Solution 1

Step 1

The prediction rule is given by $(2x_1 - x_2 - 6)$. In a linear classifier, the decision boundary is where the prediction rule equals zero:

$$2x_1 - x_2 - 6 = 0$$

We can rearrange this to express x_2 in terms of x_1:

$$x_2 = 2x_1 - 6$$

This is the equation of our decision boundary in $\mathbb{R}^2$.

Step 2

To find where the decision boundary intersects the axes:
x-axis intersection (where $x_2 = 0$):

$$0 = 2x_1 - 6$$

$$2x_1 = 6$$

$$x_1 = 3$$

So the boundary intersects the x-axis at the point $(3, 0)$.
y-axis intersection (where $x_1 = 0$):

$$x_2 = 2(0) - 6$$

$$x_2 = -6$$

So the boundary intersects the y-axis at the point $(0, -6)$.

Step 3

To determine which side of the boundary is classified as positive and which as negative, we need to examine the sign of the prediction rule $f(x) = 2x_1 - x_2 - 6$.
For a point (x_1, x_2):

- If $2x_1 - x_2 - 6 > 0$, the point is classified as positive.
- If $2x_1 - x_2 - 6 < 0$, the point is classified as negative.

Let's test a point clearly on one side of the line, such as the origin $(0, 0)$:

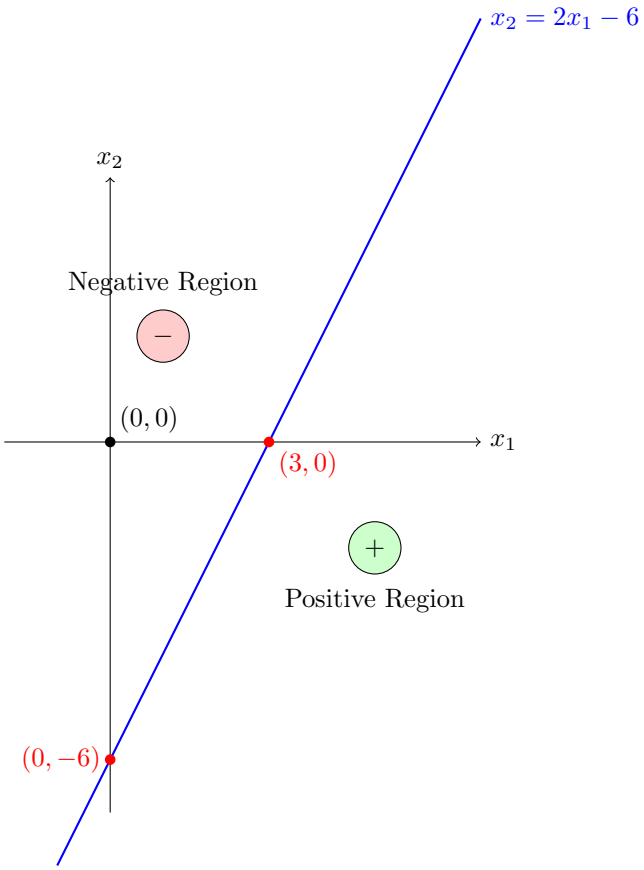
$$f(0, 0) = 2(0) - 0 - 6 = -6 < 0$$

So the origin is classified as negative. This means:

- The region below the line $x_2 = 2x_1 - 6$ is classified as positive.
- The region above the line $x_2 = 2x_1 - 6$ is classified as negative.

Step 4

Let's visualize the decision boundary:



$\therefore$ The decision boundary is the line $x_2 = 2x_1 - 6$, which intersects the x-axis at $(3, 0)$ and the y-axis at $(0, -6)$. The region below this line is classified as positive, and the region above it is classified as negative.

Solution 2 (a)

Step 1

The question states that the Perceptron algorithm converges after making k updates. We need to determine if this means the data set is definitely linearly separable.

