MACHINE LEARNING WORKSHOP

Presenter:

MOHAMMAD RASHEDI

Outline

- Introduction to Machine Learning
- Motivation
- Supervised learning
- Unsupervised learning
- Coding exercises

Definition of Machine Learning

- Arthur Samuel (1959):

Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.

Tom Mitchell (1998):

Well-posed Learning Problem: A computer program is said to learn from experience ***E*** with respect to some task ***T*** and some performance measure ***P***, if its performance on ***T***, as measured by ***P***, improves with experience ***E***.

Example

- Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam. What are the task ***T***, experience ***E***, and performance measure ***P*** in this setting?

- Classify emails as spam or not spam. → ***T***
- Watching you label emails as spam or not spam. → ***E***
- The number (or fraction) of emails correctly classified as spam/not spam. → ***P***

Traditional Programming vs. ML

Traditional programming

Machine learning

ML use cases

- ML is used when:

✓ Human expertise does not exist (navigating on Mars)

✓ Humans can't explain their expertise (speech recognition)

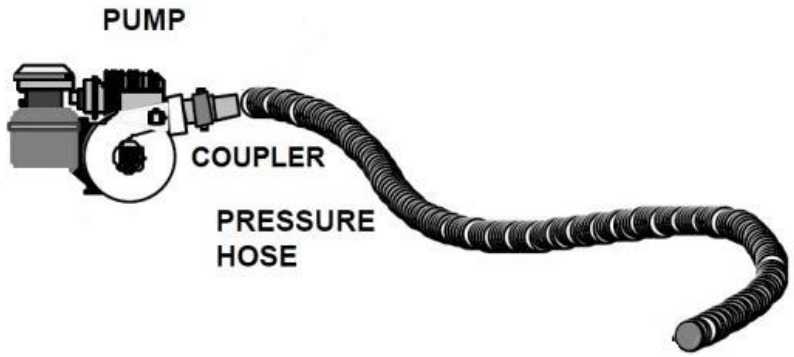
✓ Models must be customized (personalized medicine)

✓ Models are based on huge amounts of data (no governing physics)

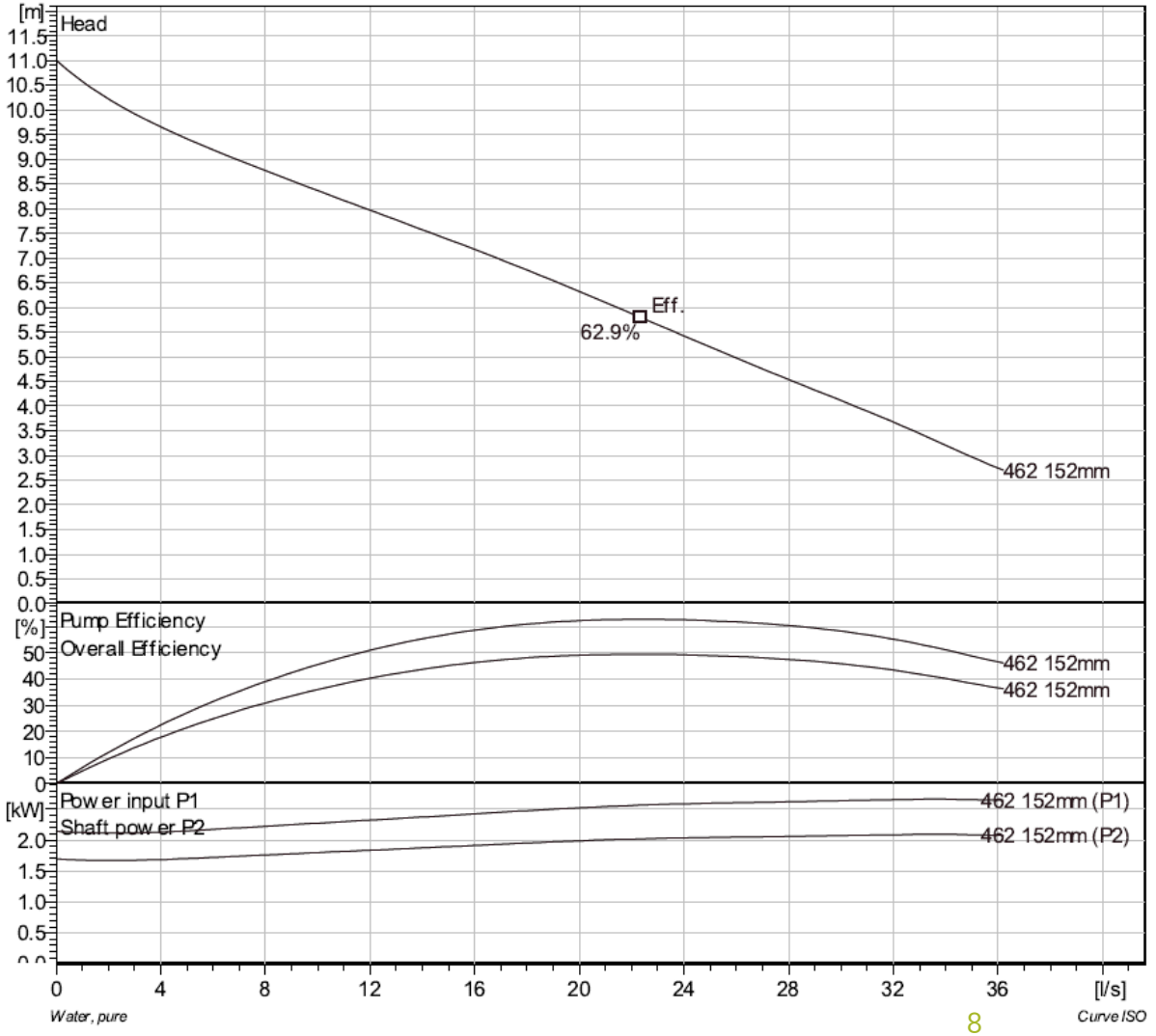
- Learning isn't always useful:

✓ There is no need to "learn" to calculate payroll

Motivation



How to find pressure and flow in pipe based on pump's input power, pipe length, etc?



Types of Machine Learning

Supervised Learning

Unsupervised Learning

Reinforcement Learning

Supervised Learning

If output is continuous → **Regression**

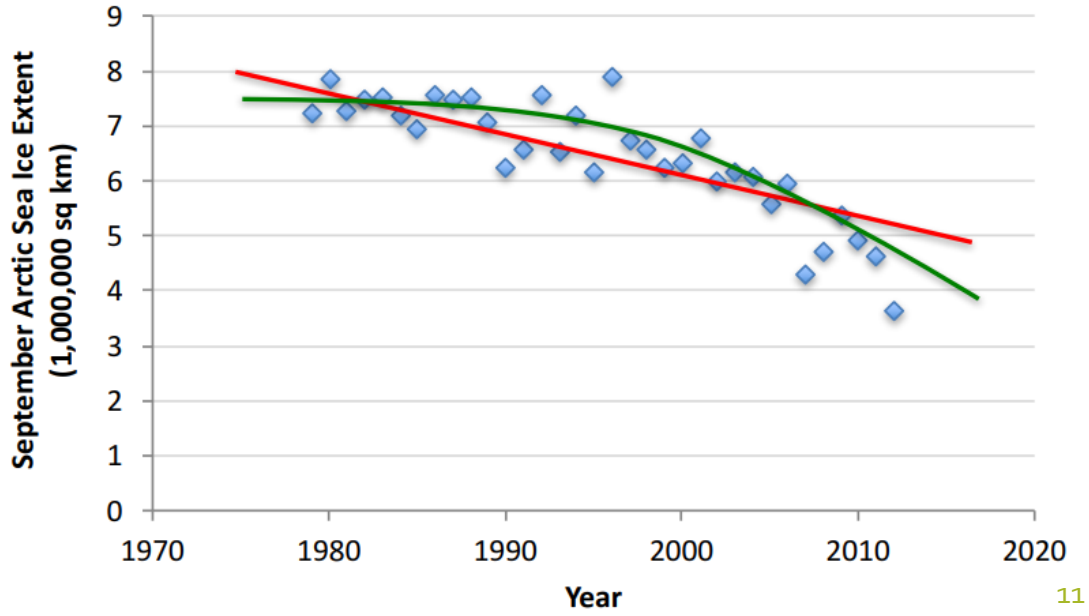
If output is discrete → **Classification**

Supervised Learning: Regression

- Given $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$
- Learn a function $f(x)$ to predict y given x

