CS446: Machine Learning **Fall 2014**

Problem Set Problem 1

Nikhil Unni *Handed In: September 11, 2014*

1. Learning Disjunctions

   a. My algorithm for learning the disjunction is actally to use only the negative examples. Because a single "truth" value in the disjunction will make the output positive, we can iterate through all the negative examples and take out all the "truths." For positive examples, a value of 1 or 0 has very little meaning if there are other features with 1

```
1. Get m training examples of n long Boolean vectors
2. Initialize a set of all 2n possible members of the target function : x_1, ¬x_1, x_2, ¬x_2, ...
3. For each training example, x:
4.     If x is labeled −1:
5.         For i = 1 to n:
6.             Remove the appropriate member, x_i, of the set, i.e. if x_2 is 1, remove x_2,
               or if x_10 is 0, remove ¬x_10
7. If the set is empty:
8.     There exists no consistent hypothesis for the training data
9. Else:
10.    For all remaining members of the set, join them as a disjunction
```

   b. The idea behind the algorithm is that it only takes a single 1 in a disjunction of booleans to make the output positive. So because of this, for each negative example, x, we take out all potential members of the final hypothesis, h, that would make $h(x)$ positive. By this principle, our final h is for sure consistent for all negative examples (unless the data are inconsistent).

   Also, because this is a disjunction, the "right" values will always remain in the final hypothesis. All incorrect additions do not affect the accuracy of the training data (although increase likelihood of false positives when testing it on outside data). For example, if the target function is $x_1 \vee \neg x_4$, the proposed hypothesis has to be of the form : $x_1 \vee \ldots \vee \neg x_4 \vee \ldots$. Just the fact that x_1 and $\neg x_4$ are in the proposed hypothesis, none of the positive examples can be misclassified. This is merely a property of the OR function - If we say the target function is f, and the remaining additions of our hypothesis are h', because $\vee$ is associative, this can be represented as $f \vee h'$. Because f is the target function, if the data are consistent, it will always evaluate to 1. This makes our hypothesis always $1 \vee h'$, which is 1, regardless of what h' is, for all positive examples in the training data. This same reasoning can be used to show that the final hypothesis is consistent

with negative examples, and that there's no way that, in the process of iterating through the training data, the current hypothesis is inconsistent with past indeces of the iteration.

It's also worth mentioning that even in the case that there are no negative examples, our hyper-overfitted hypothesis, $x_1 \vee \neg x_1 \vee x_2 \vee \neg x_2 \vee \ldots$ will simplify to 1, which will be true for all of the training data, merely because there are no negative examples.

c. Initializing the set is an $O(n)$ operation, just adding $2n$ elements to a set.
Then, we double iterate through n and m, and at each stage we remove an element of a set. If we model our "set" as an array of size $2n$, where index 0 is x_1, index 1 is $\neg x_1$, etc. we can instantly access these values (1 if present in the final function, 0 if not) because we have the current index. This makes "removal" from the set $O(1)$. So far that's $O(n) + O(n)O(m)$.
After that, we iterate through the remaining set for the final hypothesis, which is just $O(n)$. Putting that all together, and taking only the significant terms, the algorithm is $O(n^2m)$.

d. I kind of alluded to some cases in part A. But, in general, I'd say that the algorithm increases in robustness with more negative examples. The extreme case with 0 negative examples produces the function $h(x) = 1$, which will match all of the positive test examples, but if the new, unlabed datum is a negative example it will for sure be wrong.
An interesting property of the algorithm (and probably boolean functions in general?) is that it cannot produce a function that gives a false negative value. Again, this is just because the target function is always embedded in our hypothesis. However all of the "extra" function, or h', to go back to my last example, contributes to false positives when trying out on outside data. The more negative data we have in our training set out of the set of all possible negative examples, the more we minimize h', and get closer to the target function f, which, in turn, increases our accuracy for outside data.

2. Answer to problem 2

3. Answer to problem 3