Step 2

Recall the Perceptron Convergence Theorem: If a dataset is linearly separable, the Perceptron algorithm is guaranteed to converge in a finite number of updates.
However, the converse is not necessarily true. If the Perceptron algorithm converges, it only means that it found a hyperplane that correctly classifies the training data it has seen, but this doesn't guarantee that the entire dataset is linearly separable.
In this case, since we're told that the algorithm converges after cycling through all points repeatedly until convergence, this implies that the algorithm has seen all data points and found a hyperplane that correctly classifies all of them.

$\therefore$ The statement "The data set is linearly separable" is **definitely true**. The Perceptron algorithm converges if and only if the data is linearly separable. Since we're told it converges after cycling through all points repeatedly, this means a separating hyperplane exists for the entire dataset.

Solution 2 (b)

Step 1

The question asks whether the Perceptron algorithm would again converge if run with a different random permutation of the same dataset.

Step 2

From part (a), we established that the dataset is linearly separable. The Perceptron Convergence Theorem guarantees that for any linearly separable dataset, the Perceptron algorithm will converge in a finite number of updates, regardless of the order in which the data points are presented.
The order of presentation (permutation) may affect the specific hyperplane found and the number of updates required, but it does not affect whether the algorithm converges.

$\therefore$ The statement "If the process were repeated with a different random permutation, it would again converge" is **definitely true**. Since the dataset is linearly separable, the Perceptron algorithm will converge regardless of the order in which the data points are presented.

Solution 2 (c)

Step 1

The question asks whether the Perceptron algorithm would converge after exactly k updates if run with a different random permutation of the same dataset.

Step 2

The number of updates required for the Perceptron algorithm to converge depends on several factors:

- The specific data points and their labels
- The order in which the data points are presented
- The initial weight vector
- The margin of separation in the data

Different permutations of the data can lead to different update sequences and potentially different numbers of updates before convergence. While the algorithm will still converge (as established in part (b)), there is no guarantee that it will make exactly k updates with a different permutation.

$\therefore$ The statement "If the process were repeated with a different random permutation, it would again converge after making k updates" is **possibly false**. The number of updates required for convergence can vary depending on the specific permutation of the data.

Solution 2 (d)

Step 1

The question asks whether k (the number of updates until convergence) is at most n (the number of data points).

Step 2

In the Perceptron algorithm, an update occurs only when a data point is misclassified. In the worst case, every data point could be misclassified in each complete cycle through the dataset.
The algorithm cycles through all n points repeatedly until convergence. If it takes multiple cycles to converge, then the total number of updates k could be greater than n.
For example, if it takes 3 complete cycles through the dataset to converge, and several points are misclassified in each cycle, then k could be much larger than n.
The Perceptron Convergence Theorem provides an upper bound on the number of updates, which depends on the margin of separation and the maximum norm of the data points, but this bound is not simply n.

$\therefore$ The statement "k is at most n" is **possibly false**. The number of updates k can exceed the number of data points n if multiple passes through the dataset are required for convergence.

Solution 3

Step 1

Let's recall the Perceptron update rule. When a point (x_i, y_i) is misclassified:

$$w \leftarrow w + y_i x_i \tag{1}$$
$$b \leftarrow b + y_i \tag{2}$$

where w is the weight vector, b is the bias term, x_i is the feature vector, and y_i is the label ($+1$ or -1).

Step 2

We're told that the Perceptron algorithm performs $p + q$ updates in total:

- p updates on data points with label $y_i = -1$
- q updates on data points with label $y_i = +1$

Let's assume the bias term b is initialized to 0, which is a standard initialization for the Perceptron algorithm.

Step 3

For each update on a point with label $y_i = -1$, the bias is updated as:

$$b \leftarrow b + y_i \tag{3}$$
$$\leftarrow b + (-1) \tag{4}$$
$$\leftarrow b - 1 \tag{5}$$

So after p such updates, the contribution to b is $-p$.
For each update on a point with label $y_i = +1$, the bias is updated as:

$$b \leftarrow b + y_i \tag{6}$$
$$\leftarrow b + (+1) \tag{7}$$
$$\leftarrow b + 1 \tag{8}$$

So after q such updates, the contribution to b is $+q$.