✓ y is real-valued (continuous) => Regression

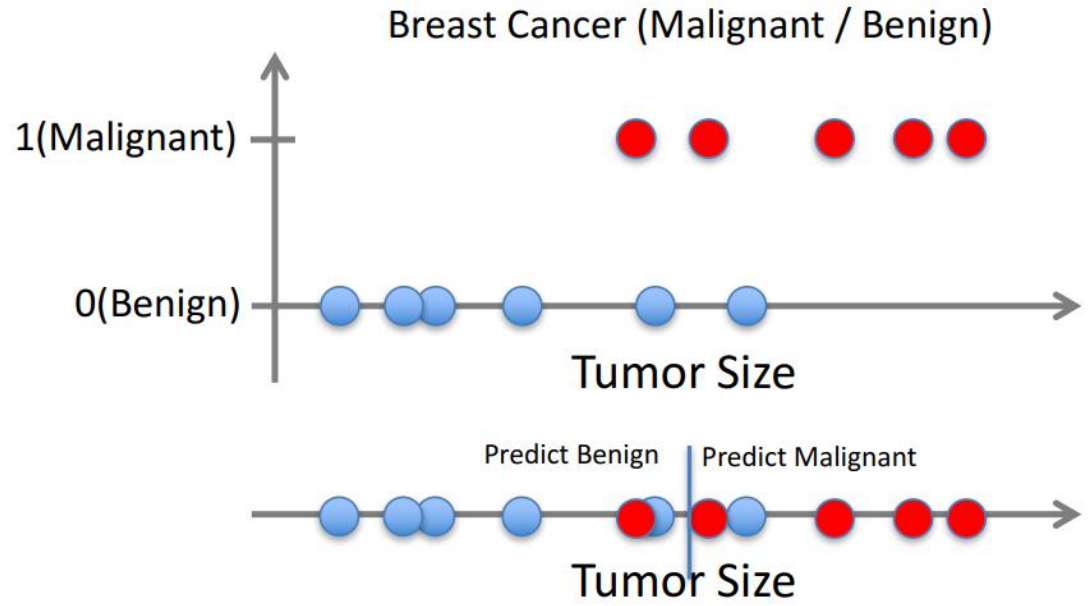
Blue dots are given right answers



Supervised Learning: Classification

- Given $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$
- Learn a function $f(x)$ to predict y given x

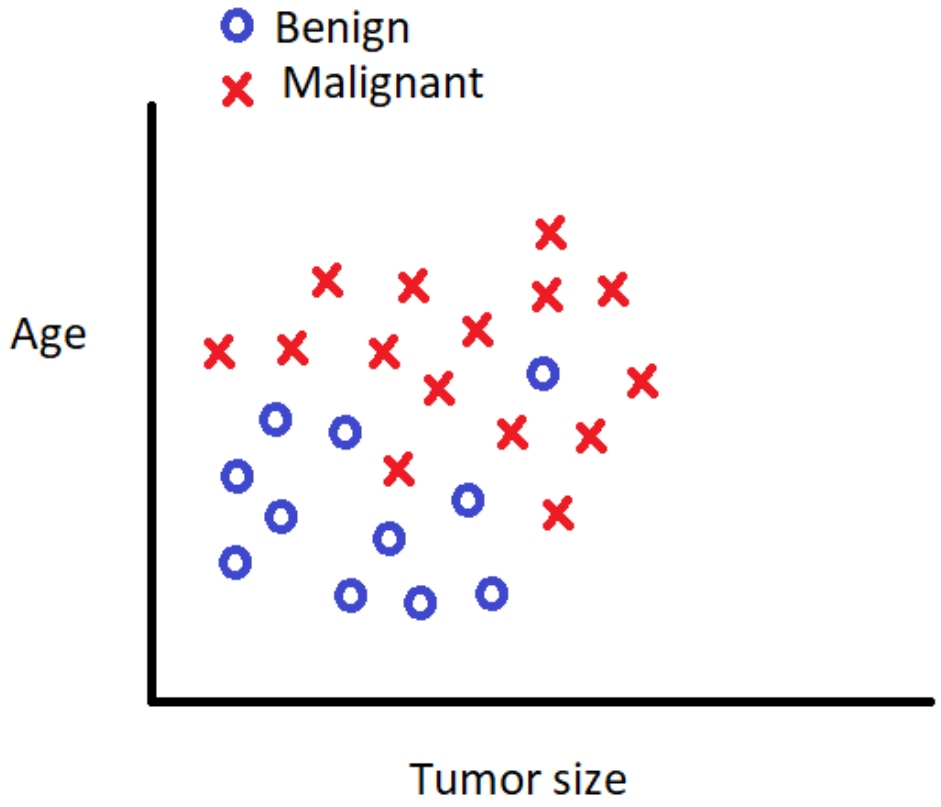
✓ y is categorical (discrete) => Classification



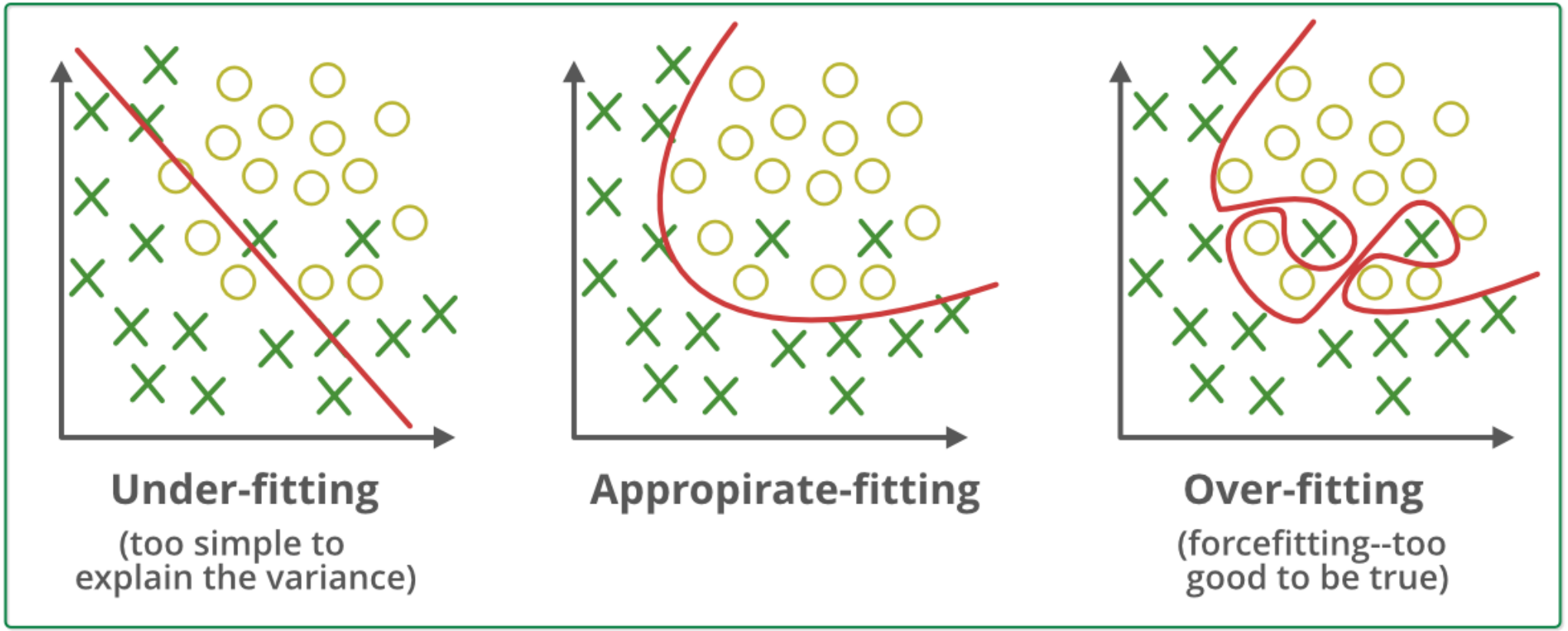
Supervised Learning: Classification

- The number of features can be more than one.

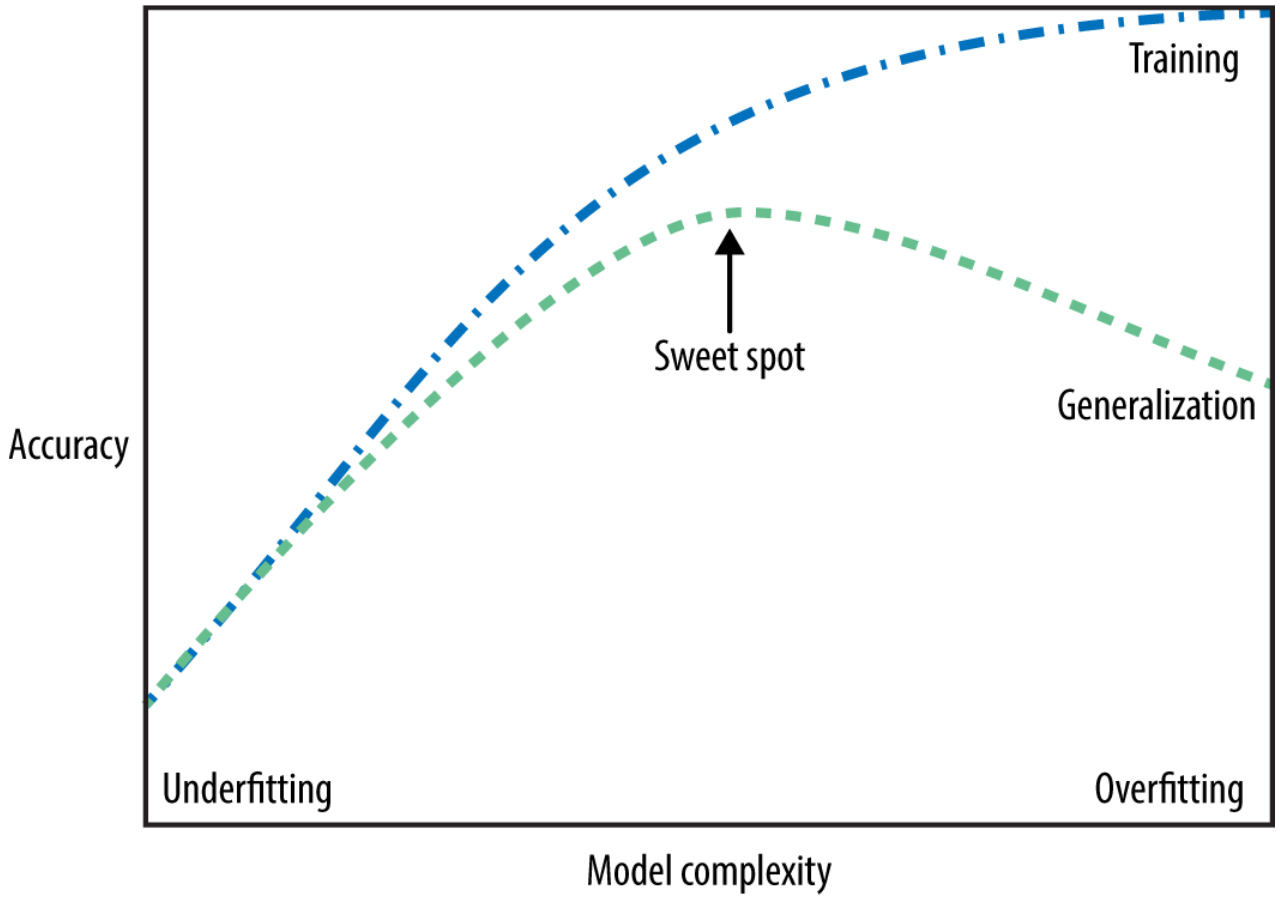
- What can be the best classifier in this example?

