Principles of Machine Learning

Week 10: Linear Classifiers with Hard and Soft Thresholds

Learning Objectives

After successfully completing this topic, you will be able to …

- Explain the drawbacks of using linear regression directly for classification problems
- Describe and implement approaches to improve on this: Linear Perceptron Classifiers and Logistic Regression
- Discuss their characteristics and limitations

Principles of Machine Learning

Classification

Classification

- Is unknown sample X or O ?

  Goal: Find model that correctly classifies a new sample, $\mathbf{x}_i$

- Training Data: $(\boldsymbol{x}_i : y_i)$

  s$_1$: (11, 3 : +1) **O**

  s$_2$: (3, 10 : +1) **O**

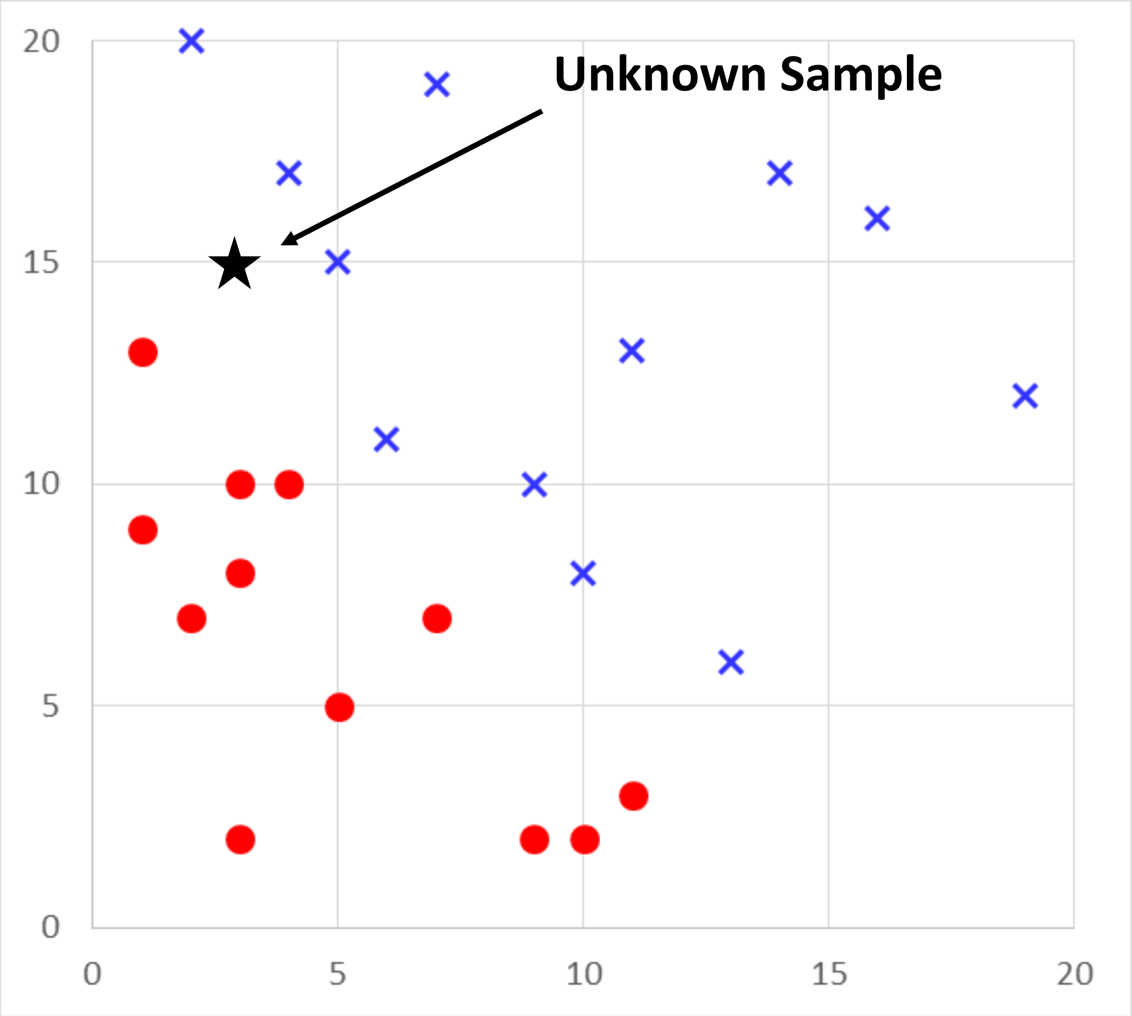
  s$_3$: (14, 17 : -1) **X**

  s$_4$: (16, 16 : -1) **X**

  …

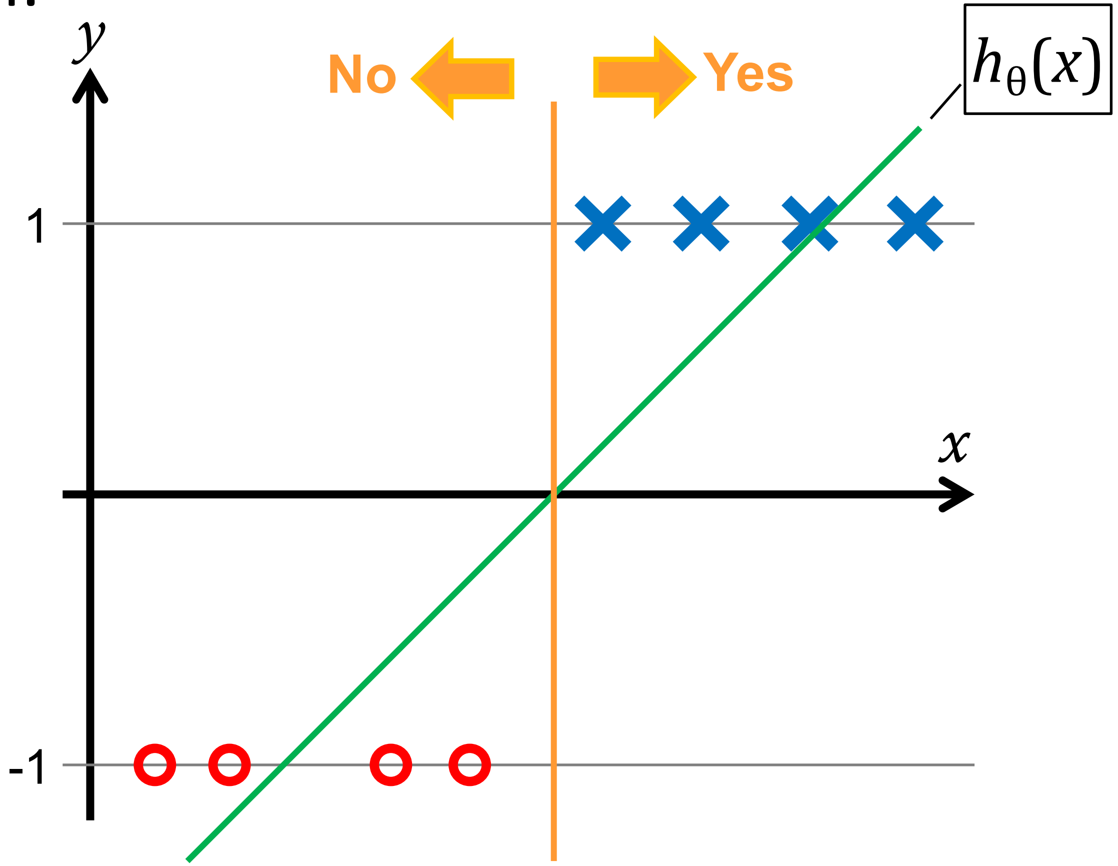
- Unknown Sample s = (3, 15: ?)

  What class does this belong to?



