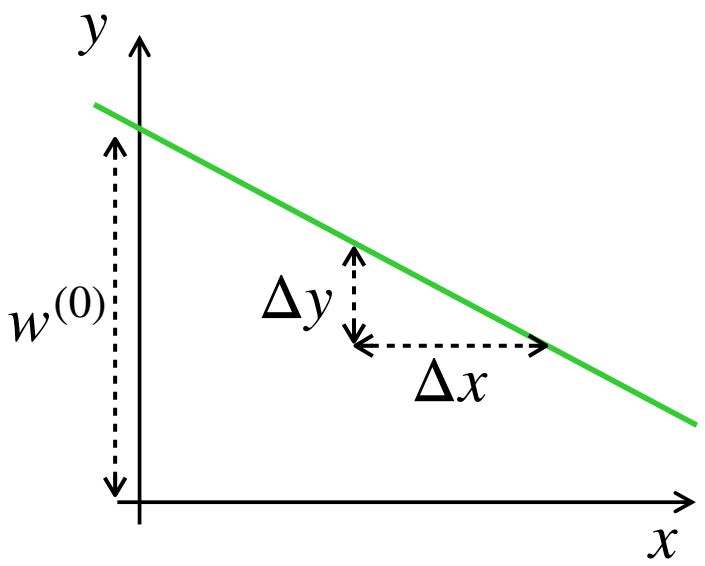
Lecture 3: Linear Models for Classification (Part 1)

Recap: A simple parametric model: The line

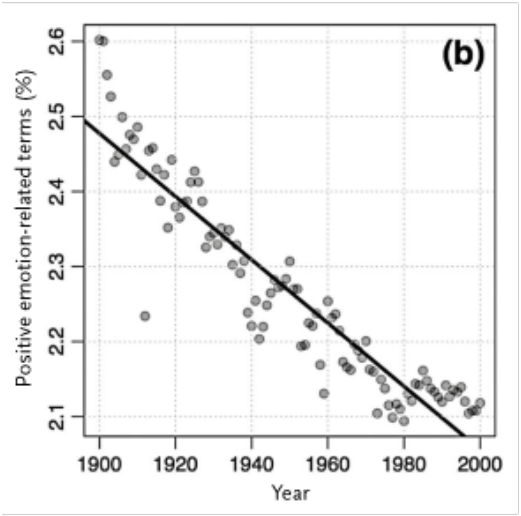


- Defined by 2 parameters
  - The y-intercept $w^{(0)}$
  - The slope $w^{(1)} = \dfrac{\Delta y}{\Delta x}$
- Mathematically, a line is expressed as

$$y = w^{(1)}x + w^{(0)}$$

Exercises: Solution

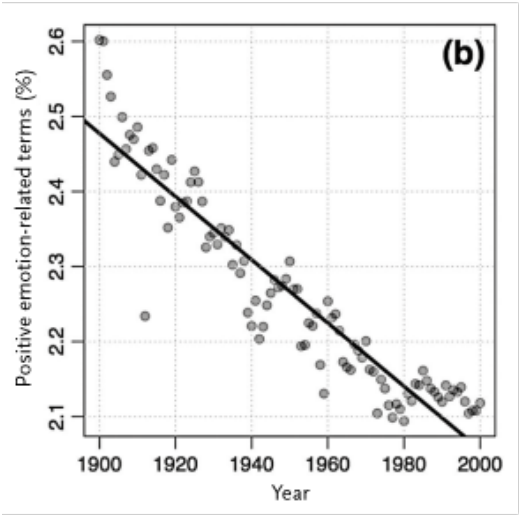
- From the trend data below



- Given an example of what numerical values one $\mathbf{x}_i$ and the corresponding y_i could roughly take
  - Solution: E.g., $\mathbf{x}_i = \begin{bmatrix} 1940 \\ 1 \end{bmatrix}, \quad y_i = 2.22$

Exercises: Solution

- From the trend data below



  - What would the matrix $\mathbf{X}$ and the vector $\mathbf{y}$ look like (using rough numerical values for a few samples)?

    Solution: $\mathbf{X} = \begin{bmatrix} 1910 & 1 \\ 1915 & 1 \\ 1920 & 1 \\ \vdots & \vdots \\ 2000 & 1 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 2.45 \\ 2.43 \\ 2.38 \\ \vdots \\ 2.11 \end{bmatrix}$

Recap: Hyperplane

- This can be generalized to higher input dimensions
- In dimension D, we can write

$$y = w^{(0)} + w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \ldots + w^{(D)}x^{(D)} = \mathbf{w}^T \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(D)} \\ 1 \end{bmatrix}$$

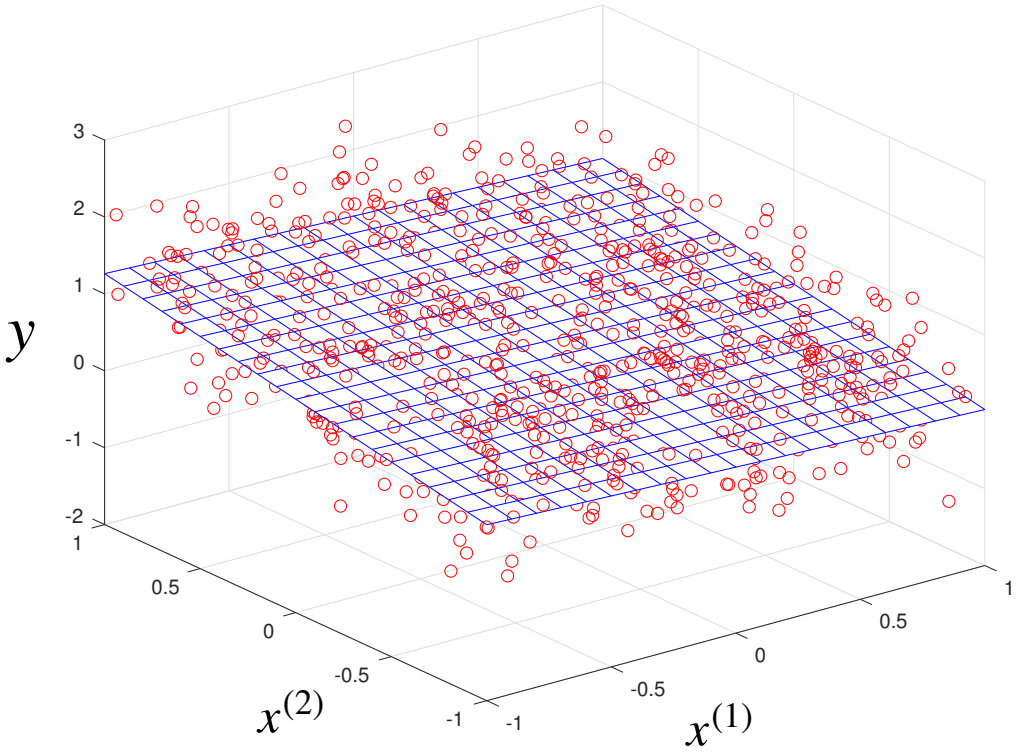
- Ultimately, whatever the input dimension, we can write

$$y = \mathbf{w}^T\mathbf{x}$$

with $\mathbf{x} \in \mathbb{R}^{D+1}$, where the extra dimension contains a 1 to account for $w^{(0)}$

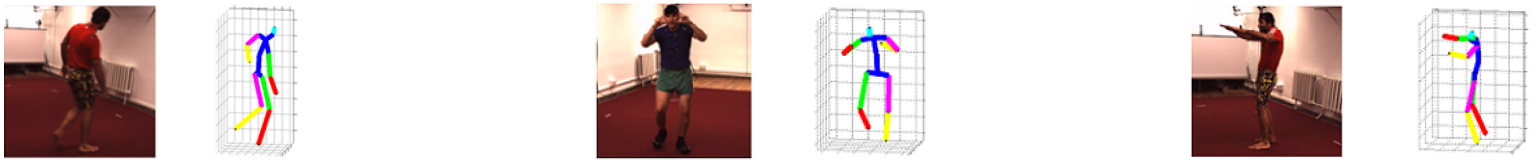
Recap: Linear regression

- Given N noisy pairs $\{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i \in \mathbb{R}^D$ (below, $D = 2$), find the parameters $\mathbf{w}^*$ that best fit these observations



Recap: Dealing with multiple output dimensions

- The previous examples assumed that the output was a single value, i.e., $y_i \in \mathbb{R}$
- In practice, however, one may want to output multiple values for a given input, i.e., $\mathbf{y}_i \in \mathbb{R}^C$, with $C > 1$
- E.g., human pose estimation
  - We aim to predict the 3D position of each joint, i.e., for J joints, $C = 3J$



Recap: Multi-output prediction

- To predict multiple values, we cannot just have a vector $\mathbf{w}$, because the product $\mathbf{w}^T\mathbf{x}_i$ yields a single value

- We therefore use a matrix $\mathbf{W} \in \mathbb{R}^{(D+1)\times C}$, such that

$$\hat{\mathbf{y}}_i = \mathbf{W}^T\mathbf{x}_i = \begin{bmatrix} \mathbf{w}_{(1)}^T \\ \mathbf{w}_{(2)}^T \\ \vdots \\ \mathbf{w}_{(C)}^T \end{bmatrix} \mathbf{x}_i$$

where each $\mathbf{w}_{(j)}$ is a $(D+1)$-dimensional vector, used to predict one output dimension

Recap: Multi-output linear regression

- We write multi-output linear regression as

$$\min_{\mathbf{W}} \sum_{i=1}^{N} \|\mathbf{W}^T \mathbf{x}_i - \mathbf{y}_i\|^2$$

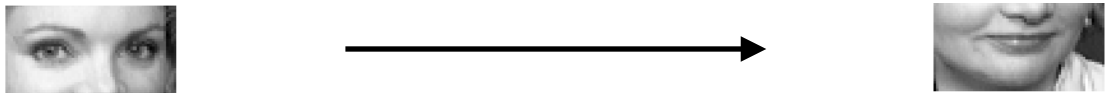
- The closed-form solution then is

$$\mathbf{W}^* = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{Y}$$

where $\mathbf{Y} \in \mathbb{R}^{N \times C}$ is a matrix stacking the N outputs

Exercises: Solution

- Given the following face completion task:
  - Task: Given the top half image of a face, predict the bottom half



  - Assuming that there are 35 training subjects with 10 images per subject, and that a complete face image is of size 28×28 pixels, what is the shape of the matrices $\mathbf{X}$ and $\mathbf{Y}$?
    - Solution: For $\mathbf{X}$, the shape is $N \times (D + 1)$. Here $N = 35 \cdot 10 = 350$, $D = 28 \cdot 14 = 392$. For $\mathbf{Y}$, the shape is $N \times C$, with $C = 28 \cdot 14 = 392$.
  - How many parameters are there to learn?
    - Solution: $\mathbf{W}$ has shape $(D + 1) \times C$. This gives $393 \cdot 392 = 154056$ parameters to learn

Goals of today's lecture

- Move from regression to classification
- Introduce the formulation for logistic regression
- Introduce basic minimization concepts
- Introduce classification evaluation metrics

From regression to classification

- Predict one discrete label for a given sample
  - E.g., binary classification for movies (like vs not like)

5	Column1	Column2
6	A very, very, very slow-moving, aimless movie about a distressed, drifting young man.	0
8	Not sure who was more lost - the flat characters or the audience, nearly half of whom walked out.	0
10	Attempting artiness with black & white and clever camera angles, the movie disappointed - became e	0
11	Very little music or anything to speak of.	0
13	The best scene in the movie was when Gerardo is trying to find a song that keeps running through his	1
15	The rest of the movie lacks art, charm, meaning... If it's about emptiness, it works I guess because it's	0
16	Wasted two hours.	0
18	Saw the movie today and thought it was a good effort, good messages for kids.	1
20	A bit predictable.	0
22	Loved the casting of Jimmy Buffet as the science teacher.	1
23	And those baby owls were adorable.	1
25	The movie showed a lot of Florida at it's best, made it look very appealing.	1
26	The Songs Were The Best And The Muppets Were So Hilarious.	1

  - E.g., multi-class image recognition

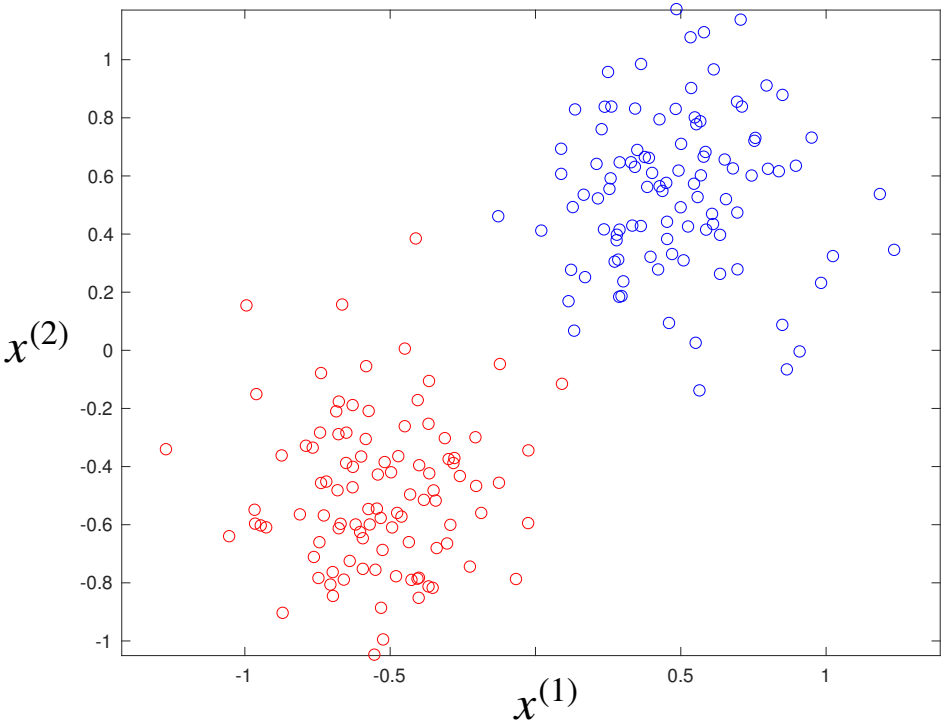

Cat


Dog


Car

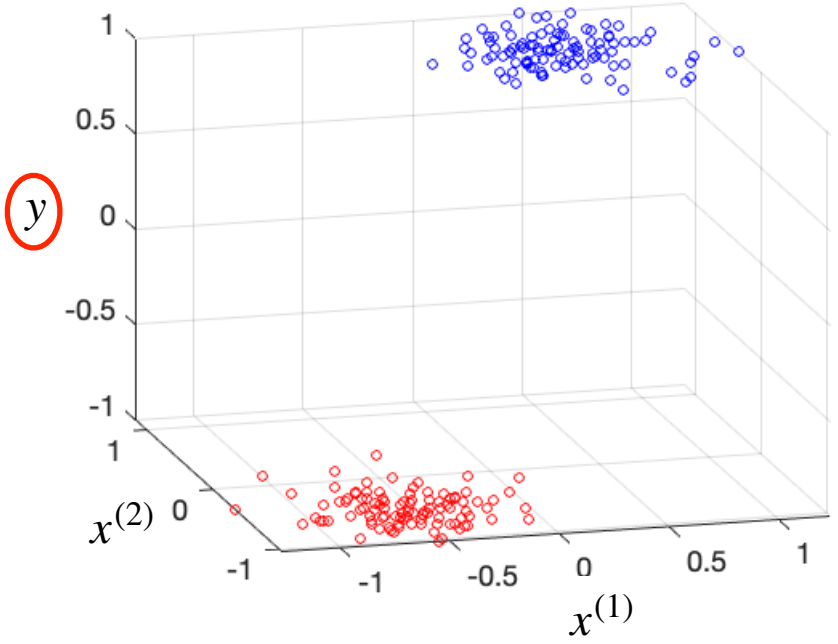
Binary classification

- Two classes (shown as different colors):
  - The label $y \in \{-1,1\}$, or $y \in \{0,1\}$
  - The samples with label 1 are called *positive* samples
  - The samples with label -1 (or 0) are called *negative* samples



