CS 6350: Homework 2

Michael Young / u0742759

1 Warm Up: Linear Classifiers and Boolean Functions

1. $\neg x_1 \vee \neg x_2 \vee x_3$: YES: $(1 - x_1) + (1 - x_2) + x_3 - 1 \geq 0$: w = [-1, -1, 1], b = 1
2. $(x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3)$: YES: The expression functionally reduces to $x_1 \wedge \neg x_3$ which can be represented by the linear expression $x_1 + (1 - x_3) \geq 2 : w = [1, 0, -1], b = -1$.
3. $(x_1 \wedge x_2) \vee \neg x3$: NO
4. $x_1 xor x_2 xor x_3$: NO
5. $\neg x_1 \wedge x_2 \wedge x_3$: YES: $(1 - x_1) + x_2 + x_3 \geq 3 : w = [-1, 1, 1], b = -2$

2 Mistake Bound Model of Learning

1. **Consider the hypothesis space Pk consisting of parity functions with at most k vari- ables. Each parity function in this set is defined by selecting at most k of the n variables, and taking the XOR of them. For example, if k = 3, the set P3 would contain functions such as x1 xor x12, xn−2 xor xn−1 xor xn, and so on.**

   1. What is the number of functions in the set P_k

      As stated in the question, our hypothesis space consists of functions that include at most k of the n variables. Because of this, the total number of functions is represented by:

$$
\begin{aligned}
&= \binom{n}{k} + \binom{n}{k-1} + \cdots + \binom{n}{1} \\
&= \frac{n!}{k!(n-k)!} + \frac{n!}{(k-1)!(n-(k-1))!} + \cdots + 1 \\
&= \sum_{j=1}^{k} \frac{n!}{j!(n-j)!}
\end{aligned}
$$

      I'm not good at math so I don't know how to reduce this further :) , but I can reason that the largest this could be is when k = n, which would make the whole thing 2^n.

2. If we use the Halving algorithm we saw in class, what is the upper bound on the number of mistakes that the algorithm would make? You can use the result we derived in class and state your answer in terms of n and k.

   As discussed in class, the Halving algorithm makes at most $\log C$ mistakes where C is the number of functions in the concept class or hypothesis space. We saw that $\binom{n}{k} \approx O(n^k)$. In the sum representing the total count of functions above we saw that that our answer was $= \binom{n}{k} + \binom{n}{k-1} + \cdots + \binom{n}{1}$, meaning the upper bound would be $O(n^{k+1})$. Thus, the upper bound on mistakes would be $= O(k \log n)$.

2. **Consider the hypothesis space H consisting of indicator functions for Boolean vectors in the n-dimensional space. That is, for every vector z ∈ {0, 1}n, the set H will contain a function fz defined as: (refer to HW pdf for the rest)**

   1. What is the number of functions in H?

      As stated in the question, for every vector $z \in \{0,1\}^n$, there is a function in H corresponding to a particular vector z. Thus, the number of functions in H is defined by how many vectors z there are. This number is defined by the dimensionality n. Thus, the number of function in our hypothesis space $H = 2^n$.

   2. Suppose you ran the Halving algorithm for this hypothesis space. How many mistakes will the algorithm make? You will need to justify your answer with a proof.

      If we are dealing with an n dimensional space, then we will have 2^n functions in our hypothesis space. For each instance x that we receive, there will be exactly one function in the hypothesis space that returns a 1 for that instance, and all other functions will return a 0. Thus, we see that no matter what our instance, the majority will always be 0. Furthermore, this majority pick of 0 as the prediction will be correct every single time, until the instance $x = z$, where z corresponds to the z in the oracle function. Once we stumble upon this instance x which equals z, we will have made a mistake, and the entire hypothesis space will be reduced to the 1 correct function. At this point, the algorithm will end. Thus, the halving algorithm paired with this particular hypothesis space will only ever make 1 mistake.

   3. Is the mistake bound you derive above the same as what we saw in class for the general Halving algorithm? If so, construct a sequences of examples that forces the algorithm to make those many mistakes for any dimensionality. If not, explain why there appears to be a discrepancy between the two mistake bounds?