Step 4

Combining the effects of all updates, and starting from $b = 0$:

$$b_{\text{final}} = 0 + (-p) + q \tag{9}$$
$$= q - p \tag{10}$$

$\therefore$ The final value of the parameter b is $q - p$.

Solution 4 (a)

Step 1

Given information:

- SVM classifier in $\mathbb{R}^2$
- Weight vector $w = (3, 4)$
- Bias term $b = -12$

The decision boundary of an SVM is defined by the equation $w \cdot x + b = 0$, where x is a point in the feature space.

Step 2

Substituting the given values:

$$w \cdot x + b = 0 \tag{11}$$
$$(3, 4) \cdot (x_1, x_2) + (-12) = 0 \tag{12}$$
$$3x_1 + 4x_2 - 12 = 0 \tag{13}$$
$$3x_1 + 4x_2 = 12 \tag{14}$$

We can rearrange to express x_2 in terms of x_1:

$$4x_2 = 12 - 3x_1 \tag{15}$$
$$x_2 = 3 - \frac{3}{4}x_1 \tag{16}$$

Step 3

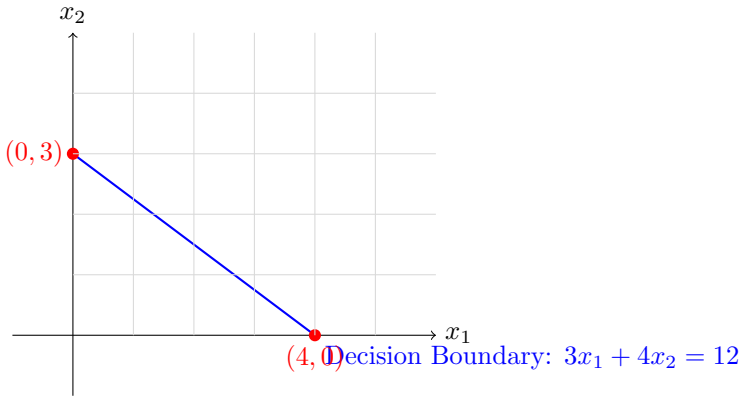
To find where the decision boundary intersects the axes:
x-axis intersection (where $x_2 = 0$):

$$0 = 3 - \frac{3}{4}x_1 \tag{17}$$
$$\frac{3}{4}x_1 = 3 \tag{18}$$
$$x_1 = 4 \tag{19}$$

So the boundary intersects the x-axis at the point $(4, 0)$.
y-axis intersection (where $x_1 = 0$):

$$x_2 = 3 - \frac{3}{4} \cdot 0 \tag{20}$$
$$x_2 = 3 \tag{21}$$

So the boundary intersects the y-axis at the point $(0, 3)$.



$\therefore$ The decision boundary is the line $3x_1 + 4x_2 = 12$ or equivalently $x_2 = 3 - \frac{3}{4}x_1$. It intersects the x-axis at $(4, 0)$ and the y-axis at $(0, 3)$.

Solution 4 (b)

Step 1

For an SVM, the left-hand and right-hand boundaries (also called the margin boundaries) are parallel to the decision boundary and are at a distance of $\frac{1}{||w||}$ from it.
The distance from a point (x_1, x_2) to the decision boundary $w \cdot x + b = 0$ is given by:

$$d = \frac{|w \cdot x + b|}{||w||} \tag{22}$$

The margin boundaries are defined by the equations:

$$w \cdot x + b = 1 \quad \text{(positive margin boundary)} \tag{23}$$
$$w \cdot x + b = -1 \quad \text{(negative margin boundary)} \tag{24}$$

Step 2

For our SVM with $w = (3, 4)$ and $b = -12$:
Positive margin boundary:

$$w \cdot x + b = 1 \tag{25}$$
$$3x_1 + 4x_2 - 12 = 1 \tag{26}$$
$$3x_1 + 4x_2 = 13 \tag{27}$$
$$x_2 = \frac{13 - 3x_1}{4} = 3.25 - \frac{3}{4}x_1 \tag{28}$$

