

Linear Models for Classification

Dan Lizotte

2017-02-02

Linear models in general HTF Ch. 2.8.3

- By linear models, we mean that the hypothesis function $h_{\mathbf{w}}(\mathbf{x})$ is a *linear function of the parameters \mathbf{w}* .
- Predictions are a *linear combination of feature values*

-

$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{k=0}^p w_k \phi_k(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{w}$$

where ϕ_k are called *basis functions*. As usual, we let $\phi_0(\mathbf{x}) = 1, \forall \mathbf{x}$, to create a bias.

- To recover degree- d polynomial regression in one variable, set

$$\phi_0(x) = 1, \phi_1(x) = x, \phi_2(x) = x^2, \dots, \phi_d(x) = x^d$$

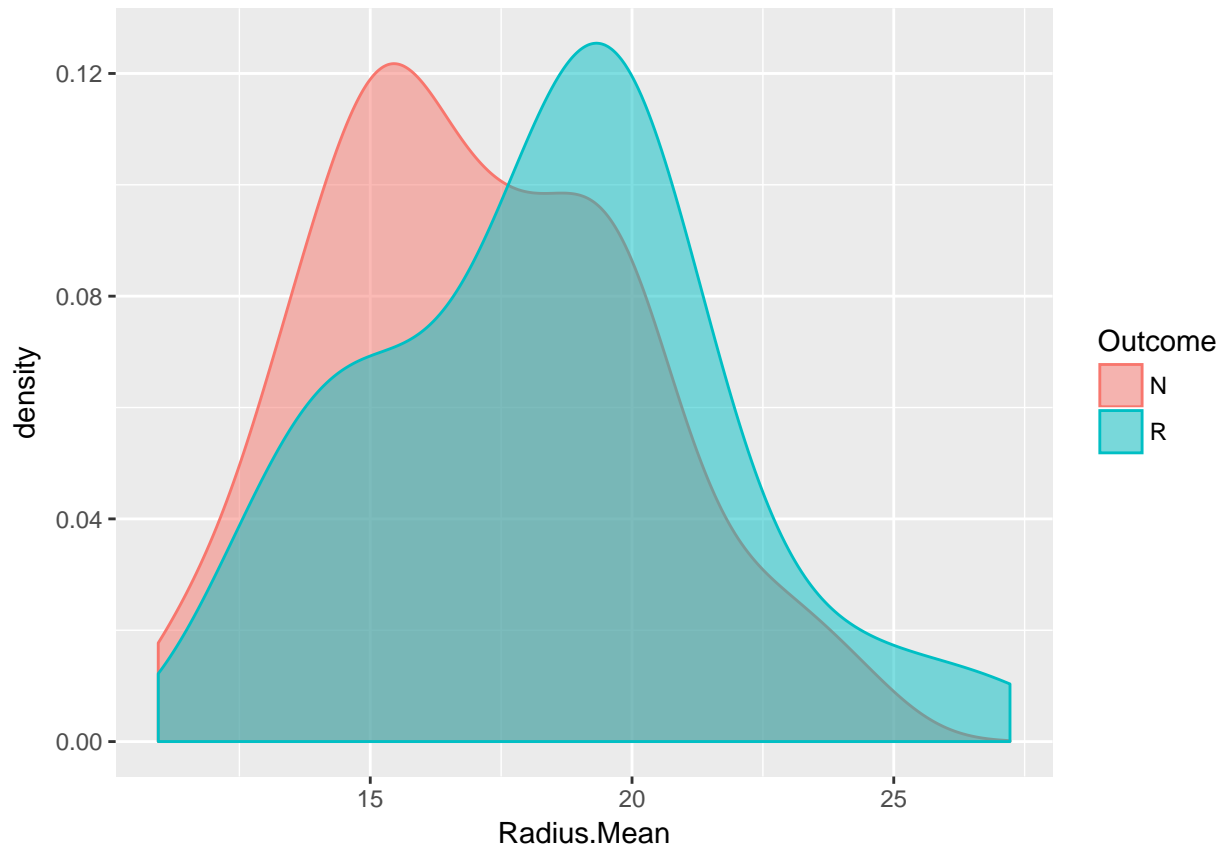
- Basis functions are *fixed* for training

Linear Methods for Classification

- Classification tasks
- Error functions for classification
- Logistic Regression
- Support Vector Machines

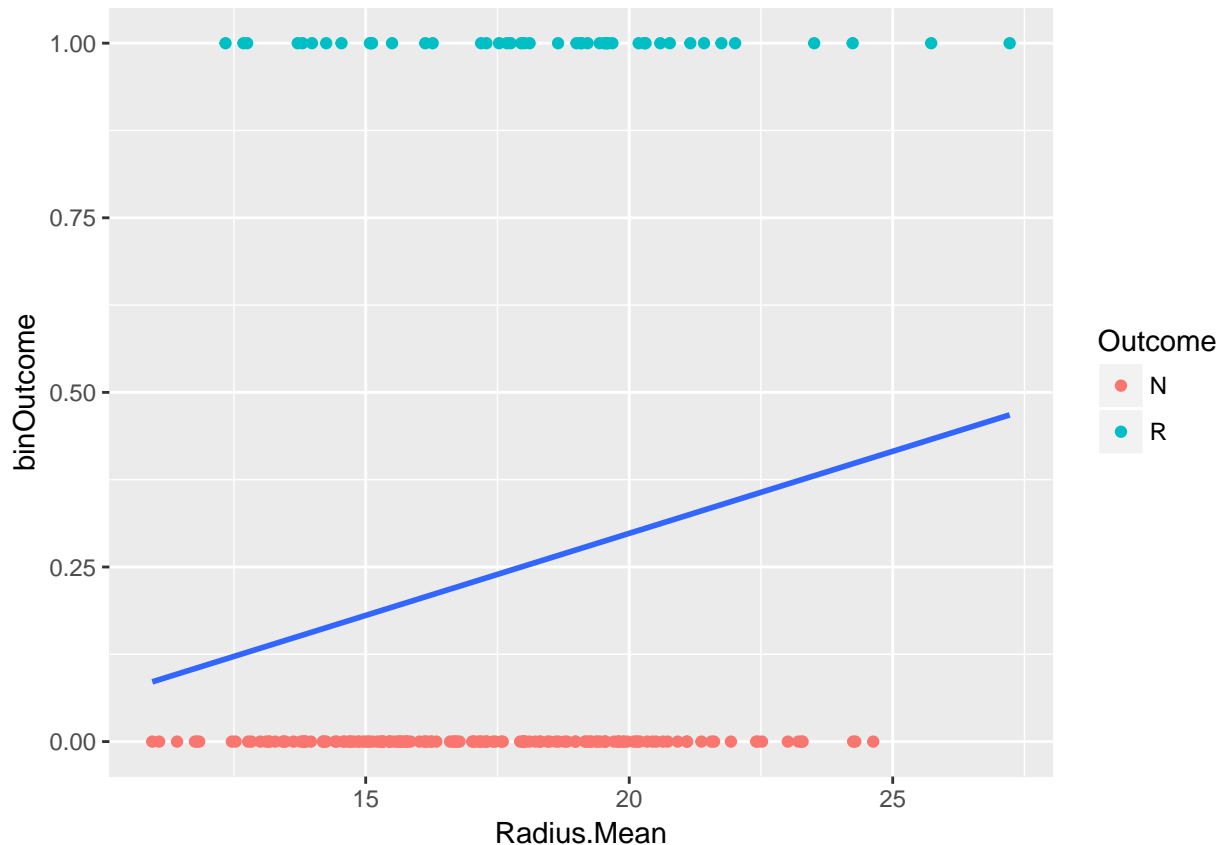
Example: Given nucleus radius, predict cancer recurrence

```
ggplot(bc, aes(Radius.Mean, fill=Outcome, color=Outcome)) + geom_density(alpha=I(1/2))
```



Example: Solution by linear regression

- Univariate real input: nucleus size
- Output coding: non-recurrence = 0, recurrence = 1
- **Sum squared error** minimized by the blue line



Linear regression for classification

- The predictor shows an increasing trend towards recurrence with larger nucleus size, as expected.
- Output *cannot be directly interpreted* as a class prediction.
- Thresholding output (e.g., at 0.5) could be used to predict 0 or 1.
(In this case, prediction would be 0 except for extremely large nucleus size.)
- Interpret as probability? Not bounded to $[0, 1]$, not consistent even for well-separated data

Probabilistic view

- Suppose we have two possible classes: $y \in \{0, 1\}$.
- The symbols “0” and “1” are unimportant. Could have been $\{a, b\}$, $\{up, down\}$, whatever.
- Rather than try to predict the class label directly, ask:
What is the *probability* that a given input \mathbf{x} to has class $y = 1$?
- Conditional Probability:

$$P(y = 1|\mathbf{X} = \mathbf{x}) = \frac{P(\mathbf{X} = \mathbf{x}, y = 1)}{P(\mathbf{X} = \mathbf{x})}$$

- Bayes' Rule

$$= \frac{P(\mathbf{X} = \mathbf{x}|y = 1)P(y = 1)}{P(\mathbf{X} = \mathbf{x}|y = 1)P(y = 1) + P(\mathbf{X} = \mathbf{x}|y = 0)P(y = 0)}$$

Probabilistic models for binary classification

- Can also write:

$$P(y = 1|\mathbf{X} = \mathbf{x}) = \sigma \left(\log \frac{P(y = 1|\mathbf{X} = \mathbf{x})}{P(y = 0|\mathbf{X} = \mathbf{x})} \right) = \sigma \left(\log \frac{P(\mathbf{X} = \mathbf{x}|y = 1)P(y = 1)}{P(\mathbf{X} = \mathbf{x}|y = 0)P(y = 0)} \right)$$

where $\sigma(a) = \frac{1}{1+\exp(-a)}$, the *sigmoid* or *logistic function*.

- Discriminative Learning:
 - Model $\log \frac{P(y=1|\mathbf{X}=\mathbf{x})}{P(y=0|\mathbf{X}=\mathbf{x})}$ (*log odds*) as a function of \mathbf{x}
 - Only models how to *discriminate* between examples of the two classes. Does not model distribution of \mathbf{x} .
- Generative Learning:
 - Model $P(y = 1), P(y = 0), P(\mathbf{X} = \mathbf{x}|y = 1), P(\mathbf{X} = \mathbf{x}|y = 0)$, then use rightmost formula above
 - Models the full joint; can actually use the model to *generate* (i.e. fantasize) data

Logistic regression HTF (Ch. 4.4)

- Represent the hypothesis as a logistic function of a linear combination of inputs:

$$h(\mathbf{x}) = \sigma(\mathbf{x}^\top \mathbf{w})$$

- Interpret $h(\mathbf{x})$ as $P(y = 1|\mathbf{X} = \mathbf{x})$, interpret $\mathbf{x}^\top \mathbf{w}$ as the log-odds
- How do we choose \mathbf{w} ?
- In the probabilistic framework, observing $\langle \mathbf{x}_i, 1 \rangle$ does not mean $h(\mathbf{x}_i)$ should be as close to 1 as possible.
- **Maximize probability the model assigns to the y_i , given the \mathbf{x}_i .**

Max Conditional Likelihood

- **Maximize probability the model assigns to the y_i , given the \mathbf{x}_i .**
- Assumption 1: Examples are i.i.d. Probability of observing all y s is product

$$\begin{aligned} P(Y_1 = y_1, Y_2 = y_2, \dots, Y_n = y_n | X_1 = \mathbf{x}_1, X_2 = \mathbf{x}_2, \dots, X_n = \mathbf{x}_n) \\ = \prod_{i=1}^n P(Y_i = y_i | X_i = \mathbf{x}_i) \end{aligned}$$

- Assumption 2:

$$\begin{aligned} P(y = 1|\mathbf{X} = \mathbf{x}) &= h_{\mathbf{w}}(\mathbf{x}) = 1/(1 + \exp(-\mathbf{x}^\top \mathbf{w})) \\ P(y = 0|\mathbf{X} = \mathbf{x}) &= (1 - h_{\mathbf{w}}(\mathbf{x})) \end{aligned}$$

Max Conditional Likelihood

- Maximize probability the model assigns to the y_i , *given the x_i* .
- More stable to maximize log probability. Note

$$\log P(Y_i = y_i | X_i = \mathbf{x}_i) = \begin{cases} \log h_{\mathbf{w}}(\mathbf{x}_i) & \text{if } y_i = 1 \\ \log(1 - h_{\mathbf{w}}(\mathbf{x}_i)) & \text{if } y_i = 0 \end{cases}$$

- Therefore,

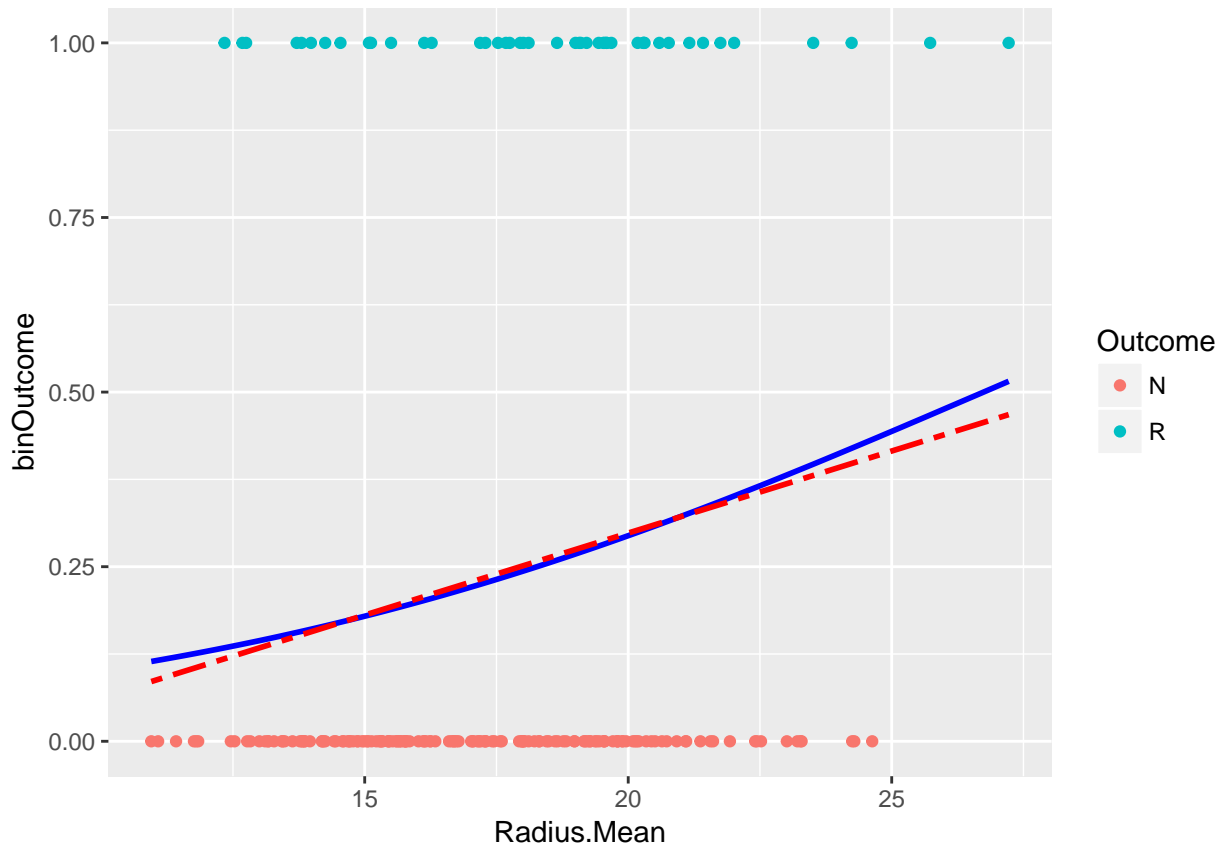
$$\log \prod_{i=1}^n P(Y_i = y_i | X_i = \mathbf{x}_i) = \sum_{i=1}^n [y_i \log(h_{\mathbf{w}}(\mathbf{x}_i)) + (1 - y_i) \log(1 - h_{\mathbf{w}}(\mathbf{x}_i))]$$

- Suggests an error

$$J(h_{\mathbf{w}}) = - \sum_{i=1}^n [y_i \log(h_{\mathbf{w}}(\mathbf{x}_i)) + (1 - y_i) \log(1 - h_{\mathbf{w}}(\mathbf{x}_i))]$$

- This is the *cross entropy*. Number of bits to transmit y_i if both parties know $h_{\mathbf{w}}$ and \mathbf{x}_i .