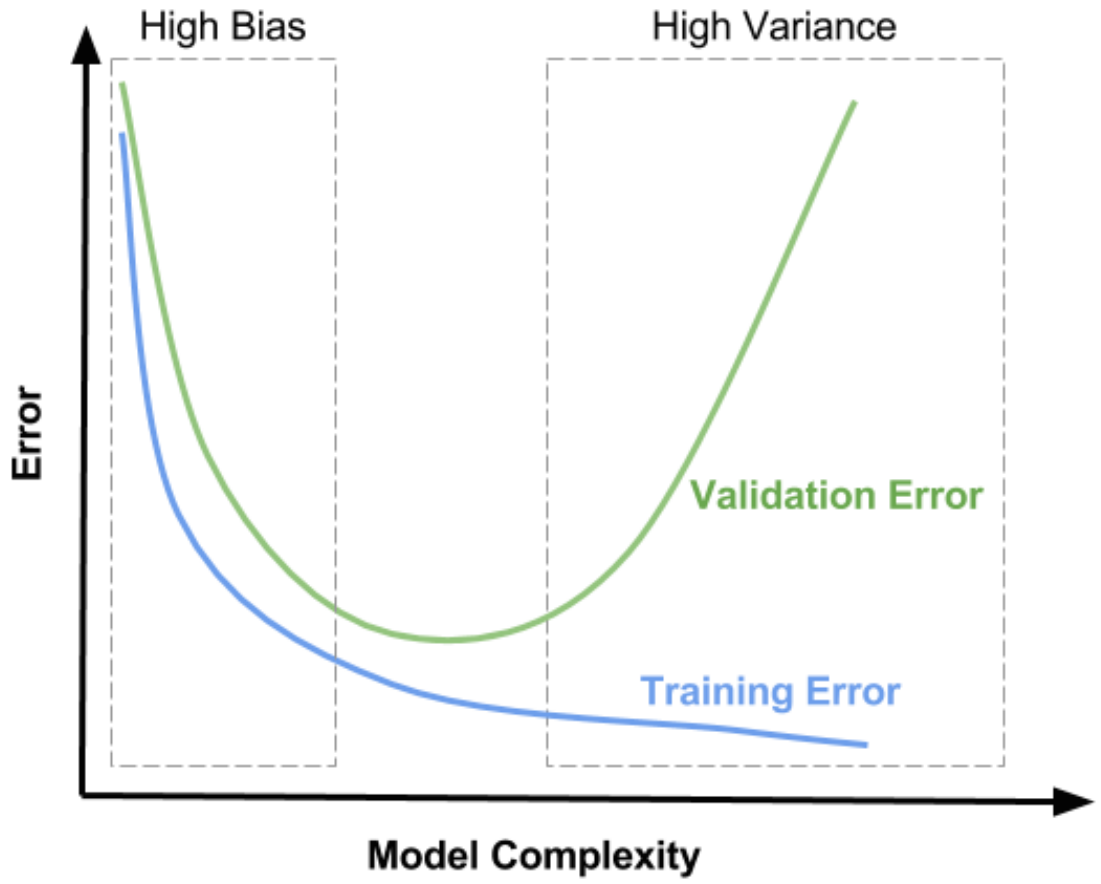


Supervised Learning: Classification



Bias-Variance trade-off in learning curve





Example:

- You're running a company, and you want to develop learning algorithms to address each of two problems:

- Problem 1:You have a large inventory of identical items. You want to predict how many of these items will sell over the next 3 months.

- Problem 2: You'd like software to examine individual customer accounts, and for each account decide if it has been hacked/compromised.

Should you treat these as classification or as regression problems?

Unsupervised Learning

- Question: Can you find some structure in the data?

Linear Regression (Least Squares)

Bivariate Regression model

(Education) x $\xrightarrow{y = \beta_0 + \beta_1 x}$ y (Income)

Multivariate regression model

(Education) x_1

(Sex) x_2

(Experience) x_3

(Age) x_4

$y = w_0 + w_1 x_1 + w_2 x_2 + \cdots$

y (Income)

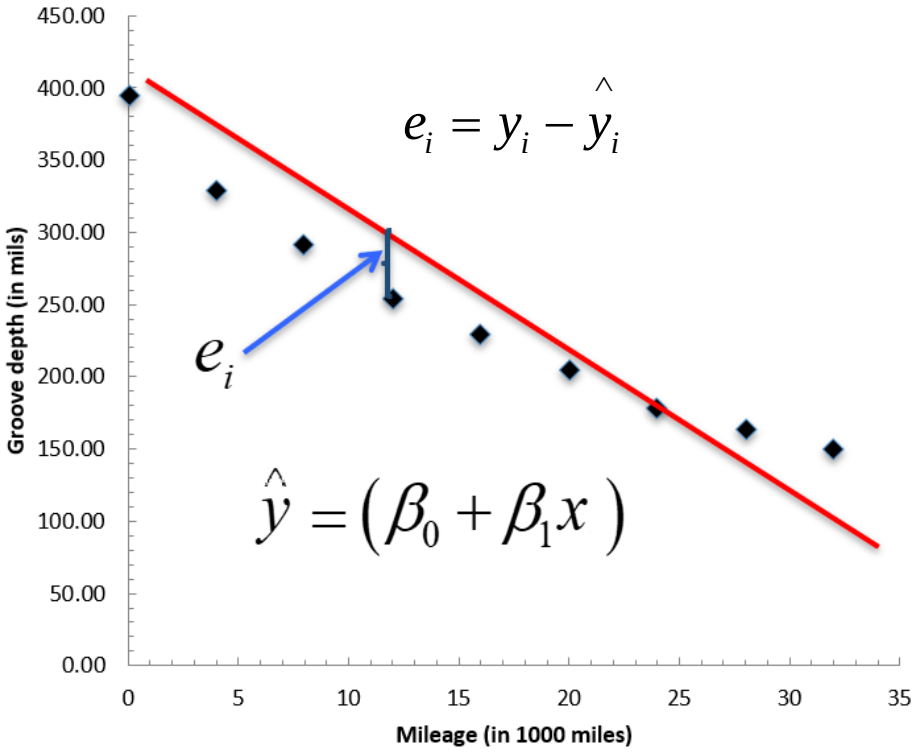
$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ x_3^{(i)} \\ x_4^{(i)} \end{bmatrix}$$

Linear Regression

- The difference between the fitted line and the real data is e_i.
- This e_i is the vertical difference.

- The objective is to minimize the sum of squares:

$$Q = \sum_{i=1}^{n} \left[y_i - \left(\beta_0 + \beta_1 x_i \right) \right]^2$$



Multivariate Linear Regression

- The model takes the form:

$$\hat{y}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- The model parameters are θs.
- The cost function will be:

$$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}(x^{(i)}) - y^{(i)})^2$$

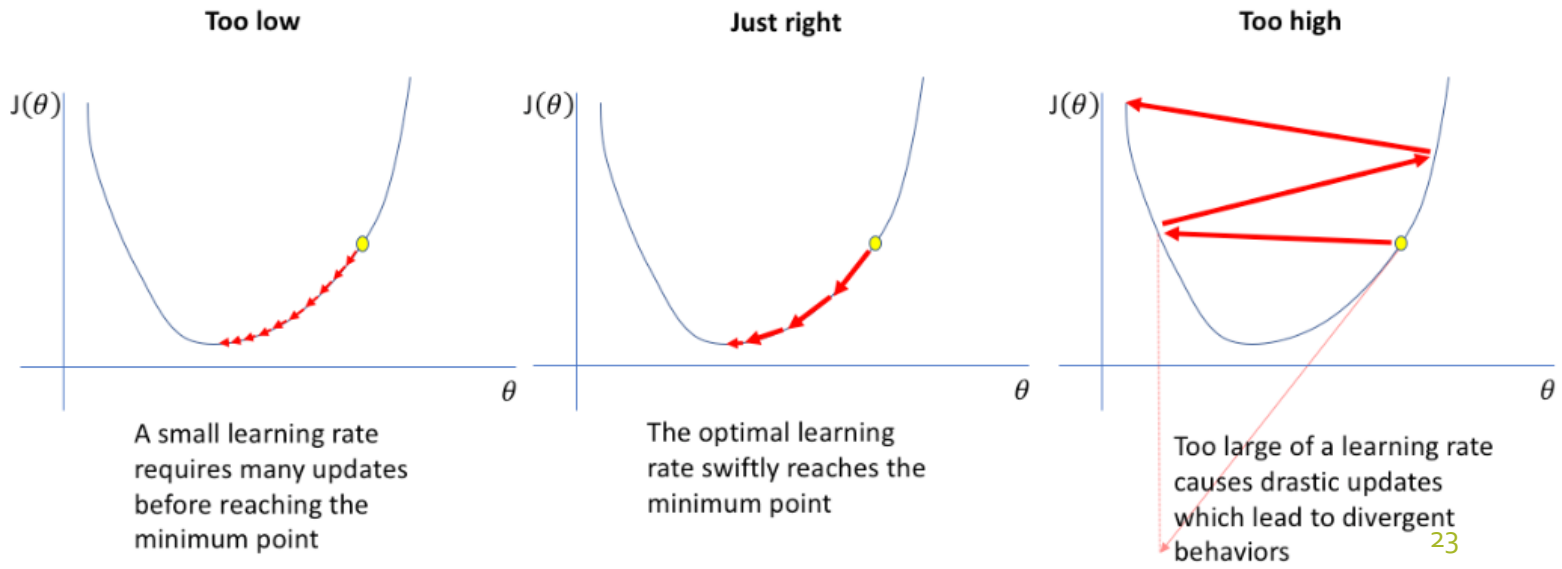
- Here, m is the number of training samples.
- To minimize the cost function, we may use gradient descent algorithm for each of the model parameters.

Gradient Descent

- This algorithm simultaneously updates every parameter θ_j.
- Repeat until convergence {

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n)\}$$

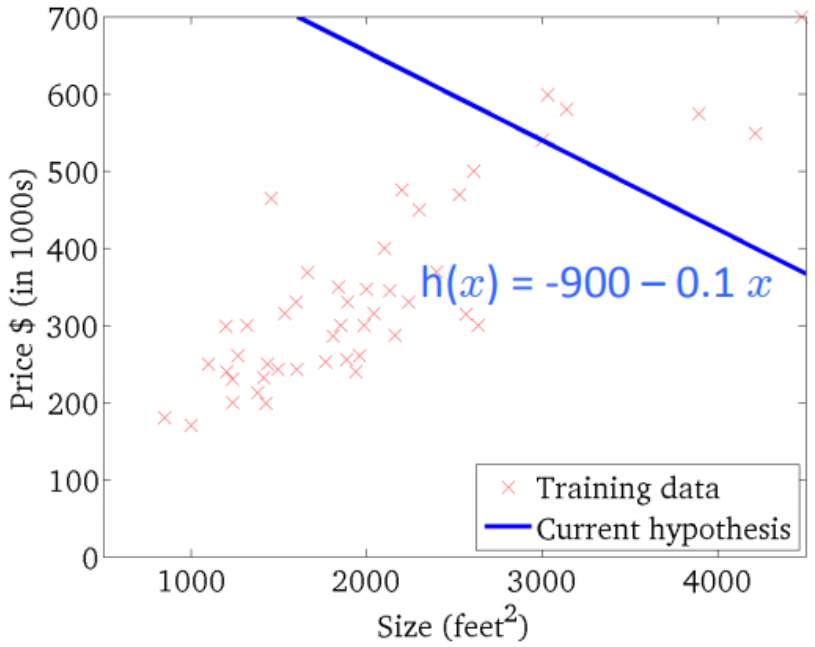
- α is the learning rate.



Gradient Descent

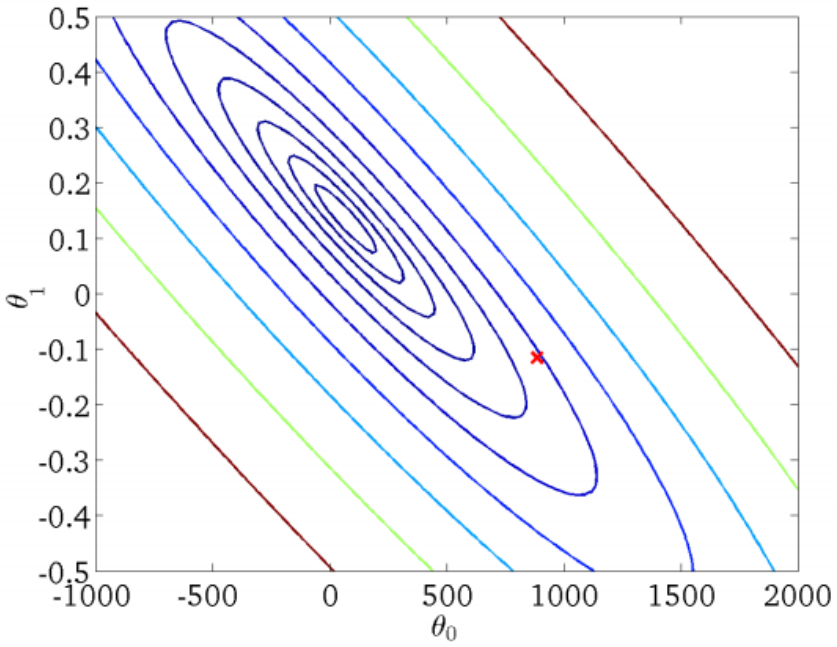
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

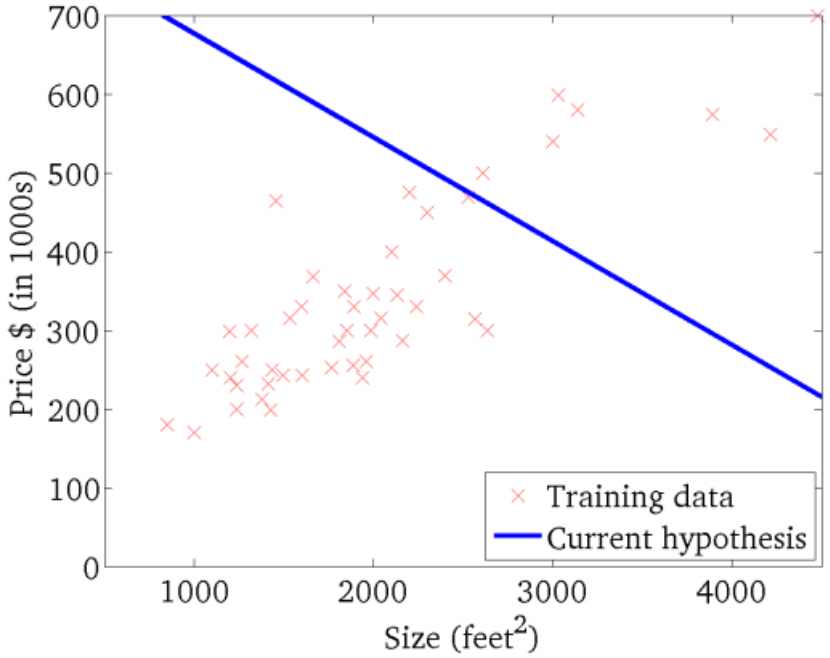
(function of the parameters θ_0, θ_1)



Gradient Descent

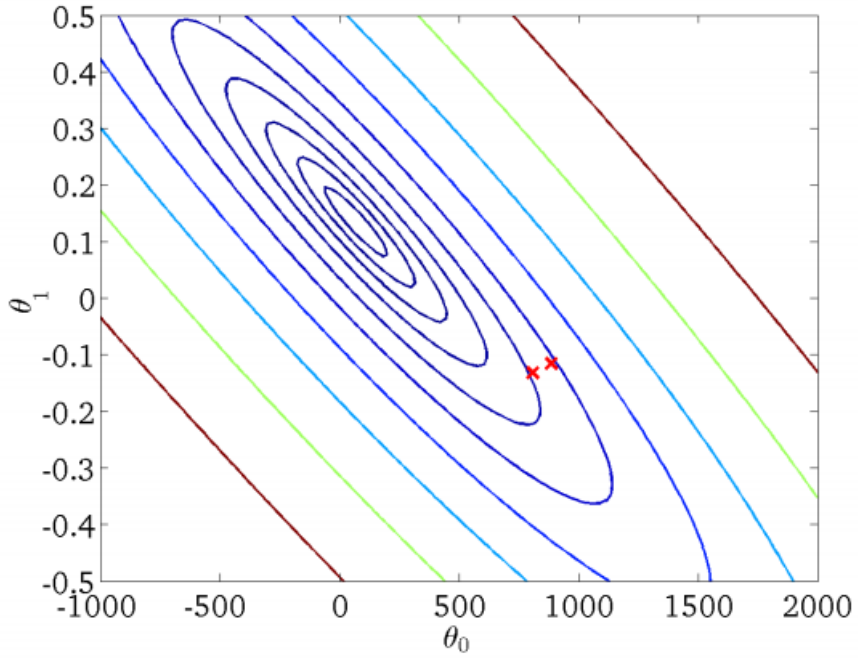
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