Negative margin boundary:

$$w \cdot x + b = -1 \tag{29}$$
$$3x_1 + 4x_2 - 12 = -1 \tag{30}$$
$$3x_1 + 4x_2 = 11 \tag{31}$$
$$x_2 = \frac{11 - 3x_1}{4} = 2.75 - \frac{3}{4}x_1 \tag{32}$$

Step 3

Let's find where these margin boundaries intersect the axes:
Positive margin boundary:
x-axis intersection ($x_2 = 0$):

$$0 = 3.25 - \frac{3}{4}x_1 \tag{33}$$

$$\frac{3}{4}x_1 = 3.25 \tag{34}$$

$$x_1 = \frac{13}{3} \approx 4.33 \tag{35}$$

y-axis intersection ($x_1 = 0$):

$$x_2 = 3.25 - \frac{3}{4} \cdot 0 = 3.25 \tag{36}$$

Negative margin boundary:
x-axis intersection ($x_2 = 0$):

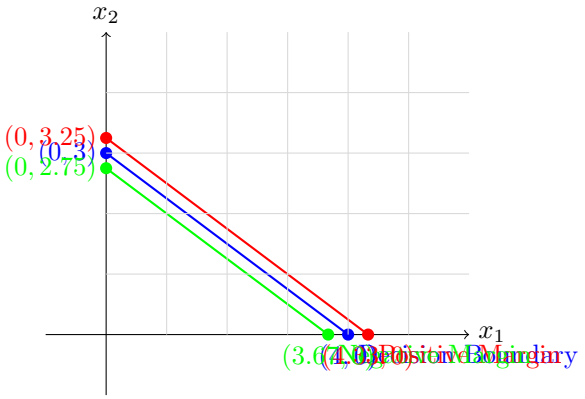
$$0 = 2.75 - \frac{3}{4}x_1 \tag{37}$$

$$\frac{3}{4}x_1 = 2.75 \tag{38}$$

$$x_1 = \frac{11}{3} \approx 3.67 \tag{39}$$

y-axis intersection ($x_1 = 0$):

$$x_2 = 2.75 - \frac{3}{4} \cdot 0 = 2.75 \tag{40}$$



∴ The positive margin boundary is the line $3x_1 + 4x_2 = 13$ or $x_2 = 3.25 - \frac{3}{4}x_1$, intersecting the x-axis at $(4.33, 0)$ and the y-axis at $(0, 3.25)$. The negative margin boundary is the line $3x_1 + 4x_2 = 11$ or $x_2 = 2.75 - \frac{3}{4}x_1$, intersecting the x-axis at $(3.67, 0)$ and the y-axis at $(0, 2.75)$.

Solution 4 (c)

Step 1

The margin of an SVM classifier is defined as the perpendicular distance between the two margin boundaries, which is equal to $\frac{2}{||w||}$.
We need to calculate $||w||$, the Euclidean norm of the weight vector:

$$||w|| = \sqrt{w_1^2 + w_2^2} \tag{41}$$
$$= \sqrt{3^2 + 4^2} \tag{42}$$
$$= \sqrt{9 + 16} \tag{43}$$
$$= \sqrt{25} \tag{44}$$
$$= 5 \tag{45}$$

Step 2

Now we can calculate the margin:

$$\text{margin} = \frac{2}{||w||} \tag{46}$$
$$= \frac{2}{5} \tag{47}$$
$$= 0.4 \tag{48}$$

$\therefore$ The margin of this SVM classifier is $\frac{2}{5} = 0.4$ units.

Solution 4 (d)

Step 1

To classify a point (x_1, x_2) using an SVM, we compute the sign of $w \cdot x + b$:

- If $w \cdot x + b > 0$, the point is classified as positive.
- If $w \cdot x + b < 0$, the point is classified as negative.
- If $w \cdot x + b = 0$, the point lies exactly on the decision boundary.