Back to the breast cancer problem



Logistic Regression:

```
## (Intercept) Radius.Mean
## -3.4671348  0.1296493
```

Least Squares:

```
## (Intercept) Radius.Mean
## -0.17166939  0.02349159
```

Supervised Learning Methods: “Objective-driven”

Mthd.	Form	Objective
OLS	$h_w(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}$ $\approx E[Y = y \mathbf{X} = \mathbf{x}] \dots$	$\sum_{i=1}^n (h_w(\mathbf{x}_i) - y_i)^2$ <p>... using a linear function</p>
LR	$h_w(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}^\top \mathbf{w}}}$ $\approx P(Y = y \mathbf{X} = \mathbf{x}) \dots$	$-\sum_{i=1}^n y_i \log h_w(\mathbf{x}_i) + (1 - y_i) \log(1 - h_w(\mathbf{x}_i))$ <p>... using sigmoid of a linear function</p>

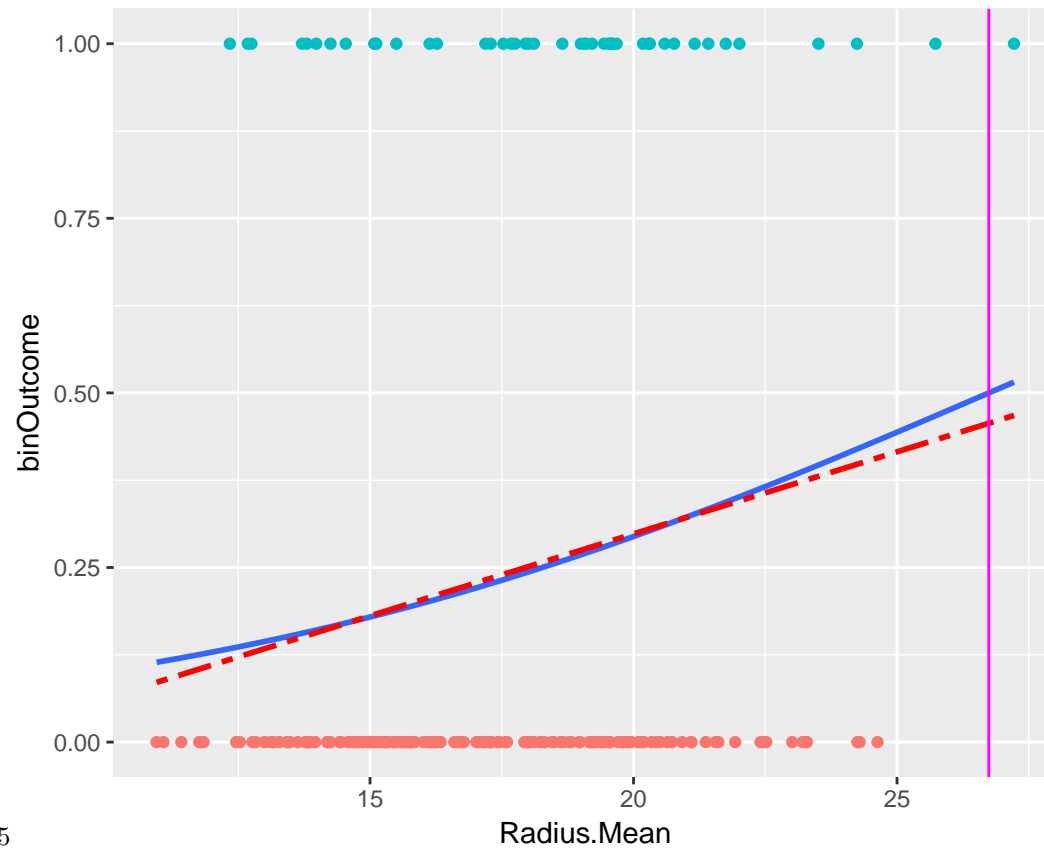
- Both model the *conditional mean of y* using a (transformed) *linear function*
- Both use *maximum conditional likelihood* to estimate

Decision boundary HTF Ch. 2.3.1,2.3.2

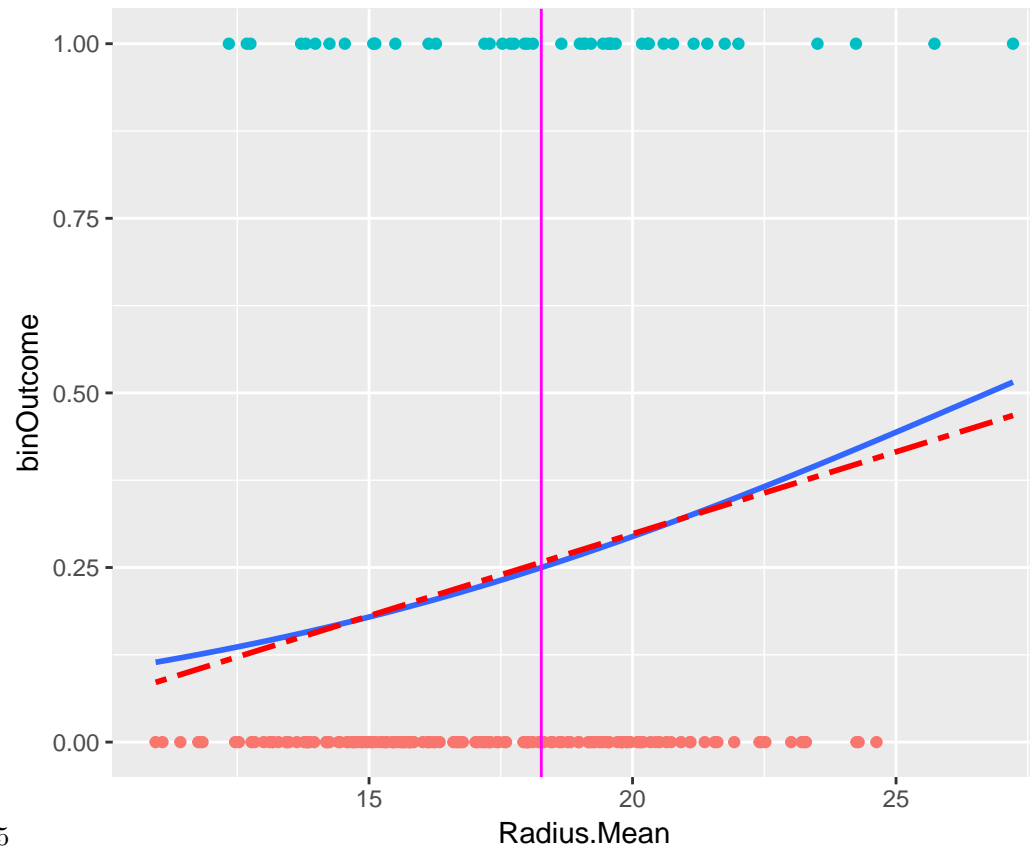
- How complicated is a classifier?
- One way to think about it is in terms of its *decision boundary*, i.e. the line it defines for separating examples
- *Linear classifiers* draw a hyperplane between examples of the different classes. *Non-linear classifiers* draw more complicated surfaces between the different classes.
- For a probabilistic classifier with a cutoff of 0.5, the decision boundary is the curve on which:

$$\frac{P(y = 1 | \mathbf{X} = \mathbf{x})}{P(y = 0 | \mathbf{X} = \mathbf{x})} = 1, \text{ i.e., where } \log \frac{P(y = 1 | \mathbf{X} = \mathbf{x})}{P(y = 0 | \mathbf{X} = \mathbf{x})} = 0$$

Decision boundary

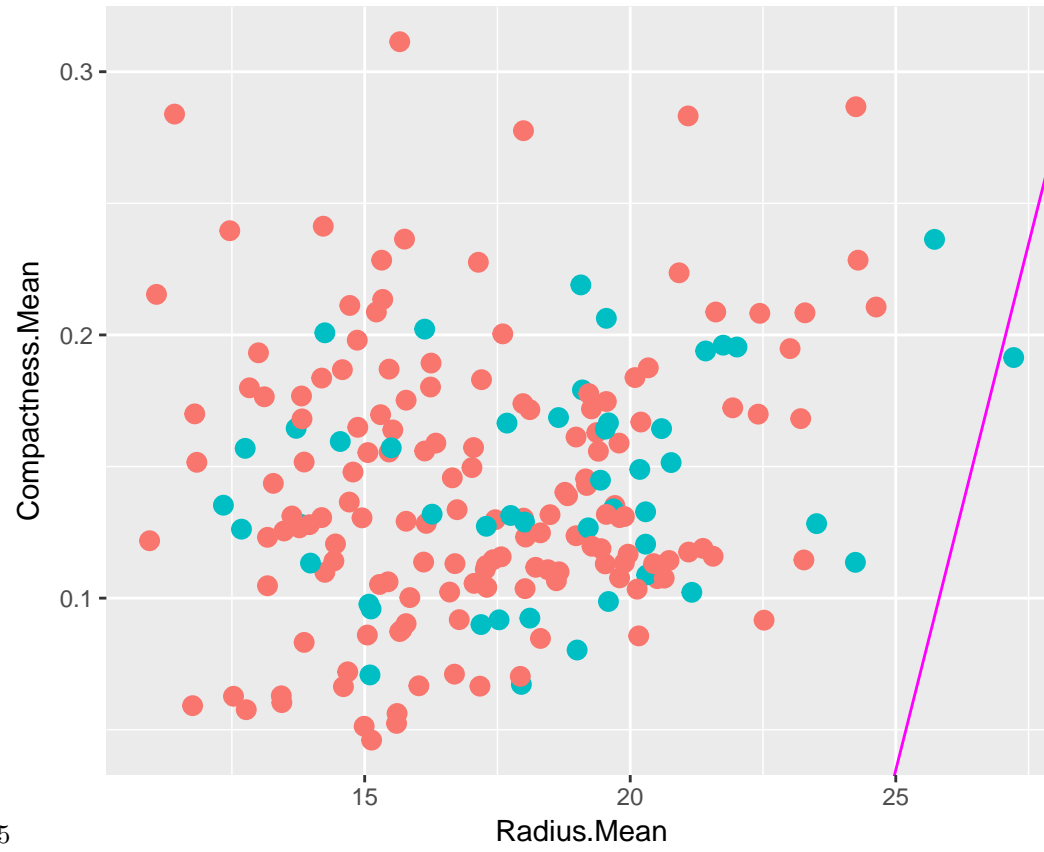


Decision boundary



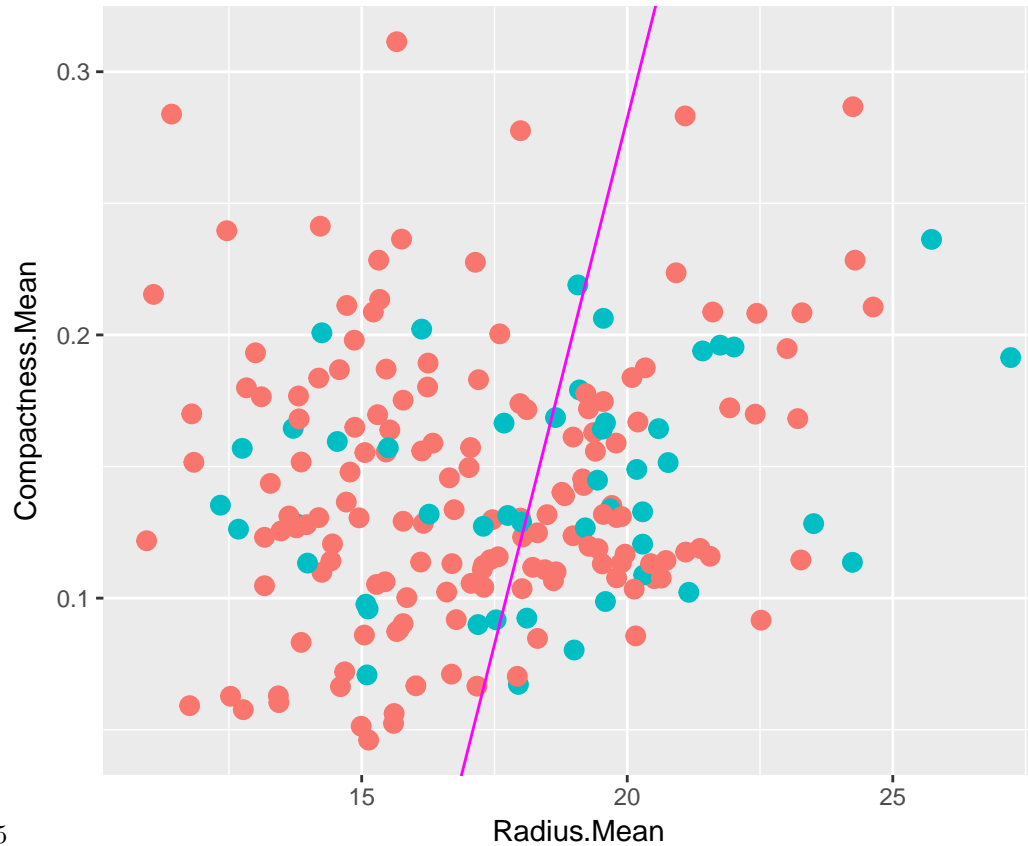
Class = R if $\Pr(Y = 1|X = x) > 0.25$

Decision boundary



Class = R if $\Pr(Y = 1|X = x) > 0.5$

Decision boundary



Class = R if $\Pr(Y = 1|X = x) > 0.25$

Supervised Learning Methods: “Objective-driven”

Mthd.	Form	Objective
OLS	$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$ $\approx E[Y = y \mathbf{X} = \mathbf{x}] \dots$	$\sum_{i=1}^n (h_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2$... using a linear function
LR	$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{x}^T \mathbf{w}}}$ $\approx P(Y = y \mathbf{X} = \mathbf{x}) \dots$	$-\sum_{i=1}^n y_i \log h_{\mathbf{w}}(\mathbf{x}_i) + (1 - y_i) \log(1 - h_{\mathbf{w}}(\mathbf{x}_i))$... using sigmoid of a linear function
SVM	$h_{\mathbf{w}}(\mathbf{x}) = \text{sgn}(\mathbf{x}^T \mathbf{w})$... using a linear function

Large Margin Classifiers: Linear Support Vector Machines

- Linear classifiers that focus on learning the *decision boundary* rather than the conditional distribution $P(Y = y|\mathbf{X} = \mathbf{x})$
 - Perceptrons
 - * Definition
 - * Perceptron learning rule

- * Convergence
- “Margin” idea and max margin classifiers
- (Linear) support vector machines
 - * Formulation as optimization problem

Marvin Minsky, 1927-2016

Perceptrons HTF Ch. 4.5

- Consider a binary classification problem with data $\{\mathbf{x}_i, y_i\}_{i=1}^n$, $y_i \in \{-1, +1\}$. **Note coding of y_i .**
- A *perceptron* (Rosenblatt, 1957) is a classifier of the form:

$$h_{\mathbf{w}, w_0}(\mathbf{x}) = \text{sign}(\mathbf{x}^T \mathbf{w} + w_0) = \begin{cases} +1 & \text{if } \mathbf{x}^T \mathbf{w} + w_0 \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Here, \mathbf{w} is a vector of weights, and w_0 is a constant offset. (**Note $x_0 = 1$ is omitted.**)

- The decision boundary is $\mathbf{x}^T \mathbf{w} + w_0 = 0$.
- Perceptrons output a **class**, not a probability
- An example (\mathbf{x}, y) is classified correctly if:

$$y \cdot (\mathbf{x}^T \mathbf{w} + w_0) > 0$$

Linear separability

- The data set is *linearly separable* if and only if there exists \mathbf{w} , w_0 such that:
 - For all i , $y_i(\mathbf{x}_i^T \mathbf{w} + w_0) > 0$.
 - Or equivalently, the 0-1 loss $\sum_i \mathbf{1}_{y_i(\mathbf{x}_i^T \mathbf{w} + w_0) < 0}$ is zero for some set of parameters (\mathbf{w}, w_0) .

Linear Separability

The Perceptron Learning Rule

- Consider the following procedure:
 1. Initialize \mathbf{w} and w_0 randomly
 2. While any training examples remain incorrectly classified
 1. Loop through all misclassified examples
 2. For misclassified example i , perform the updates:

$$\mathbf{w} \leftarrow \mathbf{w} + \delta y_i \mathbf{x}_i, \quad w_0 \leftarrow w_0 + \delta y_i$$

where δ is a step-size parameter.

- The update equation, or sometimes the whole procedure, is called the *perceptron learning rule*.
- Intuition: For positive examples misclassified as negative, change \mathbf{w} to increase $\mathbf{x}_i^T \mathbf{w} + w_0$, and vice versa

Error Minimization Interpretation

- PLR can be interpreted as a gradient descent on the following function:

$$J(\mathbf{w}, w_0) = \sum_{i=1}^n \begin{cases} 0 & \text{if } y_i(\mathbf{x}_i^T \mathbf{w} + w_0) \geq 0 \\ -y_i(\mathbf{x}_i^T \mathbf{w} + w_0) & \text{if } y_i(\mathbf{x}_i^T \mathbf{w} + w_0) < 0 \end{cases}$$

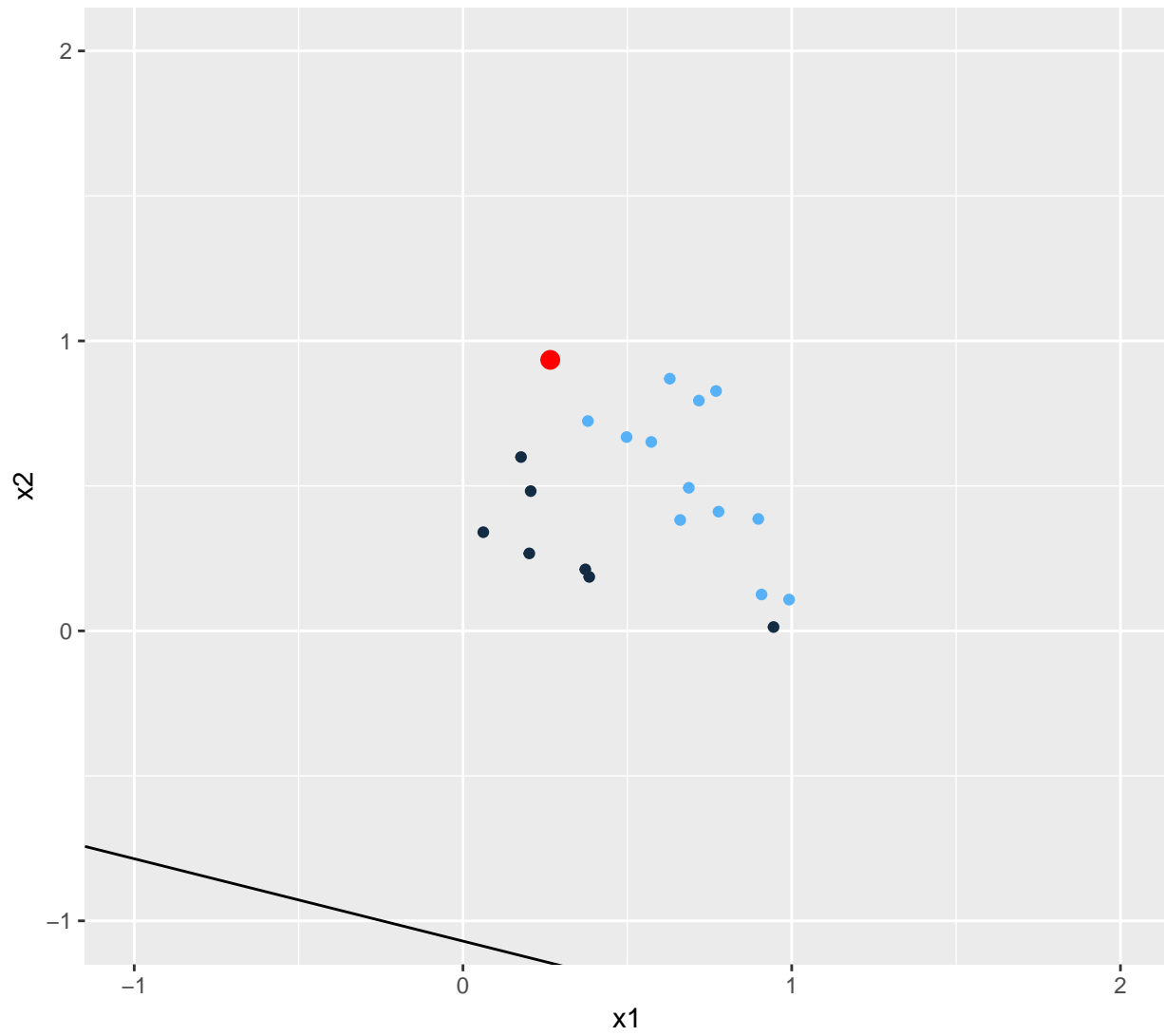
- For correctly classified examples, the error is zero.
- For incorrectly classified examples, the error is by how much $\mathbf{x}_i^T \mathbf{w} + w_0$ is on the wrong side of the decision boundary.
- Err is piecewise linear, so it has a gradient almost everywhere; stochastic gradient descent gives the perceptron learning rule.
- J is zero if and only if all examples are classified correctly – just like the 0-1 loss function.