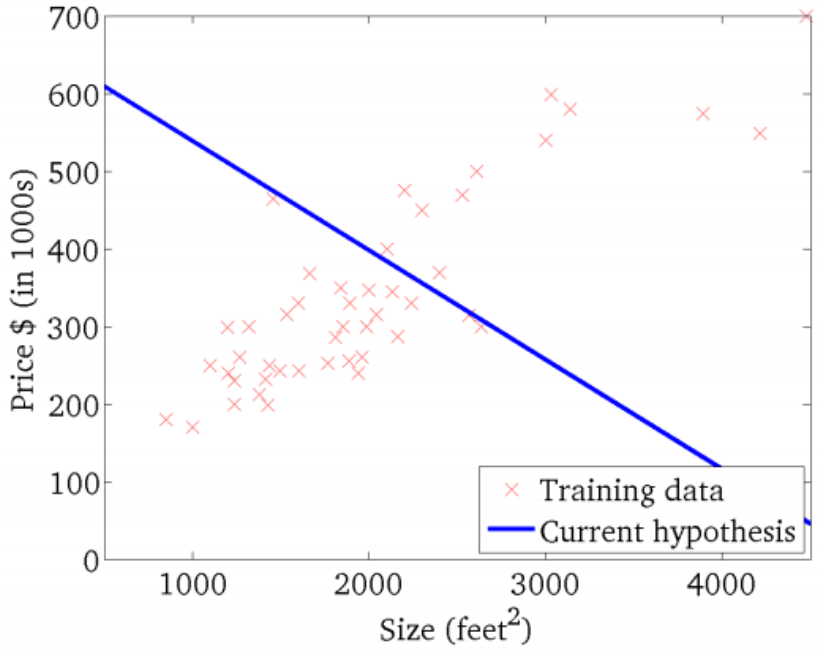
(function of the parameters θ_0, θ_1)



Gradient Descent

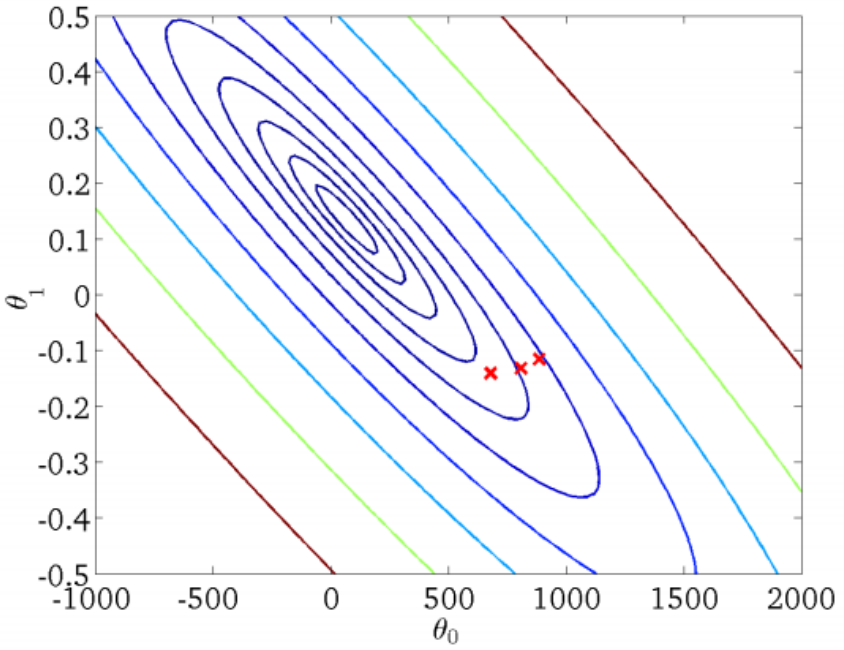
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

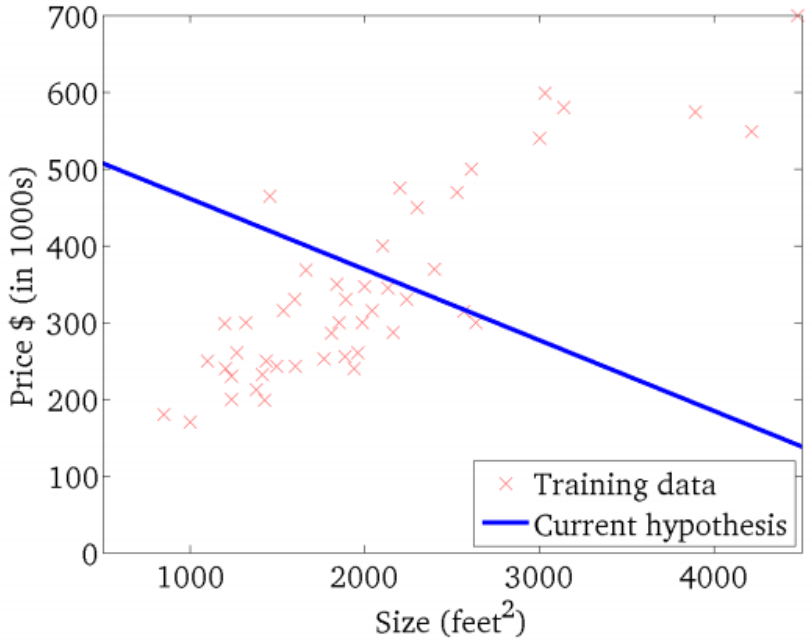
(function of the parameters θ_0, θ_1)



Gradient Descent

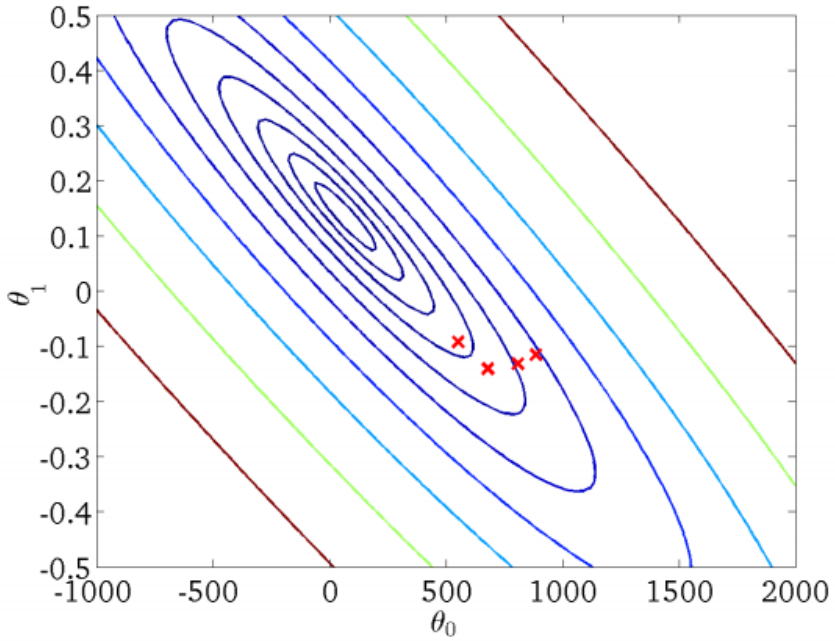
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

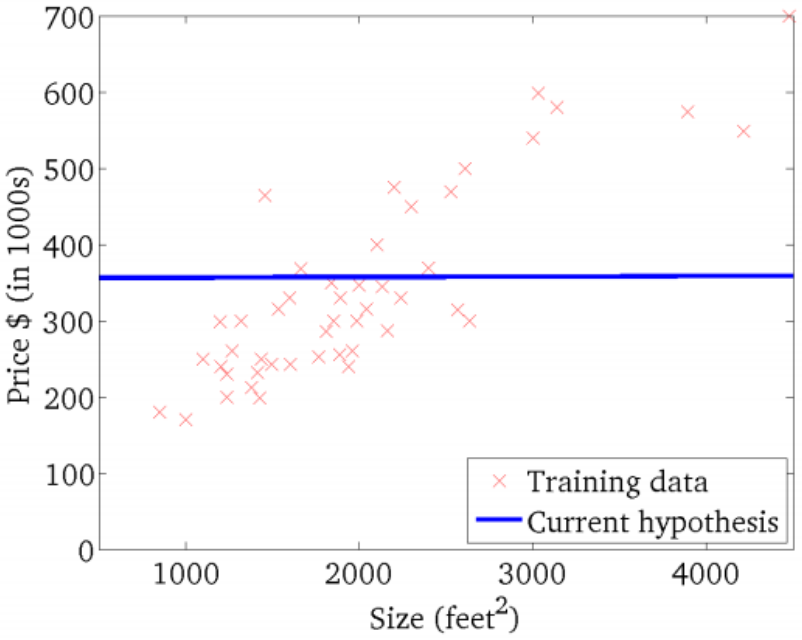
(function of the parameters θ_0, θ_1)



Gradient Descent

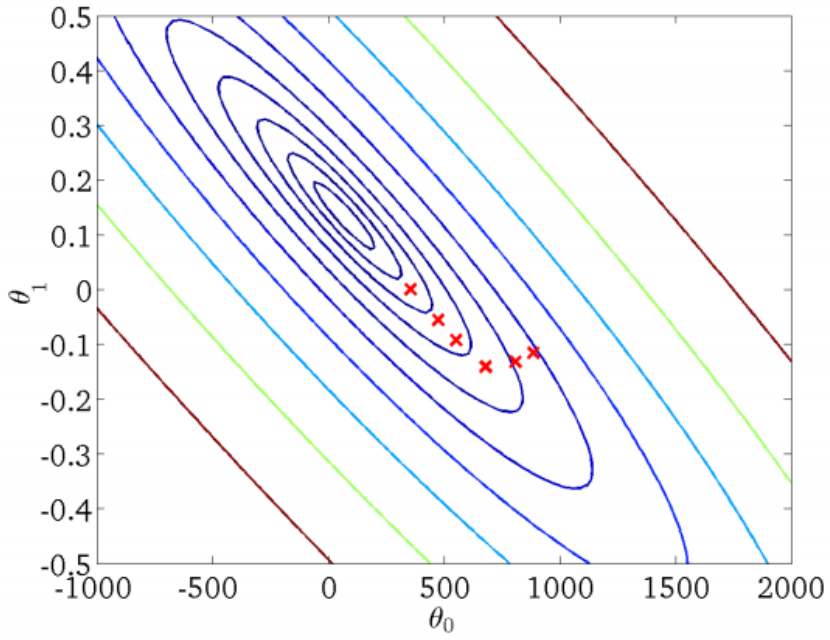
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