Step 2

For the point $(2, 2)$ with $w = (3, 4)$ and $b = -12$:

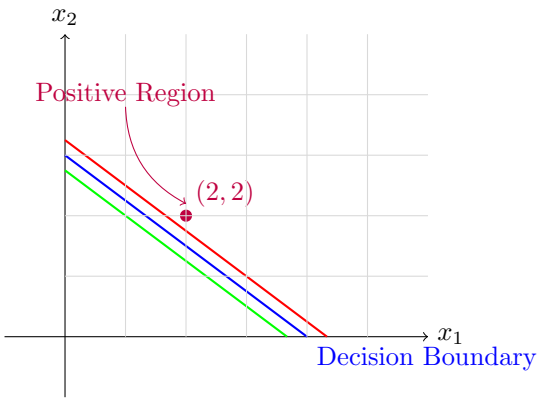
$$\begin{aligned} w \cdot x + b &= 3 \cdot 2 + 4 \cdot 2 + (-12) && (49)\\ &= 6 + 8 - 12 && (50)\\ &= 14 - 12 && (51)\\ &= 2 > 0 && (52) \end{aligned}$$

Step 3

We can also verify this geometrically. The decision boundary is the line $3x_1 + 4x_2 = 12$. For the point $(2, 2)$:

$$3 \cdot 2 + 4 \cdot 2 = 6 + 8 = 14 > 12 \tag{53}$$

This confirms that the point lies on the positive side of the decision boundary.



$\therefore$ The point $(2, 2)$ would be classified as **positive** because $w \cdot x + b = 2 > 0$.

Solution 5 (a)

Step 1

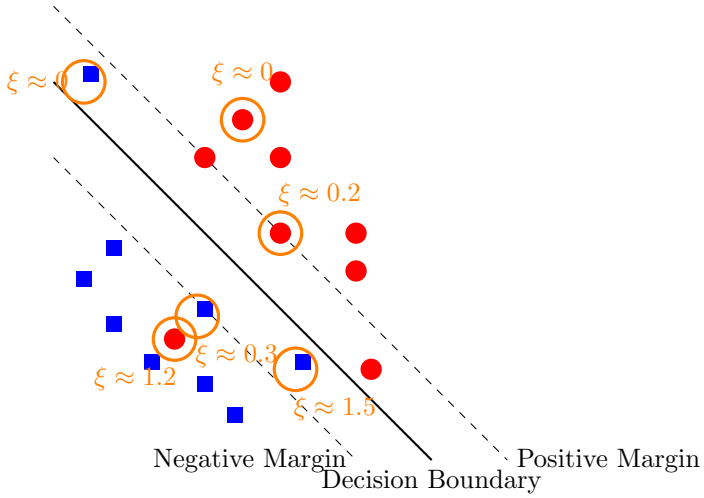
In a soft-margin SVM, the support vectors are the data points that:

- Lie exactly on the margin boundaries (slack variable $\xi_i = 0$)
- Lie between the margin boundary and the decision boundary (slack variable $0 < \xi_i < 1$)
- Lie on the wrong side of the margin boundary (slack variable $\xi_i \geq 1$)

The slack variable ξ_i represents the degree of misclassification for each point. It measures how far a point is from its correct margin boundary.

Step 2

Let's identify the support vectors in the given figure:



Step 3

Let's explain the identified support vectors:

1. **Support vectors with $\xi \approx 0$:**
   - The blue square at approximately $(0.4, 5.0)$ lies very close to the negative margin boundary.
   - The red circle at approximately $(2.5, 4.5)$ lies very close to the positive margin boundary.
   - These points have slack variables close to zero because they are almost exactly on their respective margin boundaries.
2. **Support vectors with $0 < \xi < 1$:**
   - The blue square at approximately $(1.9, 1.9)$ is between the negative margin and the decision boundary.
   - The red circle at approximately $(3, 3)$ is between the positive margin and the decision boundary.
   - These points have slack variables between 0 and 1 because they are inside the margin but on the correct side of the decision boundary.
3. **Support vectors with $\xi \geq 1$:**
   - The red circle at approximately $(1.6, 1.6)$ is on the wrong side of the decision boundary.
   - The blue square at approximately $(3.2, 1.2)$ is also on the wrong side of the decision boundary.
   - These points have slack variables greater than or equal to 1 because they are misclassified by the decision boundary.