Perceptron convergence theorem

- **If** data are linearly separable **then** the perceptron learning rule will find a separator after some finite number of updates.
- The number of updates depends on the data set, and also on the step size parameter.
- If the data is not linearly separable, there will be oscillation (which can be detected automatically).

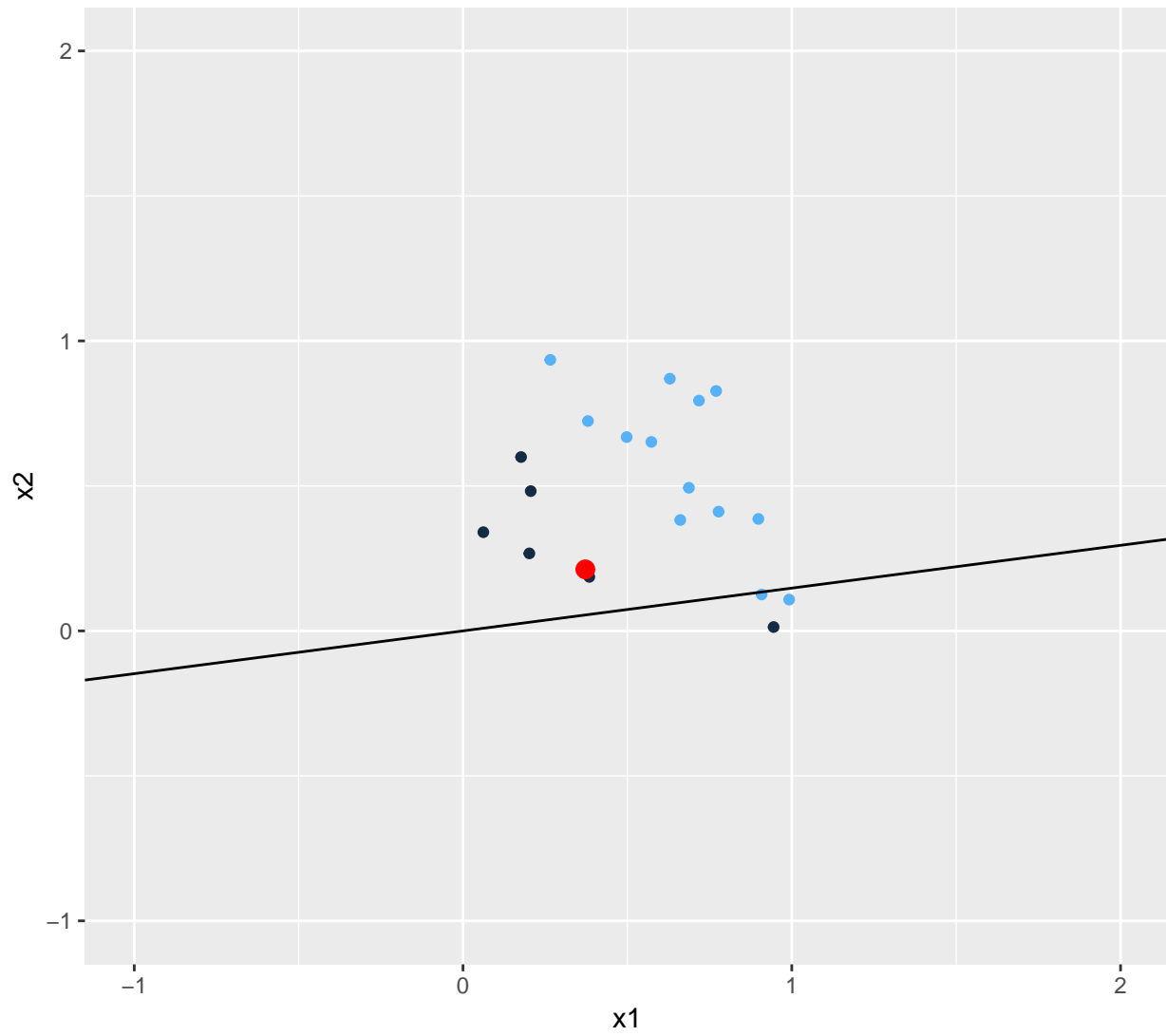
Perceptron Learning Example

Update: 1 $w: [0.266, 0.935]$, $w_0: 1.000$



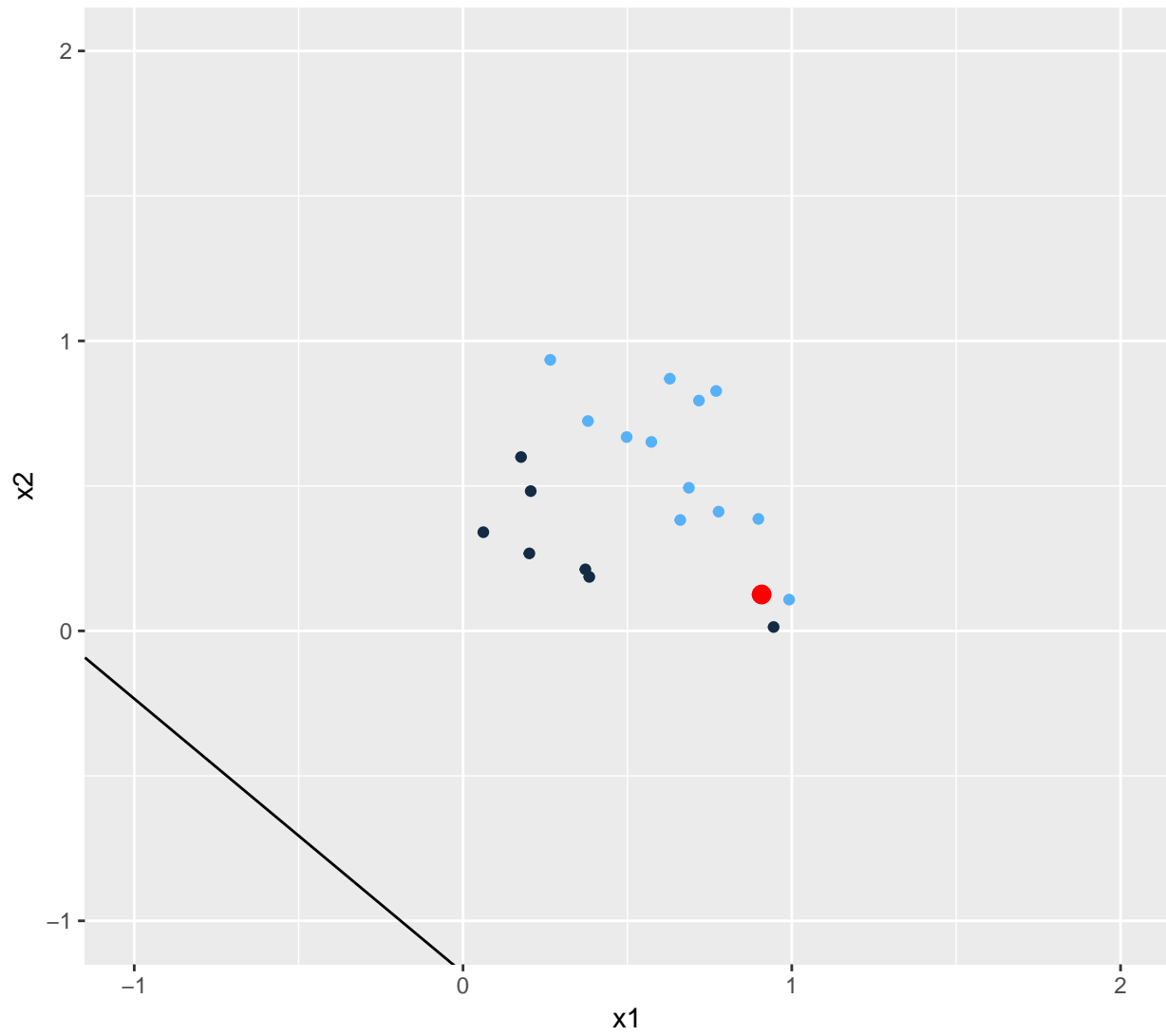
Perceptron Learning Example

Update: 2 $w: [-0.107, 0.723]$, $w_0: 0.000$



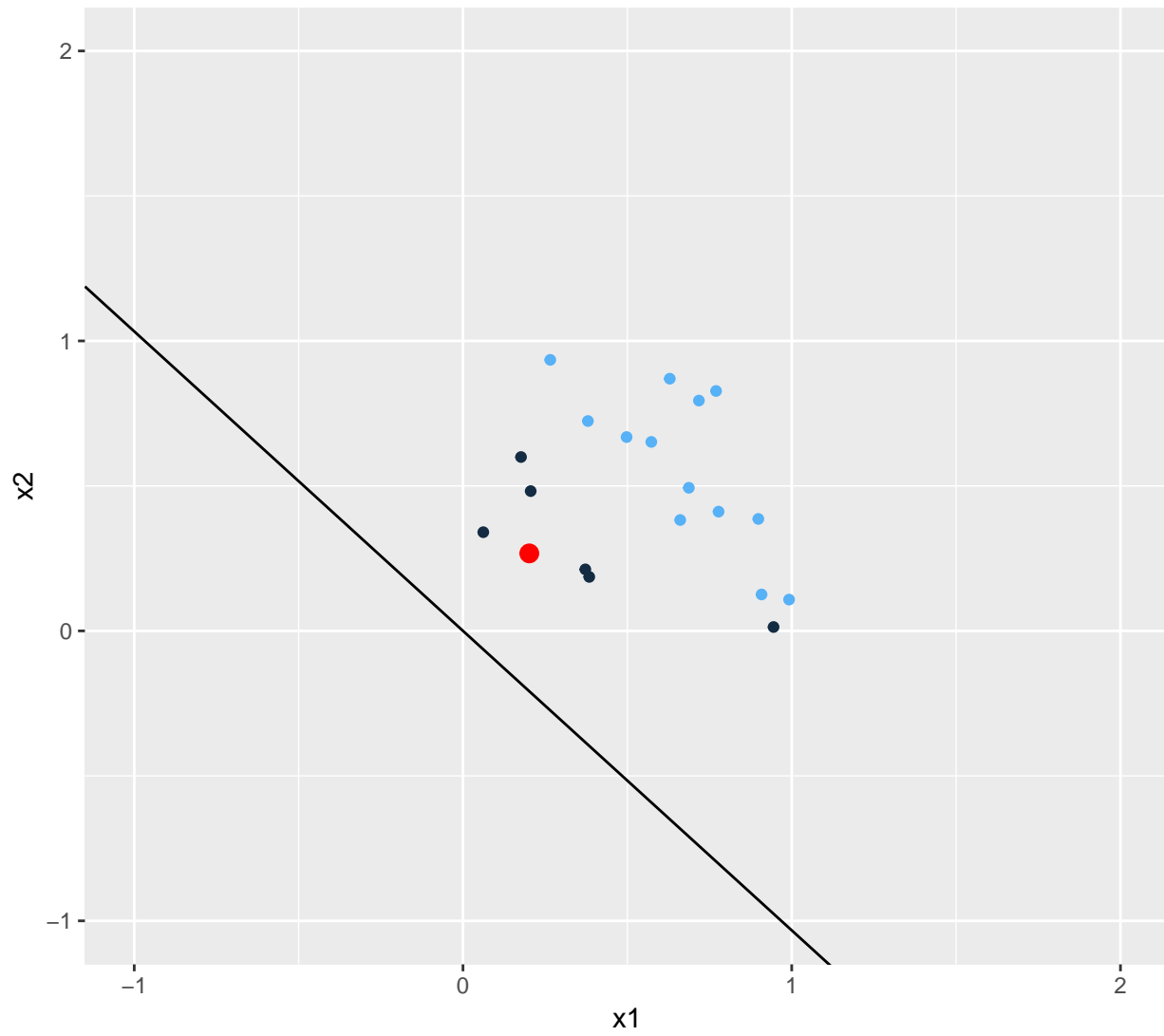
Perceptron Learning Example

Update: 3 $w: [0.802, 0.848]$, $w_0: 1.000$



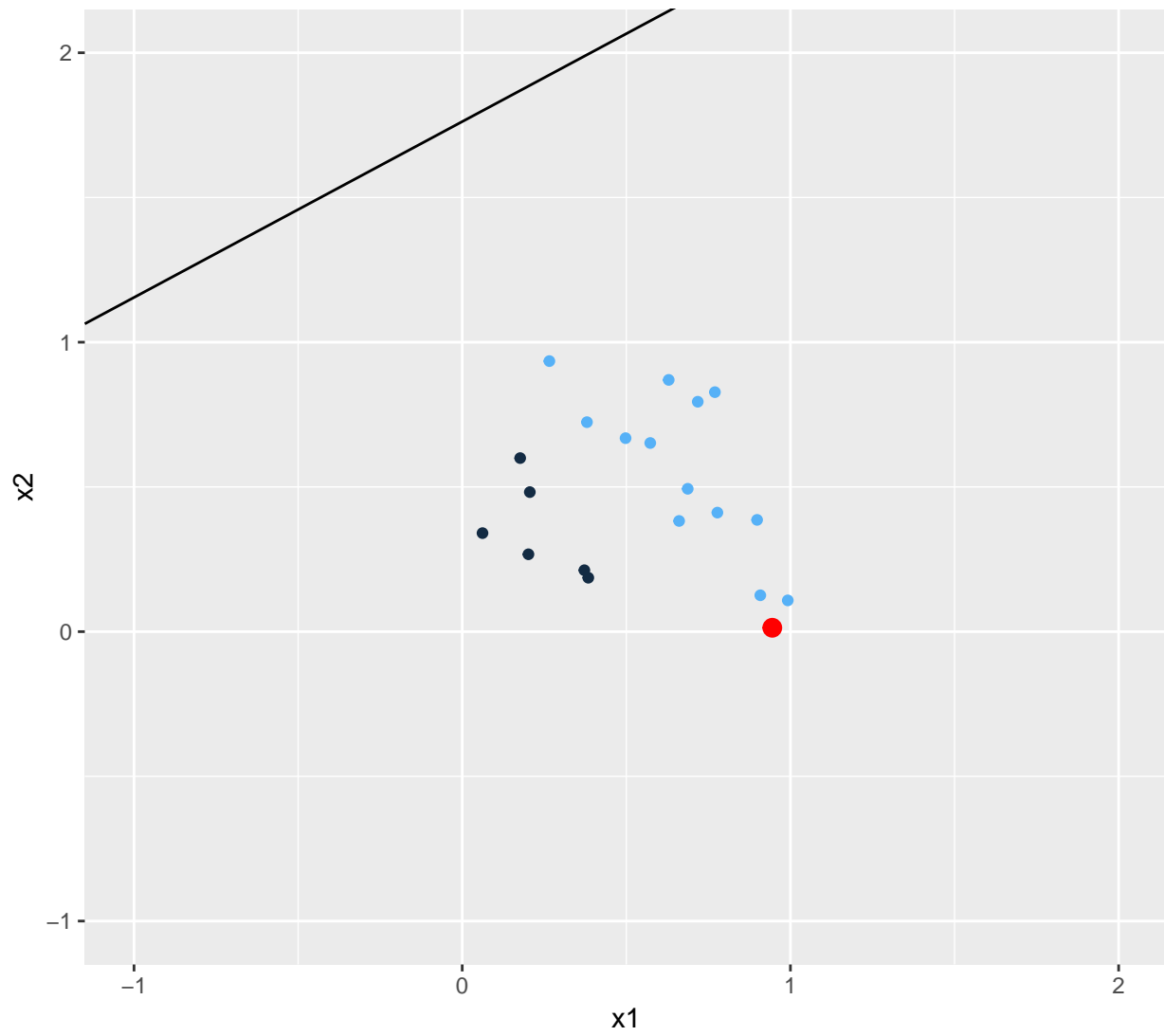
Perceptron Learning Example

Update: 4 $w: [0.600, 0.581]$, $w_0: 0.000$



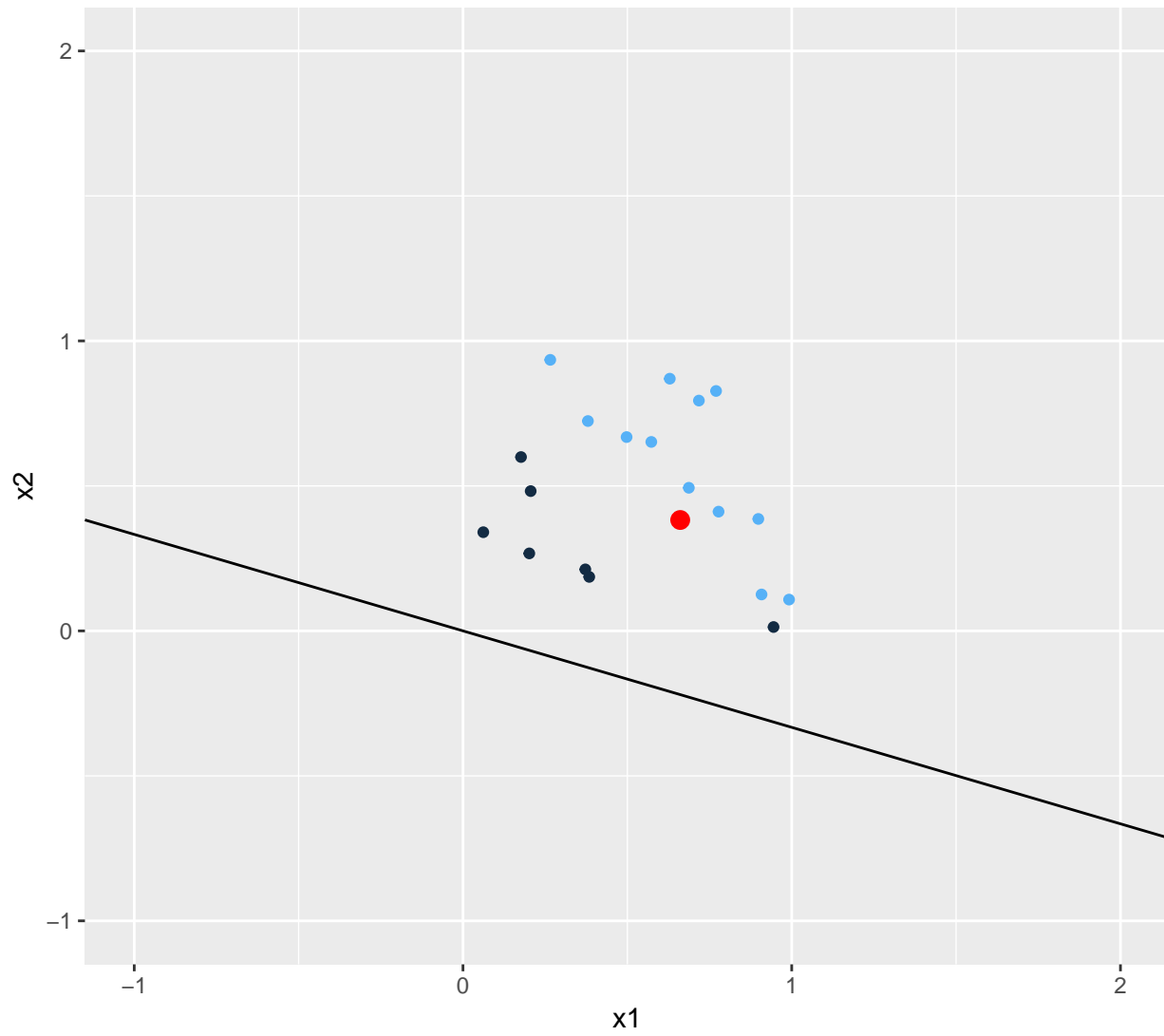
Perceptron Learning Example

Update: 5 $w: [-0.345, 0.568]$, $w_0: -1.000$



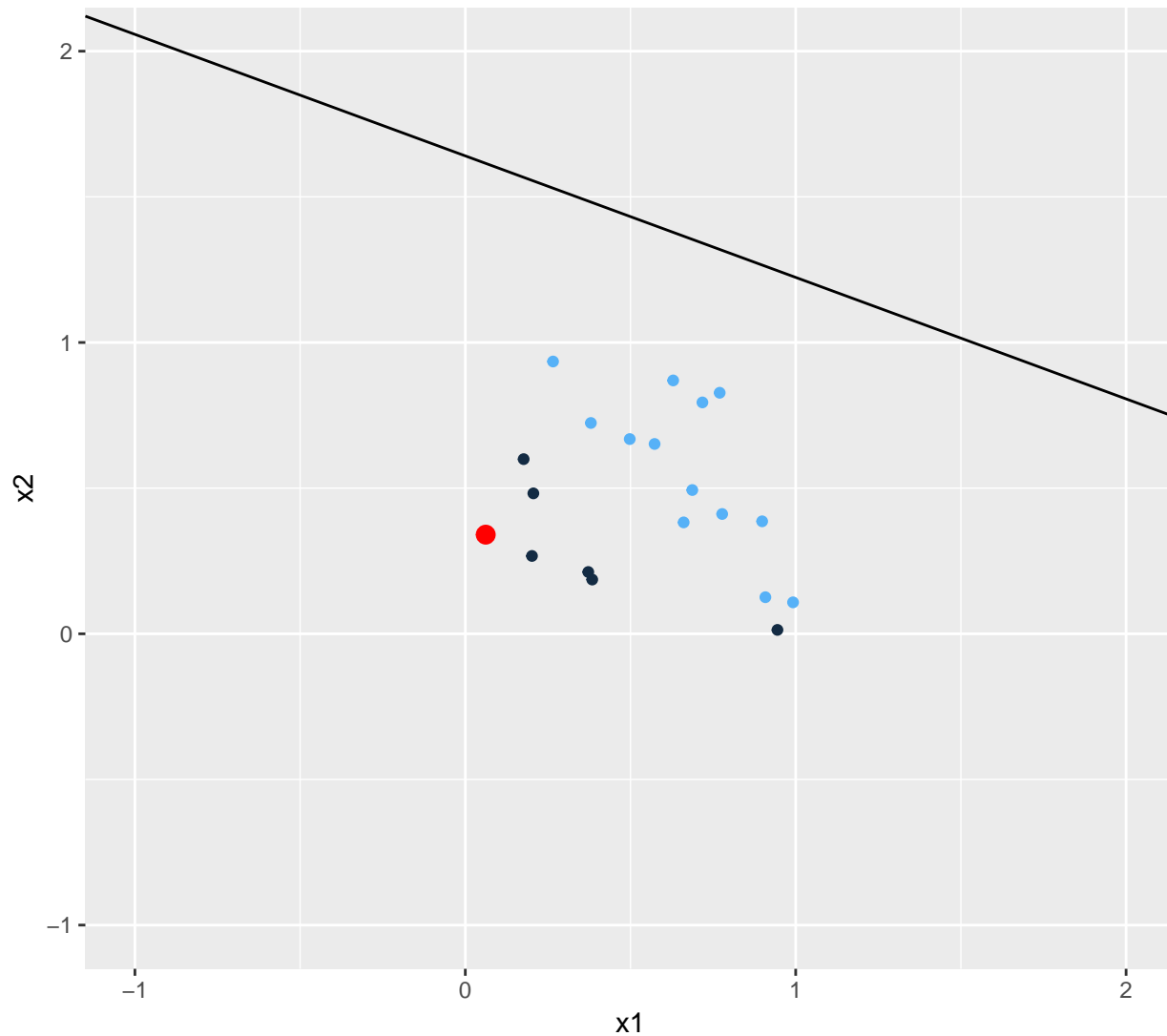
Perceptron Learning Example

Update: 6 $w: [0.316, 0.950]$, $w_0: 0.000$



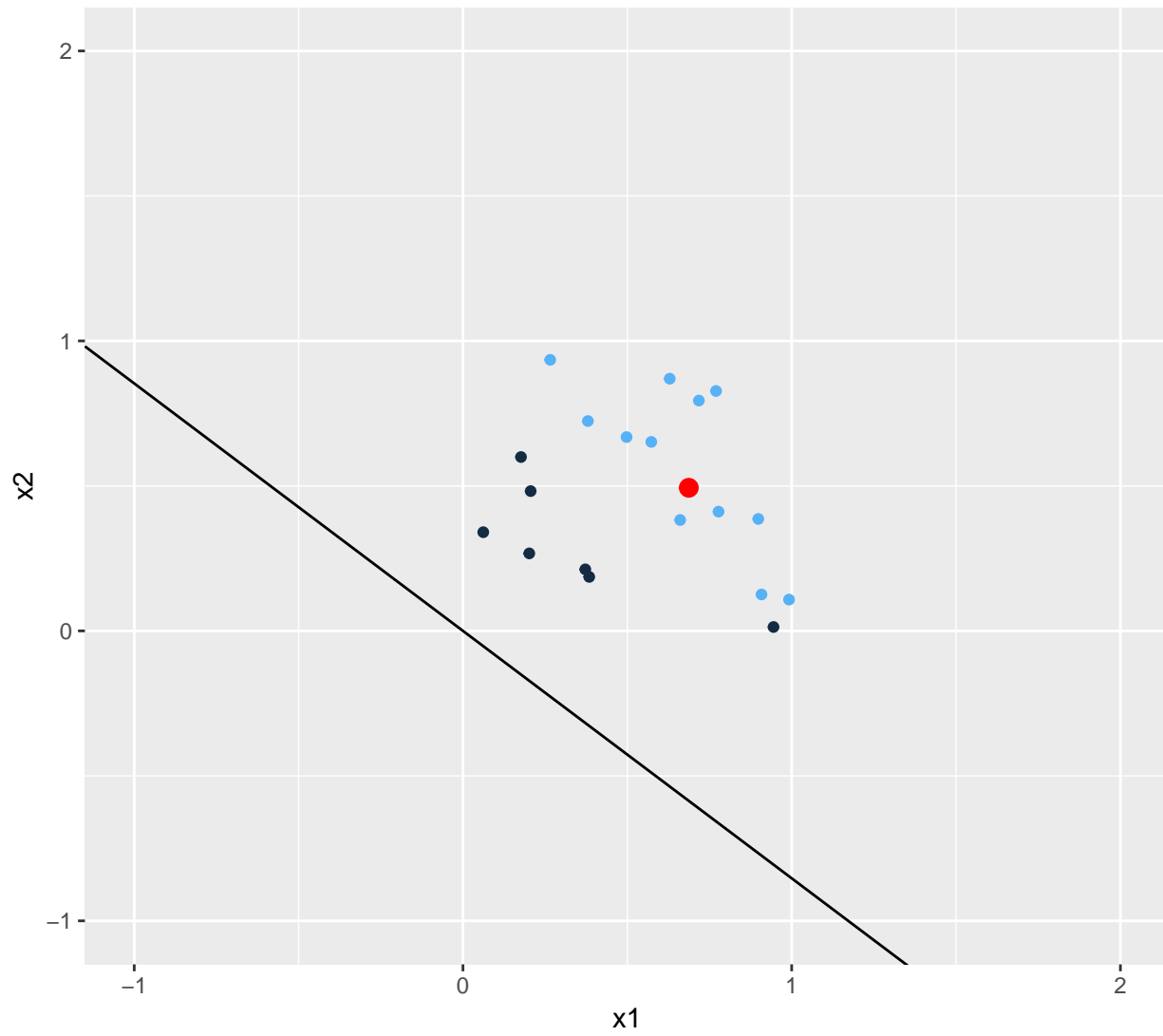
Perceptron Learning Example

Update: 7 $w: [0.254, 0.610]$, $w_0: -1.000$



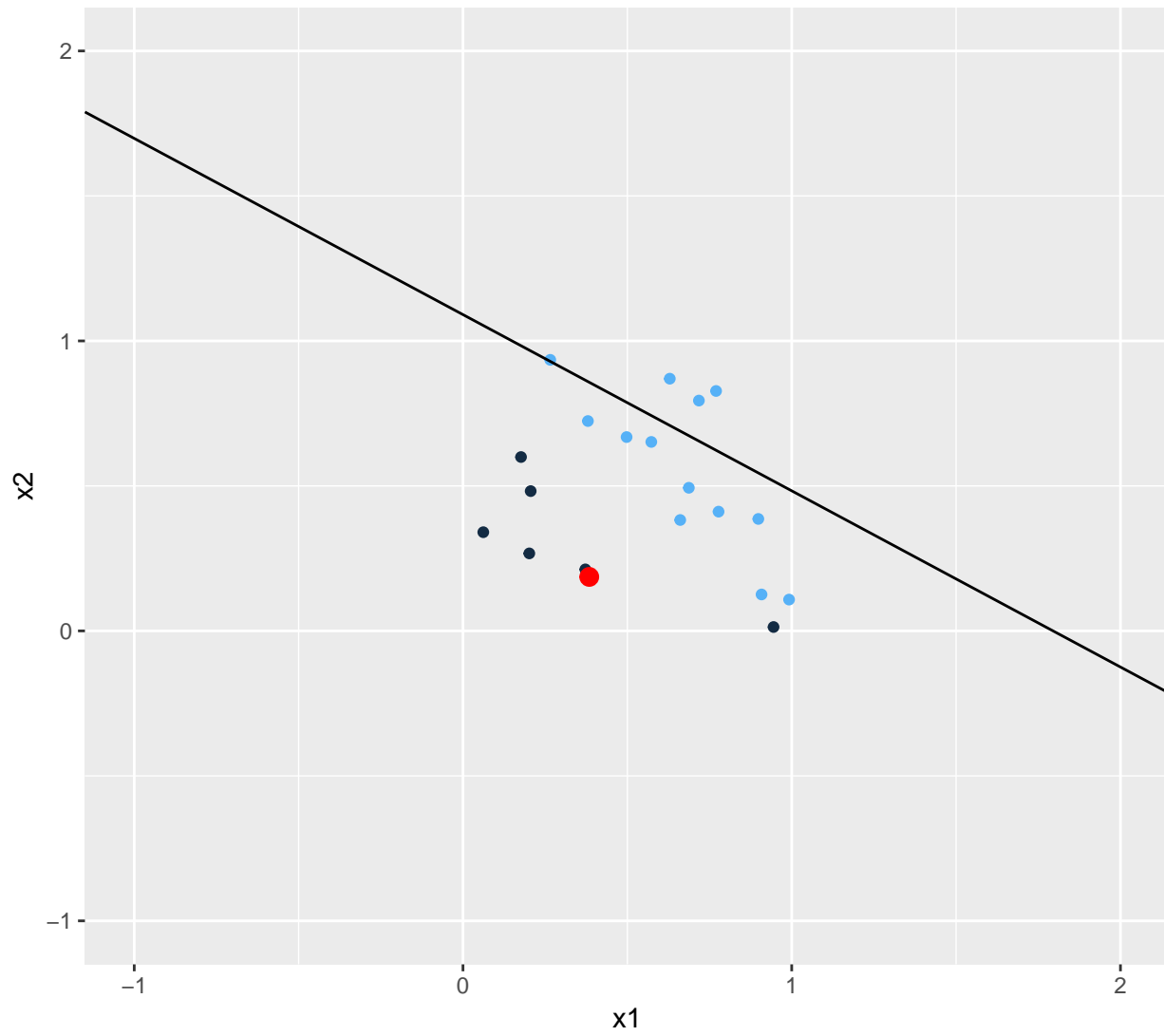
Perceptron Learning Example

Update: 8 $w: [0.941, 1.103]$, $w_0: 0.000$



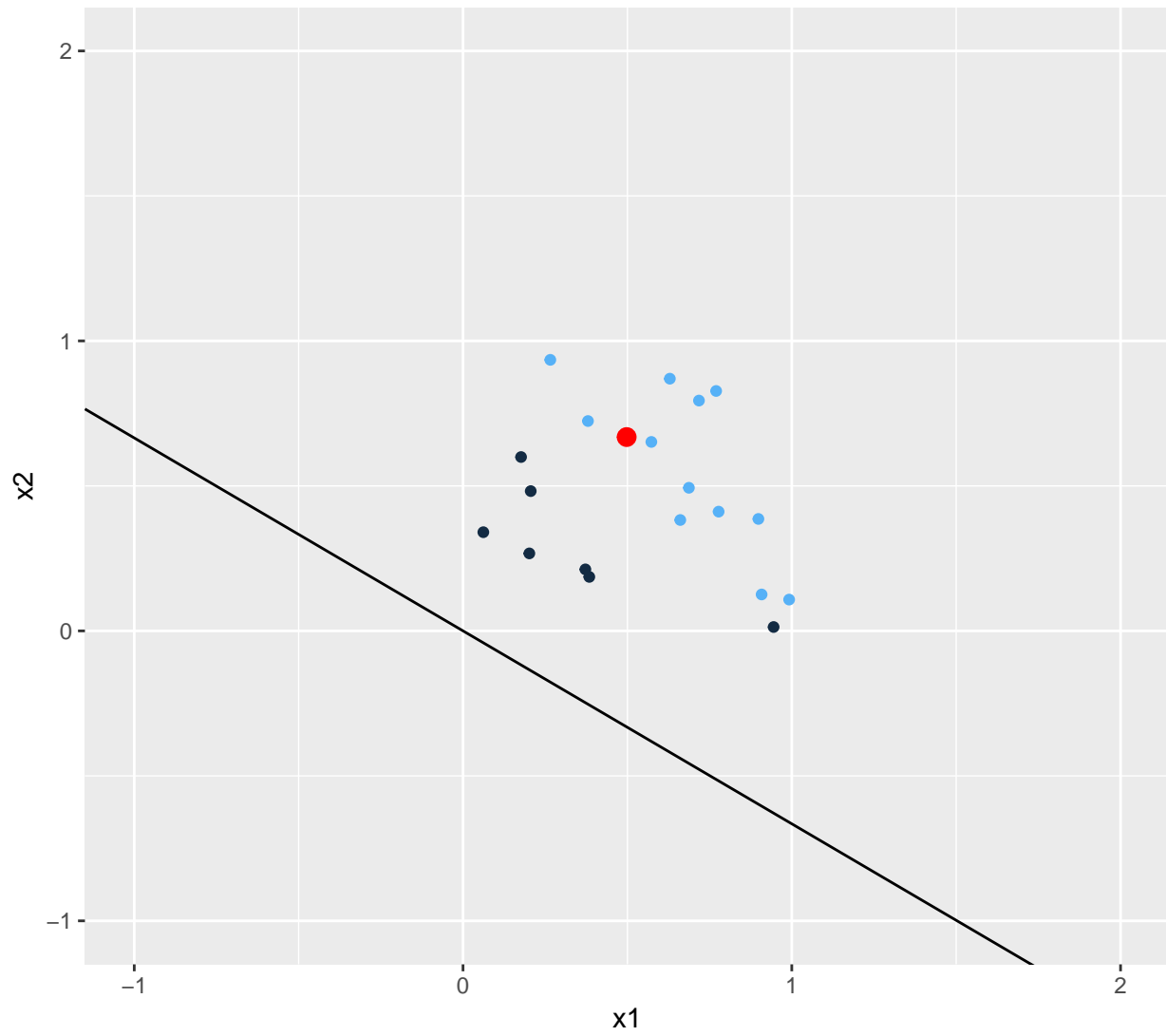
Perceptron Learning Example

Update: 9 $w: [0.557, 0.917]$, $w_0: -1.000$



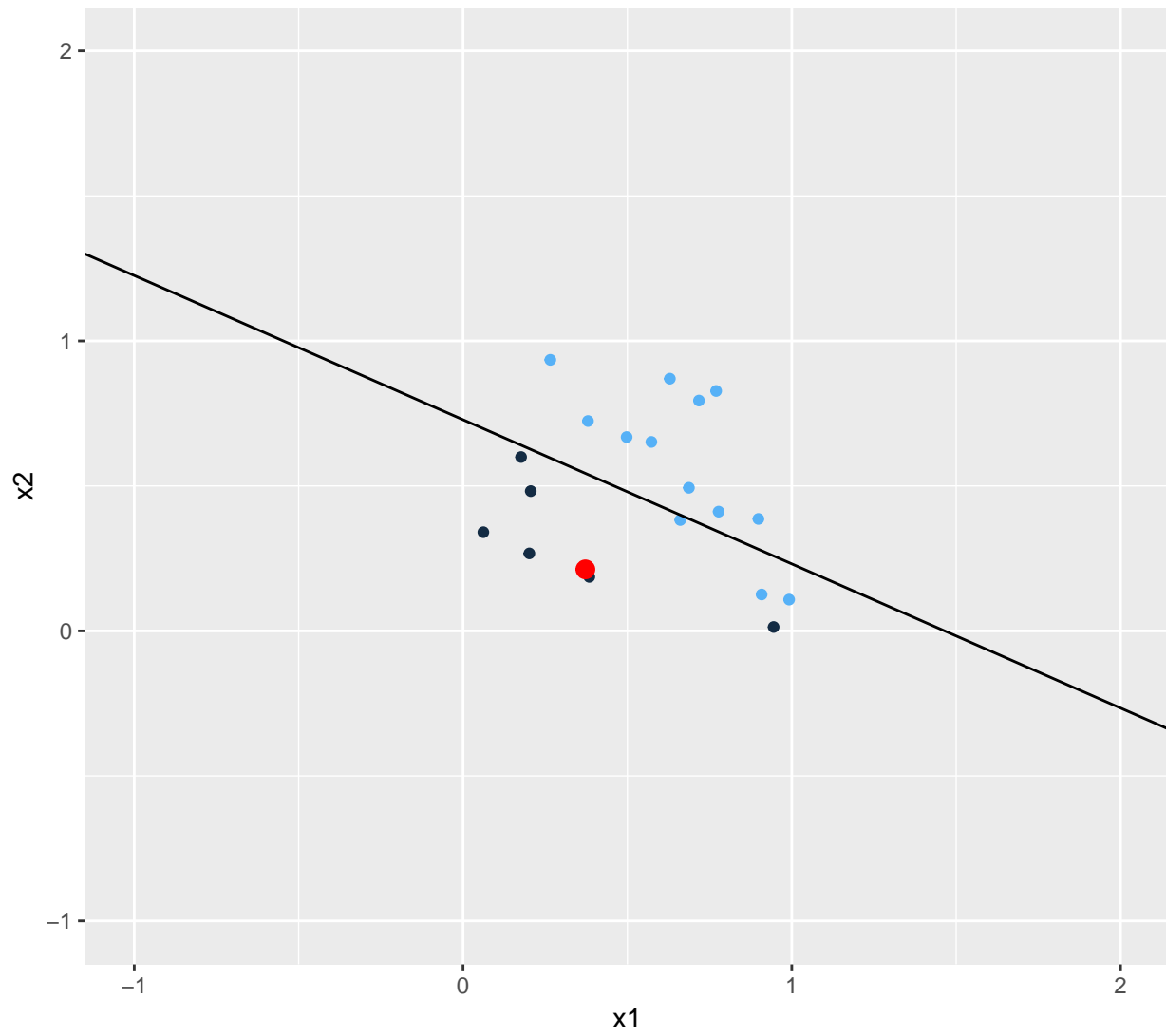
Perceptron Learning Example

Update: 10 w : [1.055, 1.585], w_0 : 0.000



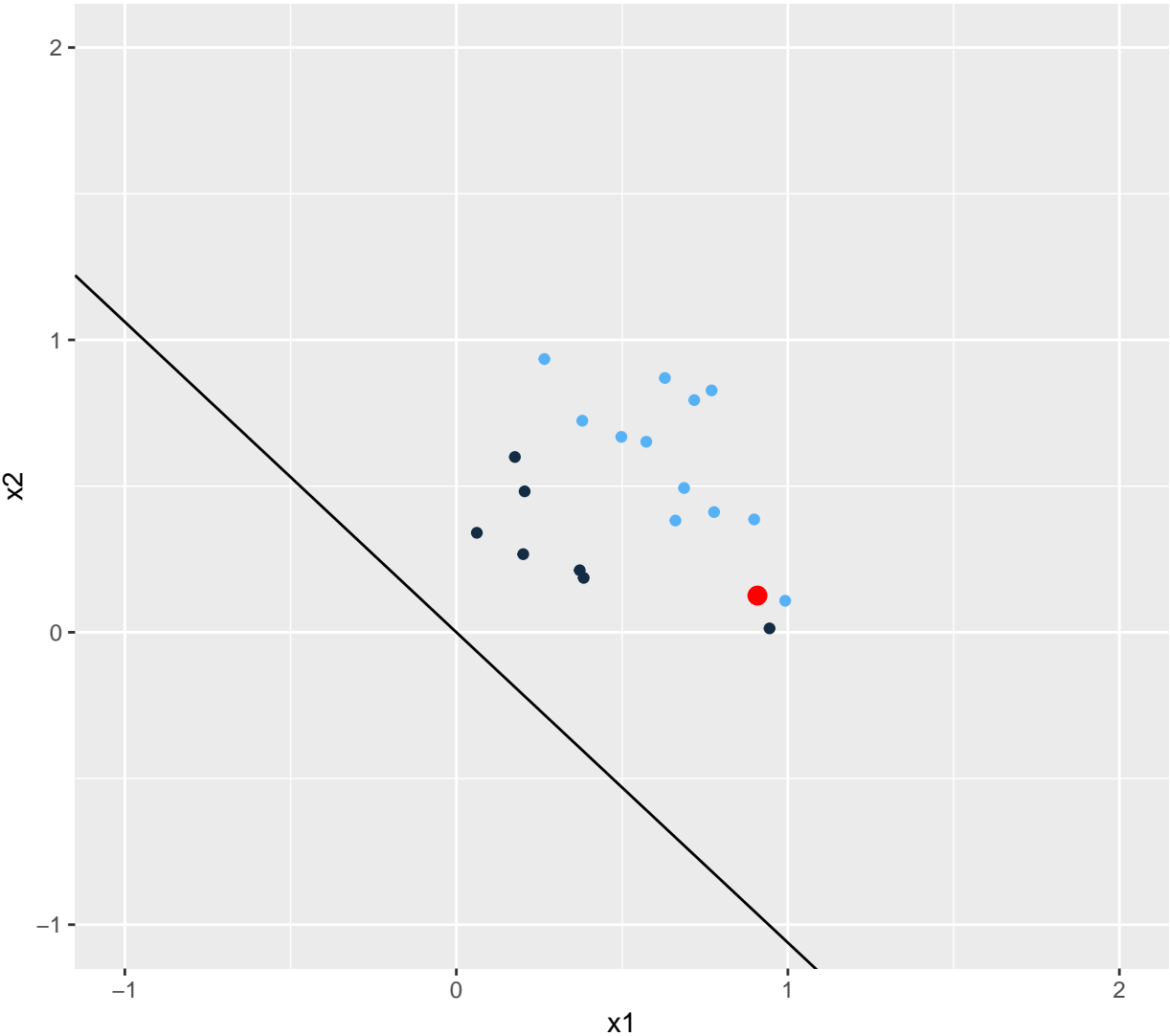
Perceptron Learning Example

Update: 11 $w: [0.683, 1.373]$, $w_0: -1.000$



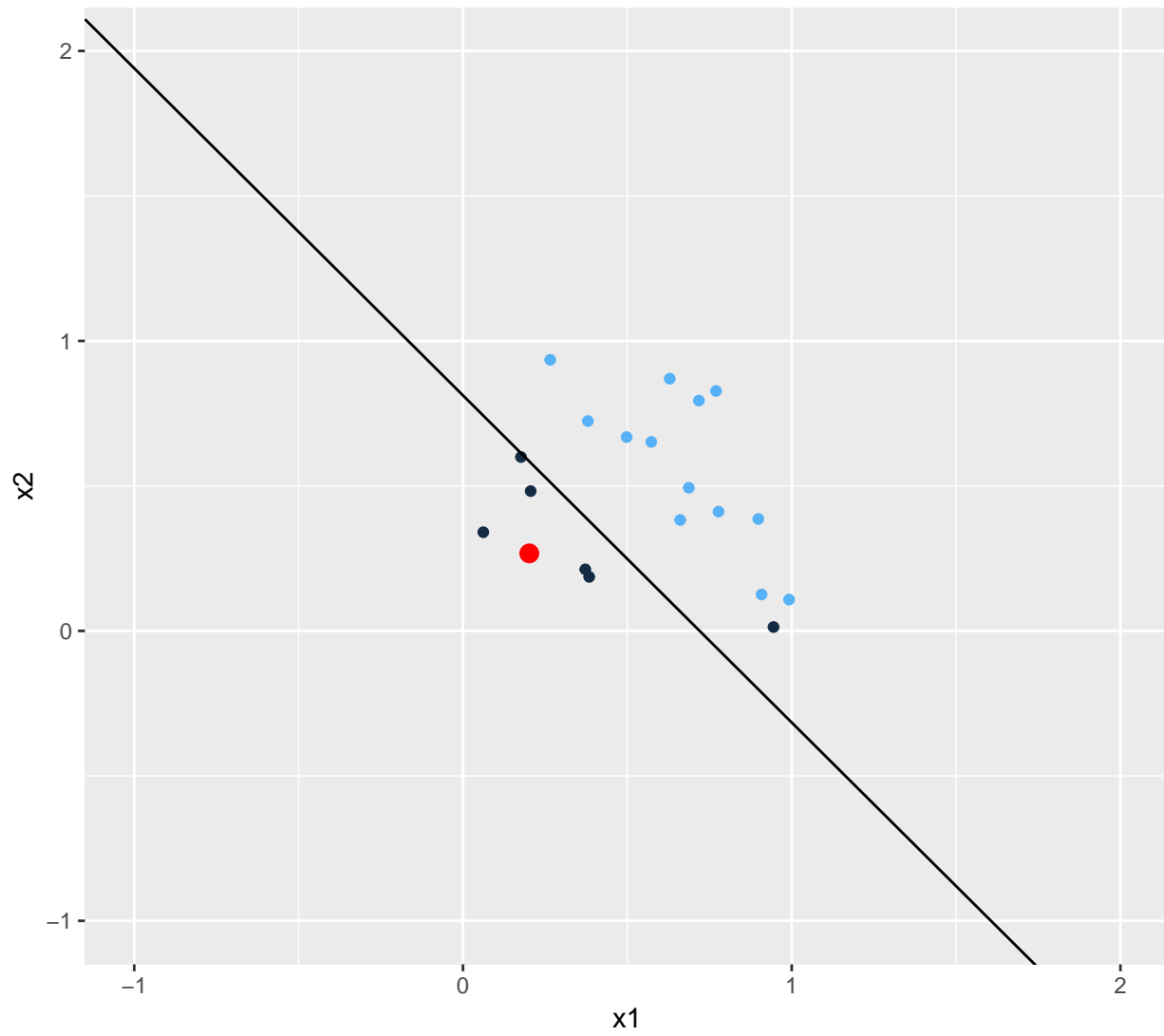
Perceptron Learning Example

Update: 12 w: [1.591,1.499], w0: 0.000



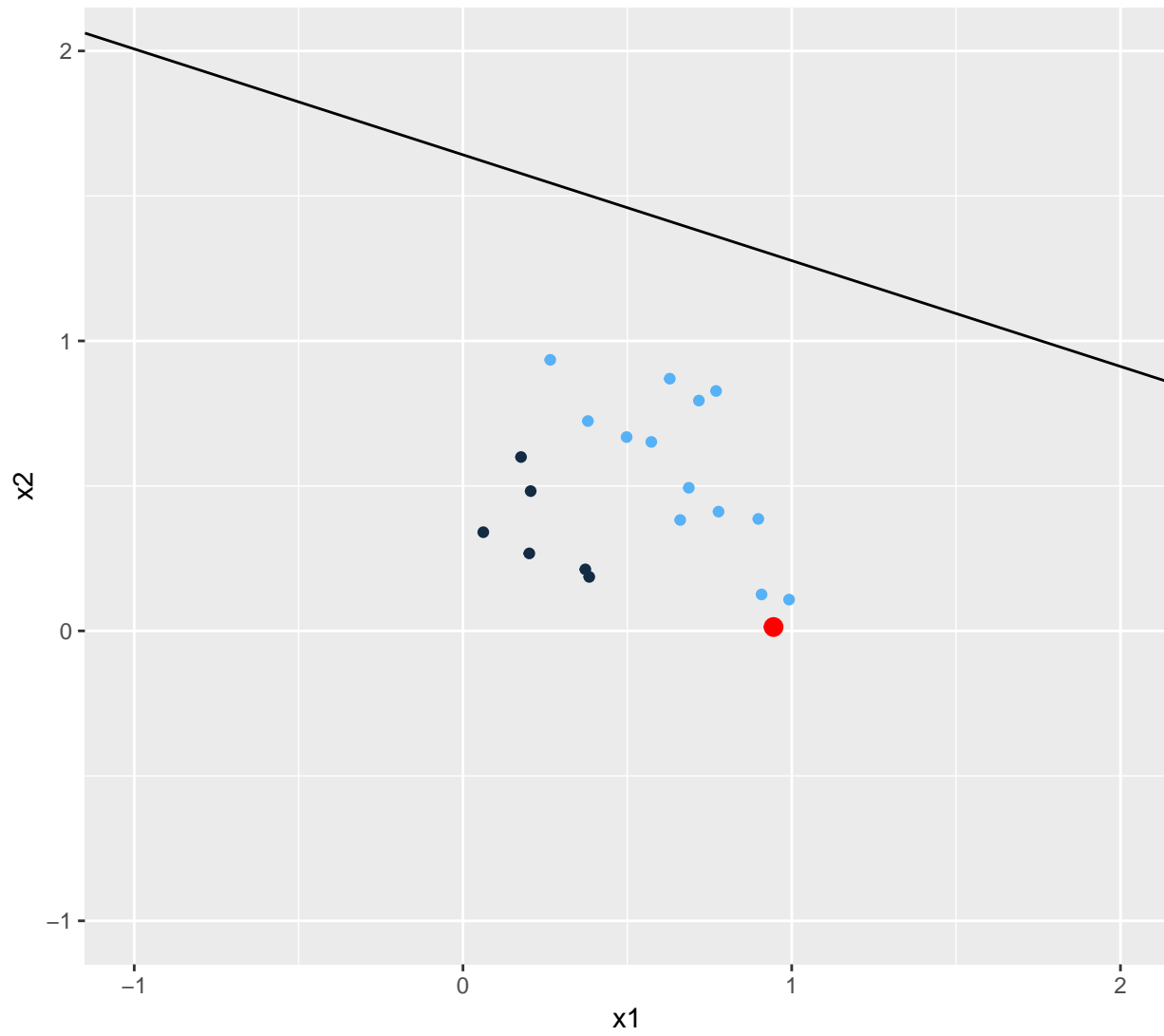
Perceptron Learning Example

Update: 13 $w: [1.389, 1.232]$, $w_0: -1.000$



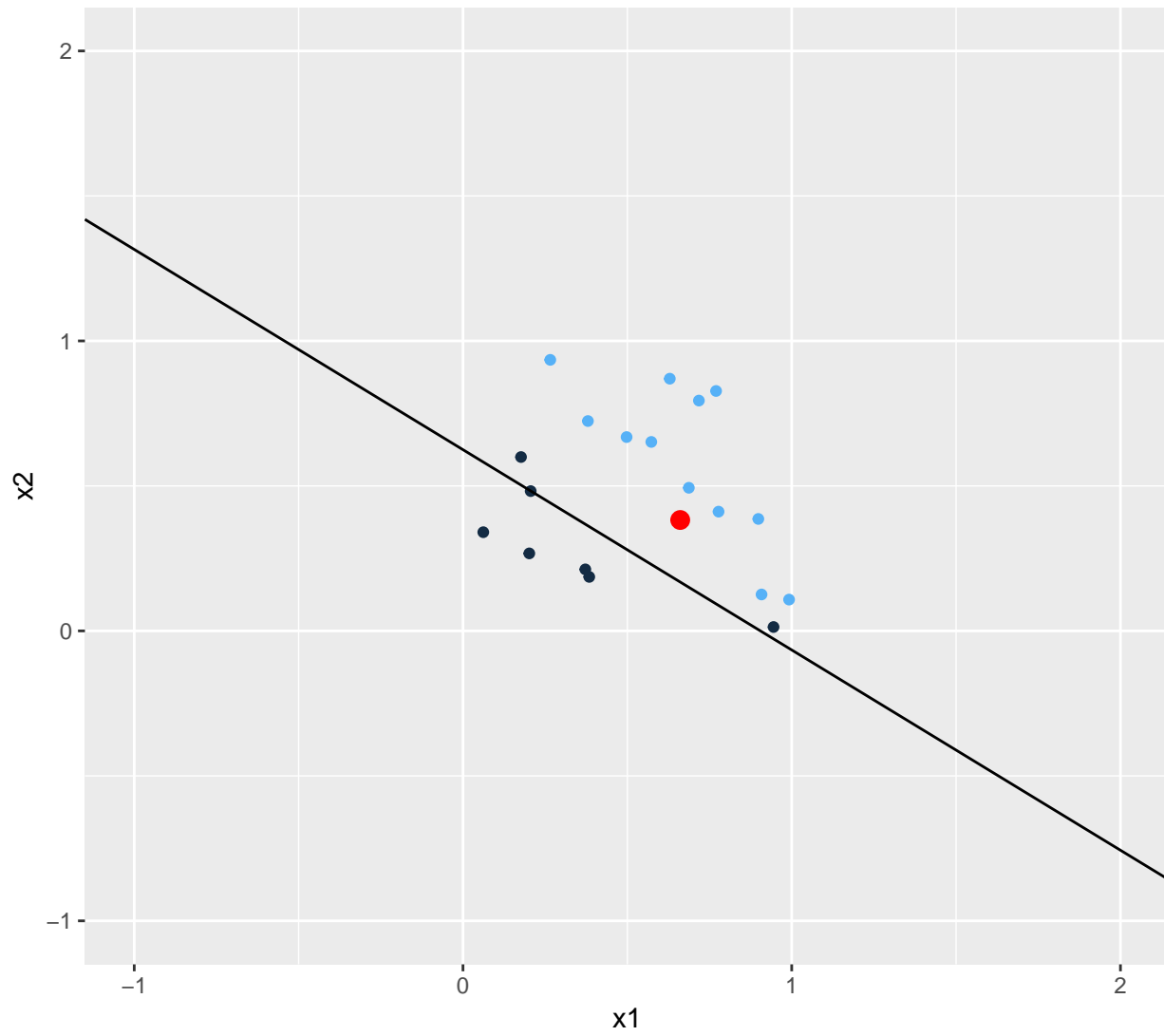
Perceptron Learning Example

Update: 14 $w: [0.445, 1.218]$, $w_0: -2.000$



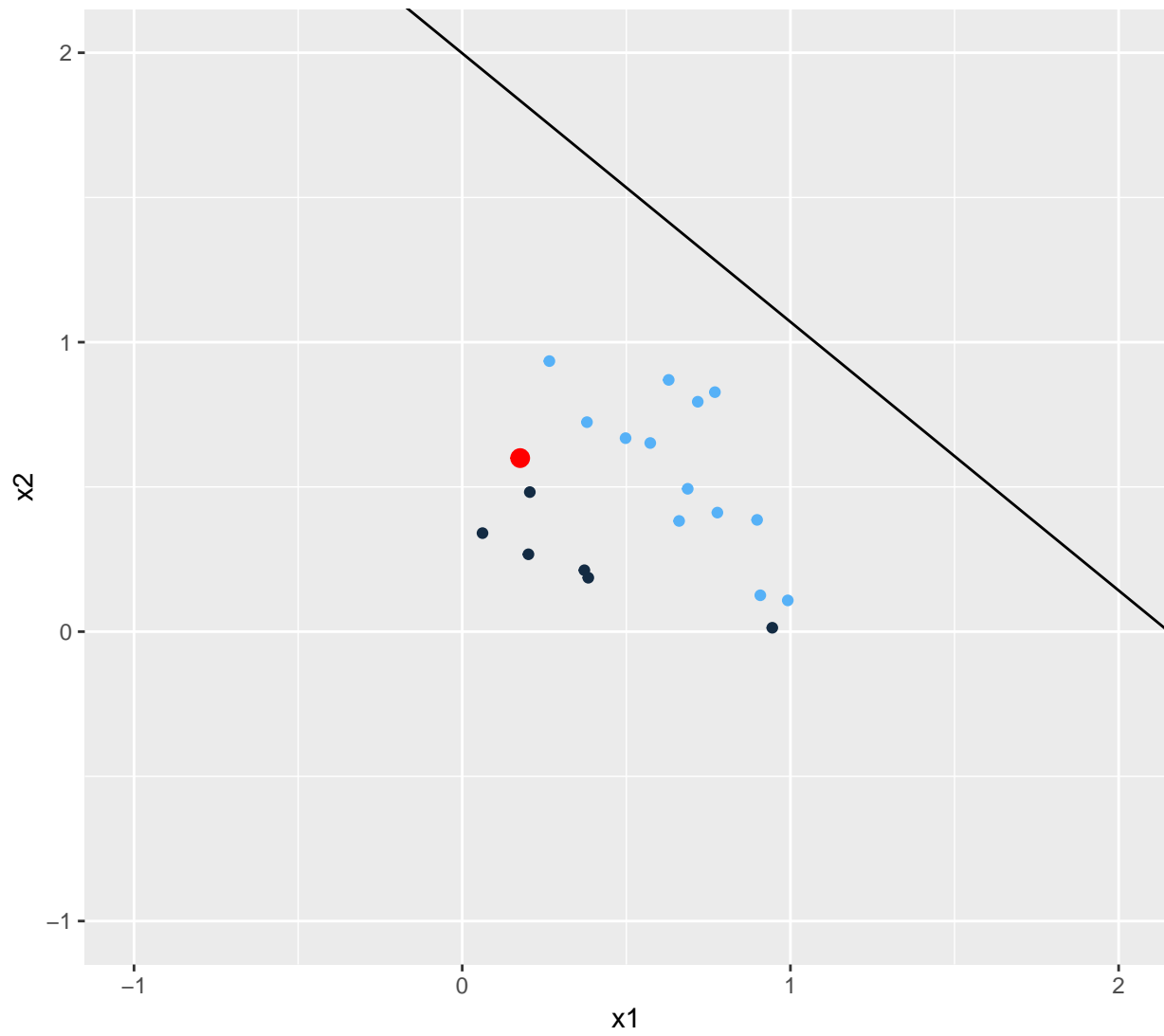
Perceptron Learning Example

Update: 15 $w: [1.105, 1.601]$, $w_0: -1.000$



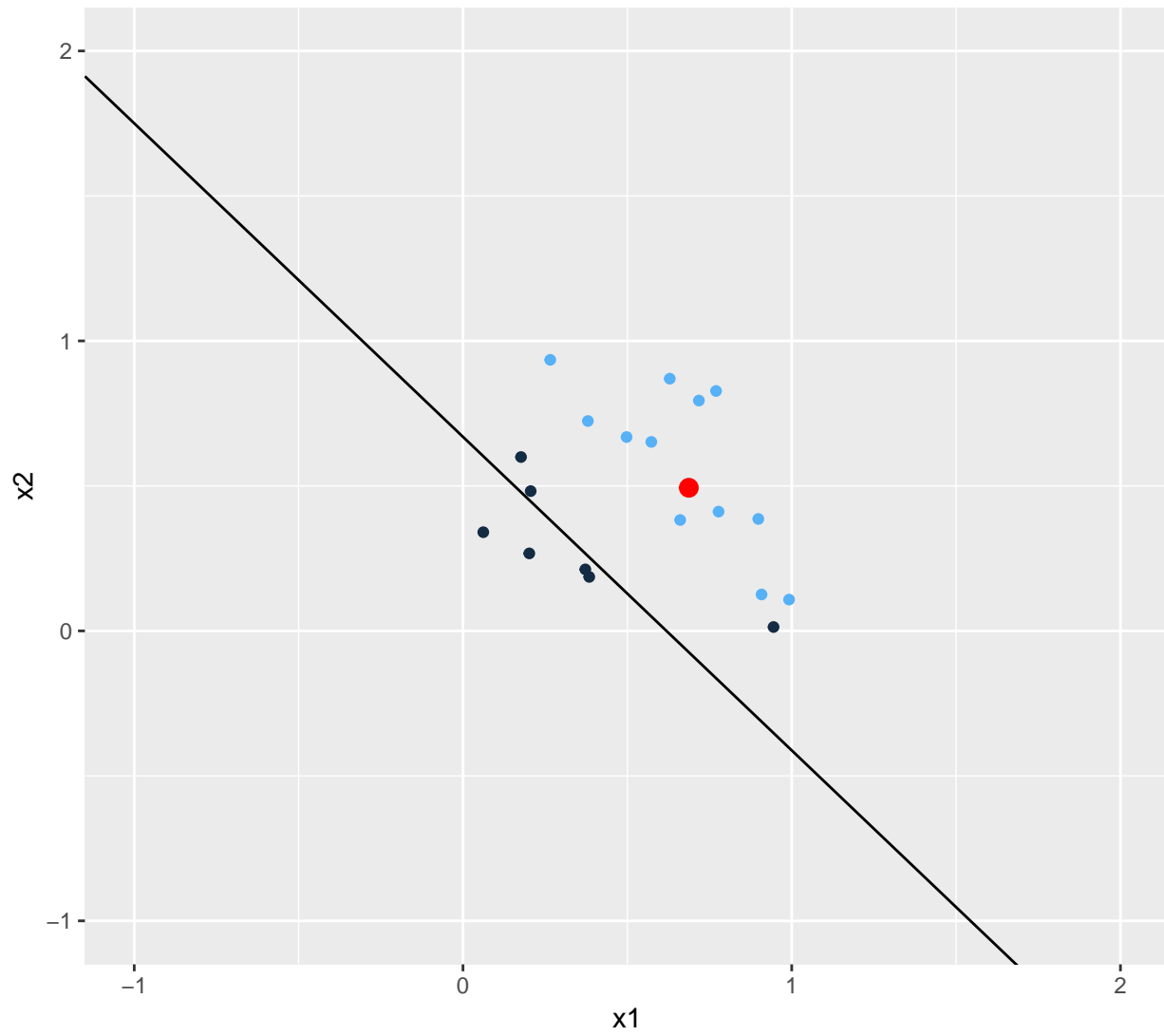
Perceptron Learning Example

Update: 16 $w: [0.929, 1.001]$, $w_0: -2.000$



