1 Linear Separability

Solution for Question 1.1

Step 1: Analyze the Boolean function

The function represents a Boolean OR operation with negated inputs. Let's create a truth table:

x_1	x_2	$\neg x_1$	$\neg x_2$	$f_1(x_1, x_2)$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

From the table, $f_1(x_1, x_2)$ outputs 1 when at least one of x_1 or x_2 is 0 and outputs 0 only when both are 1.

Step 2: Determine linear separability

We need to design a linear inequality to separate the outputs of the function. Define the decision boundary as:

$$w_1x_1 + w_2x_2 + b = 0$$

We want:

- $f_1(x_1, x_2) = 1$ when $w_1x_1 + w_2x_2 + b > 0$
- $f_1(x_1, x_2) = 0$ when $w_1x_1 + w_2x_2 + b \leq 0$

Since $f_1(x_1, x_2)$ is true for $(0, 0)$, $(0, 1)$, and $(1, 0)$ but false for $(1, 1)$, a suitable LTU can be:

$$-w_1x_1 - w_2x_2 + 1.5 > 0$$

Explanation of the decision boundary: Evaluate the inequality for each point in the truth table:

- For $(0, 0)$: $-0 - 0 + 1.5 = 1.5 > 0$ (output $= 1$)
- For $(0, 1)$: $-0 - 1 + 1.5 = 0.5 > 0$ (output $= 1$)
- For $(1, 0)$: $-1 - 0 + 1.5 = 0.5 > 0$ (output $= 1$)
- For $(1, 1)$: $-1 - 1 + 1.5 = -0.5 \leq 0$ (output $= 0$)

The inequality separates the two classes correctly.

Conclusion

The LTU for the function $f_1(x_1, x_2) = \neg x_1 \vee \neg x_2$ is:

$$-x_1 - x_2 + 1.5 = 0$$

This LTU produces the same output as the given function and correctly classifies all input combinations, confirming linear separability.

Solution for Question 1.2

Step 1: Analyze the Boolean function

This function is equivalent to the XOR function between x_1 and x_3. It outputs 1 when x_1 and x_3 have different values and 0 when they have the same value. Note that x_2 is a "don't care" variable in this case. The truth table is as follows:

x_1	x_2	x_3	$\neg x_3$	$x_1 \wedge \neg x_3$	$\neg x_1 \wedge x_3$	$f_2(x_1, x_2, x_3)$
0	*	0	1	0	0	0
0	*	1	0	0	1	1
1	*	0	1	1	0	1
1	*	1	0	0	0	0

Step 2: Determine linear separability

Since this function is an XOR between x_1 and x_3, it is **not linearly separable**. There is no linear boundary that can separate the points where the output is 1 from the points where the output is 0.

Conclusion

The function $f_2(x_1, x_2, x_3) = (x_1 \wedge \neg x_3) \vee (\neg x_1 \wedge x_3)$ is **not linearly separable**.

Solution for Question 1.3

Step 1: Analyze the function

The function outputs $+1$ if at least two of the inputs x_2, x_3, x_4 are 1. Otherwise, it outputs -1. This function does not depend on x_1. Let's create a few example cases to understand its behavior:

x_2	x_3	x_4	Number of 1's	Output f_3
0	0	0	0	-1
0	0	1	1	-1
0	1	1	2	+1
1	1	1	3	+1

Step 2: Define the linear threshold unit

We need to create a linear threshold function of the form:

$$w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b = 0$$

Since the function only depends on x_2, x_3, and x_4, we can set $w_1 = 0$. The goal is to find weights w_2, w_3, w_4 and bias b such that:

$$\text{Output} = \text{sign}(w_2x_2 + w_3x_3 + w_4x_4 + b)$$

Step 3: Construct the decision rule

We want the function to output:

- $+1$ when at least two of x_2, x_3, x_4 are 1.
- -1 when fewer than two of x_2, x_3, x_4 are 1.