      No, it's not. In class we found that the bound was $\log C$ and here we found that it's 1. In our proof for the general case of the Halving algorithm, we reasoned that every time the algorithm makes a mistake, the new concept space is at most half of the previous size of the concept space. Thus, each mistake halves the initial size of the concept class, meaning that a total of log C mistakes are made. This particular case of the Halving algorithm is different because no matter the size of concept class, there will only ever be 1 mistake, which afterwards perfectly identifies the correct function. This is of course the benefit of completely mapping every possible instance in the instance space to a corresponding function in the hypothesis space.

3. **Now, consider a different hypothesis space C that is defined very similarly to H above, except that it switches the 0's and 1's in the output of the functions. That is, the set C consists of functions gz of the form: (see hw pdf for details).**

   Is the mistake bound for the hypothesis space C the same as the one you derived above for H? Prove your claim.

   Yes, the mistake bound should be the same - 1 mistake should be made. Whereas before, the algorithm would always predict 0, and be right every time except for the 1 time the instance matched the z of the oracle, the same reasoning applies, except now, the algorithm will always choose 1 and be correct until it stumbles upon the instance x which equals the z of the oracle function.

3. The Perceptron Algorithm and its Variants: Experiments

1. Briefly describe the design decisions that you have made in your implemen- tation. (E.g, what programming language, how do you represent the vectors, etc.)

   I used python for my implementation. I stored the csv files for the training, test, and folds data as numpy arrays. This allowed for efficient matrix operations when it came time to run my different perceptron implementations. I built a Data class that had various helper functions for the data (such as shuffling). I also built a perceptron class that contained all the methods for predicting and training for each of the perceptron variants we were asked to construct.

2. Majority baseline: Consider a classifier that always predicts the most fre- quent label. What is its accuracy on test and training set?

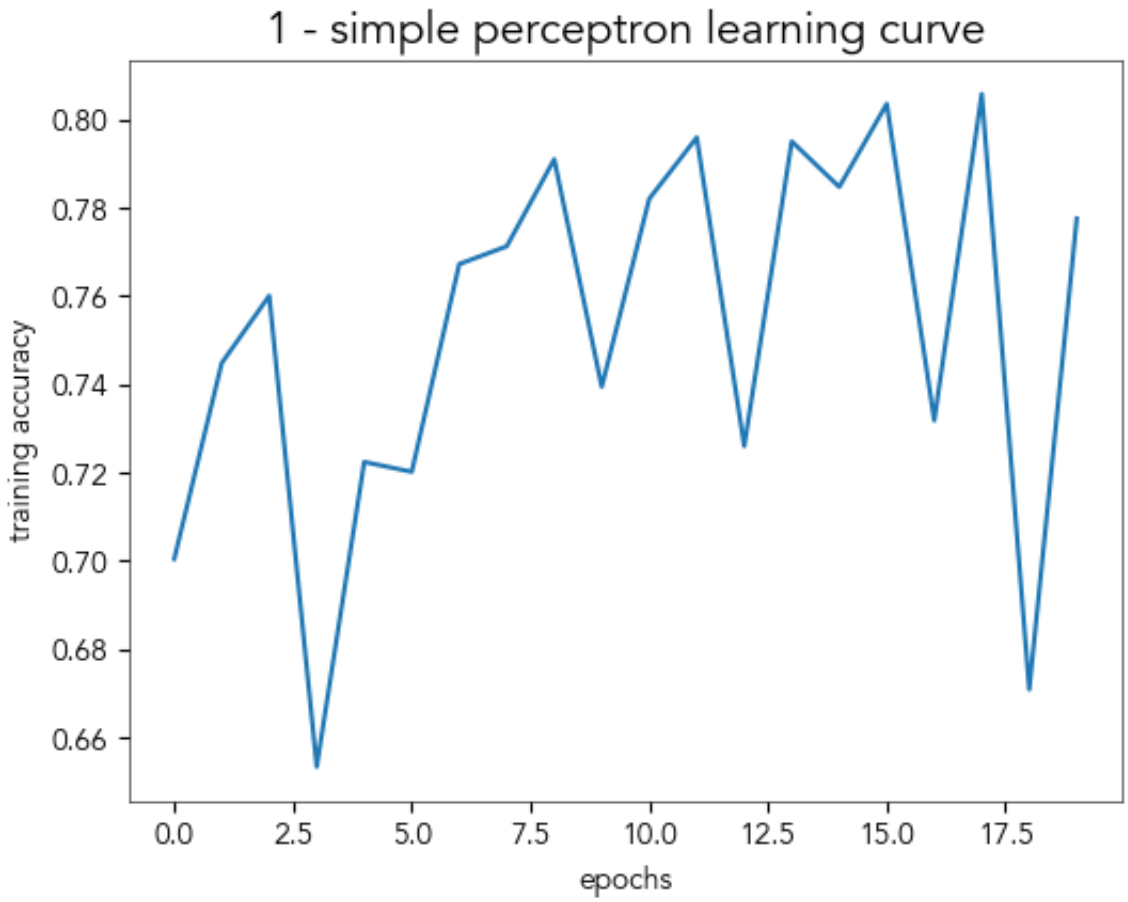
   training majority baseline: Accuracy = 0.5139013452914798, Most frequent label = 1.0