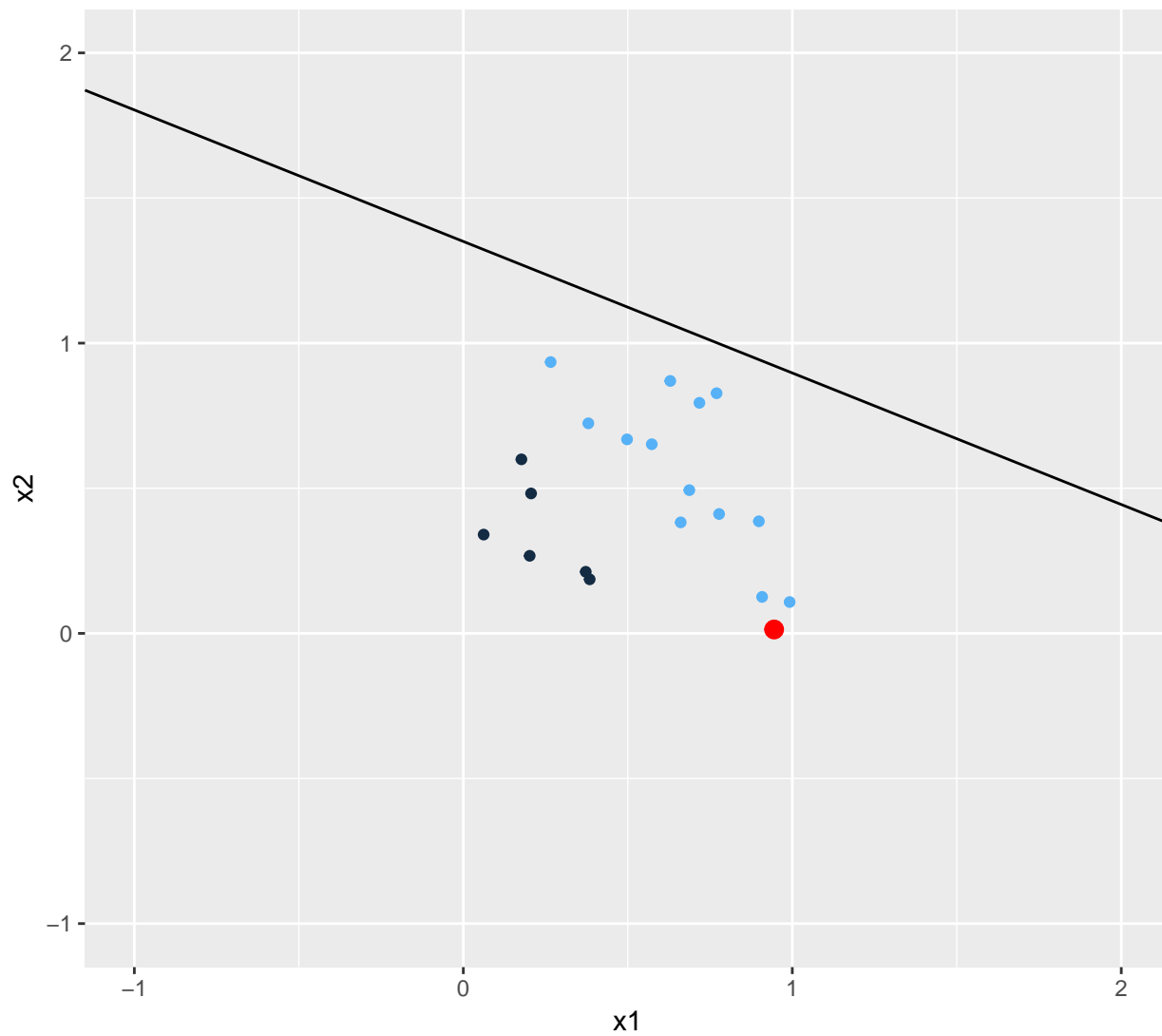
Perceptron Learning Example

Update: 17 $w: [1.616, 1.495]$, $w_0: -1.000$



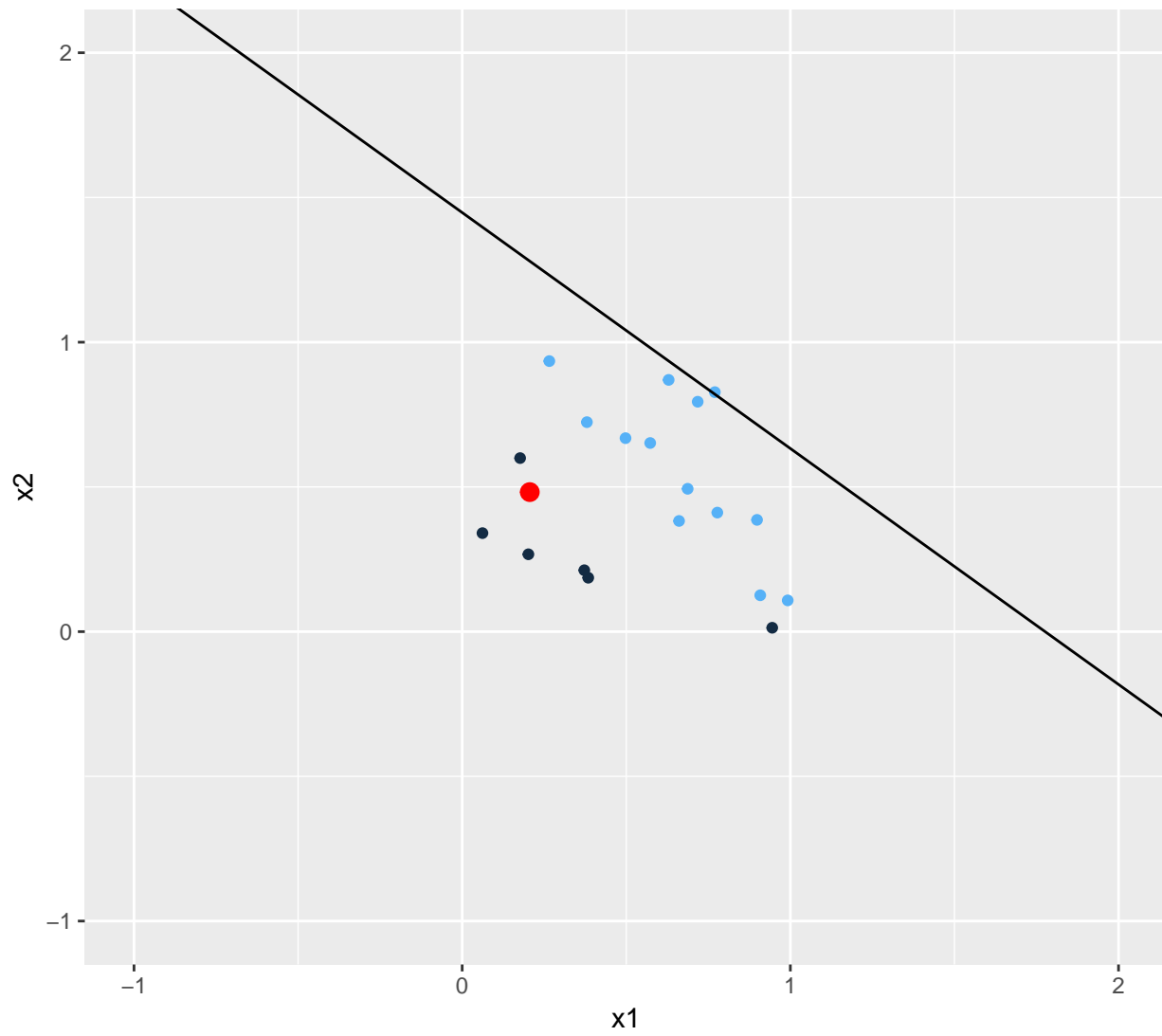
Perceptron Learning Example

Update: 18 $w: [0.671, 1.481]$, $w_0: -2.000$



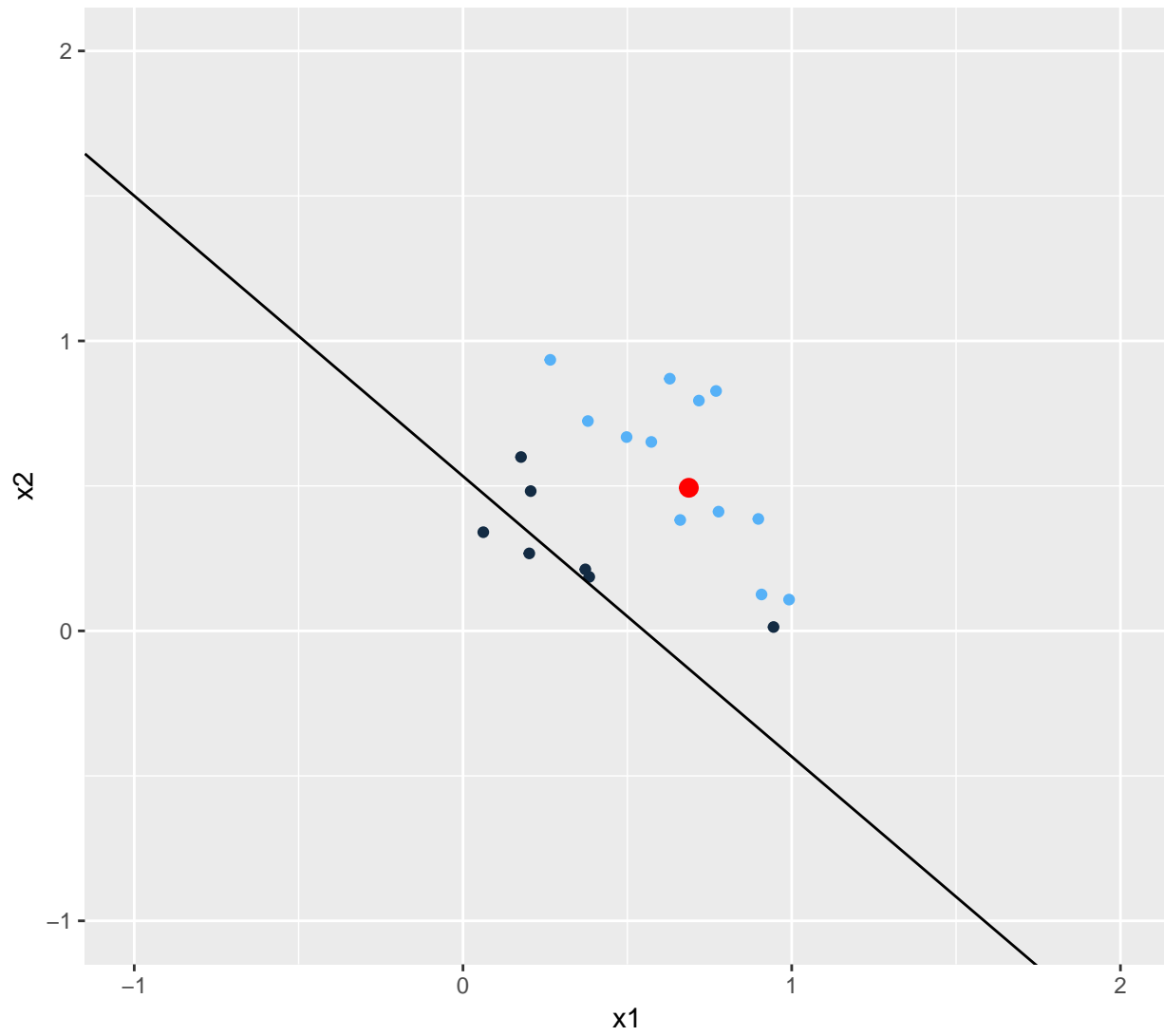
Perceptron Learning Example

Update: 20 $w: [1.126, 1.381]$, $w_0: -2.000$



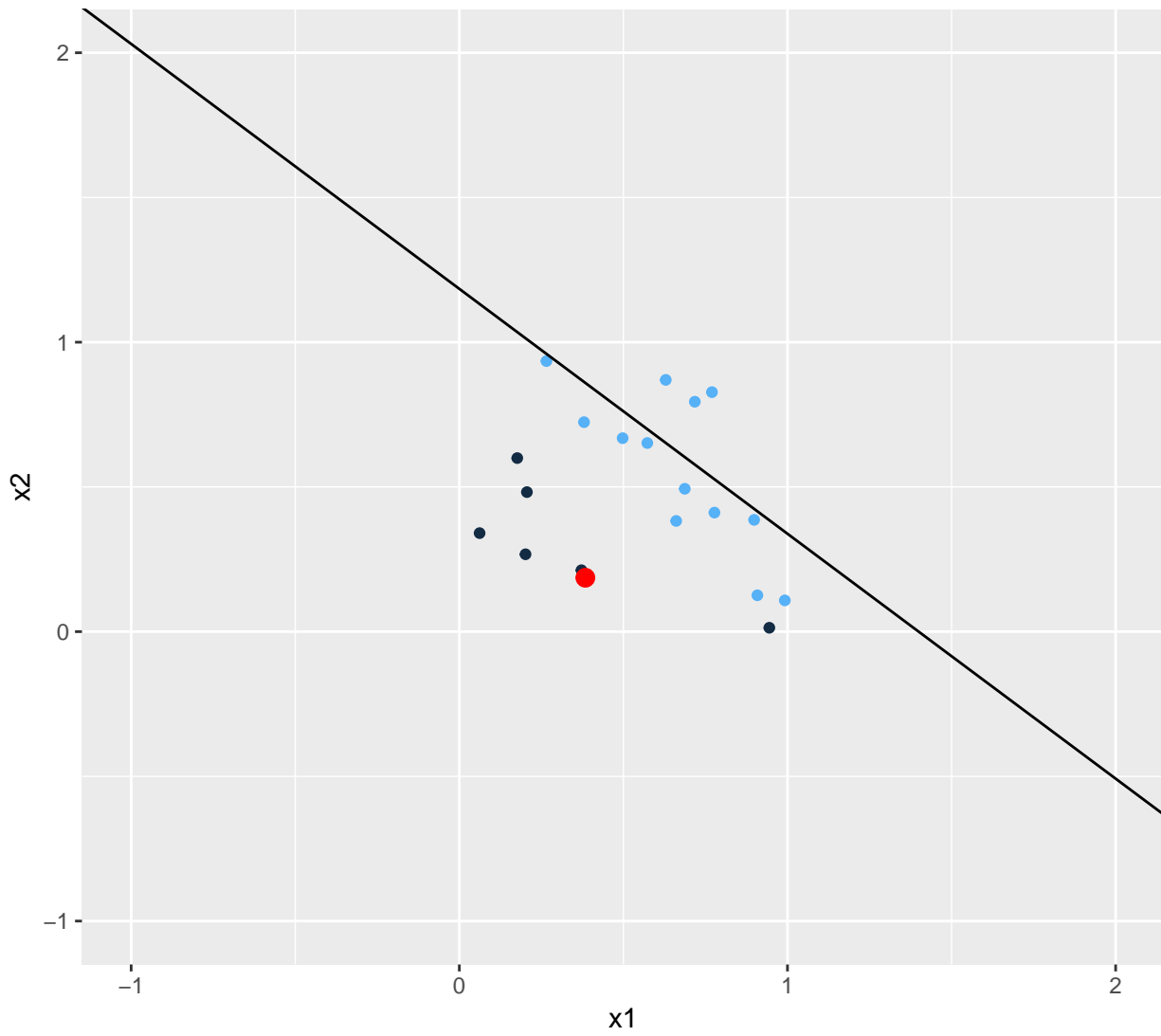
Perceptron Learning Example

Update: 21 $w: [1.813, 1.875]$, $w_0: -1.000$



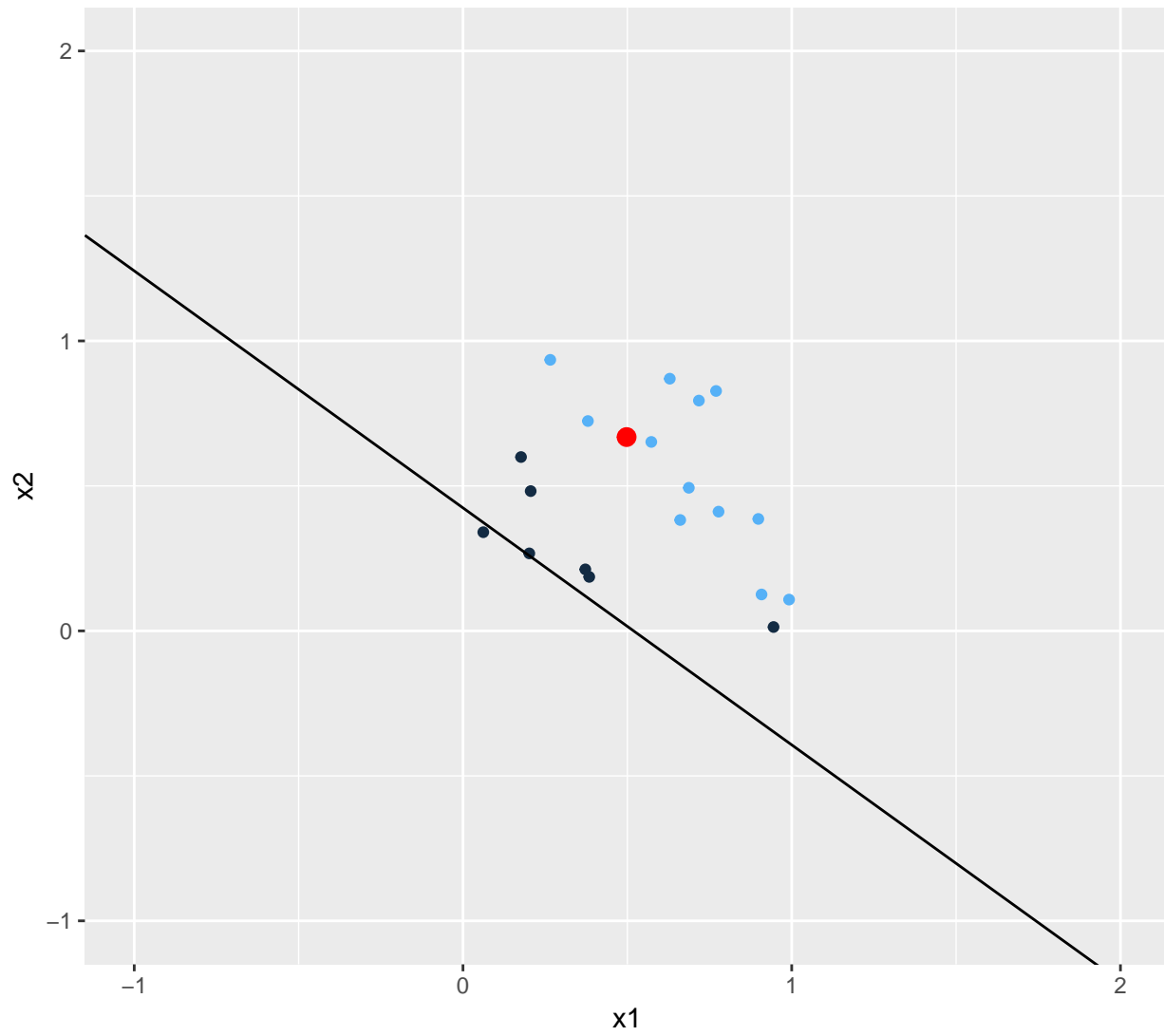
Perceptron Learning Example

Update: 22 $w: [1.429, 1.689]$, $w_0: -2.000$



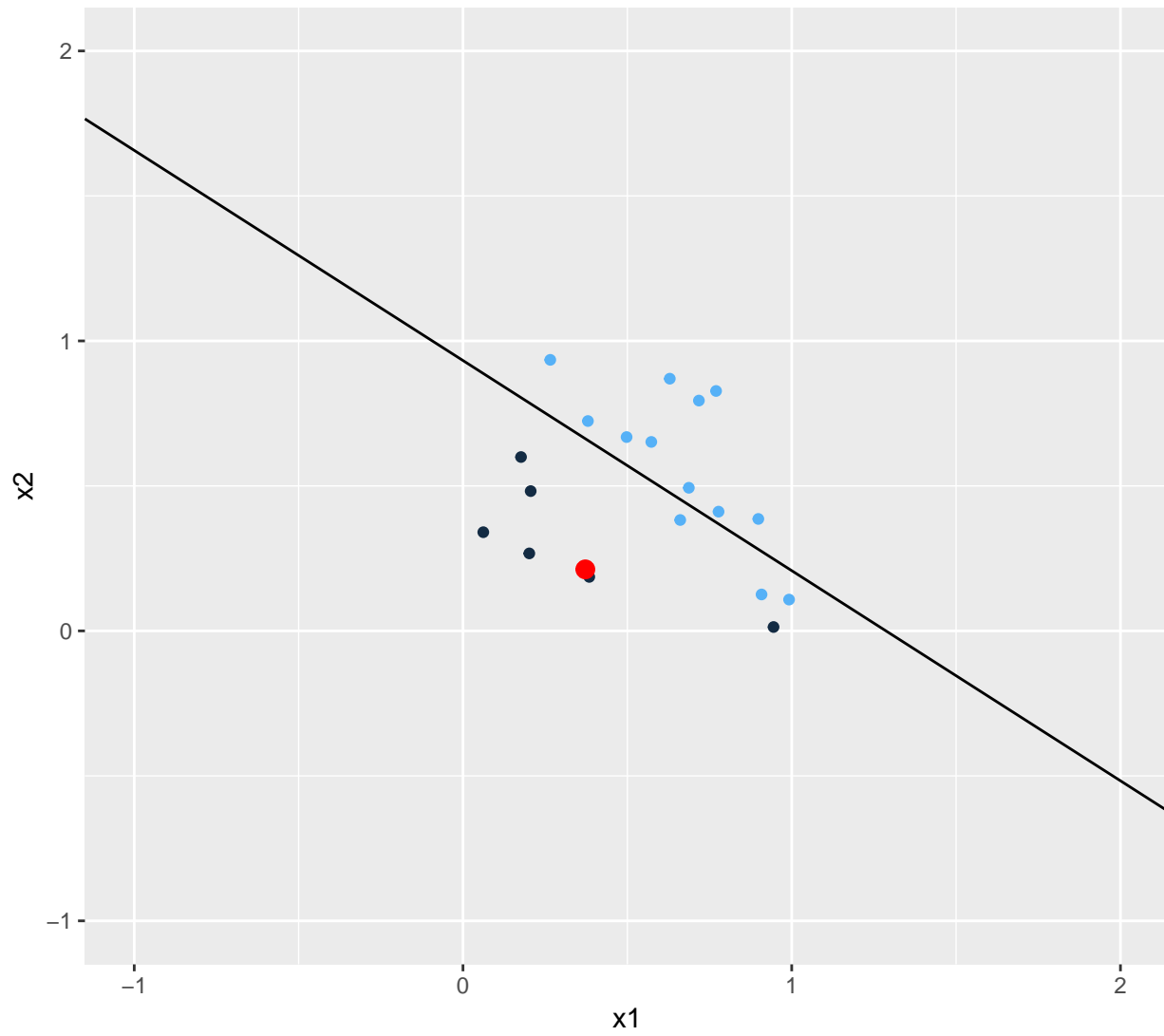
Perceptron Learning Example

Update: 23 $w: [1.927, 2.357]$, $w_0: -1.000$



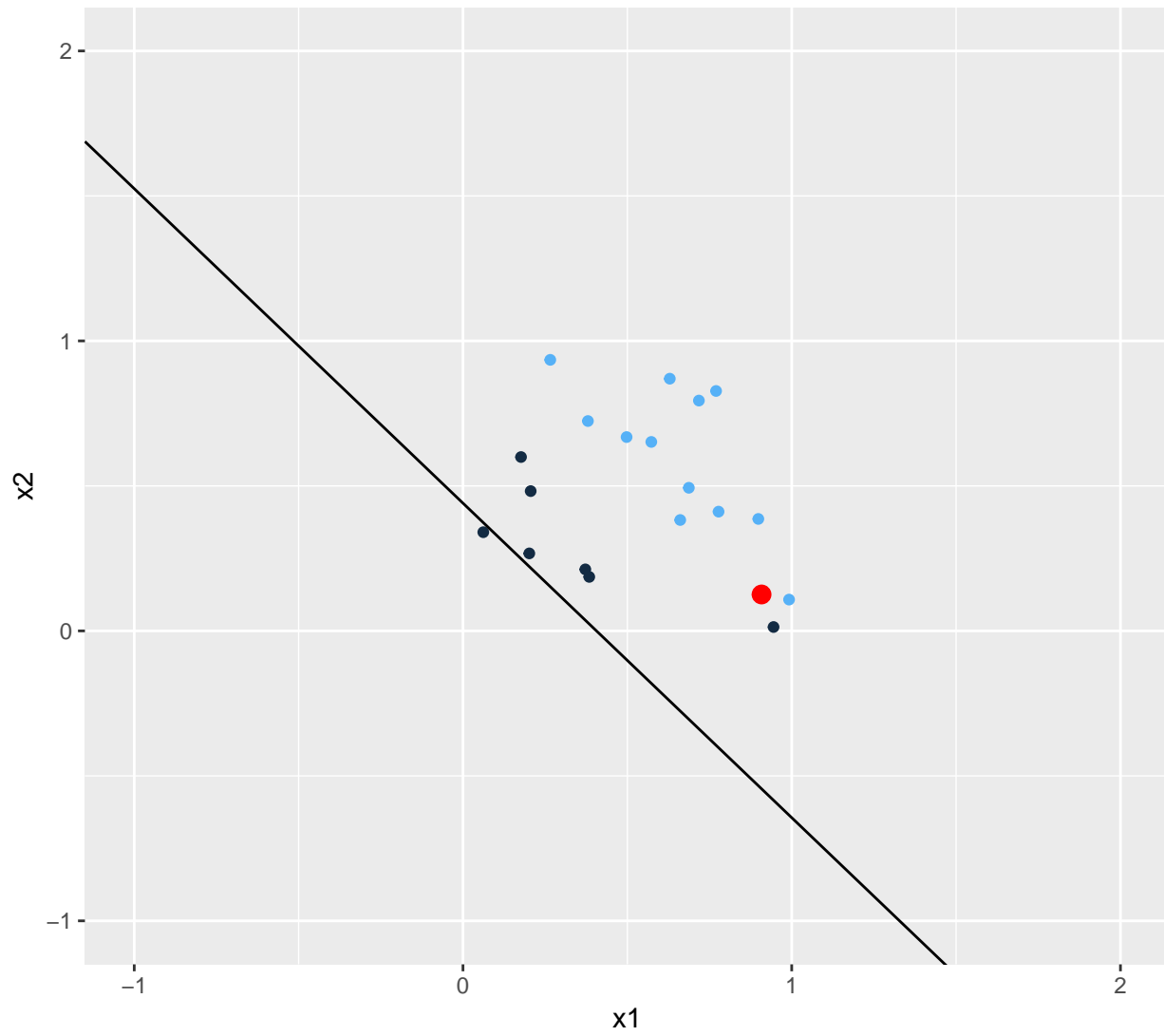
Perceptron Learning Example

Update: 24 $w: [1.554, 2.145]$, $w_0: -2.000$



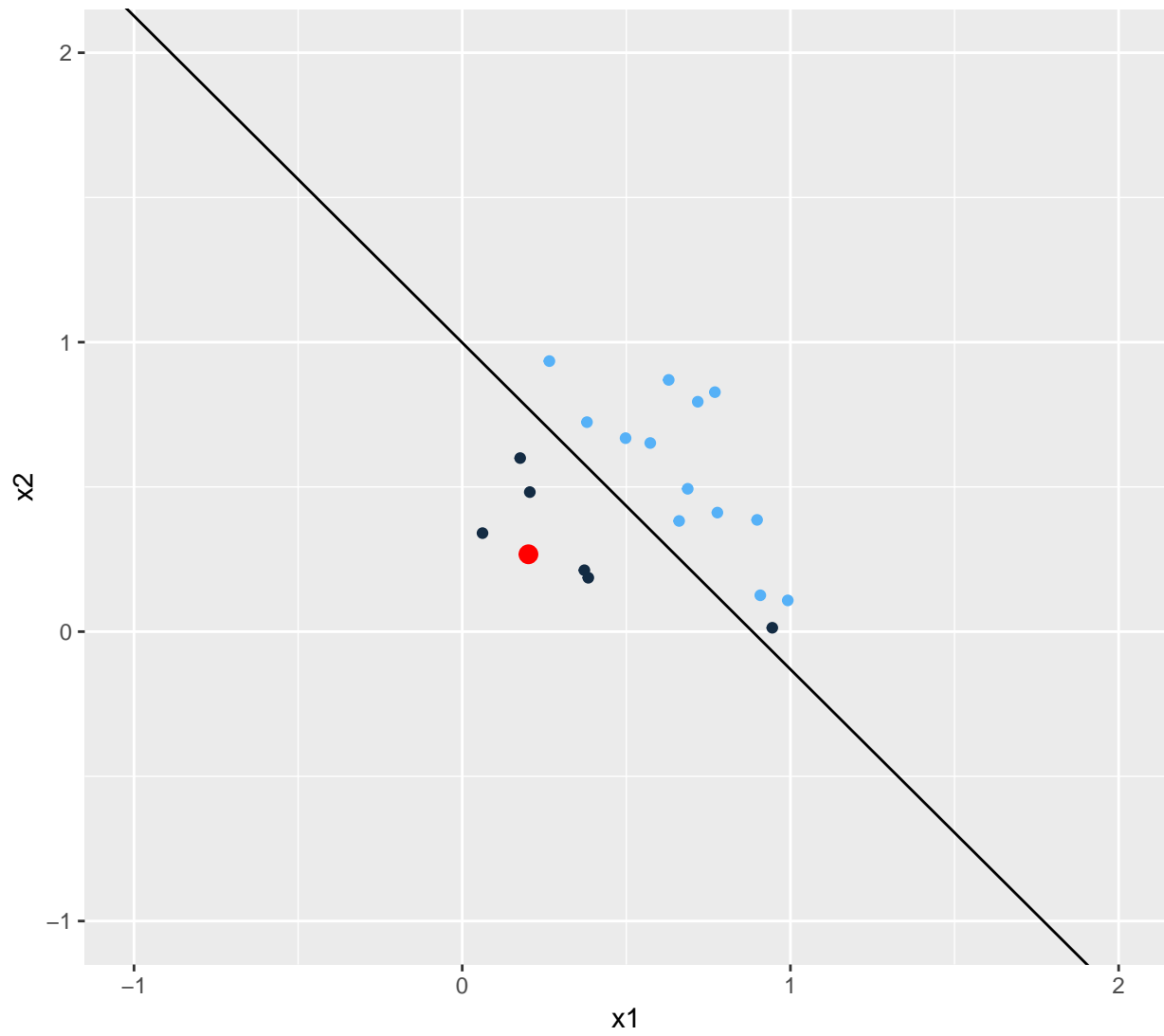
Perceptron Learning Example

Update: 25 $w: [2.463, 2.271]$, $w_0: -1.000$



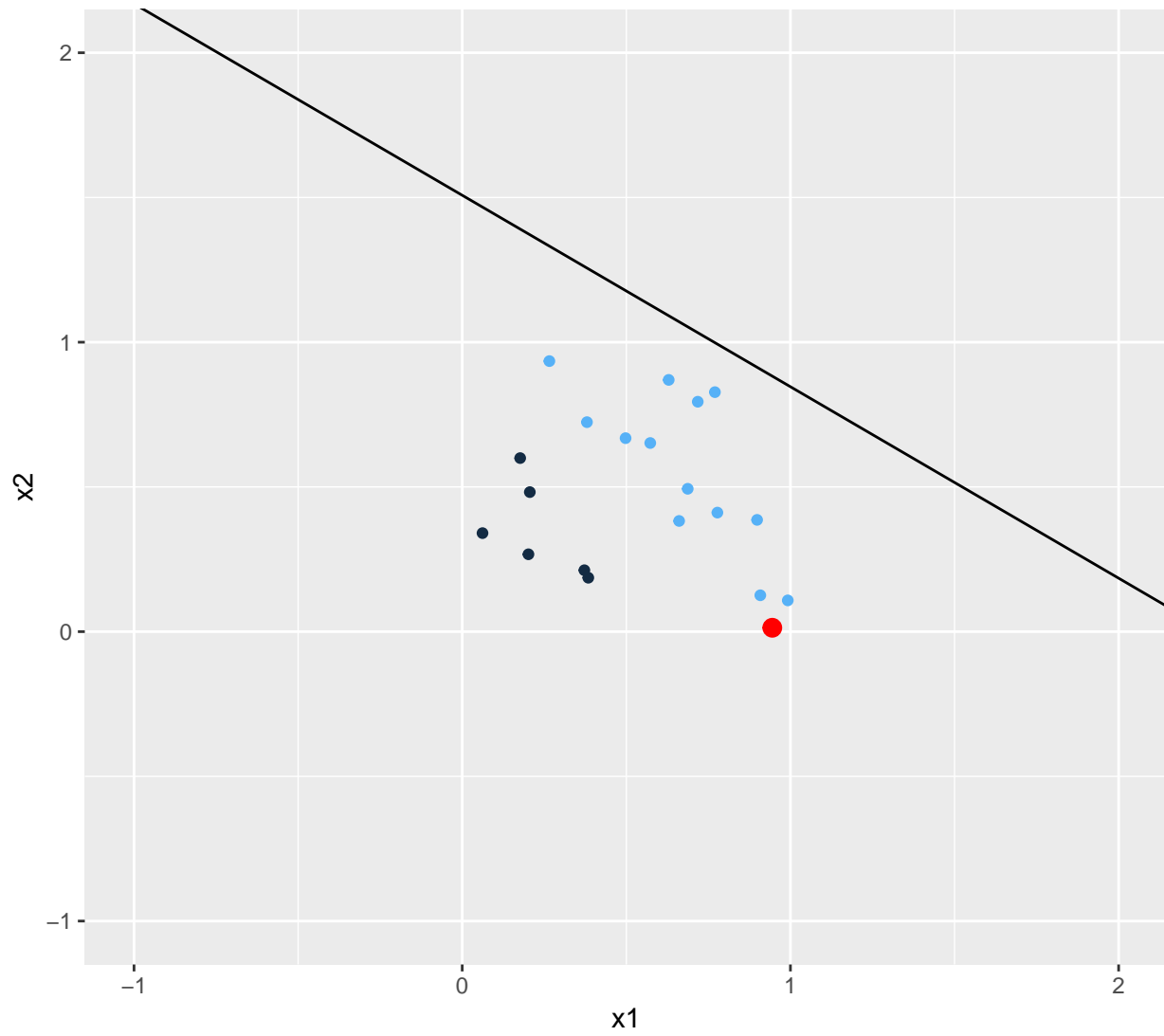
Perceptron Learning Example

Update: 26 $w: [2.261, 2.003]$, $w_0: -2.000$



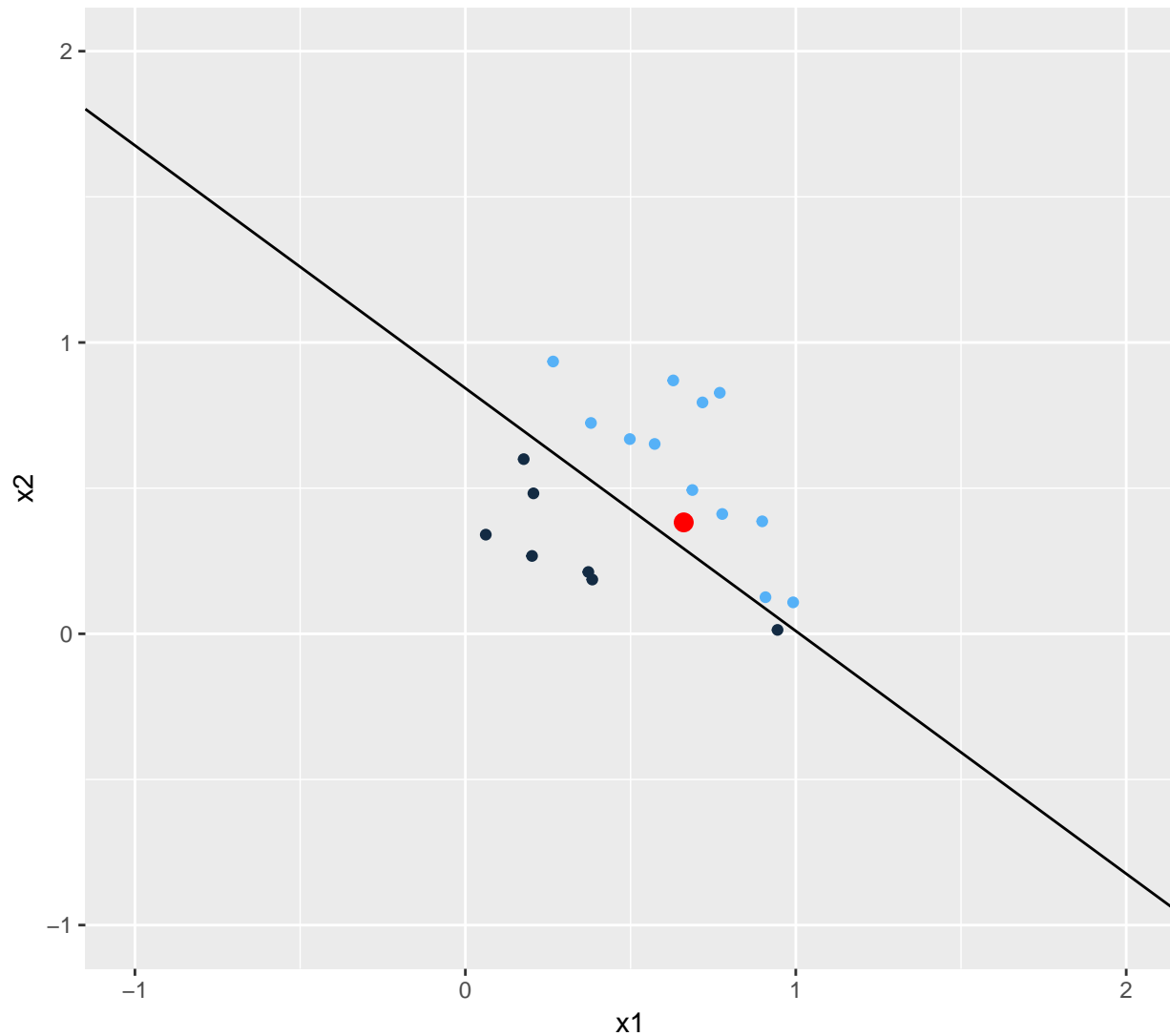
Perceptron Learning Example

Update: 27 $w: [1.316, 1.990]$, $w_0: -3.000$



Perceptron Learning Example

Update: 28 $w: [1.977, 2.372]$, $w_0: -2.000$



Weight as a combination of input vectors

- Recall perceptron learning rule:

$$\mathbf{w} \leftarrow \mathbf{w} + \delta y_i \mathbf{x}_i, \quad w_0 \leftarrow w_0 + \delta y_i$$

- If initial weights are zero, then at any step, the *weights are a linear combination of feature vectors of the examples*:

$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i, \quad w_0 = \sum_{i=1}^n \alpha_i y_i$$

where α_i is the sum of step sizes used for all updates based on example i .

- This is called the *dual representation* of the classifier.

- Even by the end of training, some example may have never participated in an update (they were always correct) so the corresponding $\alpha_i = 0$.

Examples used (bold) and not used (faint) in updates

Comment: Solutions are nonunique

Perceptron summary

- Perceptrons can be learned to fit linearly separable data, using a gradient descent rule.
- Blindingly fast