A suitable decision rule is:

$$w_2x_2 + w_3x_3 + w_4x_4 > 1.5$$

Let's set $w_2 = w_3 = w_4 = 1$ and $b = -1.5$. The function becomes:

$$f(x_2, x_3, x_4) = \text{sign}(x_2 + x_3 + x_4 - 1.5)$$

Step 4: Verify the solution

We will test the decision rule on a few cases:

- **Case:** $(x_2, x_3, x_4) = (0, 0, 0)$

$$x_2 + x_3 + x_4 - 1.5 = 0 + 0 + 0 - 1.5 = -1.5 < 0 \quad \text{(Output = -1)}$$

- **Case:** $(x_2, x_3, x_4) = (0, 0, 1)$

$$x_2 + x_3 + x_4 - 1.5 = 0 + 0 + 1 - 1.5 = -0.5 < 0 \quad \text{(Output = -1)}$$

- **Case:** $(x_2, x_3, x_4) = (0, 1, 1)$

$$x_2 + x_3 + x_4 - 1.5 = 0 + 1 + 1 - 1.5 = 0.5 > 0 \quad \text{(Output = +1)}$$

- **Case:** $(x_2, x_3, x_4) = (1, 1, 1)$

$$x_2 + x_3 + x_4 - 1.5 = 1 + 1 + 1 - 1.5 = 1.5 > 0 \quad \text{(Output = +1)}$$

The decision rule correctly classifies all cases.

Conclusion

The LTU for the function $f_3(x_1, x_2, x_3, x_4)$ is:

$$f(x_1, x_2, x_3, x_4) = \text{sign}(x_2 + x_3 + x_4 - 1.5)$$

Solution for Question 1.4

Step 1: Understanding the function

- We are working with d total features.
- Out of these d features, a set of n relevant features are selected.
- The function outputs $+1$ if at least m of the n relevant features are 1, and -1 otherwise.

This function generalizes the **threshold logic** used in previous questions, such as the **2-of-3** function in Question 1.3.

Step 2: Define the linear threshold unit

We need to define a linear threshold function of the form:

$$w_1x_1 + w_2x_2 + \cdots + w_dx_d + b = 0$$

Since only the n relevant features matter, let's denote these relevant features by $x_{r_1}, x_{r_2}, \ldots, x_{r_n}$. We can assume weights of 1 for all relevant features and 0 for irrelevant ones.

The decision rule becomes:

$$x_{r_1} + x_{r_2} + \cdots + x_{r_n} > m - 0.5$$

Here, $m - 0.5$ serves as the threshold. This rule ensures that the function outputs:

- $+1$ when at least m of the n relevant features are 1.
- -1 when fewer than m relevant features are 1.

Step 3: Verify the decision rule

We can verify this by testing a few cases:

- **Case: Fewer than m relevant features are 1**
  Suppose exactly $m - 1$ relevant features are 1. Then, the sum of the relevant features is:

  $$x_{r_1} + x_{r_2} + \cdots + x_{r_n} = m - 1$$

  Since $m - 1 \leq m - 0.5$, the output will be -1, as expected.

- **Case: At least m relevant features are 1**
  Suppose exactly m relevant features are 1. Then, the sum of the relevant features is:
  $$x_{r_1} + x_{r_2} + \cdots + x_{r_n} = m$$
  Since $m > m - 0.5$, the output will be $+1$, as expected.

Conclusion

The LTU for the **m-of-n** function is:

$$f(x_1, x_2, \ldots, x_d) = \text{sign}\left(\sum_{i=1}^{n} x_{r_i} - (m - 0.5)\right)$$

This LTU correctly classifies the inputs based on whether at least m of the n relevant features are 1.

2 Mistake Bound Model of Learning

Solution for Question 2(a)

Step 1: Understanding the concept class

- The instance space consists of all integer points (x_1, x_2) within the range $-80 \leq x_1, x_2 \leq 80$.
- The concept class C is parameterized by l, which defines a square decision boundary centered at the origin.
- For each l, the function outputs $+1$ if both $|x_1|$ and $|x_2|$ are less than or equal to l. Otherwise, the output is -1.

Step 2: Counting the number of functions

The size of the concept class $|C|$ is determined by the number of distinct values that l can take. Since l can range from 1 to 80, the number of possible functions is:
$$|C| = 80$$

Conclusion

The size of the concept class C is:

$$|C| = 80$$

Solution for Question 2(b)

Step 1: Hypothesis behavior

The hypothesis $f_l(x_1, x_2)$ predicts $+1$ when both $|x_1|$ and $|x_2|$ are within the boundary defined by l. It predicts -1 otherwise:

$$f_l(x_1, x_2) = +1 \iff |x_1| \leq l \text{ and } |x_2| \leq l$$

$$f_l(x_1, x_2) = -1 \iff |x_1| > l \text{ or } |x_2| > l$$

Step 2: Condition for a mistake

A mistake occurs when the predicted label $f_l(x_1^t, x_2^t)$ does not match the true label y_t. The two scenarios are:

- **Positive example mistake:**
  - True label $y_t = +1$
  - The hypothesis predicts -1, i.e., $|x_1^t| > l$ or $|x_2^t| > l$
- **Negative example mistake:**
  - True label $y_t = -1$
  - The hypothesis predicts $+1$, i.e., $|x_1^t| \leq l$ and $|x_2^t| \leq l$

We can generalize this with the following condition:

$$y_t \cdot f_l(x_1^t, x_2^t) \leq 0$$

This inequality checks whether the prediction is incorrect:

- If $y_t = +1$ and $f_l(x_1^t, x_2^t) = -1$, then $y_t \cdot f_l(x_1^t, x_2^t) = -1$, indicating a mistake.
- If $y_t = -1$ and $f_l(x_1^t, x_2^t) = +1$, then $y_t \cdot f_l(x_1^t, x_2^t) = -1$, also indicating a mistake.

Step 3: Express the condition explicitly

Based on the definition of $f_l(x_1, x_2)$, the prediction can be expressed as:

$$f_l(x_1^t, x_2^t) = \begin{cases} +1, & \text{if } |x_1^t| \leq l \text{ and } |x_2^t| \leq l \\ -1, & \text{otherwise} \end{cases}$$

A mistake occurs when:

$$y_t \cdot f_l(x_1^t, x_2^t) \leq 0$$

Alternatively, the mistake conditions can be expressed as:

$$\text{Mistake occurs if:} \quad (y_t = +1 \text{ and } (|x_1^t| > l \text{ or } |x_2^t| > l)) \quad \text{or} \quad (y_t = -1 \text{ and } |x_1^t| \leq l \text{ and } |x_2^t| \leq l)$$

Conclusion

The condition to check whether the hypothesis f_l makes a mistake on point (x_1^t, x_2^t) with label y_t is:

$$y_t \cdot f_l(x_1^t, x_2^t) \leq 0$$

Solution for Question 2(c)

Step 1: Understanding the role of l

The parameter l defines a square boundary centered at the origin. The hypothesis predicts $+1$ if both $|x_1|$ and $|x_2|$ are within this boundary. If a mistake occurs, it implies that either:

1. The boundary l is too small to correctly classify a positive example, or
2. The boundary l is too large, causing a negative example to be incorrectly classified as positive.

Step 2: Update rule for l

1. **Mistake on a positive example:**
   - If the true label $y_t = +1$, but $f_l(x_1^t, x_2^t) = -1$, it means that either $|x_1^t| > l$ or $|x_2^t| > l$.
   - To correctly classify this point, we need to increase l so that the condition $|x_1^t| \leq l$ and $|x_2^t| \leq l$ holds.
   - **Update rule:**
   $$l \leftarrow \max(l, |x_1^t|, |x_2^t|)$$
2. **Mistake on a negative example:**
   - If the true label $y_t = -1$, but $f_l(x_1^t, x_2^t) = +1$, it means that $|x_1^t| \leq l$ and $|x_2^t| \leq l$.
   - To correctly classify this point, we need to decrease l so that at least one of $|x_1^t|$ or $|x_2^t|$ is greater than l.
   - **Update rule:**
   $$l \leftarrow \min(l, \max(|x_1^t|, |x_2^t|) - \epsilon)$$
   Here, ϵ is a small positive value to ensure the point is outside the boundary after the update.

Step 3: General update rule

If a mistake occurs, update l based on the type of example:

$$l \leftarrow \begin{cases} \max(l, |x_1^t|, |x_2^t|), & \text{if } y_t = +1 \\ \min(l, \max(|x_1^t|, |x_2^t|) - \epsilon), & \text{if } y_t = -1 \end{cases}$$

Conclusion

The update rule for l ensures that the hypothesis boundary adapts to correctly classify the current example.

Solution for Question 2(d)

Step 1: Recap of concepts

- The hypothesis is defined by a parameter l, which determines a square boundary around the origin.
- A mistake occurs when the hypothesis incorrectly classifies a point (x_1^t, x_2^t) with label y_t.
- The algorithm updates the boundary parameter l to correct the mistake based on the type of example:
  - Increase l if the mistake occurs on a positive example.
  - Decrease l if the mistake occurs on a negative example.

