or not

thus we got labeled graphs with n vertices, also we should leave the graph with no edges because .

In conclusion

As we showed in class the PAC-learning upper bound for the sample complexity of learning H is:

הערה של עמית שגר בארמון:

קצת קשה להבין אם מדובר במקרה הrealizable או במקרה האגנוסטי כשהם לא תיארו את השאלה,

לקבוצת האיקסים הנתונה לנו כל פונקציה תחזיר 1 רק עבור x בודד מבין כל האפשריים, אבל לא ברור לי מתי זה שגיאה או לא... בגלל זה רשמתי פה את שניהם.

For the realizable case:

For the agnostic case:

(c)

For any such that we can only label but there’s no such that and thus we cannot shutter any group size 2,

Thus the VC dimension of H is 1

As we showed in class the better upper bound for the sample complexity of learning H depends on the VC dimension of H is:

הערה:

קרא הערה סעיף (b)

For the realizable case:

For the agnostic case:

6.

For

We showed in class the Preceptron performs at most updates.

Thus we should find of the Sample S.

We also showed in class Hard-SVM algorithm with input S return which is a maximal-margin separator.

Now we can describe the algorithm who returns an upper bound on Preceptron number of updates:

**Input**: a labeled sample S of labeled examples from X × Y

**Output**: an upper bound on the number of updates that the Perceptron algorithm would require if it was run on this sample.

1. runs Hard-SVM on sample S
   1. if Hard-SVM gets an error while running then the algorithm returns -1
   2. else w Hard-SVM output
      1. R sample with max norm between all samples
      2. Dxmin
      4. Return

**complexity:** Hard-SVM has polynomial runtime complexity

thus we get **polynomial runtime complexity**

7. (a)

In order to express the problem

in terms of

,

we will define variable and rewrite the problem as

,

(b)

since we only need sum of squares of , we don’t need to use , we can compile it into :

8. (a)

9.(a)

To prove that cannot be a kernel function for any feature mapping, we need to show such that .

from inner products property: . Hence:

even though for we get that

which means for some , in contradiction to inner product properties. Hence, no such exists.

(b)

Assume in contradiction that

Thus, for

for we get .

we assumed that such that and got contradiction, hence no such exists and cant be a kernel function.

(c)

let

we found some that satisfy , hence can be a kernel function.