- Solutions are non-unique

Support Vector Machines

- Support vector machines (SVMs) for binary classification can be viewed as a way of training perceptrons
- Three main new ideas:
 - A optimization criterion (the “margin”) guarantees uniqueness and has theoretical advantages
 - Natural handling nonseparable data by allowing mistakes
 - An efficient way of operating in expanded feature spaces: “kernel trick”
- SVMs can also be used for multiclass classification and regression.

Returning to the non-uniqueness issue

- Consider a linearly separable binary classification data set
- There is an infinite number of hyperplanes that separate the classes:
- Which plane is best?
- For a given plane, for which points should we be most confident in the classification?

The margin, and linear SVMs

- For a given separating hyperplane, the *margin* is two times the (Euclidean) distance from the hyperplane to the nearest training example.
- Width of the “strip” around the decision boundary containing no training examples.
- A linear SVM is a perceptron for which we choose \mathbf{w}, w_0 so that margin is maximized

Distance to the decision boundary

- Suppose we have a decision boundary that separates the data.
- Let γ_i be the distance from instance \mathbf{x}_i to the decision boundary.
- How can we write γ_i in terms of $\mathbf{x}_i, y_i, \mathbf{w}, w_0$?

Distance to the decision boundary (II)

- \mathbf{w} is normal to the decision boundary. Thus, $\frac{\mathbf{w}}{\|\mathbf{w}\|}$ is the unit normal of the boundary.
- Vector from B to \mathbf{x}_i is $\gamma_i \frac{\mathbf{w}}{\|\mathbf{w}\|}$.
- B, the point on the boundary nearest \mathbf{x}_i , is $\mathbf{x}_i - \gamma_i \frac{\mathbf{w}}{\|\mathbf{w}\|}$.
- Since B is on the boundary,

$$\left(\mathbf{x}_i - \gamma_i \frac{\mathbf{w}}{\|\mathbf{w}\|} \right)^\top \mathbf{w} + w_0 = 0$$

- Solving for γ_i yields

$$\gamma_i = \frac{\mathbf{x}_i^\top \mathbf{w} + w_0}{\|\mathbf{w}\|}$$

The margin HTF Ch. 4.5, Ch 12

- The *margin of the hyperplane* is $2M$, where $M = \min_i y_i \gamma_i$
- The most direct statement of the problem of finding a maximum margin separating hyperplane is thus