Step 2: Algorithm design

We design the algorithm in the form of pseudocode:

Algorithm: Mistake-driven learning algorithm for boundary parameter l

```
1. Initialize: $l \leftarrow 1$
2. Repeat for each input point $(x_1^t, x_2^t, y_t)$ in the dataset:
    (a) Compute the current prediction:
        $f_l(x_1^t, x_2^t) = \begin{cases} +1, & \text{if } |x_1^t| \leq l \text{ and } |x_2^t| \leq l \\ -1, & \text{otherwise} \end{cases}$
    (b) If $y_t \cdot f_l(x_1^t, x_2^t) \leq 0$ (i.e., a mistake is made):
        • If $y_t = +1$: Update $l \leftarrow \max(l, |x_1^t|, |x_2^t|)$
        • If $y_t = -1$: Update $l \leftarrow \min(l, \max(|x_1^t|, |x_2^t|) - \epsilon)$ where $\epsilon$ is a small positive value.
3. End Repeat
```

Step 3: Mistake bound analysis

1. **Positive example updates:**
   - The value of l can only increase when a mistake occurs on a positive example.
   - The largest possible value for l is 80, since the instance space is limited to $|x_1|, |x_2| \leq 80$.
   - Therefore, there can be at most **80 positive mistakes**.
2. **Negative example updates:**
   - Once l reaches a value large enough to correctly classify all positive examples, further mistakes can only occur if the algorithm incorrectly predicts $+1$ on a negative example.
   - Each time a mistake occurs on a negative example, l decreases, but it can only decrease a finite number of times before all negative examples are correctly classified.

Thus, the total number of mistakes is bounded by **at most 160 mistakes** (80 positive mistakes and 80 potential negative mistakes).

Conclusion

The mistake-driven learning algorithm is:

- **Initialize:** $l \leftarrow 1$
- **For each input:** Update l if a mistake is made.
- The algorithm can make at most **160 mistakes** on any dataset.

3 Perceptron Mistake Bound with Corrupted Examples

Solution for Step 1(a)

Step 1: Express the updated weight vector projection on u

We want to calculate $u^\top w_{k+1}$ after the update.

1. Start with the update rule:
$$w_{k+1} = w_k + y_i(x_i + r_i)$$
2. Take the dot product with u:
$$u^\top w_{k+1} = u^\top w_k + y_i \, u^\top (x_i + r_i)$$
3. Expand the second term:
$$u^\top w_{k+1} = u^\top w_k + y_i \left(u^\top x_i + u^\top r_i\right)$$

Step 2: Apply the margin condition

We are given that:

$$y_i \, u^\top x_i \geq \gamma$$

Therefore, we substitute this into the equation:

$$u^\top w_{k+1} \geq u^\top w_k + \gamma + y_i \, u^\top r_i$$

Step 3: Bound the noise term

Since r_i is a random unit vector, the value of $u^\top r_i$ is between -1 and $+1$. Therefore, we have:

$$-1 \leq u^\top r_i \leq 1$$

Thus, the inequality becomes:

$$u^\top w_{k+1} \geq u^\top w_k + \gamma - 1$$

Conclusion

The inequality that relates $u^\top w_{k+1}$ to $u^\top w_k$ is:

$$u^\top w_{k+1} \geq u^\top w_k + \gamma + y_i \, u^\top r_i$$

Alternatively, using the worst-case noise bound:

$$u^\top w_{k+1} \geq u^\top w_k + \gamma - 1$$

Solution for Question 3, Step 1(b)

From Step 1(a), we have the inequality:

$$u^\top w_{k+1} \geq u^\top w_k + \gamma + y_i \, u^\top r_i$$

Step 1: Recursively expand $u^\top w_{k+1}$

We apply the recurrence iteratively to expand $u^\top w_{k+1}$:

1. After one iteration:

$$u^\top w_1 \geq u^\top w_0 + \gamma + y_1 \, u^\top r_1$$

2. After two iterations:

$$u^\top w_2 \geq u^\top w_1 + \gamma + y_2 \, u^\top r_2$$

Expanding $u^\top w_1$ from the previous step:

$$u^\top w_2 \geq u^\top w_0 + 2\gamma + y_1 \, u^\top r_1 + y_2 \, u^\top r_2$$

3. Continuing for k iterations:

$$u^\top w_{k+1} \geq u^\top w_0 + (k+1)\gamma + \sum_{i=1}^{k+1} y_i \, u^\top r_i$$