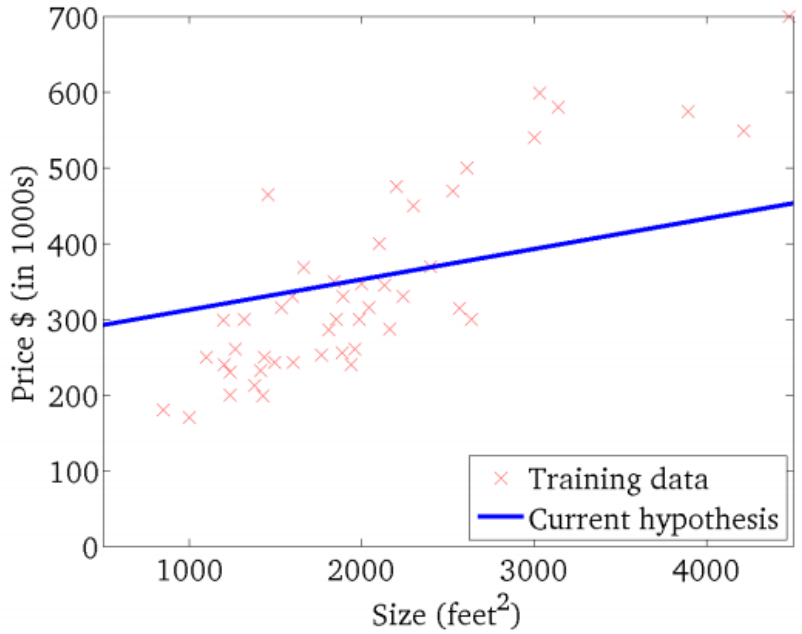
(function of the parameters θ_0, θ_1)



Gradient Descent

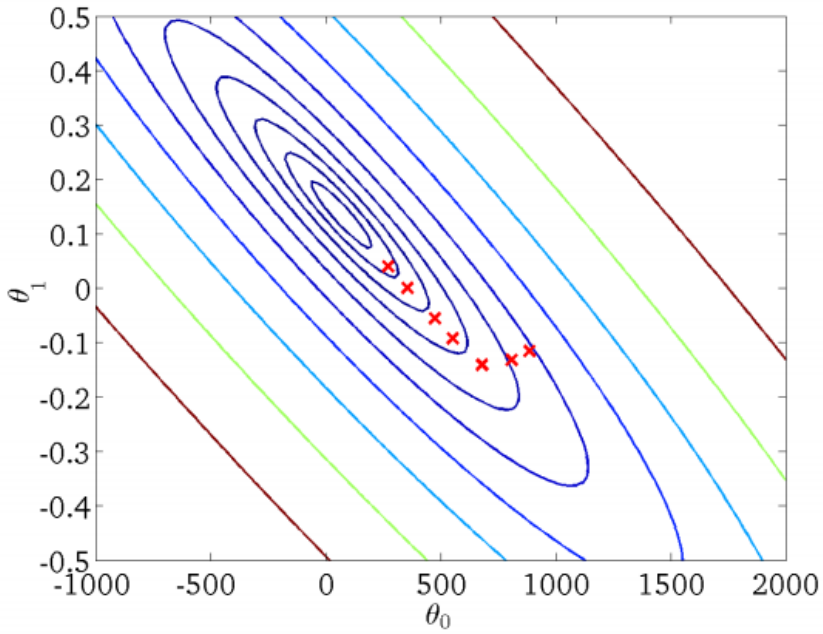
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

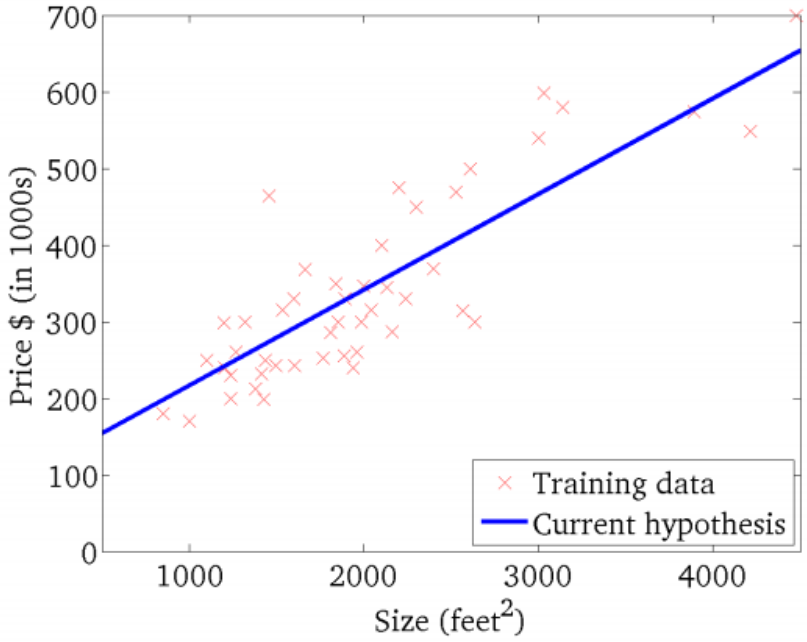
(function of the parameters θ_0, θ_1)



Gradient Descent

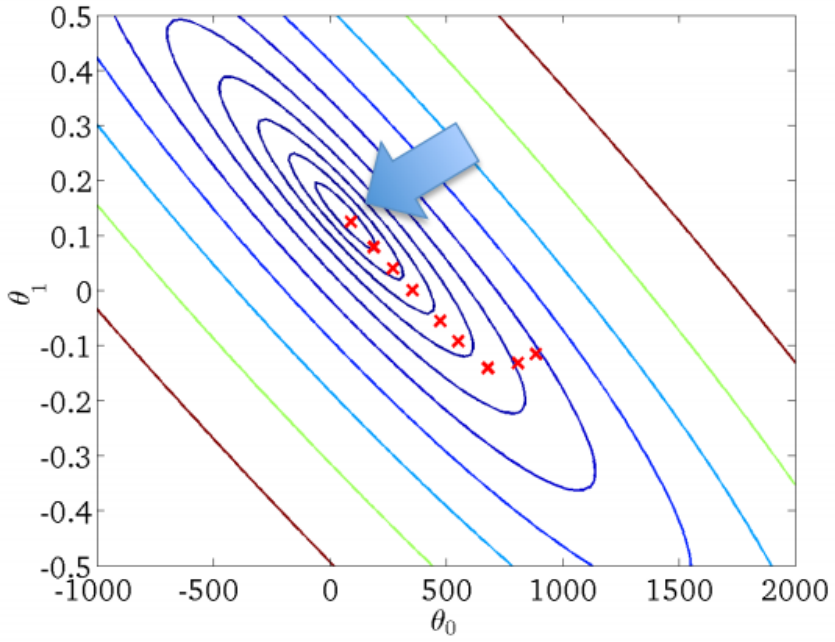
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



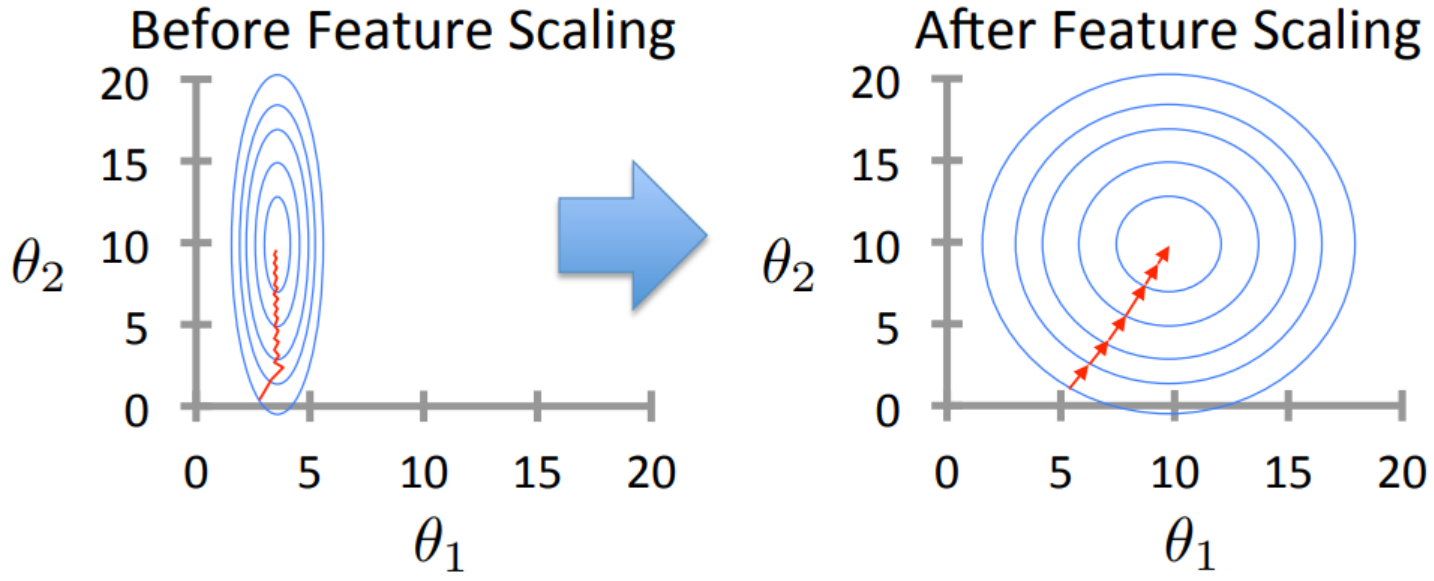
Feature scaling

- To have improved and faster learning, we need to scale the features.
- Rescale the features to have zero mean and unit variance