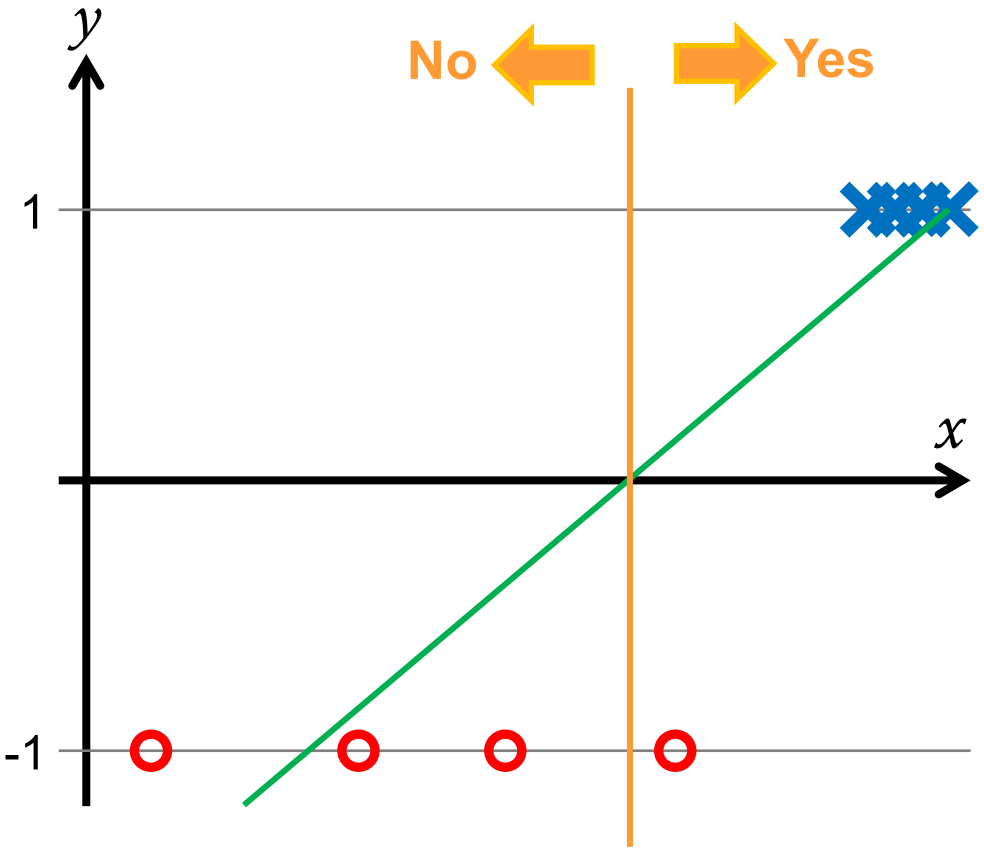
From Linear Regression to Classification (1)

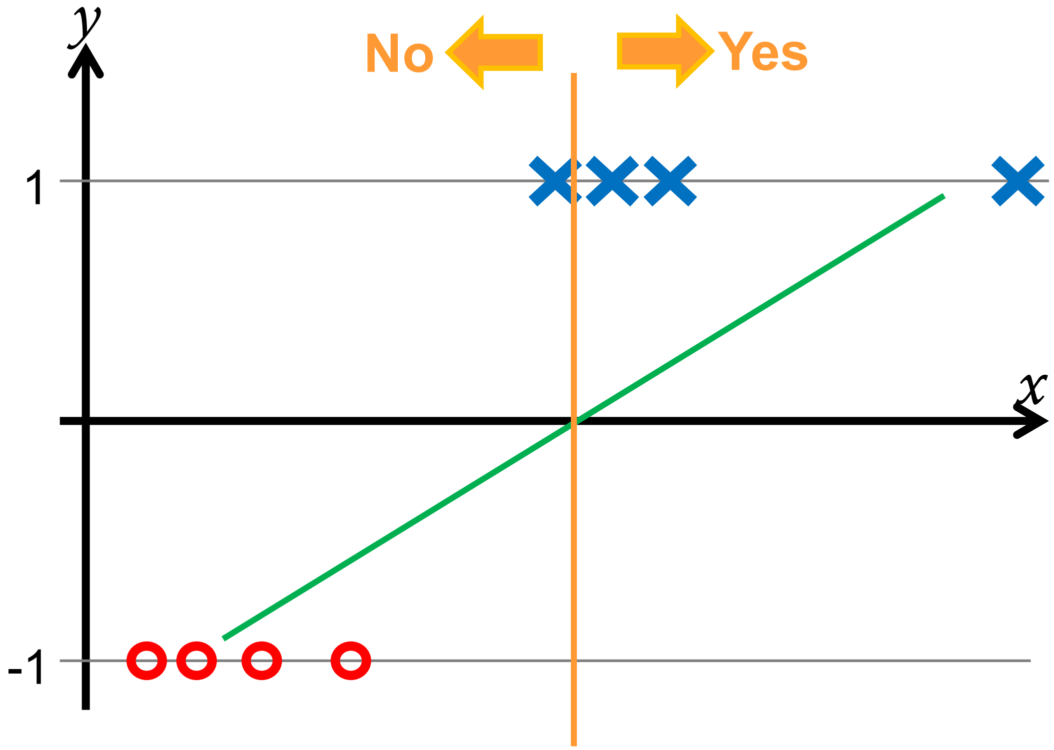
- Suppose we want to build a Yes/No **classification** model
- We know how to do linear regression:
  - Could encode No as -1 and Yes as 1
  - Perform linear regression to find $h_\theta(x)$
  - Threshold predictions:
    $h_\theta(x) >= 0 \Rightarrow$ Yes
    $h_\theta(x) < 0 \Rightarrow$ No