Step 2: Initial condition

The initial weight vector is $w_0 = 0$, so:

$$u^\top w_0 = 0$$

Thus, the inequality simplifies to:

$$u^\top w_{k+1} \geq (k+1)\gamma + \sum_{i=1}^{k+1} y_i \, u^\top r_i$$

Step 3: Bounding the noise term

The noise term $y_i \, u^\top r_i$ is bounded between -1 and $+1$. Therefore, the worst-case bound on the noise term is:

$$\sum_{i=1}^{k+1} y_i \, u^\top r_i \geq -(k+1)$$

Step 4: Final inequality

Using the worst-case bound on the noise term:

$$u^\top w_{k+1} \geq (k+1)(\gamma - 1)$$

Conclusion

The lower bound for $u^\top w_{k+1}$ in terms of k and γ is:

$$u^\top w_{k+1} \geq (k+1)(\gamma - 1)$$

Solution for Question 3, Step 2(a)

The algorithm predicts the label using the sign of:

$$f(w_k, x_i + r_i) = \text{sign}(w_k^\top (x_i + r_i))$$

A mistake occurs when the predicted label does not match the true label y_i. Therefore, we need to write an inequality that captures this mistake condition.

Step 1: Define the mistake condition

1. The Perceptron predicts the label of the example as:

$$\hat{y}_i = \text{sign}(w_k^\top (x_i + r_i))$$

2. A mistake occurs when $\hat{y}_i \neq y_i$. This is equivalent to:

$$y_i \cdot w_k^\top (x_i + r_i) \leq 0$$

Step 2: Interpretation of the inequality

- If $y_i \cdot w_k^\top(x_i + r_i) > 0$, the prediction is correct.
- If $y_i \cdot w_k^\top(x_i + r_i) \leq 0$, a mistake is made because the sign of the prediction is opposite to the true label.

Conclusion

The condition under which the Perceptron makes a mistake is given by the inequality:

$$y_i \cdot w_k^\top(x_i + r_i) \leq 0$$

Solution for Question 3, Step 2(b)

Step 1: Expand $\|w_{k+1}\|^2$

We start by expanding the norm squared of the updated weight vector:

$$\|w_{k+1}\|^2 = (w_k + y_i(x_i + r_i))^\top (w_k + y_i(x_i + r_i))$$

Expand the dot product:

$$\|w_{k+1}\|^2 = w_k^\top w_k + 2y_i w_k^\top (x_i + r_i) + y_i^2 \|x_i + r_i\|^2$$

Step 2: Simplify the expression

1. The term $w_k^\top w_k$ is just $\|w_k\|^2$.
2. The term y_i^2 is equal to 1, since $y_i \in \{-1, +1\}$. Thus, the last term simplifies to:
$$y_i^2 \|x_i + r_i\|^2 = \|x_i + r_i\|^2$$

Therefore, the expression becomes:

$$\|w_{k+1}\|^2 = \|w_k\|^2 + 2y_i w_k^\top (x_i + r_i) + \|x_i + r_i\|^2$$

Step 3: Interpretation

The inequality depends on the value of $w_k^\top(x_i + r_i)$, which affects the sign and size of the second term. However, for an upper bound, we consider the worst-case scenario where this term does not reduce the norm significantly. Hence, the relationship between $\|w_{k+1}\|^2$ and $\|w_k\|^2$ can be expressed as:

$$\|w_{k+1}\|^2 \leq \|w_k\|^2 + 2y_i w_k^\top (x_i + r_i) + R^2$$

Here, R^2 is an upper bound for $\|x_i + r_i\|^2$, where $\|x_i\| \leq R$ and $\|r_i\| = 1$.

Conclusion

The inequality that relates $\|w_{k+1}\|^2$ to $\|w_k\|^2$ is:

$$\|w_{k+1}\|^2 = \|w_k\|^2 + 2y_i w_k^\top (x_i + r_i) + \|x_i + r_i\|^2$$

Alternatively, in the upper-bound form:

$$\|w_{k+1}\|^2 \leq \|w_k\|^2 + R^2$$

Solution for Question 3, Step 2(c)

Step 1: Recursion for $\|w_{k+1}\|^2$

From Step 2(b), we have the expression:

$$\|w_{k+1}\|^2 = \|w_k\|^2 + 2y_i w_k^\top (x_i + r_i) + \|x_i + r_i\|^2$$

To find an upper bound, we assume that $y_i w_k^\top (x_i + r_i)$ does not significantly decrease the norm and focus on bounding $\|x_i + r_i\|^2$.