Binary classification as regression

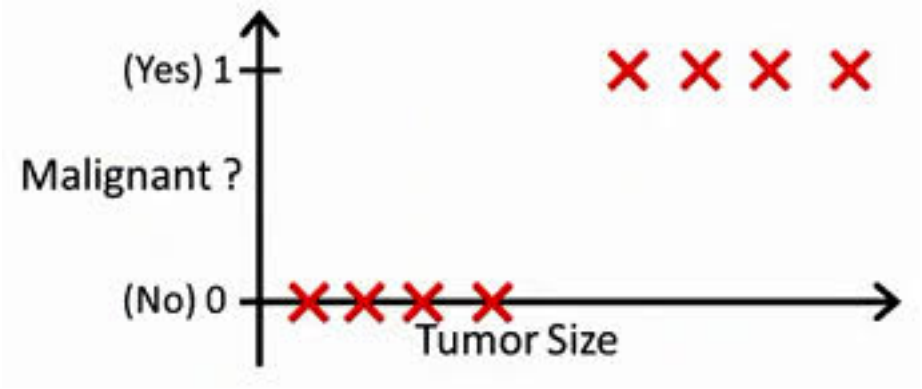
- The same data as before can also be seen in 3D
  - The label acts as the output dimension, just as in a regression problem



- This suggests that one can tackle classification as a regression task

Binary classification as regression

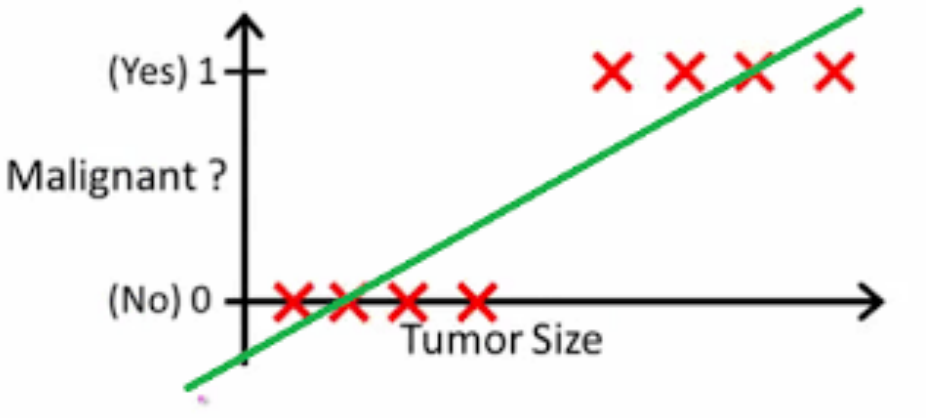
- Example (from Andrew Ng's course): Classify a tumor as benign vs malignant
  - 1D input (tumor size), 1D output (malignant vs benign)



Binary classification as regression

- Example (from Andrew Ng's course): Classify a tumor as benign vs malignant
  - 1D input (tumor size), 1D output (malignant vs benign)
  - Using a linear model, we can predict the label as

$$\hat{y} = w^{(1)}x + w^{(0)}$$



Least-square binary classification

- To fit the linear model to training data, the same strategy as for linear regression can be used:
  - Given N pairs $\{(x_i, y_i)\}$, we can use the least-square loss to write

$$\min_{\mathbf{w}} \sum_{i=1}^{N} (\mathbf{x}_i^T \mathbf{w} - y_i)^2$$

- Because it has exactly the same formulation as before, the solution is exactly the same, i.e.,

$$\mathbf{w}^* = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{y}$$

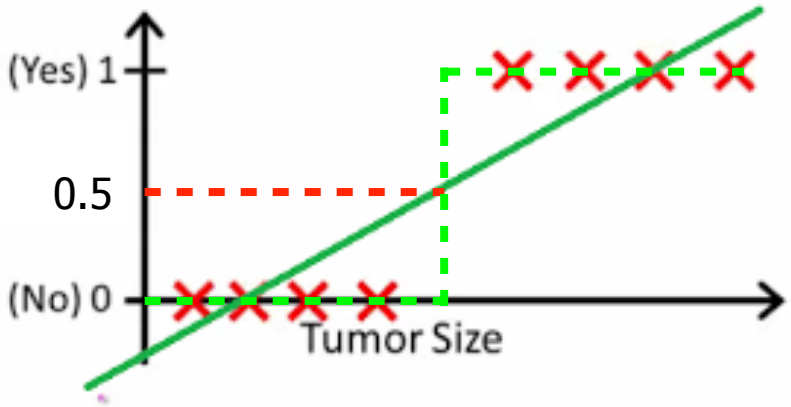
Least-square binary classification

- Linear regression outputs a continuous value
  - We would rather like to have a discrete label, 1 or 0 (-1)
  - This can be achieved by thresholding:

$$\text{label} = \begin{cases} 1 & \text{if } \hat{y} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

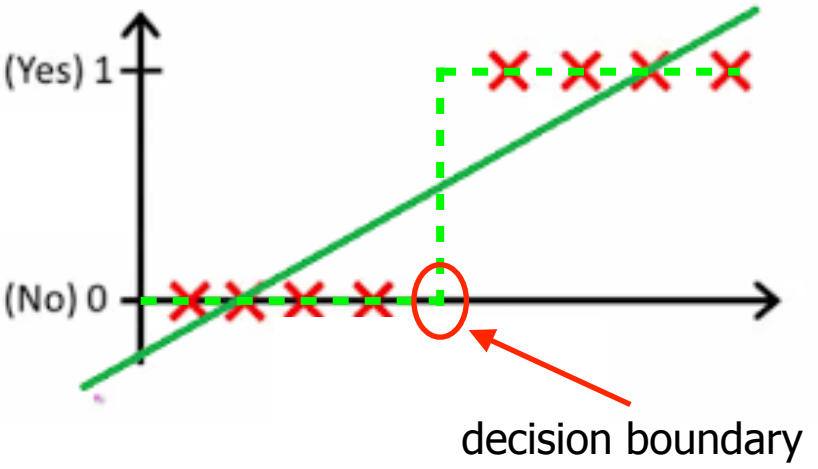
or, if the negative label is -1:

$$\text{label} = \begin{cases} 1 & \text{if } \hat{y} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$



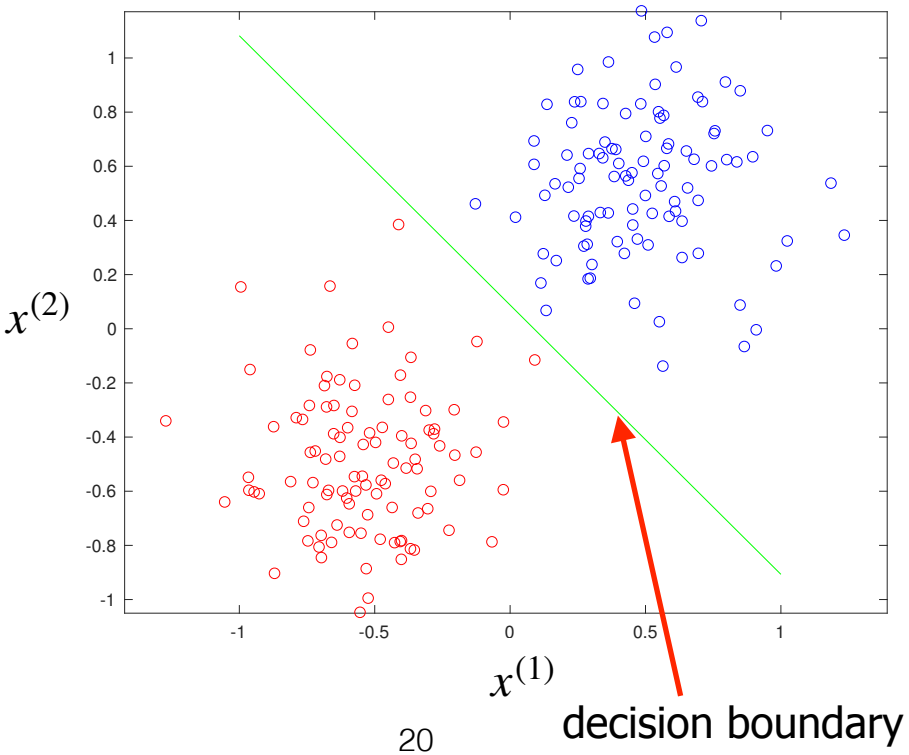
Decision boundary

- The point where the predicted label changes from 0 (or -1) to 1 forms a *decision boundary*
  - With 1D inputs, it is a single point



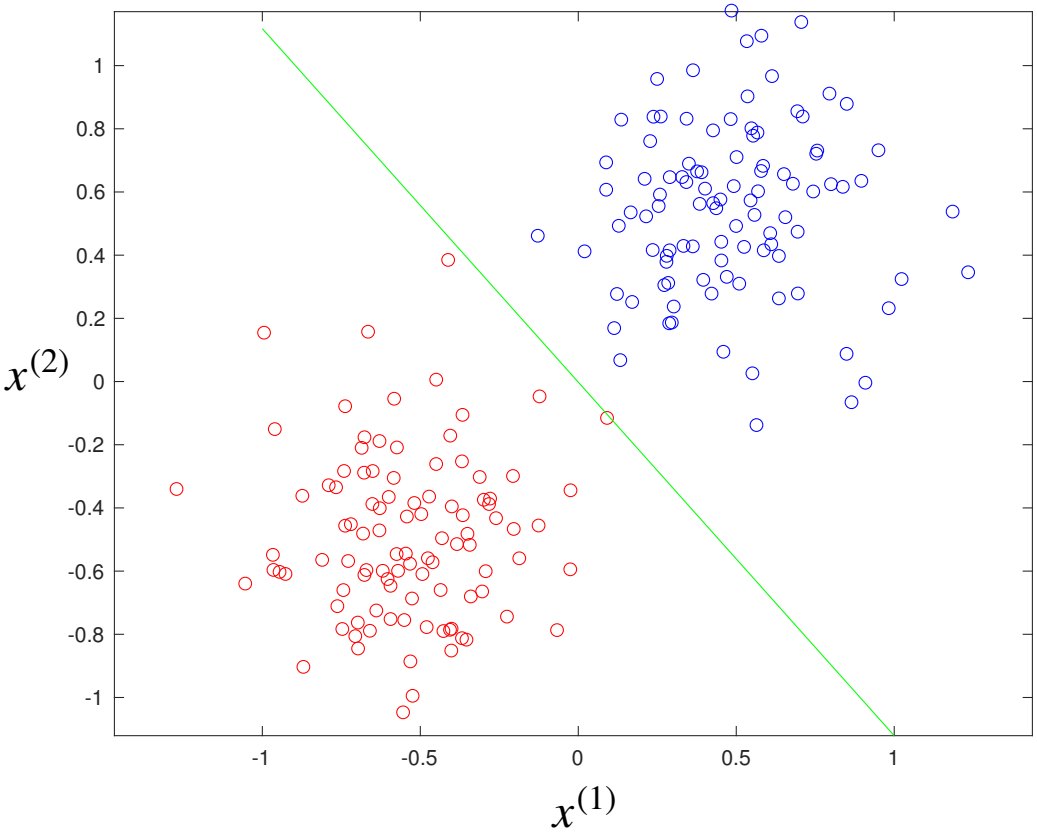
Decision boundary

- The point where the predicted label changes from 0 (or -1) to 1 forms a *decision boundary*
  - With 1D inputs, it is a single point
  - With 2D inputs, it is a line