- Unfortunately, using Linear Regression directly doesn't always work very well…

From Linear Regression to Classification (2)





From Linear Regression to Classification (3)

- The reason for the problem:

  Linear regression parameters are found without taking into account that there are only two 'real' output values, -1/1

  A threshold is applied to outputs to convert them to -1/1, but only **after** the regression hyperplane is learned

- The solution:

  Incorporate the threshold in the objective function, so that it is taken into account in the cost function and therefore becomes part of the objective to be learned

  Could use a **hard** or **soft** threshold:

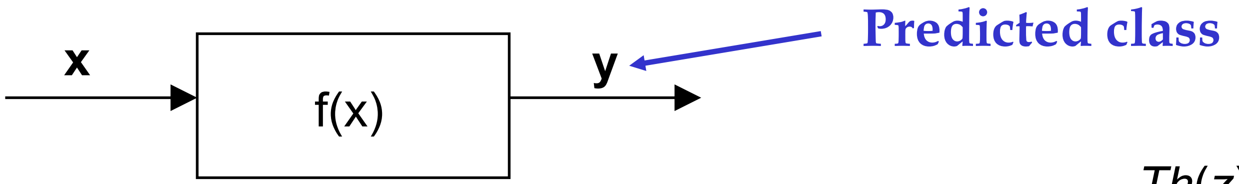
  lead to Linear Perceptron Classifiers & Logistic Regression, respectively…

Principles of Machine Learning

Linear Perceptron Classifier

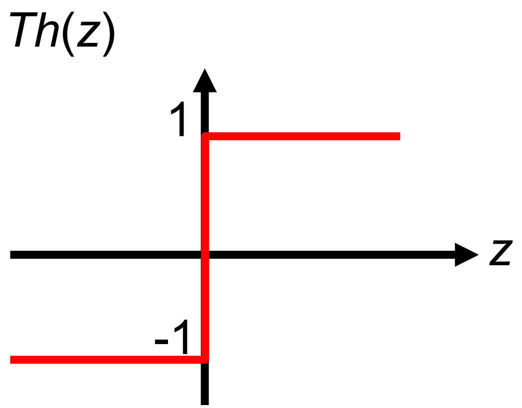
Linear Perceptron Classifier

Goal: Given sample ***x***,
find function f(***x***) that correctly classifies it



$$f(\boldsymbol{x}) = \boldsymbol{\theta} \cdot \boldsymbol{x}$$

$f(x) < 0$: sample is **X**
$f(x) \geq 0$: sample is **O**



As before, add a dummy input variable $x_0 = 1$

To classify a new sample, **x**:

- Calculate the dot product of the weight vector, **θ**, and **x**
- Apply a **hard threshold**:

$Threshold(z) = 1$ if $z \geq 0$, -1 otherwise

Building a Linear Perceptron Classifier

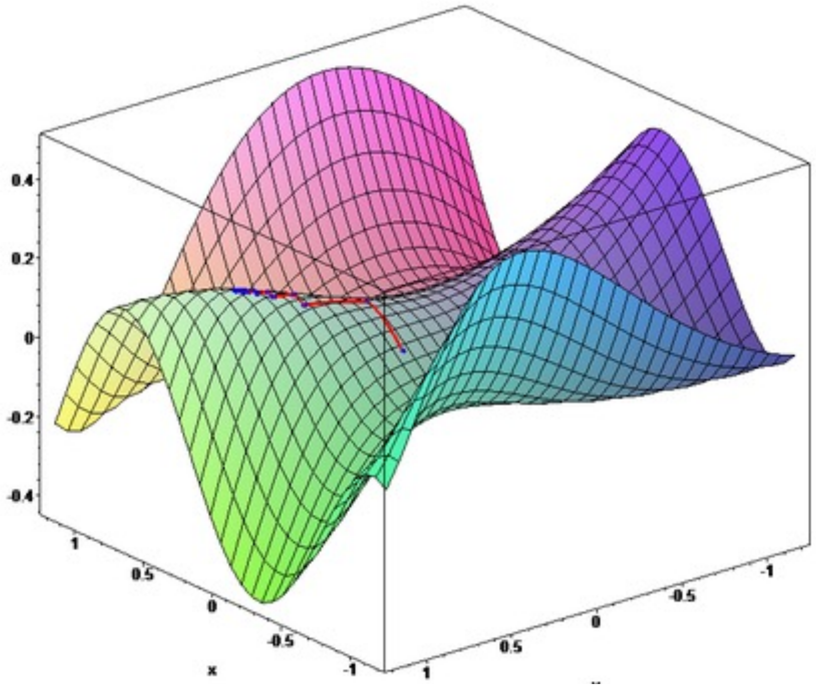
- To build a perceptron classifier:

  $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = Threshold(f(\boldsymbol{x})) = Threshold(\boldsymbol{\theta} \,.\, \boldsymbol{x})$

  – Find a function $f(\boldsymbol{x})$ (i.e. find values for θ) that correctly classifies training data when put through threshold function

- Can use Gradient Descent to find values for $\boldsymbol{\theta}$

  – But $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ is not differentiable, because of hard threshold function …



Building a Linear Perceptron Classifier: Perceptron Learning Rule

- Because $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ is not differentiable, cannot use the exact same approach that we used for Linear Regression
- Instead need **Perceptron Learning Rule** to update $\boldsymbol{\theta}$ values
  - Also called other names
  - Usually used with Stochastic Gradient Descent

$$\theta_j \leftarrow \theta_j + \alpha(y - h_{\boldsymbol{\theta}}(\boldsymbol{x})) \,.\, x_j \qquad \text{for a single training case } (\boldsymbol{x}, y)$$

- Notes:
  - For Stochastic GD, picking only one sample, so N=1
  - Converges to a solution, provided data linearly separable

How does Perceptron Learning Rule Compare to Linear Regression Update Rule?

- **Perceptron Learning Rule:**

  $\theta_j \leftarrow \theta_j + \alpha(y - h_{\boldsymbol{\theta}}(\boldsymbol{x}))\,.\,x_j$ for a single example $(\boldsymbol{x}, y)$

- Linear Regression update rule from last topic:

$$\theta_j \leftarrow \theta_j - \alpha(h_{\boldsymbol{\theta}}(\boldsymbol{x}) - y)x_j$$

Set N=1, merge equations ...

Building a Linear Perceptron Classifier: Perceptron Learning Rule

- Looks just like MLR update rule (previous topic), but behaviour is different as *h*() and *y* are -1 or 1:
  - Output correct ($h_{\boldsymbol{\theta}}(\boldsymbol{x}) = y$): weights unchanged
  - $y = 1$ but $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = -1$: increase θ_i if x_i positive
  - $y = -1$ but $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = 1$: decrease θ_i if x_i positive

$$\theta_j \leftarrow \theta_j + \alpha(y - h_{\boldsymbol{\theta}}(\boldsymbol{x})) \,.\, x_j$$

- To guarantee convergence, need to **decay** α in proportion to 1/t
  - For Perceptron, must use smaller values of α in each successive iteration
  - Unlike Linear Regression update rule: fixed α because gradients get smaller
  - t is iteration (“time step”)

Putting it Together: Linear Perceptron Classifier Using Stochastic GD with Perceptron Learning Rule

Linear Classifier Learning Algorithm:

initialise $\boldsymbol{\theta}$ to any set of valid initial values

Initialise α_0 to some step size

repeat t=1:T times, or until convergence if earlier:

  select a training example at random

  $\alpha \leftarrow \alpha_0 / t$

  simultaneously foreach θ_j in $\boldsymbol{\theta}$ do:

    $\theta_j \leftarrow \theta_j + \alpha(y - h_{\boldsymbol{\theta}}(\boldsymbol{x}))\,.\,x_j$

Linear Perceptron Classifier: Result

- Example of a successful classifier
- Data are linearly separable: can perfectly classify them
- To classify unknown sample

  x = (3, 15: ?):