Step 2: Bound $\|x_i + r_i\|^2$

Since $\|x_i\| \leq R$ and $\|r_i\| = 1$, we apply the triangle inequality:

$$\|x_i + r_i\| \leq \|x_i\| + \|r_i\| \leq R + 1$$

Thus, $\|x_i + r_i\|^2 \leq (R + 1)^2$.

Step 3: Recursively expand $\|w_{k+1}\|^2$

We apply the recurrence iteratively:

1. After one iteration:

$$\|w_1\|^2 = \|w_0\|^2 + \|x_1 + r_1\|^2$$

2. After two iterations:

$$\|w_2\|^2 = \|w_1\|^2 + \|x_2 + r_2\|^2$$

Substituting $\|w_1\|^2$:

$$\|w_2\|^2 = \|w_0\|^2 + \|x_1 + r_1\|^2 + \|x_2 + r_2\|^2$$

3. Continuing for k iterations:

$$\|w_{k+1}\|^2 = \|w_0\|^2 + \sum_{i=1}^{k+1} \|x_i + r_i\|^2$$

Step 4: Apply the bound

Using $\|x_i + r_i\|^2 \leq (R+1)^2$, we get:

$$\|w_{k+1}\|^2 \leq \|w_0\|^2 + (k+1)(R+1)^2$$

Since $w_0 = 0$, this simplifies to:

$$\|w_{k+1}\|^2 \leq (k+1)(R+1)^2$$

Conclusion

The upper bound for $\|w_{k+1}\|^2$ is:

$$\|w_{k+1}\|^2 \leq (k+1)(R+1)^2$$

Solution for Question 3, 3

- From **Step 1(b)**:
$$u^\top w_{k+1} \geq (k+1)(\gamma - 1)$$
- From **Step 2(c)**:
$$\|w_{k+1}\|^2 \leq (k+1)(R+1)^2$$

Step 1: Apply the Cauchy-Schwarz inequality

The Cauchy-Schwarz inequality gives:

$$|u^\top w_{k+1}| \leq \|u\| \, \|w_{k+1}\|$$

Since u is a unit vector, $\|u\| = 1$. Therefore:

$$u^\top w_{k+1} \leq \|w_{k+1}\|$$

Step 2: Combine the inequalities

We have:

$$u^\top w_{k+1} \geq (k+1)(\gamma - 1)$$

and:

$$\|w_{k+1}\|^2 \leq (k+1)(R+1)^2$$

Taking the square root of both sides of the second inequality:

$$\|w_{k+1}\| \leq \sqrt{(k+1)(R+1)^2} = (k+1)^{1/2}(R+1)$$

Now, combine the two inequalities:

$$(k+1)(\gamma - 1) \leq (k+1)^{1/2}(R+1)$$

Step 3: Solve for k

1. Divide both sides by $(k+1)^{1/2}$:

$$(k+1)^{1/2}(\gamma-1) \le R+1$$

2. Square both sides:

$$(k+1)(\gamma-1)^2 \le (R+1)^2$$

3. Solve for k:

$$k+1 \le \frac{(R+1)^2}{(\gamma-1)^2}$$

Thus, the bound on the number of mistakes is:

$$k \le \frac{(R+1)^2}{(\gamma-1)^2} - 1$$

Conclusion

The mistake bound for the Perceptron algorithm is:

$$k \le \frac{(R+1)^2}{(\gamma-1)^2} - 1$$

Question4. The Perceptron Algorithm and its Variants

1.

Results

From the given output:

- **Training accuracy**: 50.3%
- **Test accuracy**: 50.2%

This suggests that the dataset is **slightly imbalanced**, with the most frequent class making up approximately 50% of the labels.

Interpretation

- Since the majority class appears around 50% of the time, the baseline classifier achieves an accuracy of approximately 50% by always predicting it.
- Any real model must **outperform this baseline** to be considered effective.