   testing majority baseline: Accuracy = 0.543010752688172, Most frequent label = 1.0

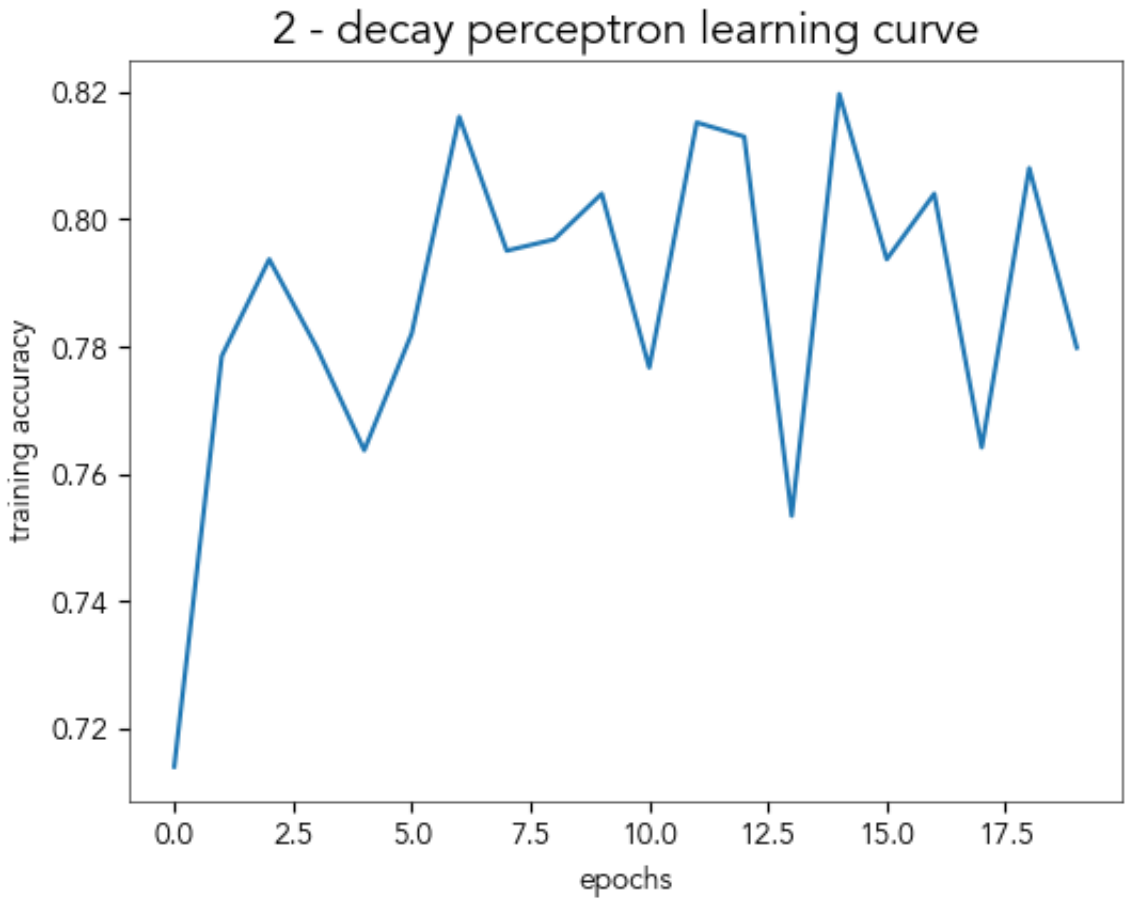
3. Results for each variant:

   1. **Simple perceptron**

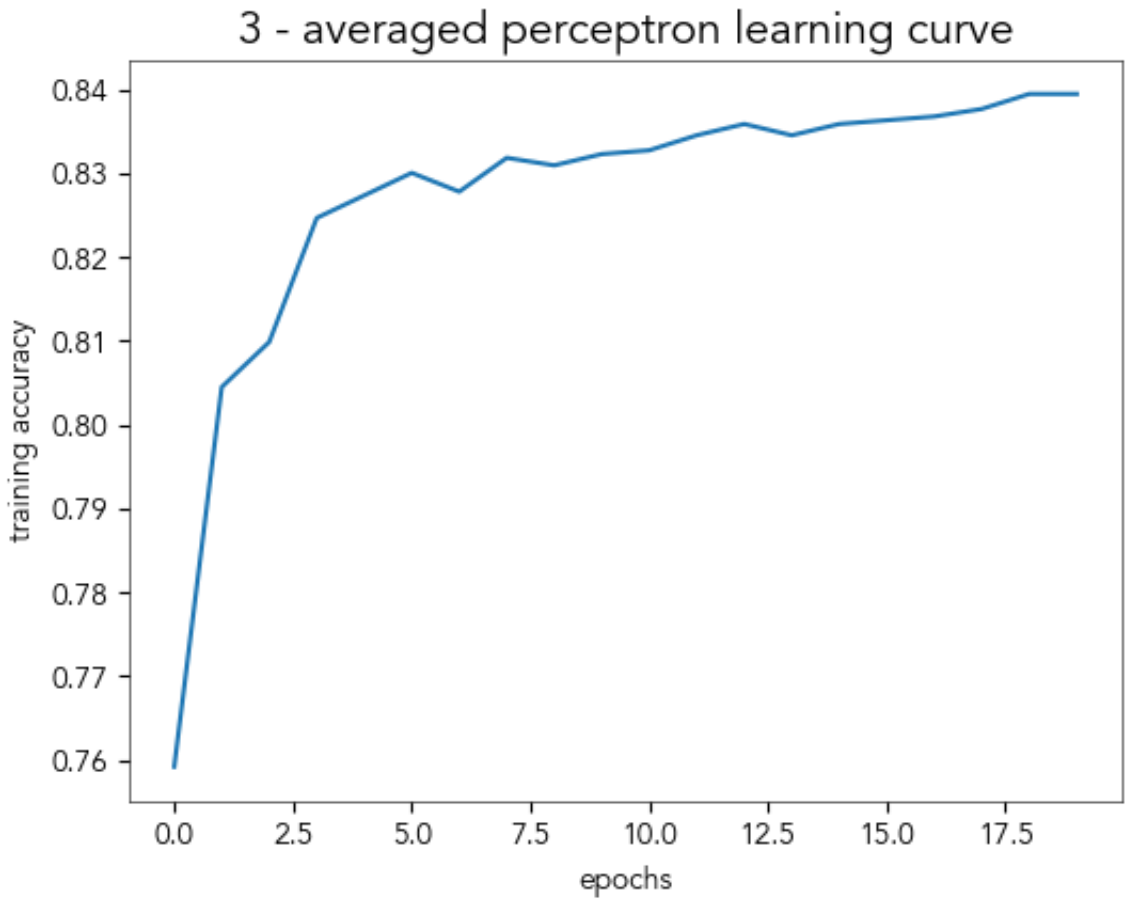
      1. Best learning rate: 0.01
      2. Cross validation accuracy w/ best hyper parameter: 0.7417040358744394
      3. Total number of updates done while training: 11974
      4. Training set accuracy: 0.805829596412556
      5. Test set accuracy: 0.7455197132616488
      6. Learning curve:



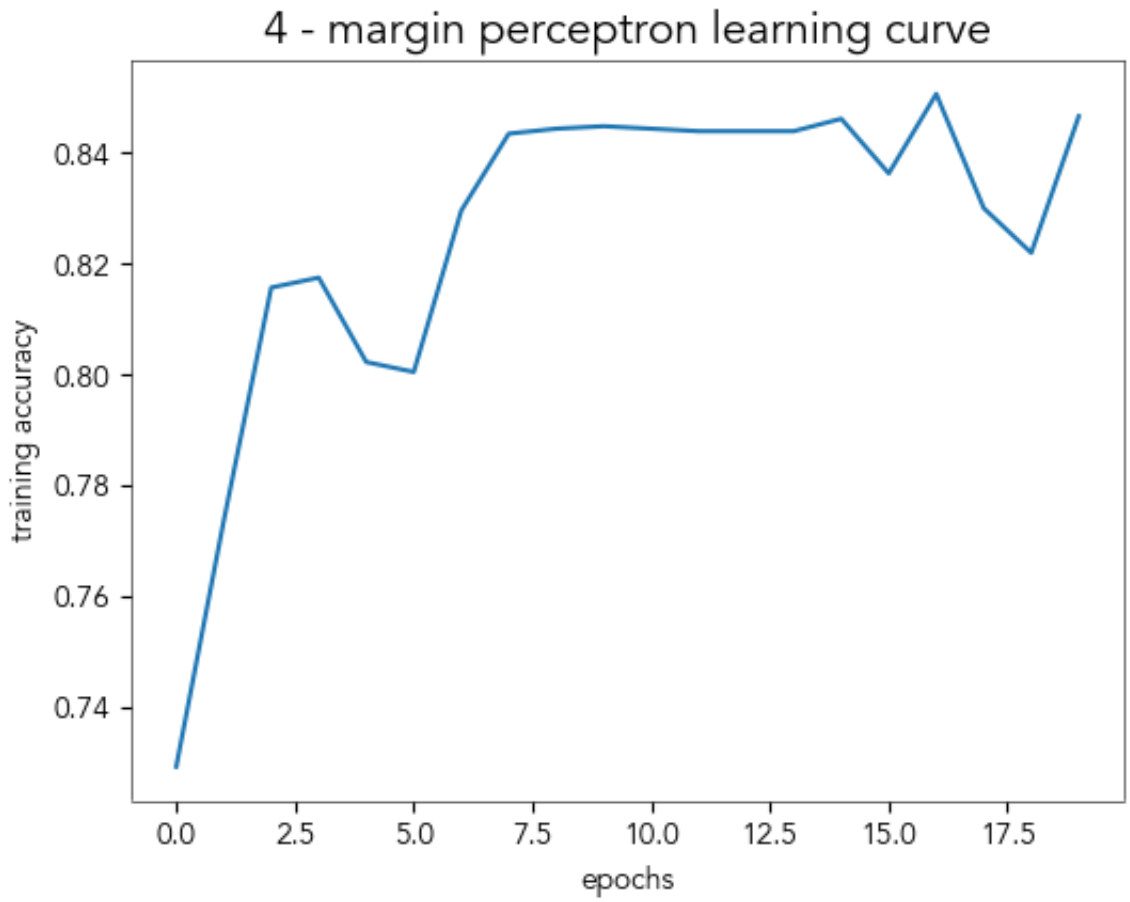
2. **Decayed learning rate**
    1. Best learning rate: 1
    2. Cross validation accuracy w/ best hyper parameter: 0.7632286995515695
    3. Total number of updates done while training: 9920
    4. Training set accuracy: 0.8197309417040358
    5. Test set accuracy: 0.7885304659498208
    6. Learning curve:



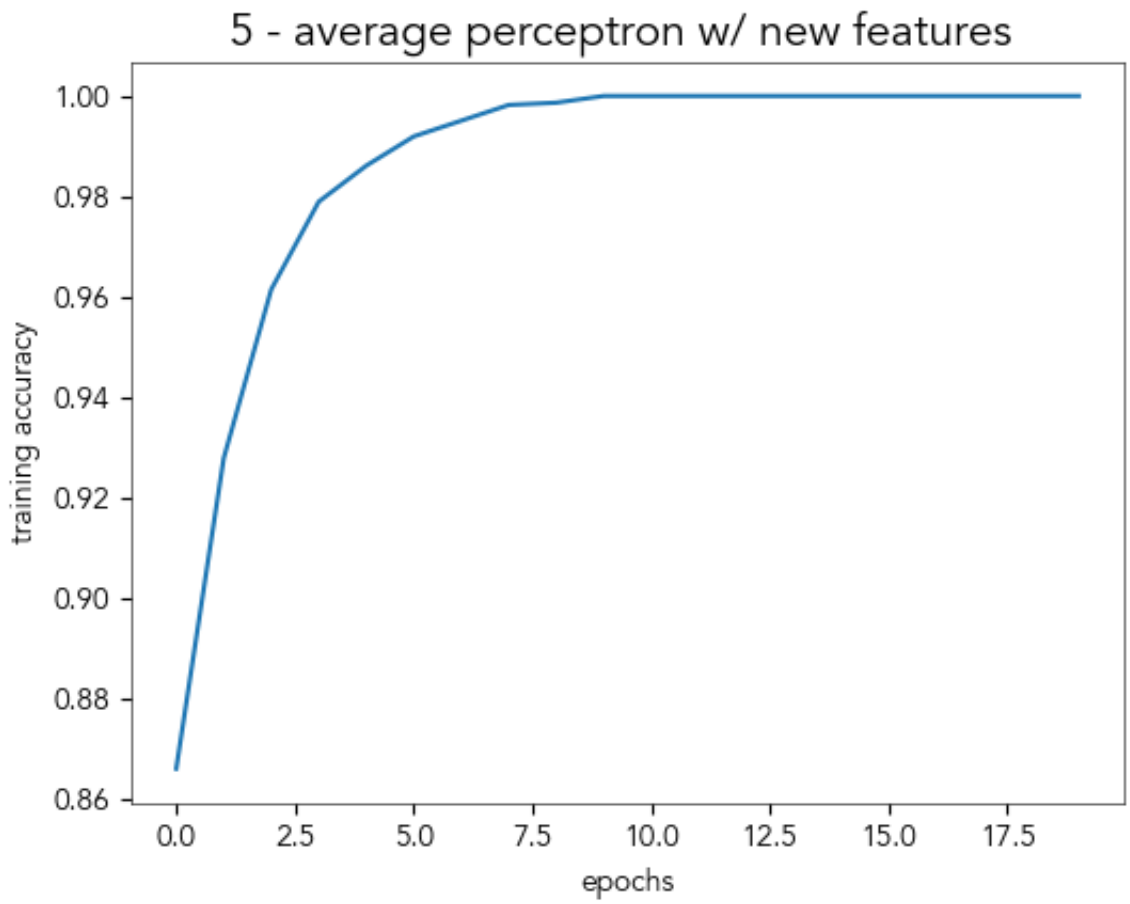
3. **Averaged Perceptron**
    1. Best learning rate: 1
    2. Cross validation accuracy w/ best hyper parameter: 0.7690582959641254
    3. Total number of updates done while training: 11788
    4. Training set accuracy: 0.8394618834080717
    5. Test set accuracy: 0.8028673835125448
    6. Learning curve:



4. **Margin perceptron**
    1. Best learning rate: 0.1, Best margin: 1
    2. Cross validation accuracy w/ best hyper parameters: 0.768609865470852
    3. Total number of updates done while training: 17917
    4. Training set accuracy: 0.8506726457399103
    5. Test set accuracy: 0.8046594982078853
    6. Learning curve:



5. **Extra credit**
    1. Best learning rate: 1
    2. Cross validation accuracy w/ best hyper parameter: 0.8188340807174889
    3. Total number of updates done while training: 2181
    4. Training set accuracy: 1.0
    5. Test set accuracy: 0.8405017921146953
    6. Learning curve:


5 - average perceptron w/ new features
training accuracy
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epochs