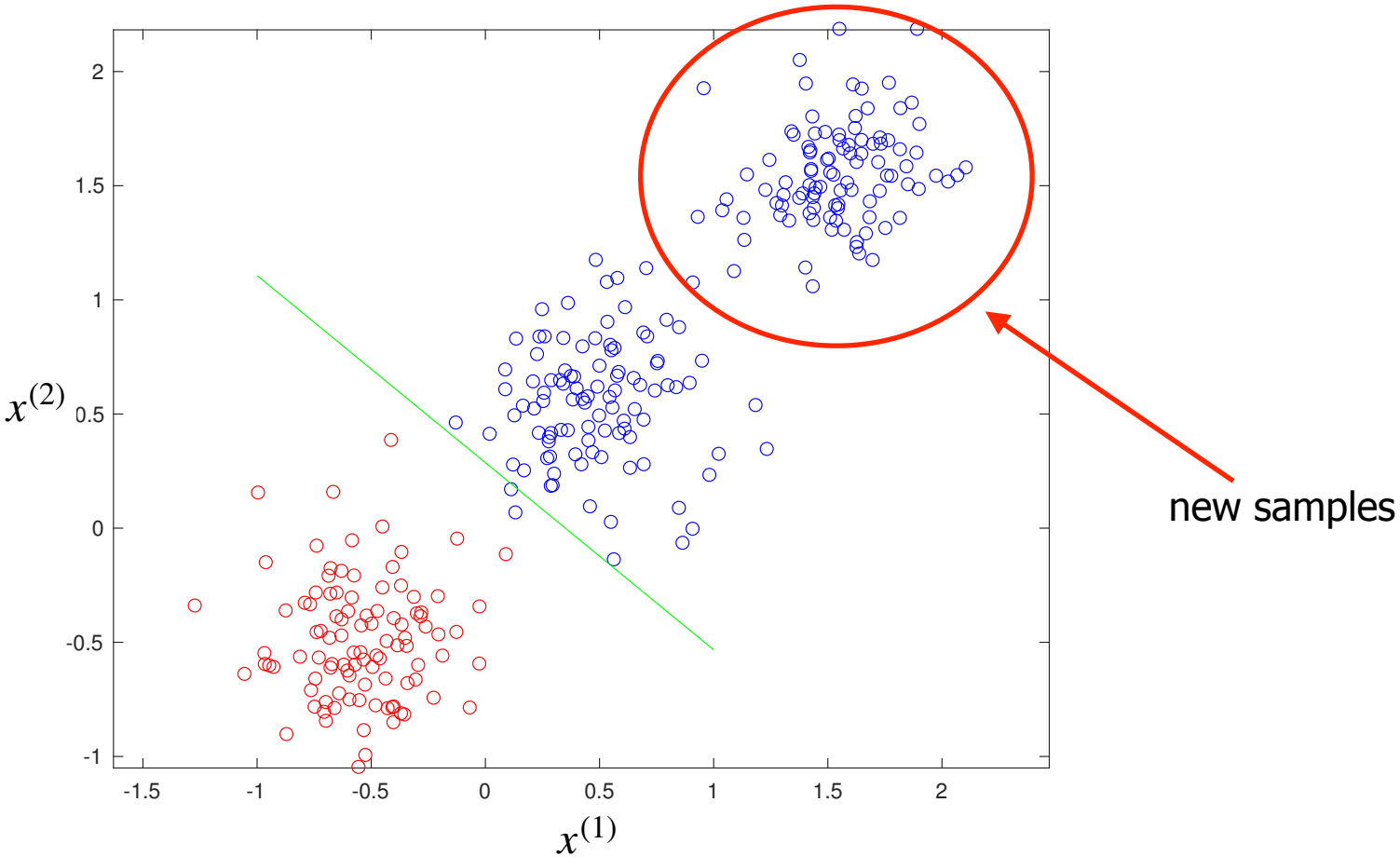
Least-square binary classification: Example

- 2D inputs, 2 classes
  - All training samples are correctly classified



Least-square binary classification: Failure

- However, adding new samples far from the decision boundary makes some of the training samples be misclassified

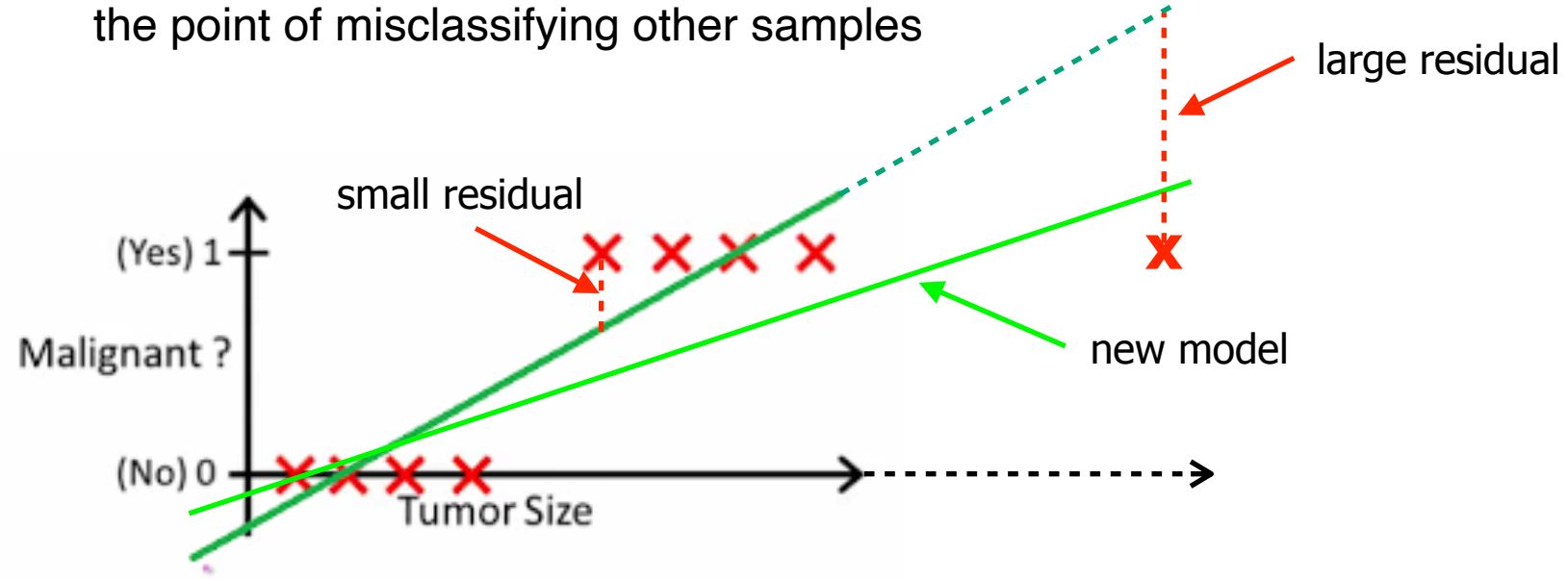


Failure: Geometric interpretation

- http://digitalfirst.bfwpub.com/stats_applet/stats_applet_5_correg.html

Failure: Geometric interpretation

- Back to the tumor prediction example
  - The residuals for “normal” samples are quite small
  - However, a sample with a much larger tumor size (with true label 1) would have a much larger residual
  - Training with such a sample would strongly affect the model parameters, to the point of misclassifying other samples



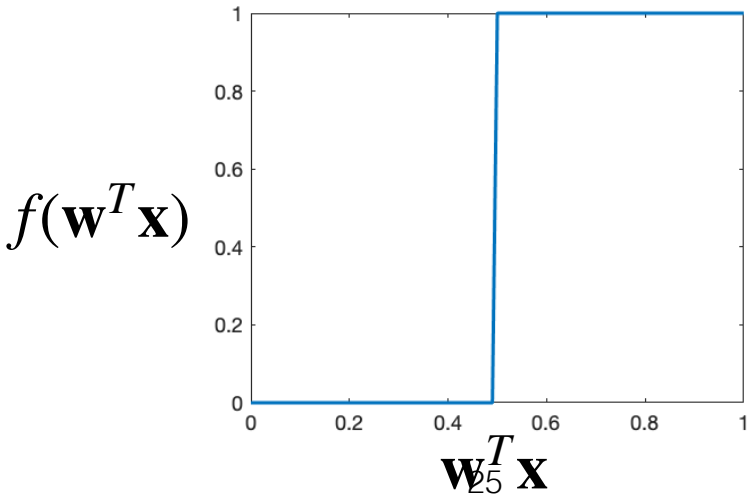
  - This is because least-square classification fits a line to non-linear data

Adding non-linearity

- The output of the linear model $\mathbf{w}^T\mathbf{x}$ is a continuous value
- What we would really like is a discrete output, e.g.,

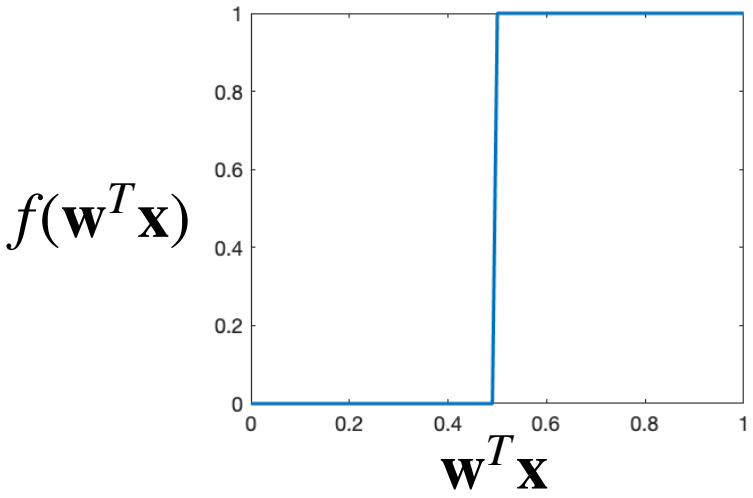
$$y = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

- This can be achieved by passing the output of the linear model through a step function



Step function: Drawback

- The gradient of the step function is not continuous
  - It is zero almost everywhere, except at $\mathbf{w}^T\mathbf{x} = 0.5$

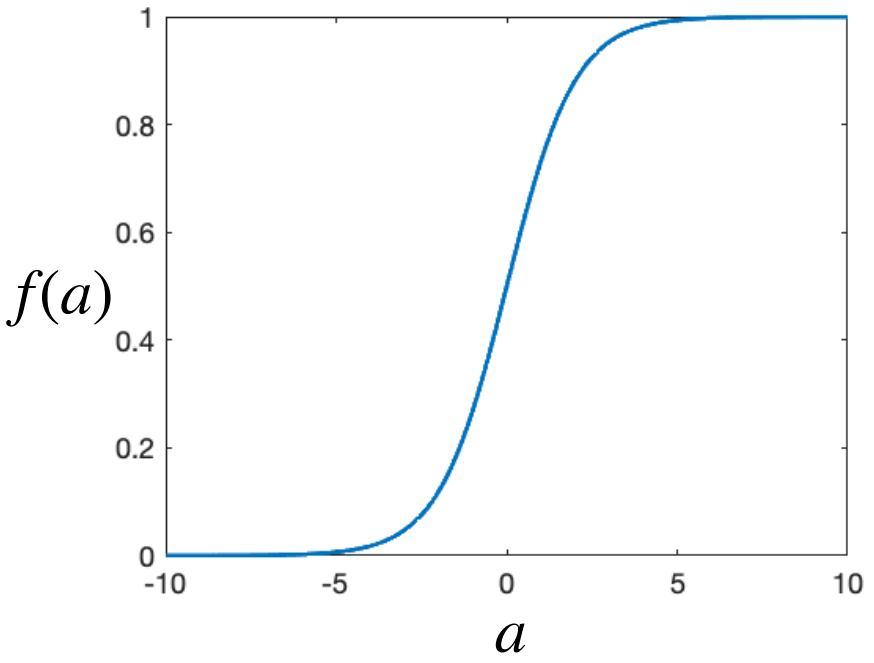


- This behavior typically makes optimization difficult
  - The Perceptron algorithm can nonetheless handle the 2-class case, but we will not cover it in this course

Adding non-linearity

- Instead, one can use a smooth approximation of the step function, such as the *logistic sigmoid function*

$$f(a) = \frac{1}{1 + \exp(-a)}$$

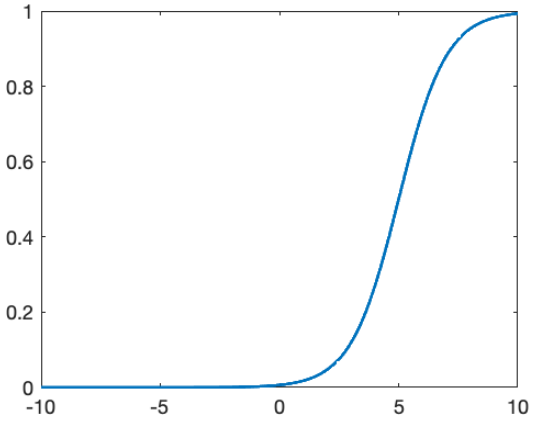


Logistic function: Effect of w

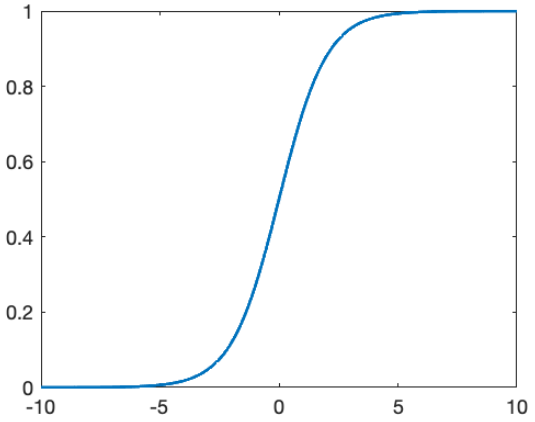
- With a 1D input x, applying the logistic function to the output of a linear model gives a prediction