- A perceptron or other machine learning model should aim to significantly exceed 50.2% accuracy, otherwise, it is not learning meaningful patterns beyond guessing the majority class.

Question 2.a

Programming Language

The models are implemented in **Python**, leveraging its flexibility and extensive libraries for numerical computation.

Data Representation

- Feature vectors (x) and labels (y) are represented using **NumPy arrays** (`np.ndarray`), ensuring efficient matrix operations.
- Weights (w) and biases (b) are also stored as **NumPy arrays**, allowing optimized computations.

Initialization Strategy

- Weights (w) are initialized randomly between $[-0.01, 0.01]$ to break symmetry:

```
    self.w = np.random.uniform(-0.01, 0.01, num_features)
```

- Bias (b) is initialized similarly:

```
    self.b = np.random.uniform(-0.01, 0.01)
```

- For the **Aggressive Perceptron**, weights are initialized to zero:

```
    self.w = np.zeros(num_features)
    self.b = 0
```

Update Mechanism

0.1 Simple Perceptron

```
self.w += self.lr * y[i] * x[i]
self.b += self.lr * y[i]
```

0.2 Decay Perceptron

```
learning_rate = self.lr / (1 + self.t)
self.t += 0.001
```

0.3 Margin Perceptron

```
if y[i] * (np.dot(self.w, x[i]) + self.b) < self.mu:
    self.w += learning_rate * y[i] * x[i]
    self.b += learning_rate * y[i]
```

0.4 Averaged Perceptron

```
self.avg_weights += self.w
self.avg_bias += self.b
self.count += 1
```

0.5 Aggressive Perceptron

```
eta = (self.mu - margin) / (np.dot(x[i], x[i]) + 1)
self.w += eta * y[i] * x[i]
self.b += eta * y[i]
```

Data Processing

- **Shuffling:** The dataset is shuffled between epochs to prevent order bias:

```
    x, y = shuffle_data(x, y)
```

- **Batch Processing:** The perceptron updates weights after every individual sample.

Prediction Strategy

- The models use the `sign` function to classify data:

```
    return np.sign(np.dot(x, self.w) + self.b)
```

- The **Averaged Perceptron** uses the **average weights** for final predictions:

```
    final_weights = self.avg_weights / self.count
    final_bias = self.avg_bias / self.count
    return np.sign(np.dot(x, final_weights) + final_bias)
```

Question 2.b

Best Hyperparameters

The optimal hyperparameters for each Perceptron variant, determined via cross-validation, are summarized in Table 1.

Perceptron Variant	**Learning Rate (lr)**	**Margin (mu)**
Simple Perceptron	0.1	0.0
Decay Perceptron	0.01	0.0
Margin Perceptron	0.01	10.0
Averaged Perceptron	0.01	0.0
Aggressive Perceptron	0.0	0.1

Table 1: Best hyperparameters from cross-validation.

Question 2.c

Cross-Validation Accuracy

The cross-validation accuracy corresponding to the best hyperparameters for each variant is presented in Table 2.

Perceptron Variant	**Accuracy**
Simple Perceptron	0.782
Decay Perceptron	0.777
Margin Perceptron	0.865
Averaged Perceptron	0.819
Aggressive Perceptron	0.797

Table 2: Cross-validation accuracy for the best hyperparameters.

Among all variants, the Margin Perceptron with $\mu = 10.0$ and $lr = 0.01$ achieved the highest accuracy of 0.865.

Question 2.d

Total Number of learning rate Updates

The total number of updates the learning algorithm performed on the training set for each variant is presented in Table 3.

Perceptron Variant	**Total Updates**
Simple Perceptron	8992
Decay Perceptron	6303
Margin Perceptron	38387
Averaged Perceptron	9100
Aggressive Perceptron	32476

Table 3: Total number of updates performed during training.

Question 2.e

Development Set Accuracy

The accuracy on the development set after training for 20 epochs for each Perceptron variant is shown in Table 4.

Perceptron Variant	Development Set Accuracy
Simple Perceptron	0.965
Decay Perceptron	0.968
Margin Perceptron	0.885
Averaged Perceptron	0.989
Aggressive Perceptron	0.998

Table 4: Development set accuracy for each Perceptron variant.

Question 2.f

Test Set Accuracy

The accuracy on the test set after training for 20 epochs for each Perceptron variant is shown in Table 5.