  f(x) = 3 + 15 – 16

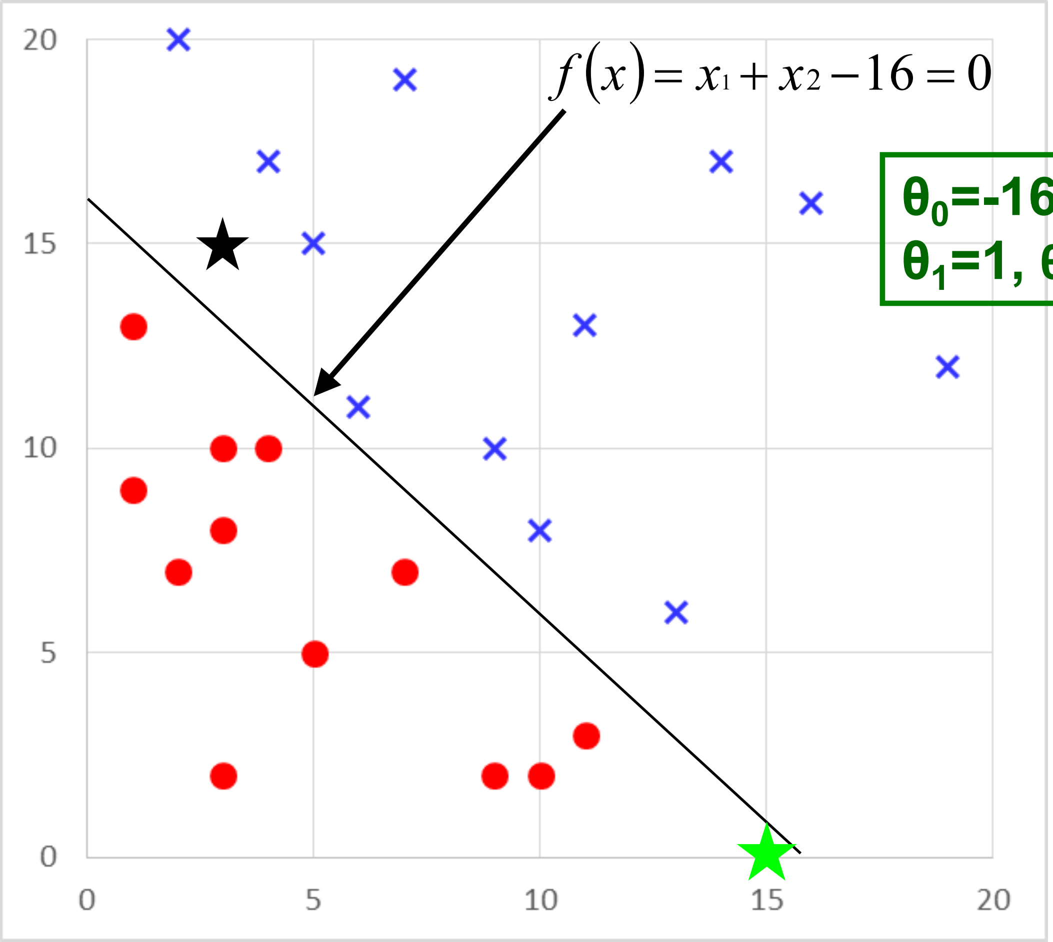
  f(x) = 2

  f(x) > 0 ⇒ sample is **X**
- Another unknown sample

  Green star: x= (15,0)

  f(x) = -1 ⇒ sample is O
- However, many functions could separate this data….