$\therefore$ The support vectors are the points circled in orange in the figure above, with their approximate slack variable values indicated. Support vectors include points exactly on the margin boundaries ($\xi \approx 0$), points between the margin and decision boundary ($0 < \xi < 1$), and misclassified points ($\xi \geq 1$).

Solution 5 (b)

Step 1

Recall the optimization problem for soft-margin SVM:

$$\min_{w,b,\xi} \frac{1}{2}||w||^2 + C\sum_{i=1}^{n} \xi_i$$

subject to:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad \forall i$$

The parameter C controls the trade-off between maximizing the margin (minimizing $||w||^2$) and minimizing the classification error (minimizing $\sum \xi_i$).

Step 2

When C is increased:

- The penalty for misclassification and margin violations becomes higher.
- The optimization will prioritize reducing the slack variables ξ_i over maximizing the margin.
- This means the model will try harder to correctly classify all training points, potentially at the expense of a smaller margin.

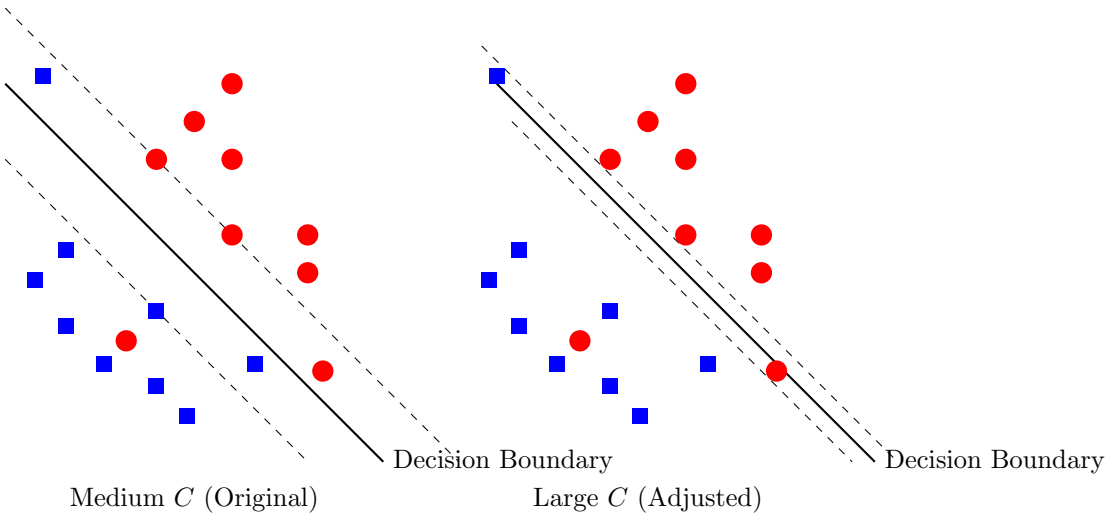
Step 3

Let's compare the effects of different values of C:

Small C	**Medium** C	**Large** C
Larger margin	Balanced trade-off	Smaller margin
More misclassifications	Some misclassifications	Fewer misclassifications
Underfitting risk	Good generalization	Overfitting risk

In our specific example, increasing C would likely:

- Make the decision boundary adjust to reduce the misclassifications (the red circle at $(1.6, 1.6)$ and the blue square at $(3.2, 1.2)$).
- Potentially reduce the margin between the dashed lines to better accommodate the training points.



$\therefore$ If the factor C in the soft-margin SVM optimization problem were increased, we would expect the margin to **decrease**. This is because a larger C places more emphasis on correctly classifying all training points, even if it means having a smaller margin.

Solution 6 (a)

Step 1

Let's recall the dual form of the Perceptron algorithm:
In the dual form, we represent the weight vector w as a linear combination of the training examples:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

where α_i counts how many times example i has been misclassified during training.