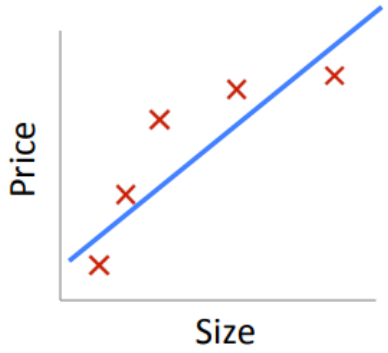
$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$$

$$s_j = \sqrt{\frac{\sum_{i=1}^{m}\left(x_j^{(i)} - \mu_j\right)^2}{m}}$$

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j} \text{ for all } j$$

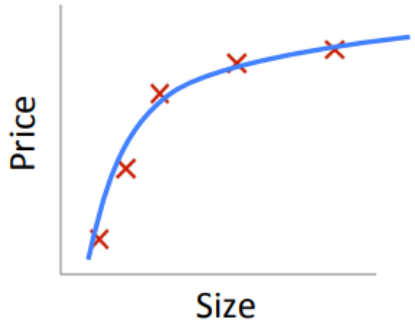


Overfitting Issue



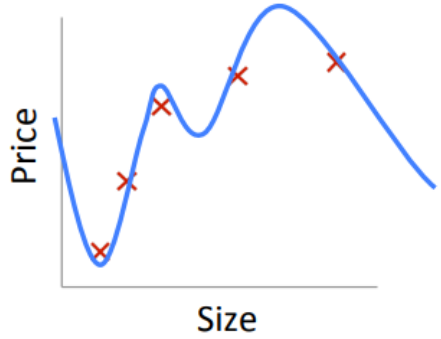
$\theta_0 + \theta_1 x$

Underfitting
(high bias)



$\theta_0 + \theta_1 x + \theta_2 x^2$

Correct fit



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Overfitting
(high variance)

- Overfitting:

  The learned model may fit the training set very well ($J(\theta) \approx 0$), but fails to generalize to new examples.

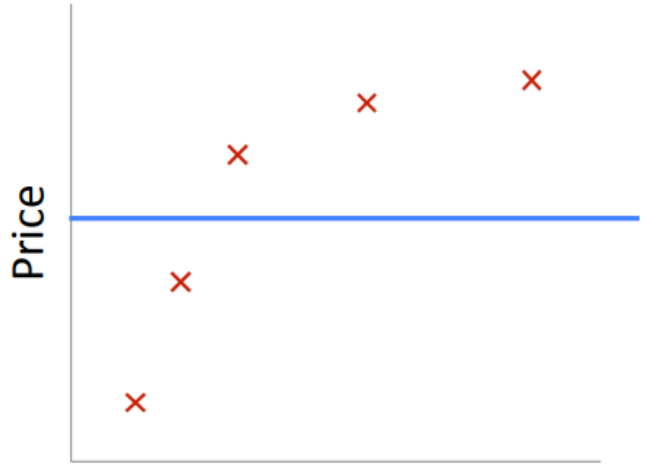
Regularization

- A method to automatically control the complexity of the learned model.
- The idea is to penalize the objective function for large values of θ_j
- $J(\theta) = \underbrace{\frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}\left(x^{(i)}\right) - y^{(i)}\right)^2}_{\text{Model fit to data}} + \underbrace{\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2}_{\text{Regularization}}$

- λ is the regularization parameter ($\lambda \geq 0$)
- No regularization on θ_0

Understanding The Regularization

- $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}\left(x^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$
- What happens if we set the tuning parameter λ to be huge?



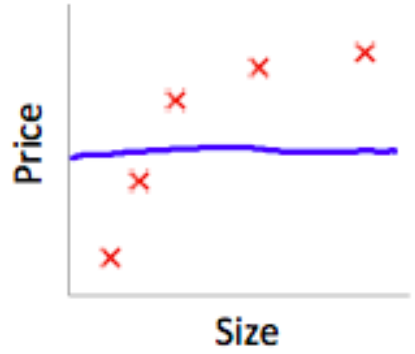
$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

0 0 0 0

Understanding The Regularization

Model: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

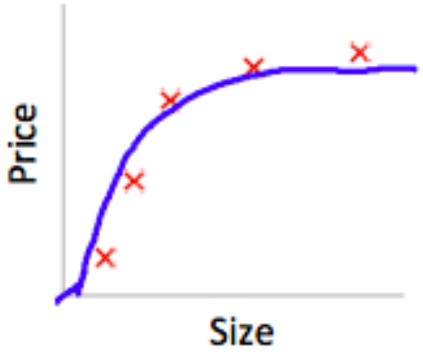


Large λ

High bias (underfit)

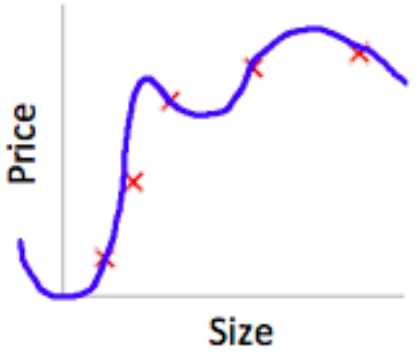
$\lambda = 10000.\ \theta_1 \approx 0, \theta_2 \approx 0, \ldots$

$h_\theta(x) \approx \theta_0$



Intermediate λ

"Just right"



Small λ

High variance (overfit)

Ridge and Lasso Regressors

- Ridge regression (L2 regularization):

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}\left(x^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

- Lasso regression (L1 regularization):

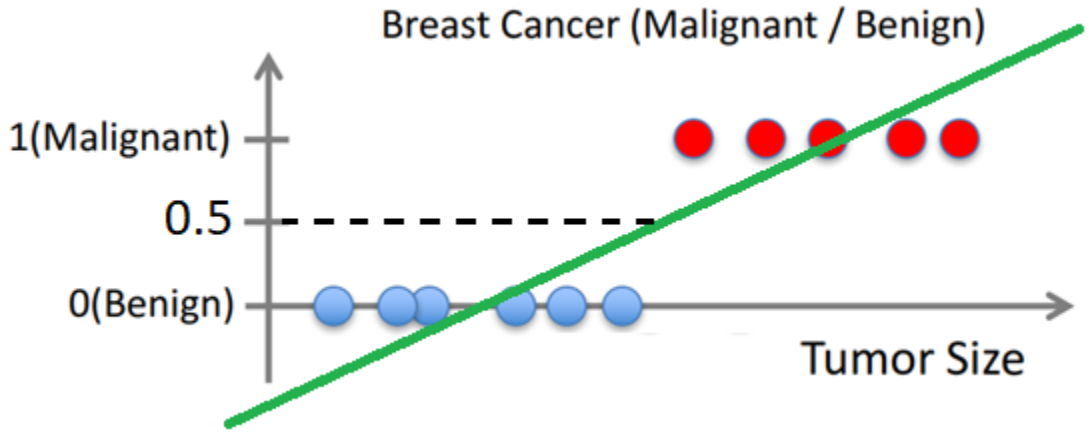
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}\left(x^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{m}\sum_{j=1}^{n}|\theta_j|$$

Linear Regression for Classification?

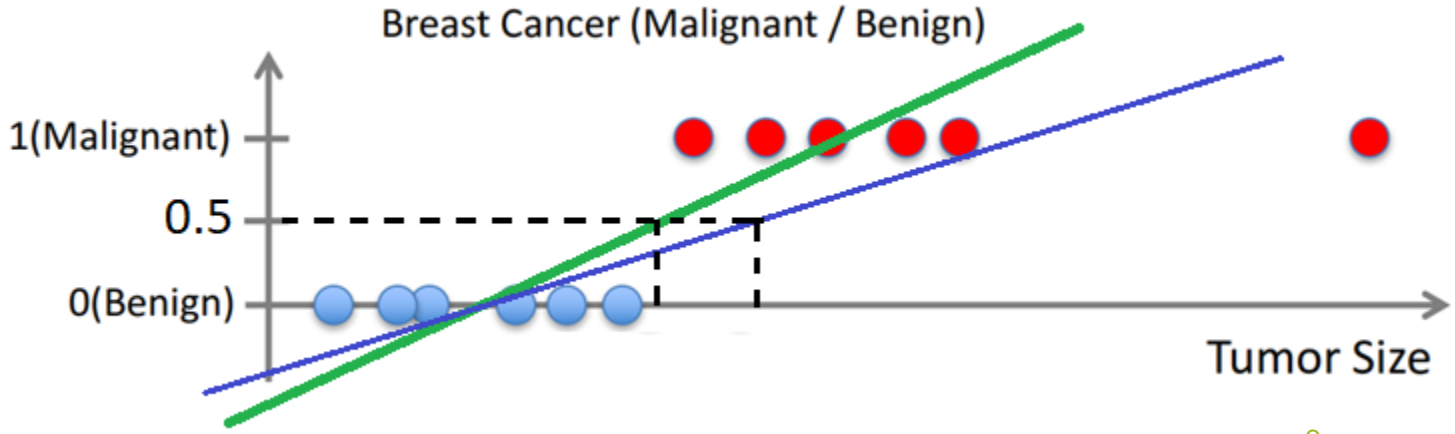
- Threshold classifier output $y(x)$ at 0.5

  If $\hat{y}(x) \geq 0.5$, predict $y = 1$

  If $\hat{y}(x) < 0.5$, predict $y = 0$



What happens if a new but far point is added?



