CS 6350/DS 4350: HW4 Linear concepts/classes with perceptron algorithms

Name: Jackson Switzer

Student ID: u1608753

Question 1 [50 pts]— Linear Separability & LTUs

Inputs and labels are in $\{-1, +1\}$.

Q1.1 [10 pts]

$f_1(x_1, x_2) = x_1 \wedge \neg x_2$ (i.e., +1 iff $x_1 = +1$ and $x_2 = -1$; else -1).

Answer

$f_1(x_1, x_2) = \text{sign}(\ [1, -1]\mathbf{x} - 1\ )$

	x_1 = 1	**x_1 = -1**
x_2 = 1	sign[(1)(1) + (-1)(1) - 1] = -1	sign[(1)(-1) + (-1)(1) - 1] = -1
x_2 = -1	sign[(1)(1) + (-1)(-1) - 1] = +1	sign[(1)(-1) + (-1)(-1) - 1] = -1

Q1.2 [10 pts]

$f_2(x_1, x_2, x_3)$ is odd parity on $\{x_1, x_2, x_3\}$: +1 iff an odd number of $\{x_1, x_2, x_3\}$ are +1; otherwise −1. (Equivalently, $f_2(\mathbf{x}) = x_1x_2x_3$.)

Answer

Not linearly separable. The justification is on the lecture slides titled "Linear Modeling," slide numbers 48 and 49. Parity is not linearly separable. When you plot it, you can see why; the sign alternates between every octant, so there is no way to draw a single plane separating all the positives from the negatives.

Q1.3 [10 pts]

$f_3(x_1, x_2, x_3, x_4) = +1$ iff at least three of $\{x_1, x_2, x_3, x_4\}$ are $+1$; otherwise -1.

Answer

$f_3(x_1, x_2, x_3, x_4) = \text{sign}([1, 1, 1, 1]\mathbf{x} - 1.5)$

x_1	x_2	x_3	x_4	f_3
-1	-1	-1	-1	-1
1	-1	-1	-1	-1
-1	1	-1	-1	-1
-1	-1	1	-1	-1
-1	-1	-1	1	-1
1	1	-1	-1	-1
1	-1	1	-1	-1
1	-1	-1	1	-1
-1	1	1	-1	-1
-1	1	-1	1	-1
-1	-1	1	1	-1
1	1	1	-1	1
1	1	-1	1	1
1	-1	1	1	1
-1	1	1	1	1
1	1	1	1	1

Q1.4 [10 pts]

General m-of-n. Let the d input features be $x_1, \ldots, x_d$ and let $R \subseteq \{1, \ldots, d\}$ be the set of n relevant indices. Define

$f_{\text{m-of-n}}(x) = \{ +1$ if at least m of $\{x_j : j \in R\}$ are $+1$; -1 otherwise

Write a single LTU $h(x) = \text{sign}(w^T x + b)$ (specify all w_j and b) that realizes $f_{\text{m-of-n}}$ for the given m, n, R, and briefly explain why your choice avoids ties.

Answer

$f_3(x_1, x_2, x_3, x_4) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$ where

- **w** is a vector with d elements w_j
- $w_j = 1$ if $j \in R$; otherwise, $w_j = 0$
- $b = n - 2m + 0.5$

This works, first, because the irrelevant indices are ignored by setting their weights equal to 0. Second, the relevant indices are summed (+1 if x_j=1, -1 if x_j=-1), since all their weights are 1. Third, the bias term adjusts the result, so the threshold crosses over zero (without ever landing on it) at the correct number of terms—when at least m of the relevant indices are 1.

This choice avoids ties because the elements of **x** and **w** can only be integers, so $\mathbf{w}^T\mathbf{x}$ is always an integer. By setting $b = n - 2m + 0.5$, we offset the function so it always lands halfway between integers, meaning it can equal 0.5 or -0.5, but never 0.

Question 2 [100 pts +10 pts for bonus]— Coding of Perceptron Algorithm.

Q2.2.1 Simple Perceptron [25 points]

Majority baseline test accuracy: 0.553
Majority baseline development accuracy: 0.558252427184466

a) Briefly describe the design decisions that you have made in your implementation. (e.g., what programming language, how do you represent the vectors, etc.)
All of the models are implemented in Python and the vectors are represented using NumPy arrays. For each epoch, I shuffled the data, and then the activation function <w_t, x_i> + b is multiplied by y_i. If that is less than or equal to 0, then the prediction is incorrect and we must update the weights using the perceptron update formulas. I needed to keep track of the number of steps also, so that weights could carry forward through each epoch.
b) The best hyperparameters
    - lr: 1.0
    - Number of epochs: 2
c) The cross-validation accuracy for the best hyperparameter(s): 0.989
d) Development set accuracy with 2 epochs: 0.985
e) Test set accuracy: 0.976
f) Total number of updates the final learning algorithm performs on the training set (across all epochs): 96

Q2.2.2 Decaying the learning rate [25 points]

Majority baseline test accuracy: 0.553
Majority baseline development accuracy: 0.558252427184466

a) Briefly describe the design decisions that you have made in your implementation.
   Decaying the learning rate required only a small adjustment to the basic perceptron algorithm. By adding an "if" statement before the activation function and update rule, I verified whether the decay_lr parameter was True, and if it was, then I slightly lowered the learning rate. This is inside the outer "for" loop, the one for epochs, because the instructions said to put it there, but we could also have had a time step for each sample.
b) The best hyperparameters
   - lr: 1.0
   - Number of epochs: 2
c) The cross-validation accuracy for the best hyperparameter(s): 0.990
d) Development set accuracy with 2 epochs: 0.995
e) Test set accuracy: 0.966

Total number of updates the final learning algorithm performs on the training set (across all epochs): 83

Q2.2.3 Margin Perceptron [25 points]

Majority baseline test accuracy: 0.553
Majority baseline development accuracy: 0.558252427184466

a) Briefly describe the design decisions that you have made in your implementation. (e.g., what programming language, how do you represent the vectors, etc.)
The margin perceptron just required another small change to the existing perceptron. Instead of checking if y_i times the activation function is less than 0, now we check if it's less than mu. If it is, we update the weights. That's the only difference.
b) The best hyperparameters
    - lr: 0.01
    - mu: 1.0
    - Number of epochs: 1
c) (CV) The cross-validation accuracy for the best hyperparameter(s): 0.995
d) (epochs) Development set accuracy with 2 epochs: 0.981
e) (final) Test set accuracy: 0.985

Total number of updates the final learning algorithm performs on the training set (across all epochs): 161

Q2.2.4 Averaged Perceptron [25 points]

Majority baseline test accuracy: 0.553
Majority baseline development accuracy: 0.558252427184466

a) Briefly describe the design decisions that you have made in your implementation. (e.g., what programming language, how do you represent the vectors, etc.)
   The averaged perceptron is a bit different from the others; that's why it requires its own class. First, I went back to the basic perceptron, with no decay or margin. Then, after every example, I add the weights and bias to their cumulative sums self.w_sum and self.b_sum, so that after all the epochs are finished we can divide the sum of the weights by the number of samples to get the average weights for the whole training process.
b) The best hyperparameters
   - lr: 1.0
   - Number of epochs: 8
c) (CV) The cross-validation accuracy for the best hyperparameter(s): 0.989
d) (epochs) Development set accuracy with 8 epochs: 0.981
e) (final) Test set accuracy: 0.981

Total number of updates the final learning algorithm performs on the training set (across all epochs): 213

Q2.2.5 Aggressive Perceptron with Margin [Bonus 10 points]