Step 2

In the standard Perceptron algorithm, when a point (x_i, y_i) is misclassified, we update:

$$w \leftarrow w + y_i x_i$$

In the dual form, this corresponds to incrementing α_i by 1:

$$\alpha_i \leftarrow \alpha_i + 1$$

Initially, all α_i values are set to 0. Each time a point is misclassified, its corresponding α_i is incremented by 1.

Step 3

Since the algorithm performs k updates in total, and each update increments exactly one α_i by 1, we know that:

- Each α_i represents the number of times example i was misclassified
- Each α_i must be a non-negative integer
- Some examples might never be misclassified, so their α_i remains 0
- Some examples might be misclassified multiple times, so their α_i could be greater than 1

$\therefore$ The statement "Each α_i is either 0 or 1" is **possibly false**. While some α_i values may be 0 (never misclassified) or 1 (misclassified once), it's possible for an example to be misclassified multiple times during training, resulting in $\alpha_i > 1$.

Solution 6 (b)

Step 1

We know that the Perceptron algorithm makes a total of k updates, and each update corresponds to incrementing exactly one α_i by 1.

Step 2

Initially, all α_i values are set to 0. After k updates, each incrementing one α_i by 1, the sum of all α_i values must be equal to k:

$$\sum_{i=1}^{n} \alpha_i = k$$

This is because each update contributes exactly 1 to the sum of α_i values, and there are k updates in total.

$\therefore$ The statement "$\sum_i \alpha_i = k$" is **necessarily true**. Since each of the k updates increments exactly one α_i by 1, and all α_i values start at 0, the sum of all α_i values must equal k.

Solution 6 (c)

Step 1

We need to determine the maximum number of nonzero coordinates in the vector α.

Step 2

A coordinate α_i is nonzero if and only if the corresponding example (x_i, y_i) has been misclassified at least once during training.
In the worst case, each of the k updates could be applied to a different example, resulting in k different examples having their α_i incremented to 1.
However, it's also possible that some examples are misclassified multiple times, which would result in fewer than k nonzero coordinates in α.

Step 3

Let's consider a simple example to illustrate:
Suppose we have 5 training examples and the algorithm makes $k = 3$ updates:

- If the updates are applied to examples 1, 2, and 3, then $\alpha = (1, 1, 1, 0, 0)$ with 3 nonzero coordinates.
- If the updates are applied to examples 1, 1, and 2, then $\alpha = (2, 1, 0, 0, 0)$ with 2 nonzero coordinates.

In general, the number of nonzero coordinates in α is at most k (when each update is applied to a different example) and at least 1 (when all updates are applied to the same example).

$\therefore$ The statement "α has at most k nonzero coordinates" is **necessarily true**. Since there are k updates in total, at most k different examples can be misclassified, resulting in at most k nonzero α_i values.

Solution 6 (d)

Step 1

We need to determine whether the convergence of the Perceptron algorithm implies that the training data is linearly separable.

Step 2

Recall the Perceptron Convergence Theorem: If the training data is linearly separable, then the Perceptron algorithm will converge in a finite number of updates.
The converse is also true: If the Perceptron algorithm converges, then the training data must be linearly separable. This is because:

- Convergence means the algorithm has found a weight vector w and bias b such that all training examples are correctly classified.
- This means there exists a hyperplane defined by $w \cdot x + b = 0$ that separates the positive and negative examples.
- By definition, this makes the data linearly separable.

Step 3

It's important to note that if the data is not linearly separable, the Perceptron algorithm will never converge - it will continue to make updates indefinitely as it cycles through the data.
Since we're told that the algorithm converges after k updates, this implies that the algorithm has found a separating hyperplane, which means the data must be linearly separable.

$\therefore$ The statement "The training data must be linearly separable" is **necessarily true**. The convergence of the Perceptron algorithm implies that it has found a hyperplane that correctly classifies all training examples, which by definition means the data is linearly separable.