$$\hat{y} = \frac{1}{1 + \exp(-w^{(1)}x - w^{(0)})}$$

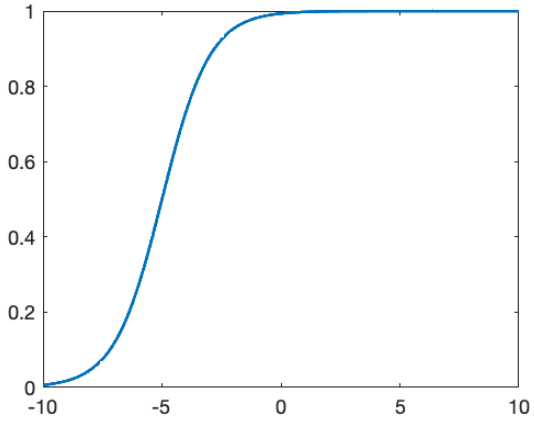
- Effect of $w^{(0)}$



$w^{(0)} = -5,\ w^{(1)} = 1$



$w^{(0)} = 0,\ w^{(1)} = 1$



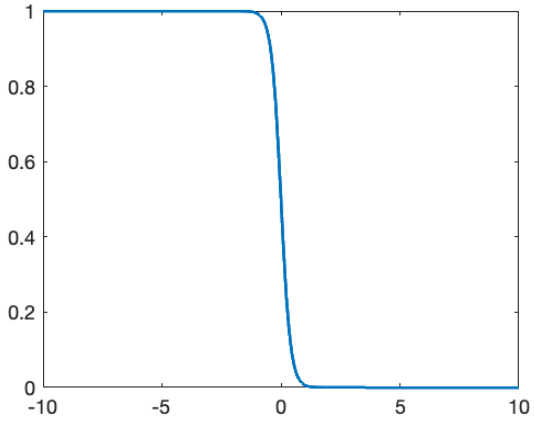
$w^{(0)} = 5,\ w^{(1)} = 1$

Logistic function: Effect of w

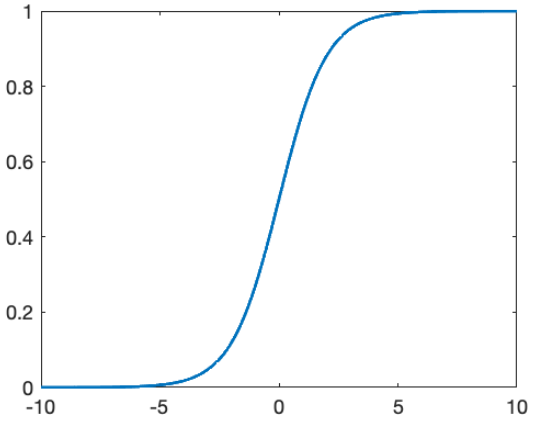
- With a 1D input x, applying the logistic function to the output of a linear model gives a prediction

$$\hat{y} = \frac{1}{1 + \exp(-w^{(1)}x - w^{(0)})}$$

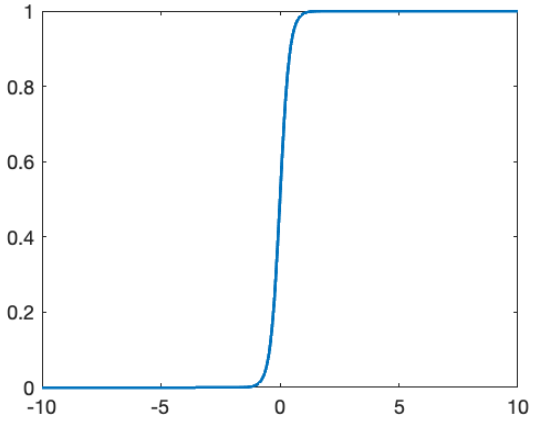
- Effect of $w^{(1)}$



$w^{(0)} = 0, w^{(1)} = -5$



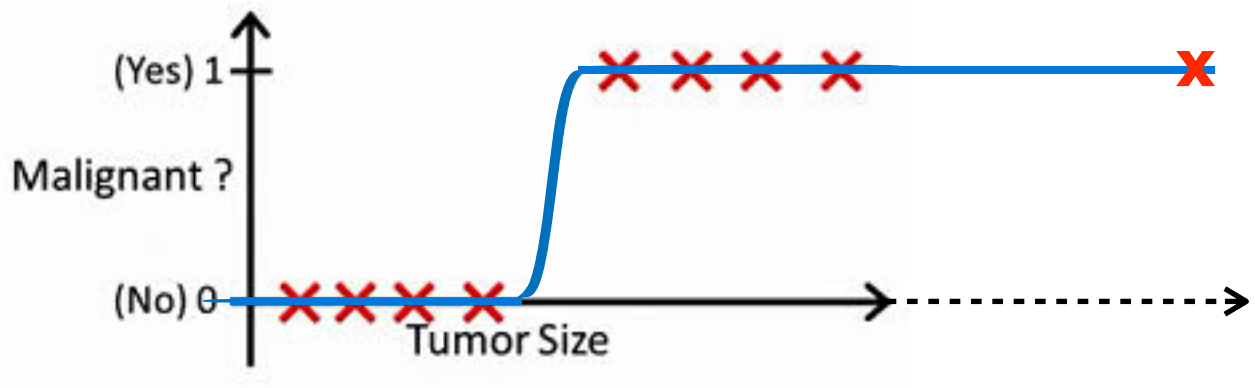
$w^{(0)} = 0, w^{(1)} = 1$



$w^{(0)} = 0, w^{(1)} = 5$

Fitting the tumor data

- Applying a logistic function to the output of the linear model can fit the data much better
- Even in the presence of outliers



Probabilistic interpretation

- In the binary case, with D dimensional inputs, we can write the prediction of the model as

$$\hat{y} = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

- Such a prediction can be interpreted as the probability that $\mathbf{x}$ belongs to the positive class (or as a score for the positive class)
- The probability to belong to the negative class (or score for the negative class) is then computed as $1 - \hat{y}$

Logistic regression

- The model whose prediction is defined as

$$\hat{y} = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

  is called *logistic regression*

- Note that this name is slightly misleading, as this is truly a classification model, not a regression one

- At its heart, logistic regression still makes use of the linear model, but followed by a nonlinearity to better encode the underlying binary classification problem

Logistic regression: Training

- Given N training samples, to learn the parameters $\mathbf{w}$ of the logistic regression model, we need to define a loss function

- Instead of using the L^2 loss as in the least-square case, this is achieved by making use of the probabilistic interpretation
  - Because knowledge of probabilities is not required for this course, the derivation of the loss function that follows will be a bit hand-wavy.
  - Nevertheless, this loss remains intuitive: The loss is such that it aims to find the parameters that maximize the probability (or score) of the correct class for each training sample

Logistic regression: Training

- Specifically, the logistic regression loss is derived by first writing the following likelihood (think of it as a value representing how well the true labels are predicted given $\mathbf{w}$)

$$p(\mathbf{y} \mid \mathbf{w}) = \prod_{i=1}^{N} \hat{y}_i^{y_i} \cdot (1 - \hat{y}_i)^{1-y_i}$$

- Because a product of terms between 0 and 1 can be unstable to optimize, we take the logarithm of this likelihood, and take its negative to define an empirical risk to minimize, written as

$$R(\mathbf{w}) = -\sum_{i=1}^{N} \left(y_i \ln(\hat{y}_i) + (1 - y_i) \ln(1 - \hat{y}_i) \right)$$

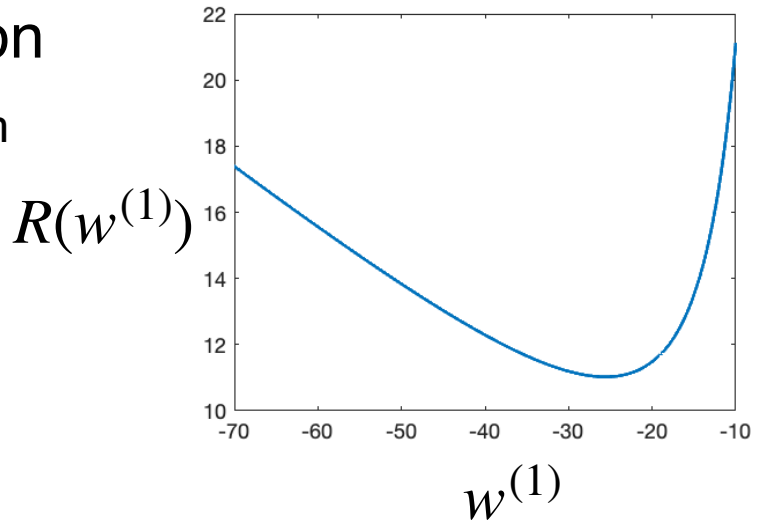
- This loss function is known as the *cross-entropy*

Logistic regression: Training

- Since the true y_i is either 0 or 1, the cross-entropy can be re-written as

$$R(\mathbf{w}) = -\sum_{i \in \text{positive samples}} \ln(\hat{y}_i) - \sum_{i \in \text{negative samples}} \ln(1 - \hat{y}_i)$$

- The cross-entropy is a convex function
  - However, its minimization has no closed-form solution
- We optimize it via gradient descent
  - Let us now review this algorithm



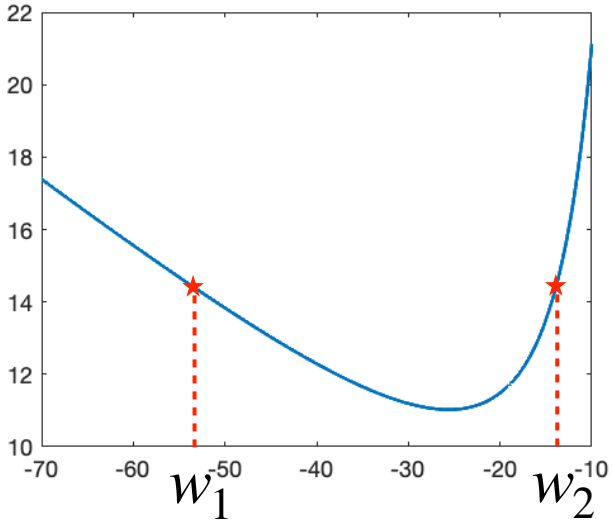
Interlude

Iterative, Gradient-based Optimization

Minimizing a function

- With 1D functions for which the solution cannot be computed in closed form, one can nonetheless use the derivative
  - Recall that it represents the slope at a given point, and thus indicates in which direction to move to reach a lower function value

At w_1, the slope is negative. However, one should move in the positive direction ($\Delta w > 0$) to go towards the minimum



At w_2, the slope is positive. However, one should move in the negative direction ($\Delta w < 0$) to go towards the minimum

- This indicates that one should move in the direction opposite to the derivative for minimization

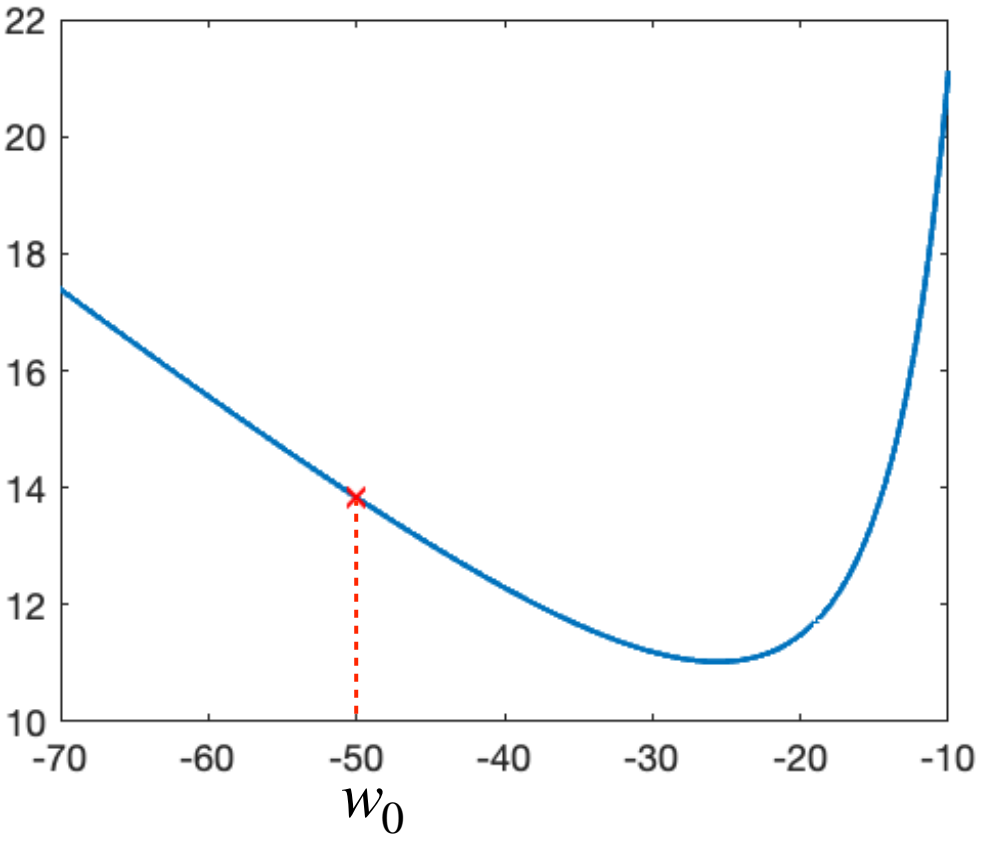
Minimizing a function

- Algorithm:
  1. Initialize w_0 (e.g., randomly)
  2. While not converged
     - 2.1. Update $w_k \leftarrow w_{k-1} - \eta \frac{dR(w_{k-1})}{dw}$

- η defines the step size of each iteration
  - In ML, it is often referred to as *learning rate*

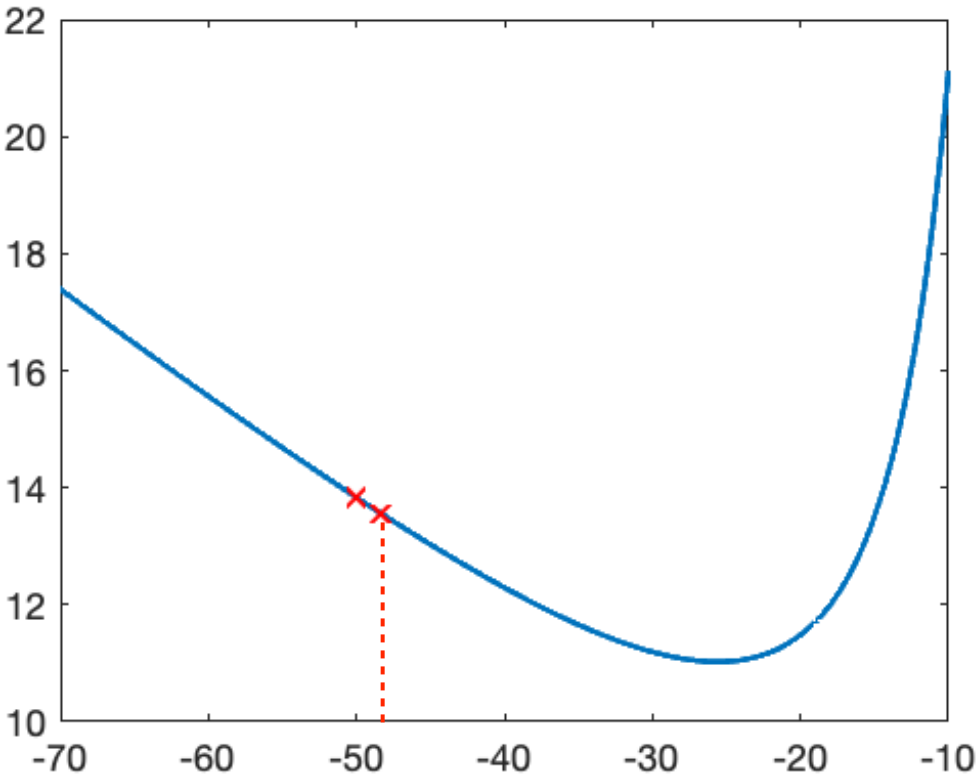
Minimizing a convex function

- Example:



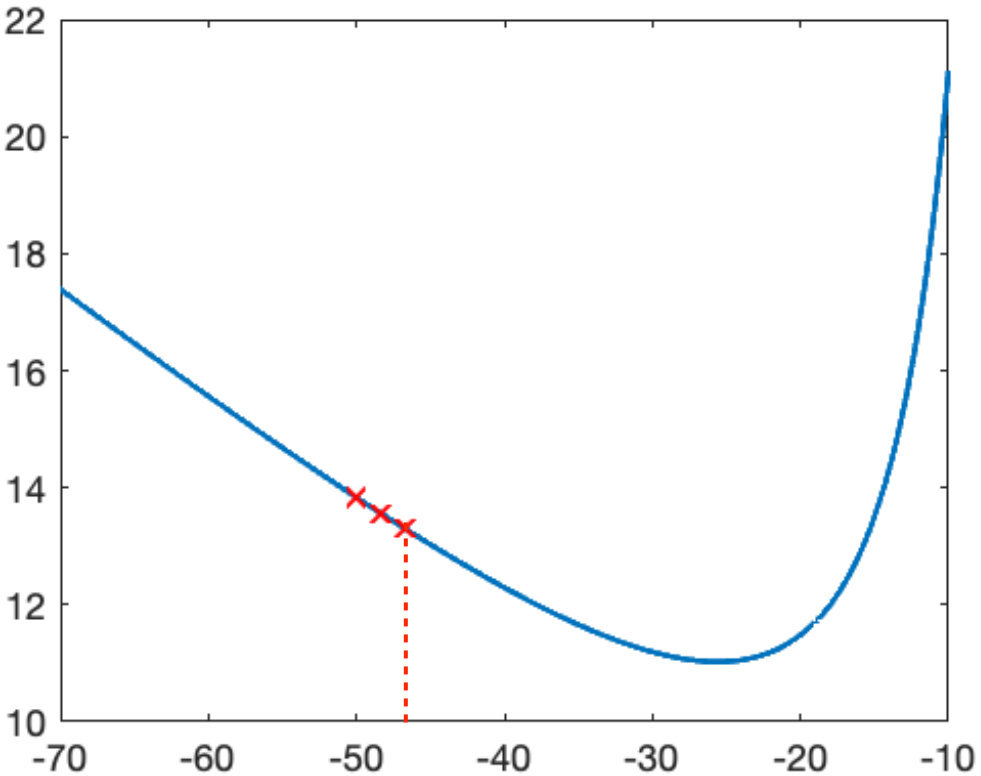
Minimizing a convex function

- Example:



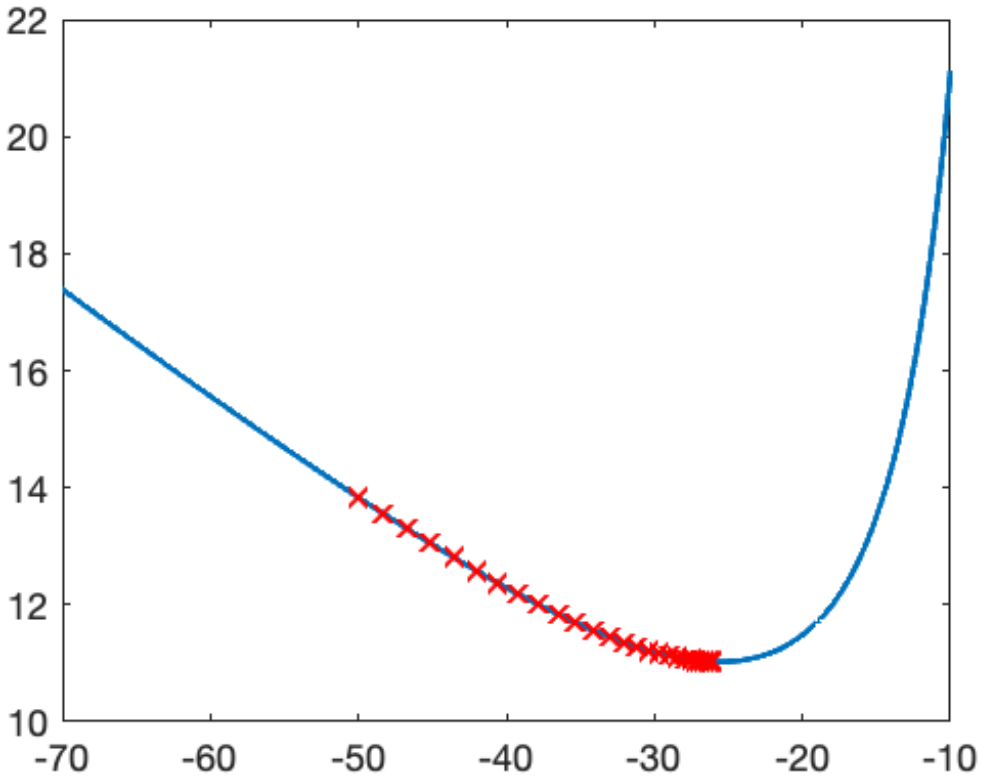
Minimizing a convex function

- Example:



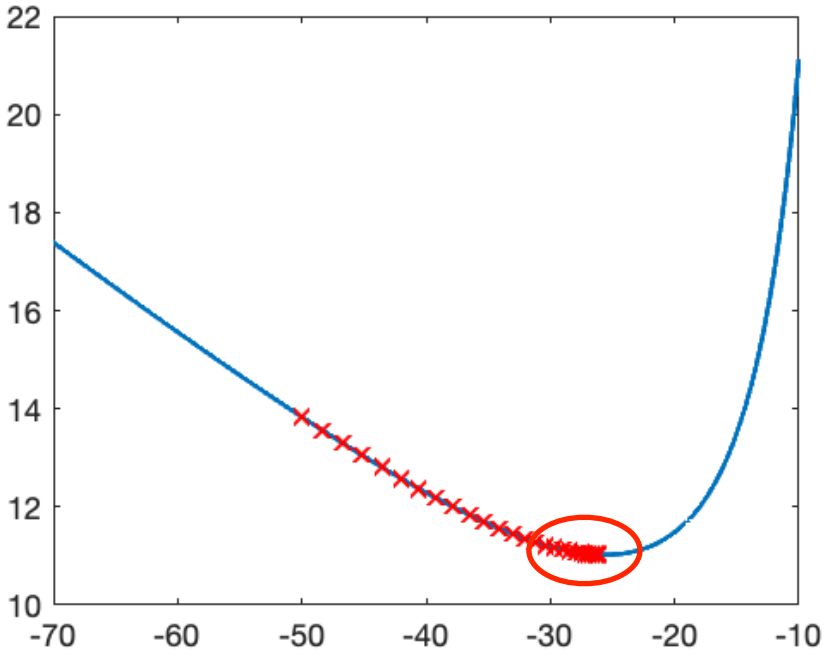
Minimizing a convex function

- Example:



Minimizing a convex function

- Convergence:
  - Because the derivative becomes smaller $\left(\frac{dR(w)}{dw} \rightarrow 0\right)$, the update Δw becomes smaller, and the algorithm converges



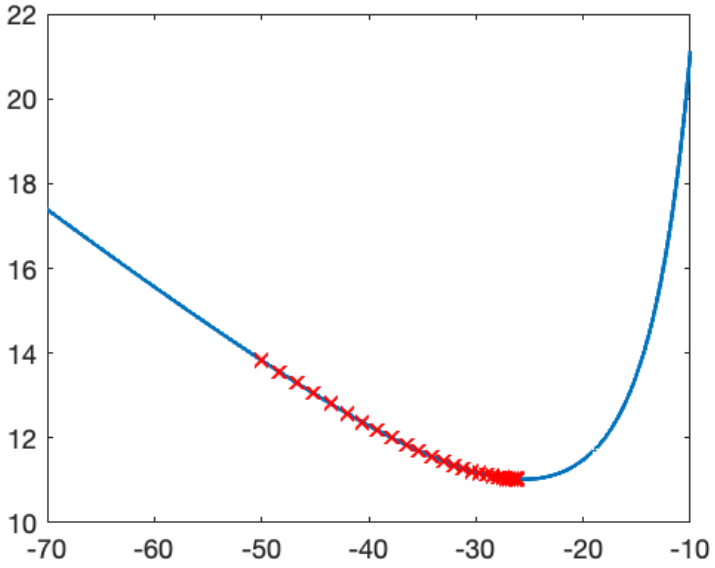
Minimizing a function

- Potential convergence criteria:
  - Change in function values less than threshold: $|R(w_{k-1}) - R(w_k)| < \delta_R$
  - Change in parameter value less than threshold: $|w_{k-1} - w_k| < \delta_w$
  - Maximum number of iterations reached
    - No guarantee to have reached the minimum

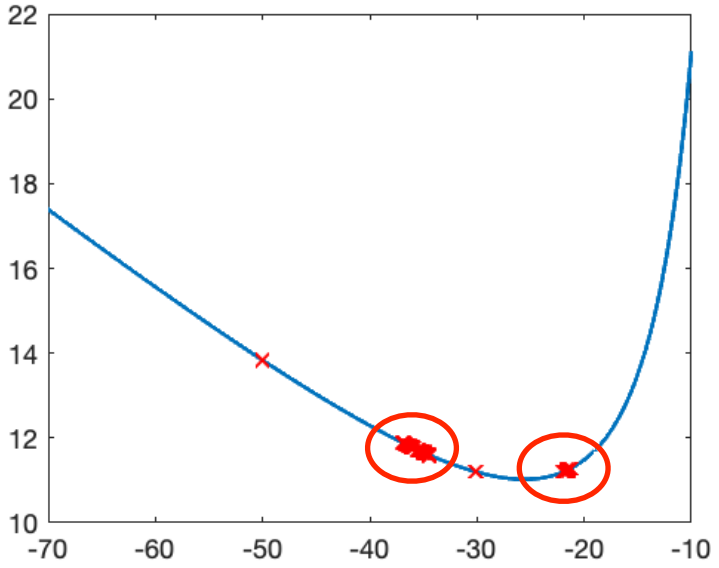
Minimizing a convex function

- Influence of η:

$\eta = 10$



$\eta = 120$

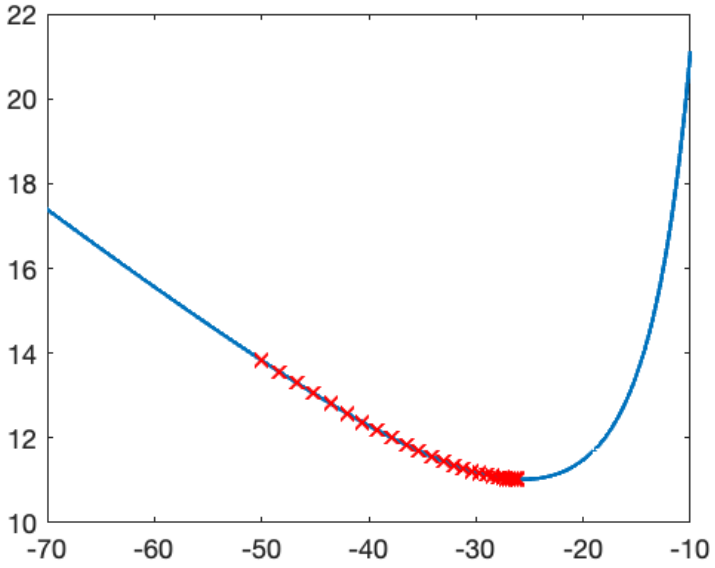


The steps are too large and the algorithm starts jumping between these two points

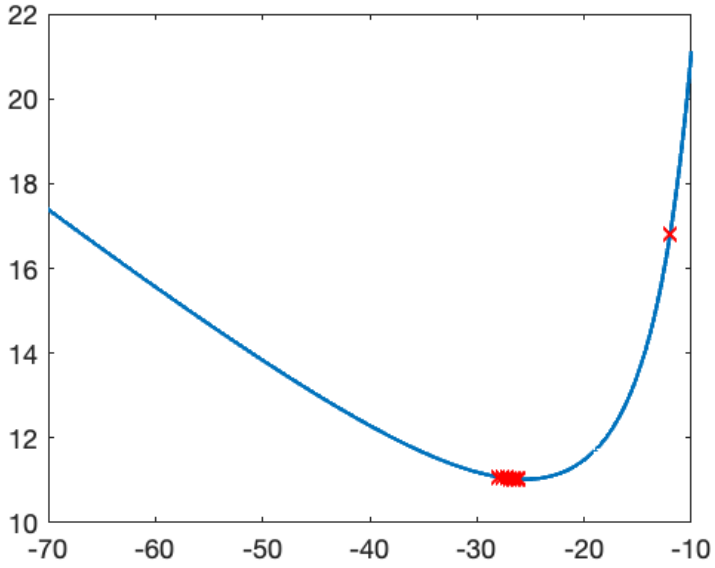
Minimizing a convex function

- Influence of the initial w_0:

$w_0 = -50$



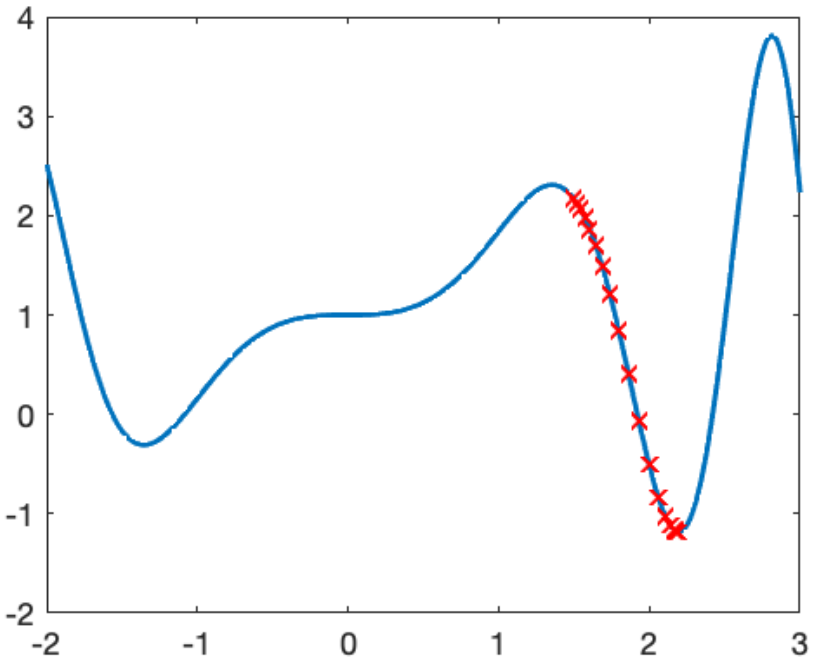
$w_0 = -12$



Still converges to the same point, but faster

Minimizing a non-convex function

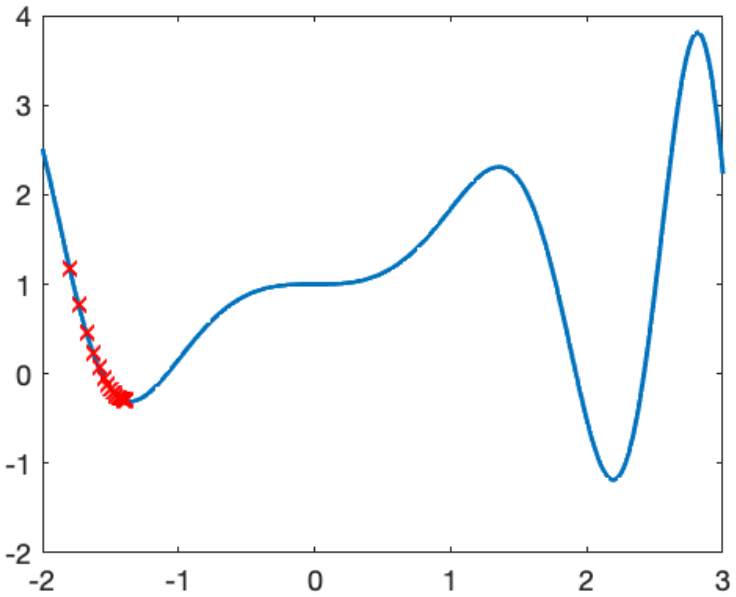
- The same iterative algorithm as before can be used



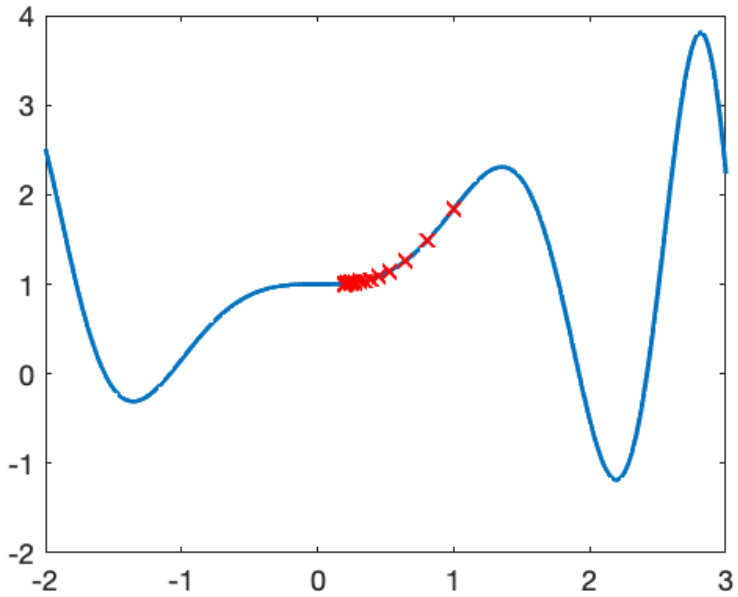
Minimizing a non-convex function

- However, different initializations yield different results
  - Before, $w_0 = 1.5$ and optimization reached the global minimum

$w_0 = -1.8$
Local minimum



$w_0 = 1$
Saddle point

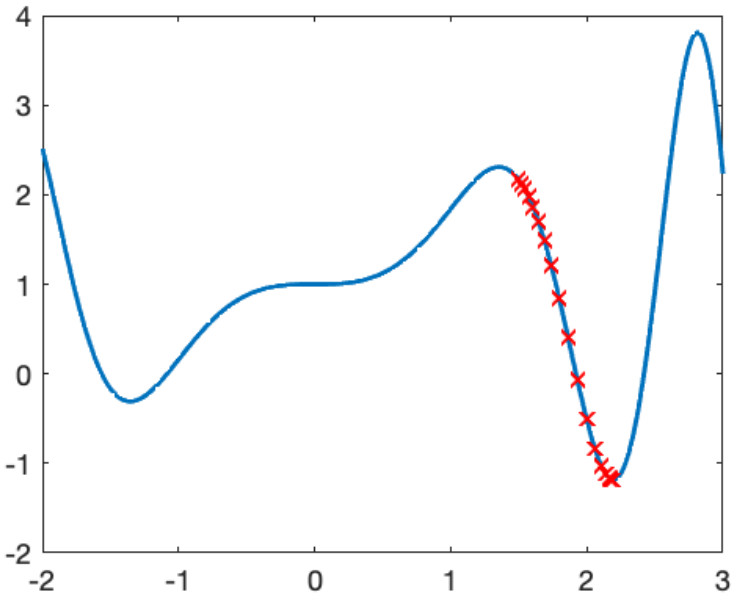


Minimizing a non-convex function

- And the results remain sensitive to the learning rate η
  - Note that η is problem-dependent and the values used here are very different from those used in the convex example before

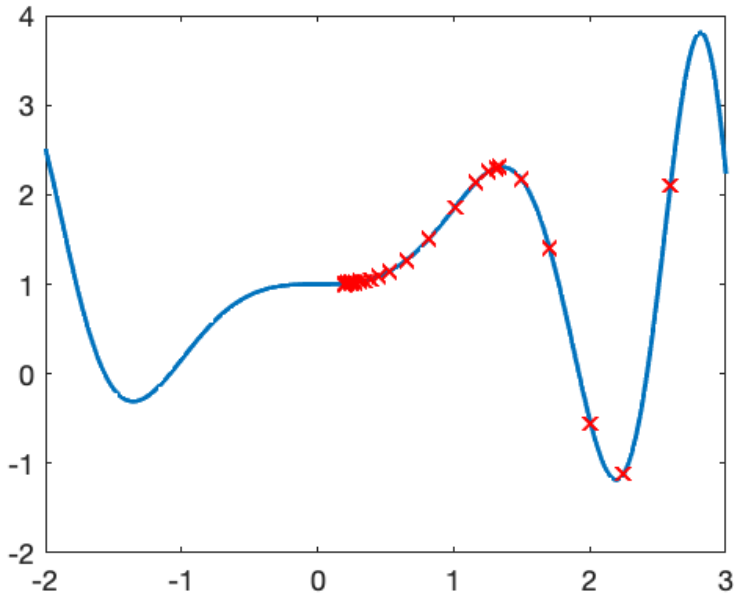
$\eta = 0.01, \ w_0 = 1.5$

Global minimum



$\eta = 0.1, \ w_0 = 1.5$

Saddle point



Recap: Gradient

- In practice, we have more than one parameter, i.e., $\mathbf{w} \in \mathbb{R}^D$, and we thus use the function gradient:

$$\nabla R(\mathbf{w}) = \begin{bmatrix} \frac{\partial R}{\partial w^{(1)}} \\ \frac{\partial R}{\partial w^{(2)}} \\ \vdots \\ \frac{\partial R}{\partial w^{(D)}} \end{bmatrix} \in \mathbb{R}^D$$

where $\frac{\partial R}{\partial w^{(j)}}$ denotes the partial derivative w.r.t. variable $w^{(j)}$

Recap: Properties of gradient

- The gradient at a point $\mathbf{w}$ has the direction of greatest increase of the function at $\mathbf{w}$
- Its magnitude is the rate of increase in that direction

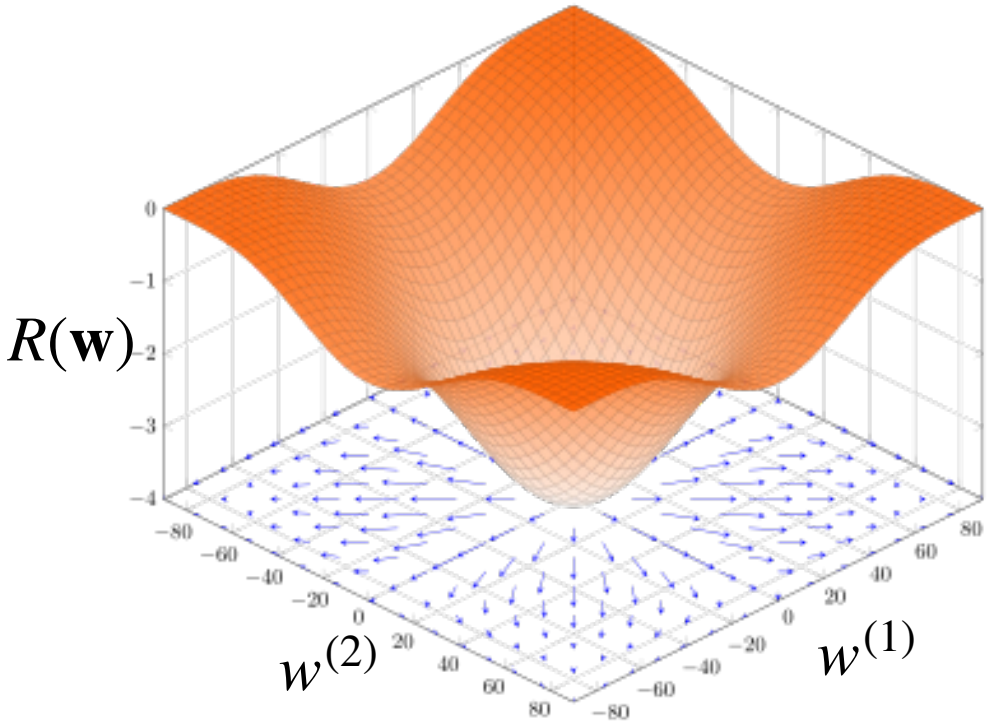


Figure from Wikipedia

Minimizing multi-variate functions

- As in the 1D case, to minimize a function that has no closed-form solution, we can exploit the fact that the gradient points in the direction of maximum function increase

- To decrease the function, we therefore want to move in the direction opposite to the gradient

- This yields an iterative algorithm that generalizes the one-variable one we saw before to multiple variables
  - This is called *gradient descent* (or steepest descent)

Gradient descent

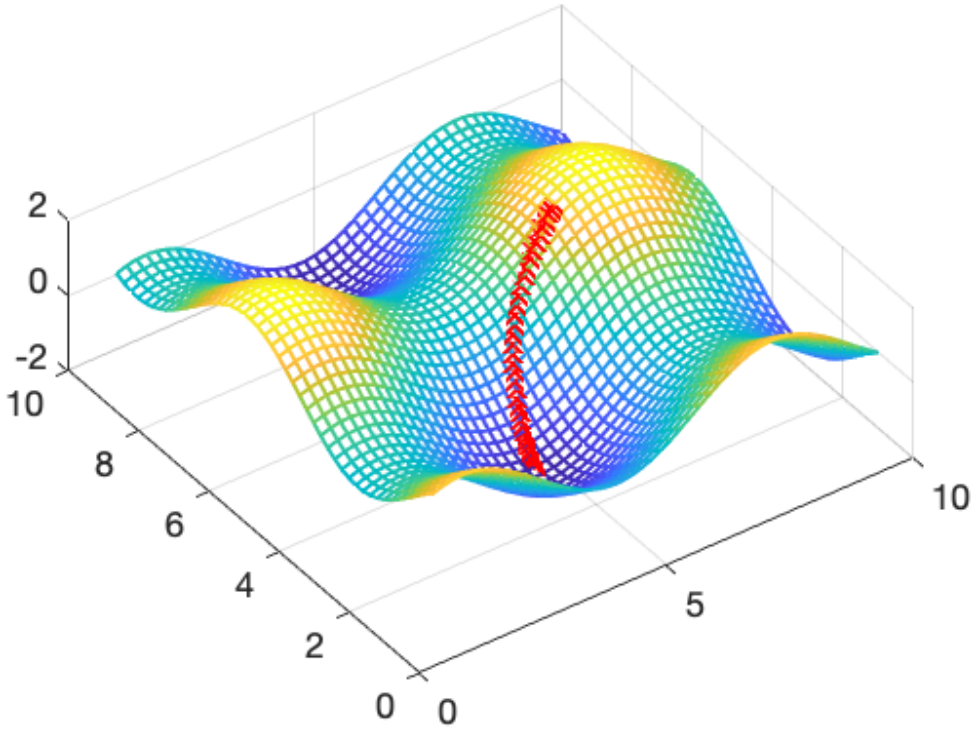
- Algorithm:
  1. Initialize $\mathbf{w}_0$ (e.g., randomly)
  2. While not converged
     - 2.1. Update $\mathbf{w}_k \leftarrow \mathbf{w}_{k-1} - \eta \nabla R(\mathbf{w}_{k-1})$

- The learning rate η has the same effect as in the 1D case
- Convergence can again be measured by
  - Thresholding the change in function value
  - Thresholding the change in parameters, e.g., $\|\mathbf{w}_{k-1} - \mathbf{w}_k\| < \delta_w$