$$\max_{\mathbf{w}, w_0} \min_i y_i \gamma_i \equiv \max_{\mathbf{w}, w_0} \min_i y_i \frac{\mathbf{x}_i^\top \mathbf{w} + w_0}{\|\mathbf{w}\|}$$

- This turns out to be inconvenient for optimization, however

Treating the γ_i as constraints

- From the definition of margin, we have:

$$M \leq y_i \gamma_i = y_i \frac{\mathbf{x}_i^\top \mathbf{w} + w_0}{\|\mathbf{w}\|} \quad \forall i$$

- This suggests:

$$\begin{array}{ll} \text{maximize } M & \text{with respect to } M, \mathbf{w}, w_0 \\ \text{subject to } M \leq y_i \frac{\mathbf{x}_i^\top \mathbf{w} + w_0}{\|\mathbf{w}\|} & \text{for all } i \end{array}$$

- Problems:
 - \mathbf{w} appears nonlinearly in the constraints.
 - This problem is underconstrained. If (\mathbf{w}, w_0, M) is an optimal solution, then so is $(\beta \mathbf{w}, \beta w_0, M)$ for any $\beta > 0$.

Adding a constraint

Let's add the constraint that $M = 1/\|\mathbf{w}\|$:

- This allows us to rewrite the objective function:
- This is really nice because the constraints are linear.

$$\begin{array}{l} \text{maximize } \frac{1}{\|\mathbf{w}\|} \quad \text{with respect to } \mathbf{w}, w_0 \\ \text{subject to } \frac{1}{\|\mathbf{w}\|} \leq y_i \frac{\mathbf{x}_i^\top \mathbf{w} + w_0}{\|\mathbf{w}\|} \text{ for all } i \end{array}$$

which is the same as

$$\begin{array}{l} \text{maximize } \frac{1}{\|\mathbf{w}\|} \quad \text{with respect to } \mathbf{w}, w_0 \\ \text{subject to } 1 \leq y_i (\mathbf{x}_i^\top \mathbf{w} + w_0) \text{ for all } i \end{array}$$

Final formulation

- Let's minimize $\|\mathbf{w}\|^2$ instead of maximizing $\frac{1}{\|\mathbf{w}\|}$. (Taking the square is a monotone transformation, as $\|\mathbf{w}\|$ is positive, so this doesn't change the optimal solution.)
- This gets us to:

$$\begin{array}{l} \text{minimize } \|\mathbf{w}\|^2 \text{ w.r.t. } \mathbf{w}, w_0 \\ \text{subject to } y_i (\mathbf{x}_i^\top \mathbf{w} + w_0) \geq 1 \end{array}$$

- This we can solve! How?
 - It is a convex *quadratic programming* (QP) problem—a standard type of optimization problem for which many efficient packages are available.

Example

We have a solution, but no “support vectors” yet...

What are “Support Vectors”?