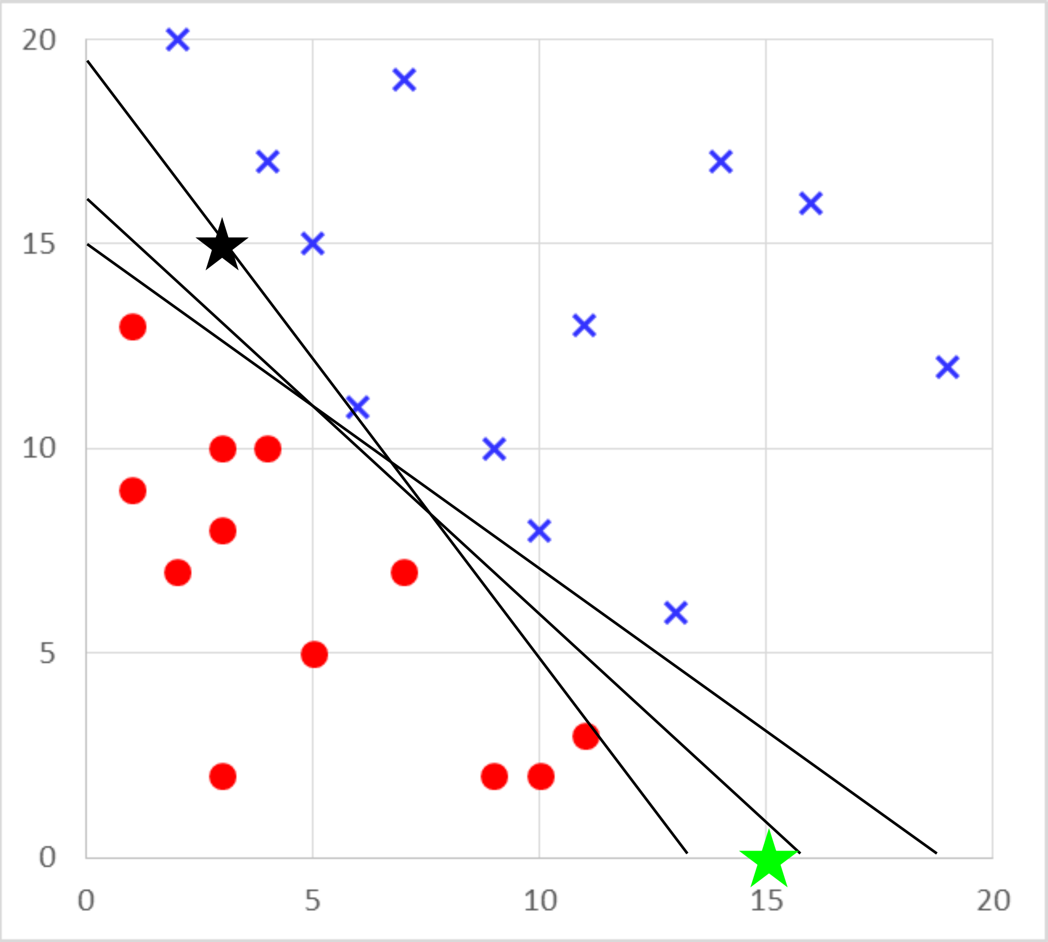
★ = Unknown Sample



θ_0=-16,
θ_1=1, θ_2=1

Linear Perceptron Classifier: Which is Best?

- There are many linear classifiers to choose from
  - Which one you find will depend on parameter search settings
- All work equally on linearly separable training data
  - But some will output a different prediction for unknown samples
  - Because Perceptron Rule works with 1/0 values, converges fully as soon boundary line found to fully separate data
  - Where it stops depends on if approaching from “red side” or “blue side”



Avoiding This Behaviour …

- Would like to find a boundary line that falls between the two classes, separating them as well as possible
  - To address this, need a different approach: **Linear Support Vector Machine**
  - Finds the maximum margin hyperplane separating classes
  - Not within the scope of this module

- The Logistic Classifier (coming up next) uses a “soft margin” that tends to push boundary away from nearest training data
  - However Logistic Classifier is **not** guaranteed to maximize the separating plane, whereas SVM is guaranteed to.

Principles of Machine Learning

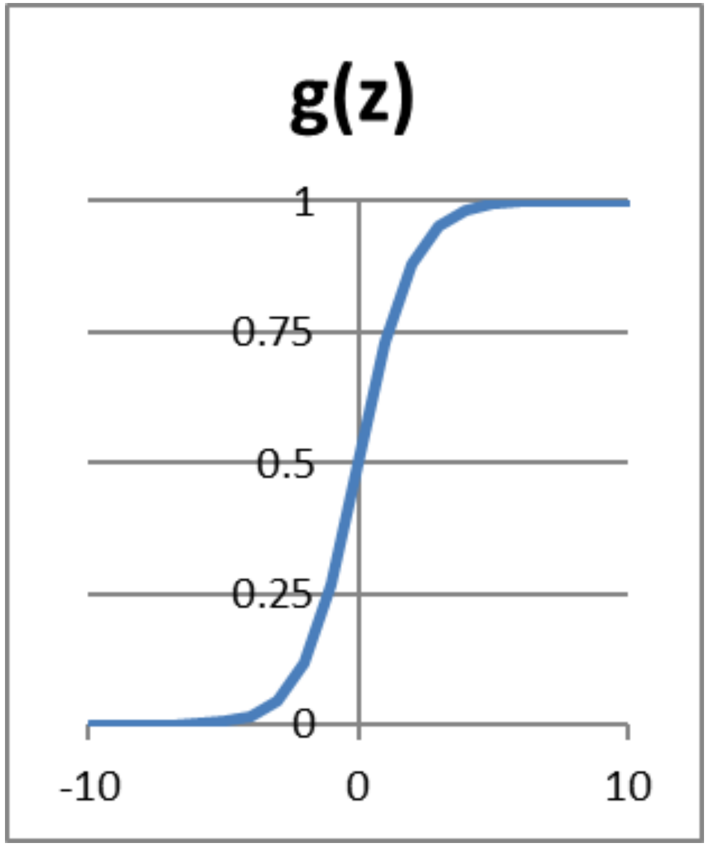
Soft Thresholds & Logistic Regression

Soft Thresholds & Logistic Regression

- Instead of hard threshold used in Linear Classifier, could use a ***soft*** one
  - Allow $h_\theta(x)$ to take on values in range [0,1]
  - Have it switch rapidly from 0 to 1 (almost step function)
- Go from the linear regression formula:

  $h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$
- To this:

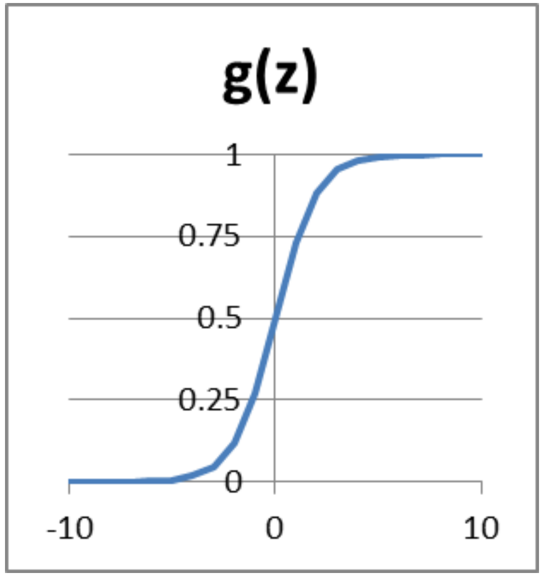
  $h_{\boldsymbol{\theta}}(\mathbf{x}) = g(\boldsymbol{\theta} \cdot \mathbf{x})$ where $g(z) = \dfrac{1}{1+e^{-z}}$



- $g(z)$ is called the sigmoidal or logistic function

Logistic Regression

- How we interpret the output:
  $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ is an estimate of the probability that y=1 for input $\boldsymbol{x}$, given the parameters $\boldsymbol{\theta}$



- As before, can derive a cost function for $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ and optimise parameters with gradient descent
  - The cost function is differentiable: can use standard GD as with Linear Regression

- Important: Logistic Regression is used for classification tasks, not for regression tasks!

Logistic Regression Cost Function [1]

- Probability that y=1 (Positive Class) for a case x is given by $h_{\boldsymbol{\theta}}(\boldsymbol{x})$

  $P(y=1 \mid \boldsymbol{x}) = h_{\boldsymbol{\theta}}(\boldsymbol{x})$

- Therefore, probability that y=0 (Negative class) is $1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})$

  $P(y=0 \mid \boldsymbol{x}) = 1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})$

- We can combine these equations to cover both y=1 and y=0:

  $P(y \mid \boldsymbol{x}) = (h_{\boldsymbol{\theta}}(\boldsymbol{x}))^{y} \, (1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}))^{1-y}$

- Starting from this, a cost function can be defined, though I won't show its derivation (I include the index (i) for a training instance):

$$J(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N} y^{(i)} \log\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)\right) + (1 - y^{(i)}) \log\left(1 - h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)\right)$$

Extra material

Logistic Regression Cost Function [2]

$$J(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N} y^{(i)} \log\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)\right) + (1 - y^{(i)}) \log\left(1 - h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)\right)$$

Extra material

- Behaviour:
  - As $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ tends to the correct value (either y=1 or y=0), $J(\boldsymbol{\theta})$ tends to 0
  - As $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ tends to the wrong value, $J(\boldsymbol{\theta})$ tends towards infinity
- The partial derivative of this cost function is:

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left(h_\theta\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)\, x_j^{(i)}$$

- Surprisingly, this looks identical to the linear regression case, though the hypothesis function is different
- Note: There are various definitions and derivations; I am following Stanford ones: http://ufldl.stanford.edu/tutorial/supervised/LogisticRegression/

Comparing Linear Classifiers With Hard and Logistic Thresholds

- Both find a hyperplane between classes, with threshold function to convert real number to 0/1
  - Both assume that classes are linearly separable
  - Neither attempt to maximise margin, though sigmoid can push Logistic Regressor boundary out from cases closest to it
- Parameters found with Gradient Descent
  - Logistic: Standard GD approach
  - Hard: Use a different update rule (Perceptron Update Rule) and have to decay α
- Logistic Regression outputs probabilities
  - Reflects uncertainty close to decision boundaries
- Logistic Regression has better convergence and behaves better when data are not linearly separable

Learning Objectives Review

In this topic you have learned to …

- Explain the drawbacks of using linear regression directly for classification problems
- Describe and implement approaches to improve on this: Linear Perceptron Classifiers and Logistic Regression
- Discuss their characteristics and limitations

Final Note – these are important foundational concepts:
Limitations of Linear Classifiers addressed with SVMs;
Logistic Regression leads to ideas in Neural Networks.