Minimizing a non-convex function

- Example: Sine- and cosine-based function of 2 variables

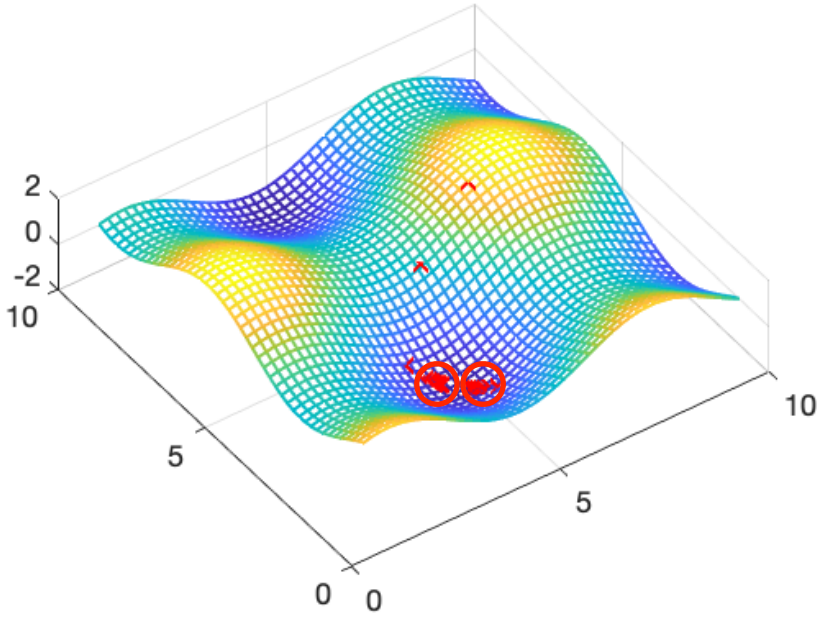
$$R(\mathbf{w}) = \sin\left(w^{(1)}\right) + \cos\left(w^{(2)}\right) \qquad \nabla R(\mathbf{w}) = \begin{bmatrix} \cos\left(w^{(1)}\right) \\ -\sin\left(w^{(2)}\right) \end{bmatrix}$$



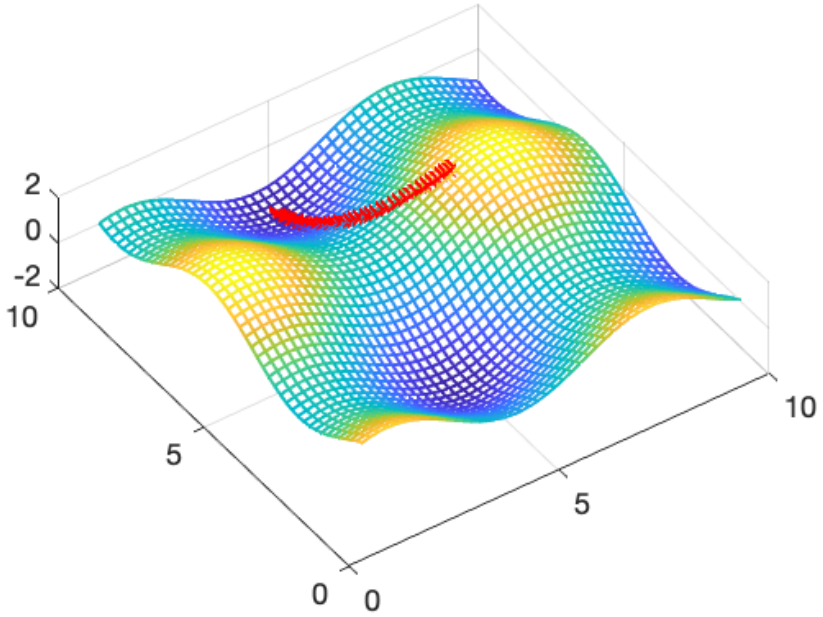
Gradient descent

- The same issues as in the 1D case can arise:
  - Different learning rates can yield different results
  - Different initializations can yield different results

$\eta = 2$ (vs 0.1)
Jumps between 2 solutions



$\mathbf{w}_0 = \begin{bmatrix} 7 \\ 6.5 \end{bmatrix}$ (vs $\begin{bmatrix} 7 \\ 6 \end{bmatrix}$)



End of the interlude

Back to Logistic Regression

Logistic regression: Training

- With binary ground-truth labels y_i, the cross-entropy can be written as

$$R(\mathbf{w}) = -\sum_{i \in \text{positive samples}} \ln(\hat{y}_i(\mathbf{w})) - \sum_{i \in \text{negative samples}} \ln(1 - \hat{y}_i(\mathbf{w}))$$

- To minimize it via gradient descent, we need to compute its gradient w.r.t. $\mathbf{w}$

Logistic regression: Gradient

- To compute the gradient w.r.t. $\mathbf{w}$, let us first look at the derivative of the logistic function

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

- One can derive that

$$\frac{d\sigma(a)}{da} = \sigma(a) \cdot (1 - \sigma(a))$$

Logistic regression: Gradient

- Then, looking at the positive samples and following the chain rule, we have

$$\nabla\left(-\sum_{i\in\text{positive}}\ln\sigma(\mathbf{w}^T\mathbf{x}_i)\right)=-\sum_{i\in\text{positive}}\frac{\partial\ln\sigma}{\partial\sigma}\frac{\partial\sigma(a_i)}{\partial a_i}\frac{\partial a_i}{\partial\mathbf{w}}$$

$$=-\sum_{i\in\text{positive}}\frac{1}{\sigma(\mathbf{w}^T\mathbf{x}_i)}\sigma(\mathbf{w}^T\mathbf{x}_i)\left(1-\sigma(\mathbf{w}^T\mathbf{x}_i)\right)\mathbf{x}_i$$

$$=-\sum_{i\in\text{positive}}\left(1-\hat{y}_i\right)\mathbf{x}_i=\sum_{i\in\text{positive}}\left(\hat{y}_i-1\right)\mathbf{x}_i$$

- Similarly, for the negative examples, we have

$$\nabla\left(-\sum_{i\in\text{negative}}\ln(1-\sigma(\mathbf{w}^T\mathbf{x}))\right)=\sum_{i\in\text{negative}}\hat{y}_i\mathbf{x}_i$$

Logistic regression: Gradient

- Finally, this can be put back together to obtain

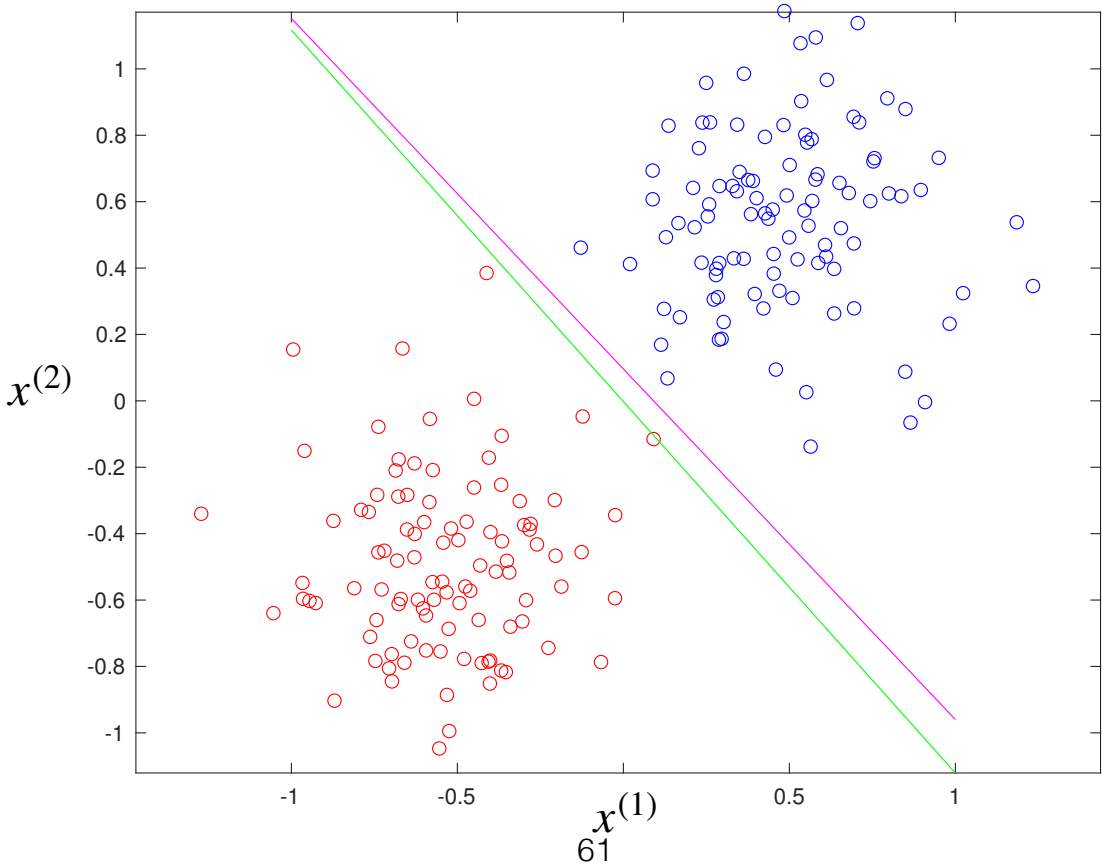
$$\nabla R(\mathbf{w}) = \sum_{i=1}^{N} \left(\hat{y}_i - y_i\right) \mathbf{x}_i$$

where y_i is the true label for sample i, i.e., 1 for positive samples and 0 for negative samples

- Intuitively, the gradient is given by the error of the current prediction $\left(\hat{y}_i - y_i\right)$, multiplied by the input

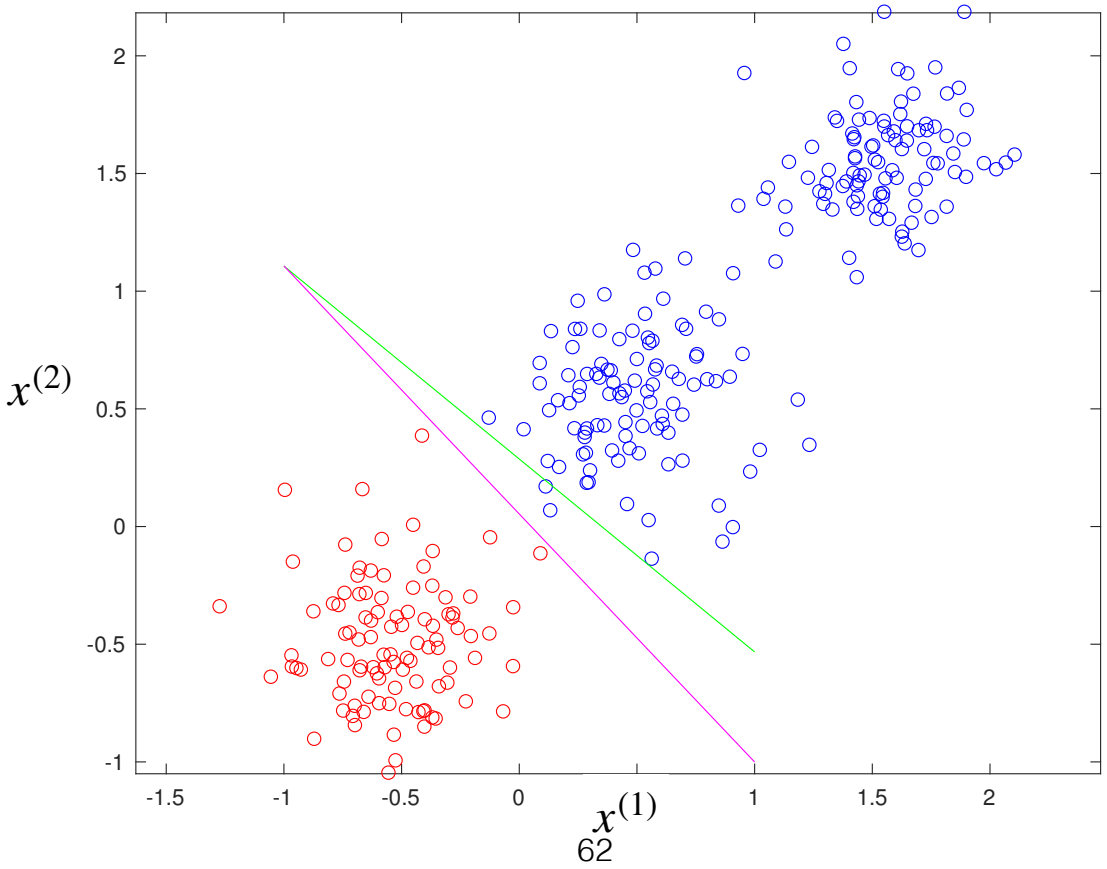
Logistic regression

- Example: 2D inputs, 2 classes
  - Least-square classification in green, logistic regression in magenta



Logistic regression

- Now, with additional samples, the boundary remains correct
  - Least-square classification in green, logistic regression in magenta