$$\begin{array}{l} \text{minimize } \frac{1}{2} \|\mathbf{w}\|^2 \text{ w.r.t. } \mathbf{w}, w_0 \quad \text{subject to } y_i (\mathbf{x}_i^\top \mathbf{w} + w_0) \geq 1 \end{array}$$

- Turns out (HTF Ch. 4.5.2) we can write:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i, \quad \text{where } \alpha_i \geq 0$$

- As for the perceptron with zero initial weights, the optimal solution for \mathbf{w} and w_0 is a linear combination of the \mathbf{x}_i .
- The output is therefore:

$$h_{\mathbf{w}, w_0}(\mathbf{x}) = \text{sign} \left(\sum_{i=1}^n \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + w_0 \right)$$

- Output depends on weighted dot product of input vector with training examples

Solving “the dual”

- We can actually solve directly for the α_i (again see HTF Ch. 4.5.2):

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

with constraints: $\alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$

- This is also a QP

The support vectors

- Suppose we find optimal $\boldsymbol{\alpha}$ s (e.g., using a standard QP package)
- The α_i will be > 0 only for the points for which $y_i(\mathbf{x}_i^T \mathbf{w} + w_0) = 1$
- These are the points lying on the edge of the margin, and they are called *support vectors*, because they define the decision boundary
- The output of the classifier for query point \mathbf{x} is computed as:

$$\text{sgn} \left[\left(\sum_{i=1}^n \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) \right) + w_0 \right]$$

Hence, the output is determined by computing the *dot product of the point with the support vectors!*

Example

Support vectors are in bold

But why all this work?

- SVMs are a state-of-the-art for classification when you don't need probability estimates
- Intuitively, the large-margin property makes sense. Theory backs this up.
- SVMs offer “off-the-shelf” *non*-linear classification without having to do explicit feature construction, as we will see.

Soft margin classifiers

- Recall that in the linearly separable case, we compute the solution to the following optimization problem:

$$\begin{array}{ll} \min & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{s.t.} & y_i (\mathbf{x}_i^T \mathbf{w} + w_0) \geq 1 \end{array} \quad \text{w.r.t. } \mathbf{w}, w_0$$