Perceptron Variant	Test Set Accuracy
Simple Perceptron	0.793
Decay Perceptron	0.823
Margin Perceptron	0.888
Averaged Perceptron	0.862
Aggressive Perceptron	0.827

Table 5: Test set accuracy for each Perceptron variant.

Question 2.g

The learning curves for each Perceptron variant, showing the development set accuracy across 20 epochs, are presented below:

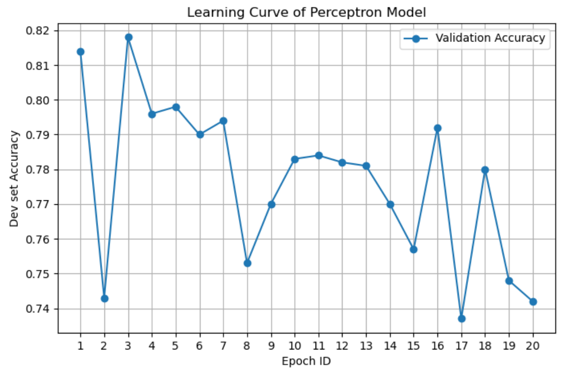

Figure 1: Learning Curve for Simple Perceptron

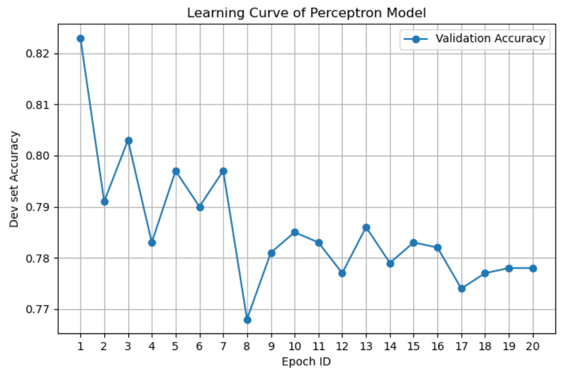

Figure 2: Learning Curve for Decay Perceptron

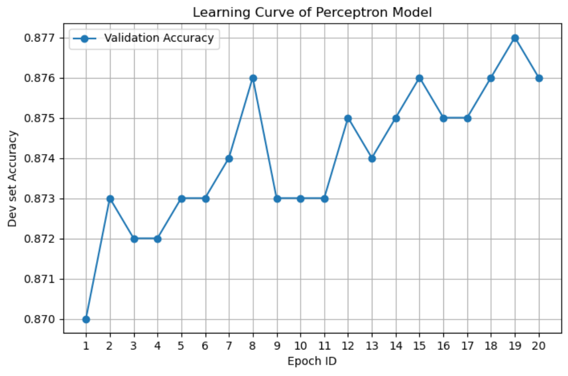

Figure 3: Learning Curve for Margin Perceptron

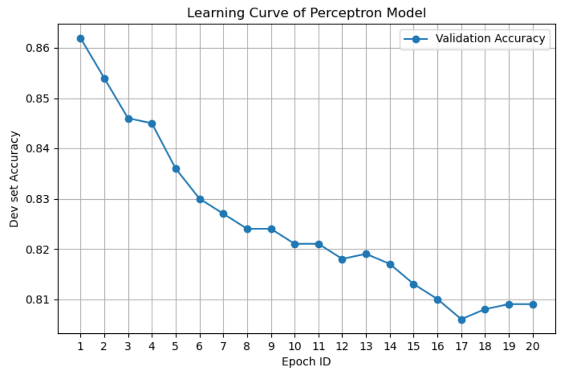

Figure 4: Learning Curve for Averaged Perceptron

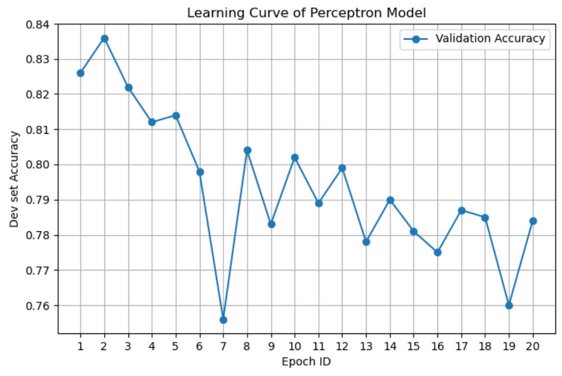

Figure 5: Learning Curve for Aggressive Perceptron