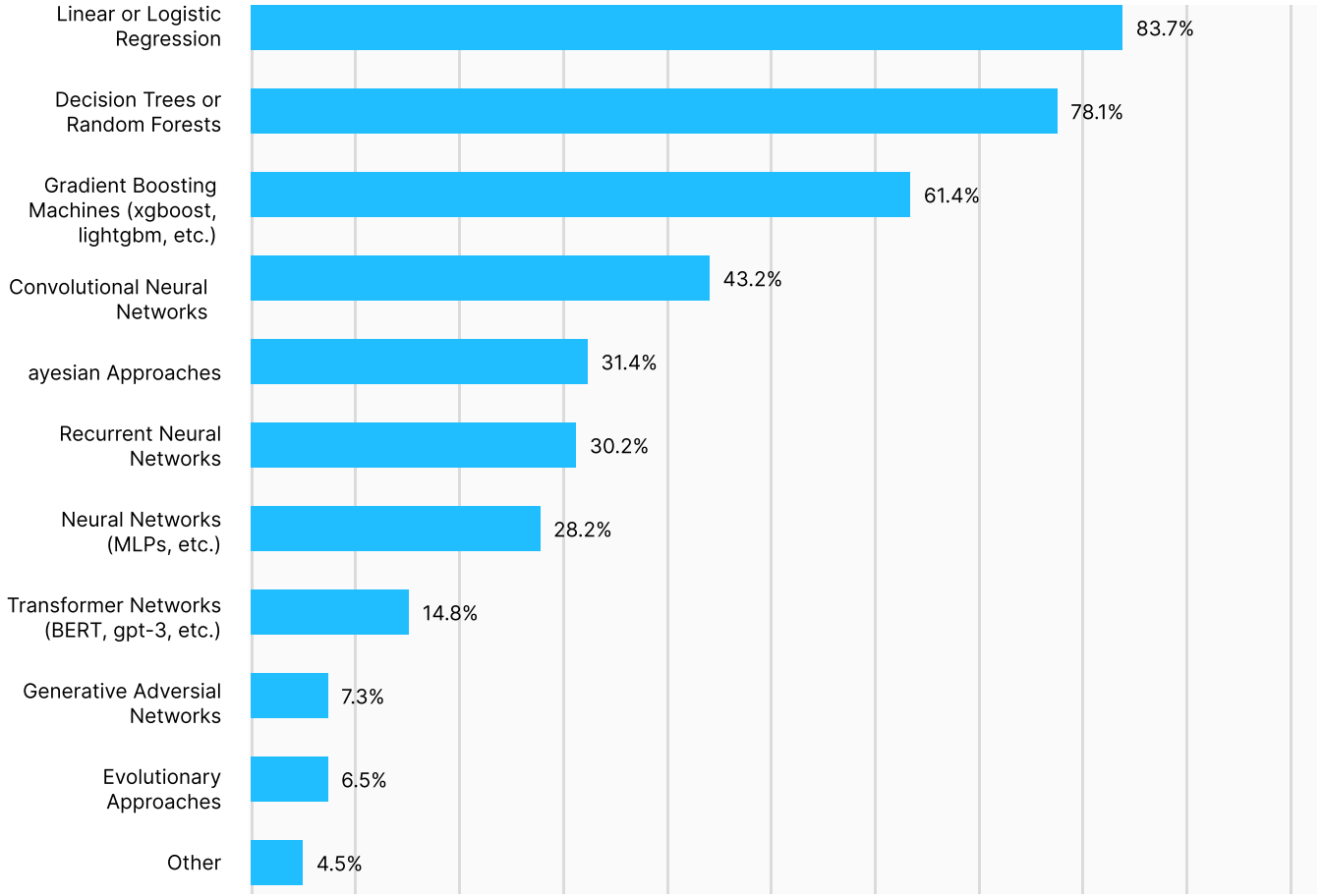


Logistic regression: Demo

- https://playground.tensorflow.org/#activation=sigmoid&batchSize=10&dataset=gauss&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=&seed=0.92907&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false

Logistic regression in practice

- According to a 2020 survey among data scientist, the answer to the question "What kind of data science methods are used at work" gave the statistics



https://www.kaggle.com/kaggle-survey-2020

Exercises

- Describe two main differences between linear regression and logistic regression.

- Describe one similarity between linear regression and logistic regression.

Recap: Model evaluation

- Once an ML model is trained, one would typically understand how well it performs on unseen test data
  - At this stage, the parameters of the model are fixed
  - Recall that the training and testing data must be separated!

- During this evaluation, one compares the predictions of the model with the true annotations of the test data
  - In contrast to the training stage, the model parameters are *not* updated
  - The evaluation metric may directly be the loss function, but may also differ from it

Classification evaluation

- Confusion matrix: Binary case (e.g., spam vs non-spam email)

True class → / Hypothesized class ↓	Pos	Neg
Yes	TP	FP
No	FN	TN
	P=TP+FN	N=FP+TN

TP: True positive (number of samples correctly classified as positive)
FP: False positive (number of samples incorrectly classified as positive)

TN: True negative (number of samples correctly classified as negative)
FN: False negative (number of samples incorrectly classified as negative)

P: Total number of positive samples
N: Total number of negative samples

Figure from N. Japkowicz's ICMLA tutorial

Classification evaluation

- Confusion matrix: Binary case
  - Numerical example for a dataset with 500 positive and 500 negative samples

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

Figure from N. Japkowicz's ICMLA tutorial

Classification metrics

- Accuracy: Percentage of correctly classified samples

$$Acc = \frac{TP + TN}{P + N}$$

- Exercise: Compute the accuracy for the following data

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

Figure from N. Japkowicz's ICMLA tutorial

What's wrong with accuracy?

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

True class →	Pos	Neg
Yes	400	300
No	100	200
	P=500	N=500

- The two classifiers (matrices) above yield the same accuracy
- But their behavior is very different
  - Left: Weak positive recognition rate, but strong negative recognition rate
  - Right: Strong positive recognition rate, but weak negative recognition rate

Figure from N. Japkowicz's ICMLA tutorial

Classification metrics

- Precision: Percentage of samples classified as positives that are truly positives

$$Precision = \frac{TP}{TP + FP}$$

- Exercise: Compute the precision for this data

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

Figure from N. Japkowicz's ICMLA tutorial

Classification metrics

- Recall: Percentage of positives samples that are correctly classified as positives

$$Recall = \frac{TP}{P}$$

- Exercise: Compute the recall for this data

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

Figure from N. Japkowicz's ICMLA tutorial

What's wrong with precision/recall?

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

True class →	Pos	Neg
Yes	200	100
No	300	0
	P=500	N=100

- The two classifiers (matrices) above yield the same precision and recall (0.667 and 0.4) (The datasets are different)
- But their behavior is very different
  - Same positive recognition rate
  - Very different negative recognition rates (strong on left, but nil on right)
- Accuracy would clearly show this

Figure from N. Japkowicz's ICMLA tutorial

Classification metrics

- False positive rate: Percentage of negative samples that are incorrectly classified as positives

$$FP\ rate = \frac{FP}{N}$$

- Example:

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

$$FP\ rate = \frac{100}{500} = 0.2$$

Figure from N. Japkowicz's ICMLA tutorial

Classification metrics

- F1 score: Single number that combines precision and recall

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

- Example:

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

$$F1 = 2 \cdot \frac{0.667 \cdot 0.4}{0.667 + 0.4} = 0.5$$

Figure from N. Japkowicz's ICMLA tutorial

ROC curve

- Many classifiers output a score/confidence for their predictions (e.g., $\hat{y}$ is a continuous value between 0 and 1 in logistic regression)

- The decision between positive and negative can be computed by thresholding this score:

$$\text{label}_i = 1 \text{ if } \hat{y}_i \geq \tau$$

where τ is a given threshold

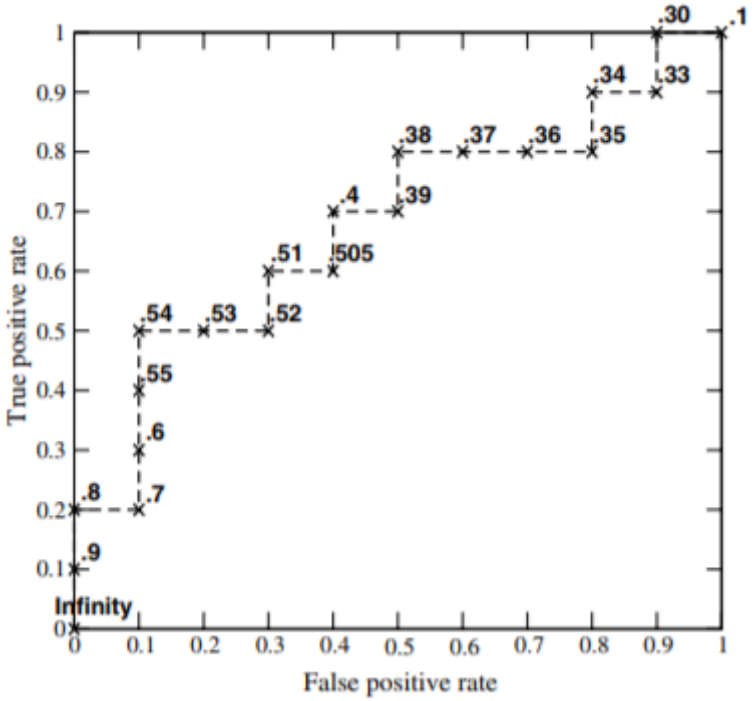
- The Receiver Operating Characteristic (ROC) curve plots the true positive rate (recall) as a function of the false positive rate, obtained by varying the score threshold

Constructing an ROC curve

- Example with 10 positive and 10 negative samples

Inst#	Class	Score	Inst#	Class	Score
1	p	.9	11	p	.4
2	p	.8	12	n	.39
3	n	.7	13	p	.38
4	p	.6	14	n	.37
5	p	.55	15	n	.36
6	p	.54	16	n	.35
7	n	.53	17	p	.34
8	n	.52	18	n	.33
9	p	.51	19	p	.30
10	n	.505	20	n	.1

Samples sorted by score
(*Class* denotes the true label)

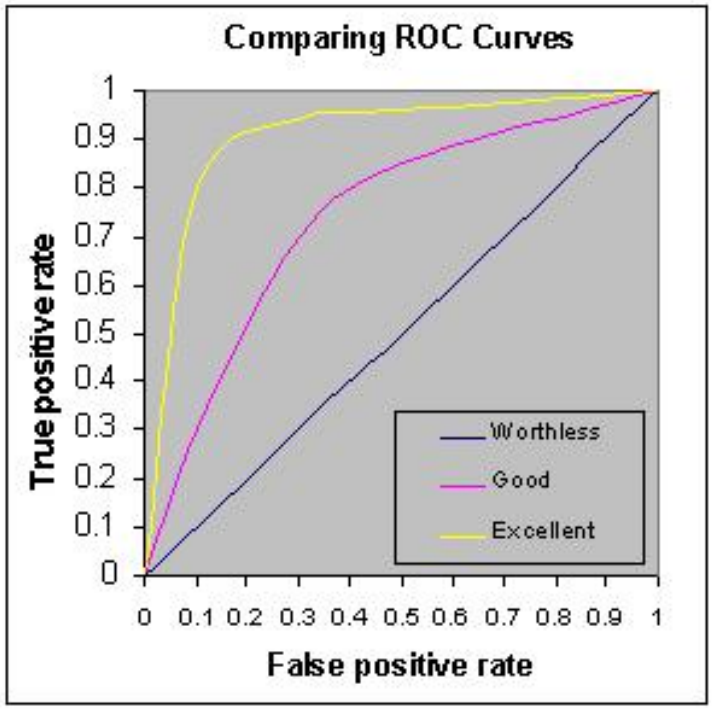


ROC curve

Figure from Z. Waks's slides

Classification metrics

- ROC curves

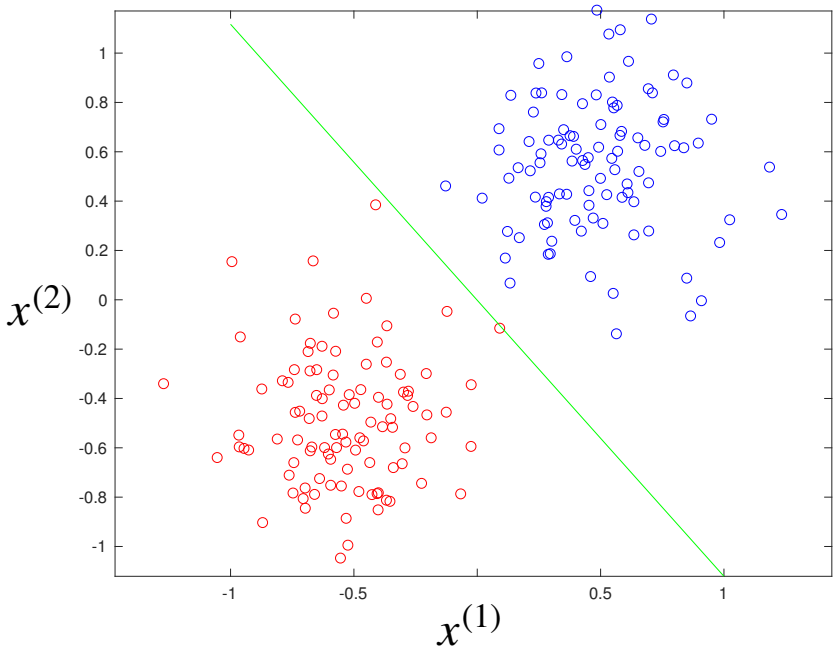


- The Area Under the ROC Curve (AUC) can act as a metric
  - The maximum AUC is 1
  - Random predictions give and AUC of 0.5

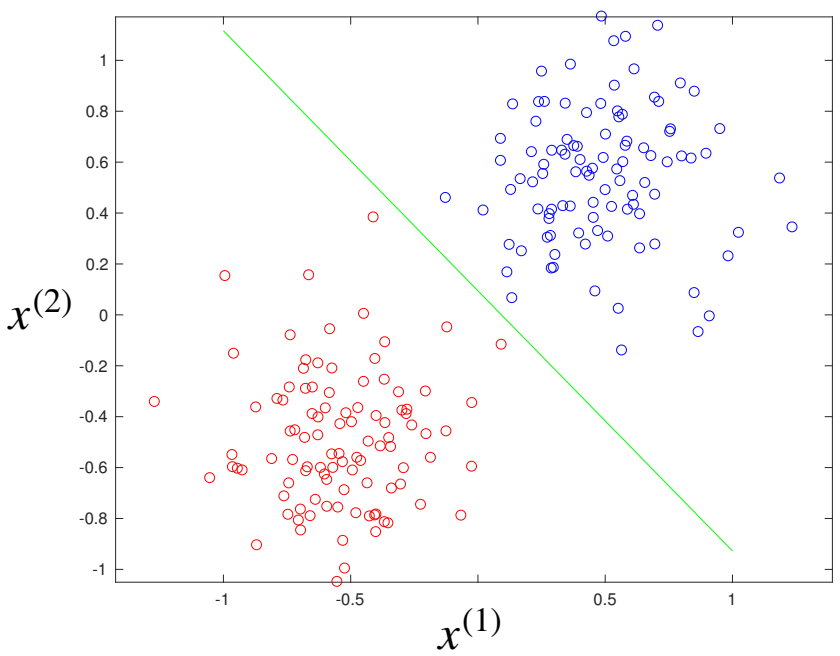
Figure from Z. Waks's slides

Decision boundaries

- Different classifiers have different decision boundaries



Least-square classification



Logistic regression

- Can we define what a good decision boundary is?
  - We will discuss this further next week