- What if we can't satisfy the constraints?

Soft margin classifiers

- To allow misclassifications, we relax the constraints to:

$$y_i(\mathbf{x}_i^T \mathbf{w} + w_0) \geq 1 - \xi_i$$

- If $\xi_i \in (0, 1)$, the data point is within the margin
- If $\xi_i \geq 1$, then the data point is misclassified
- We define the *soft error* as $\sum_i \xi_i$; each ξ_i is a *slack variable*

Problem formulation with soft errors

- Instead of:

$$\begin{array}{ll} \min & \frac{1}{2} \|\mathbf{w}\|^2 \text{ w.r.t. } \mathbf{w}, w_0 \\ \text{s.t.} & y_i(\mathbf{x}_i^T \mathbf{w} + w_0) \geq 1 \end{array}$$

we want to solve:

$$\begin{array}{ll} \min & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i \text{ w.r.t. } \mathbf{w}, w_0, \xi_i \\ \text{s.t.} & y_i(\mathbf{x}_i^T \mathbf{w} + w_0) \geq 1 - \xi_i, \xi_i \geq 0 \end{array}$$

- Note that soft errors include points that are misclassified, as well as points within the margin
- There is a linear penalty for both categories
- The choice of the *constant C controls boundary-fitting*

A built-in boundary-fitting knob

$$\begin{array}{ll} \min & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i \\ \text{w.r.t.} & \mathbf{w}, w_0, \xi_i \\ \text{s.t.} & y_i(\mathbf{x}_i^T \mathbf{w} + w_0) \geq 1 - \xi_i, \xi_i \geq 0 \end{array}$$

- If C is 0, there is no penalty for soft errors, so the focus is on maximizing the margin, even if this means more mistakes
- If C is very large, the emphasis on the soft errors will decrease the margin, if this helps to classify more examples correctly.
- Internal cross-validation is a good way to choose C appropriately

Dual form for the soft margin problem

- Like before, we can formulate a “dual” problem that identifies the support vectors:

Primal form:

$$\min \quad \|\mathbf{w}\|^2 + C \sum_i \xi_i \text{ w.r.t. } \mathbf{w}, w_0, \xi_i$$

Primal form:

s.t. $y_i(\mathbf{x}_i^T \mathbf{w} + w_0) \geq (1 - \xi_i), \xi_i \geq 0$

Dual form:

max $\sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n y_i y_j \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j)$ w.r.t. α_i
s.t. $0 \leq \alpha_i \leq C, \sum_{i=1}^n \alpha_i y_i = 0$

- All the previously described machinery can be used to solve this problem