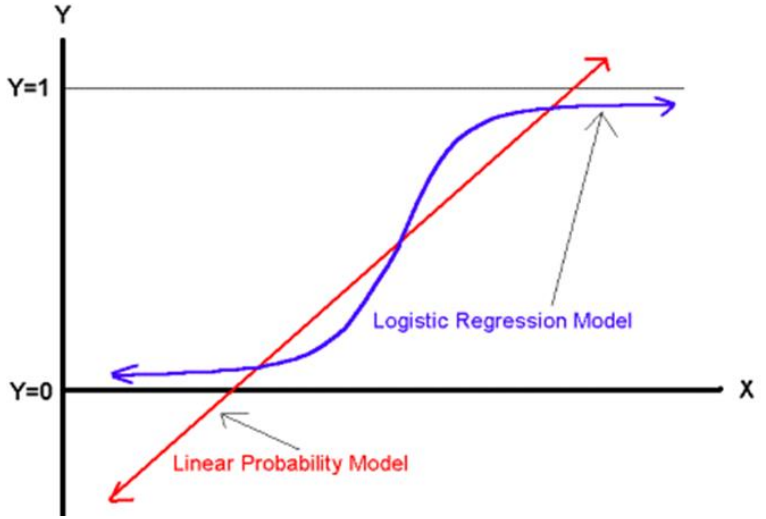
Logistic Regression

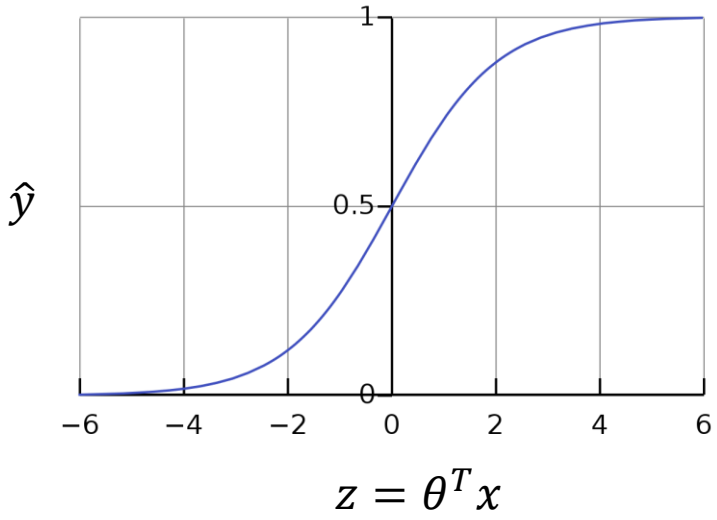
- In classification we want the output of the model to be as $0 \leq \hat{y}(x) \leq 1$

- In linear regression we had the output as $\hat{y}(x) = \theta^T x$

- To limit the output in the range $[0,1]$, we can use logistic function

$$g(z) = \frac{1}{1 + \exp(-z)}$$





Logistic Regression

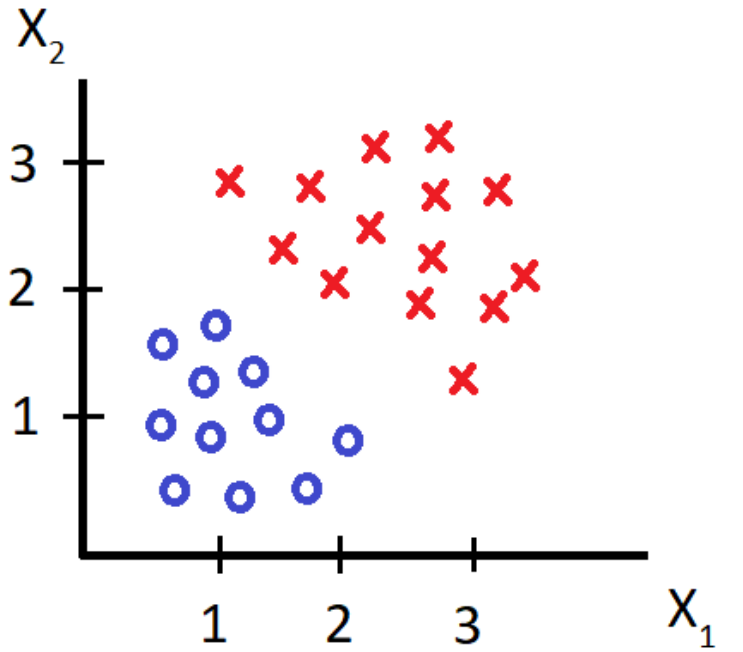
- An interpretation of the logistic function can be $\begin{cases} \hat{y}(\theta^T x) \geq 0.5 & if \quad \theta^T x \geq 0 \\ \hat{y}(\theta^T x) < 0.5 & if \quad \theta^T x < 0 \end{cases}$

$$\hat{y}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

If $\theta_0 = 0$, $\theta_1 = 1$, and $\theta_2 = 1$ then,

predicts $y = 1$ if $-3 + x_1 + x_2 \geq 0$

predicts $y = 0$ if $-3 + x_1 + x_2 < 0$

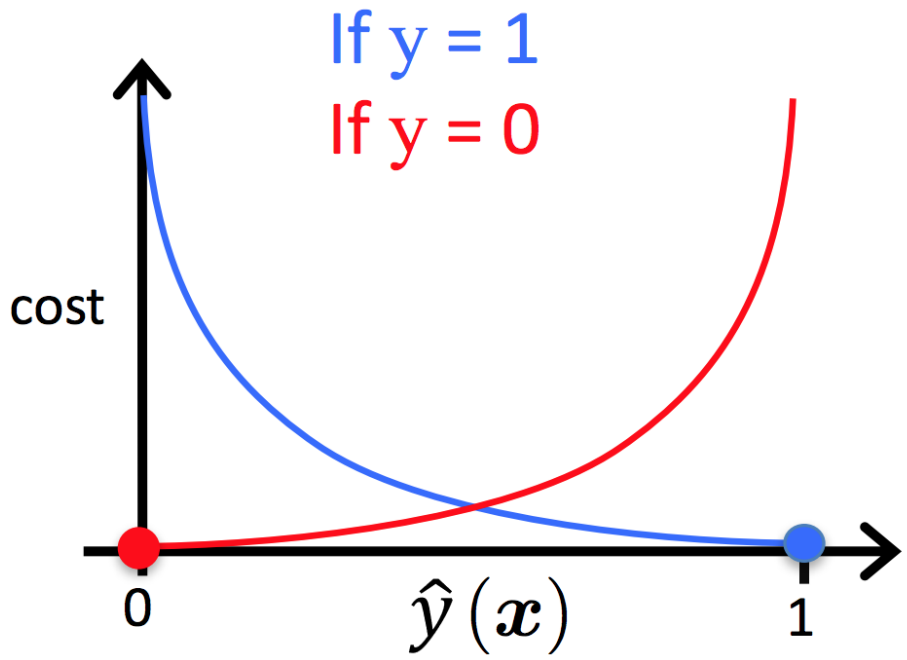


Cost Function for Logistic Regression

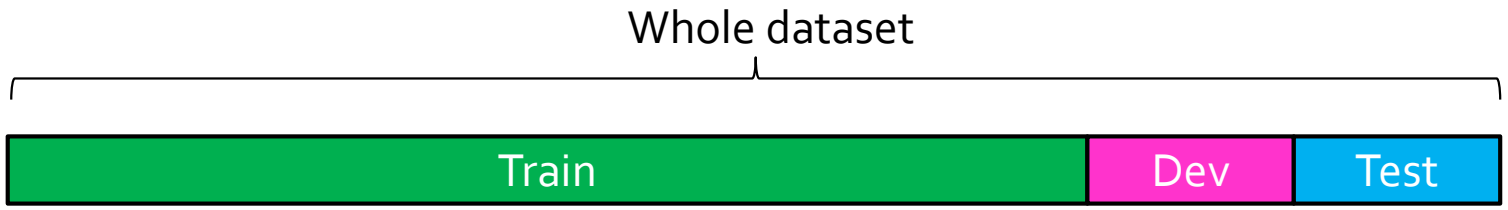
- Cost function for linear regression: $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}(x^{(i)}) - y^{(i)})^2$
- Cost function for logistic regression:

$$cost\ (\hat{y}, y) = \begin{cases} -\log \hat{y}(x) & if \quad y = 1 \\ -\log(1 - \hat{y}(x)) & if \quad y = 0 \end{cases}$$

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log(\hat{y}(x^{(i)})) + (1 - y^{(i)}) \log(1 - \hat{y}(x^{(i)}))]$$



Train/Dev/Test sets



Previously:
- 70% for training
- (no dev set)
- 30% for testing

Machine learning with small dataset:

100-10,000 samples
- 60% for training
- 20% for development
- 20% for testing

Machine learning with big dataset:

1 million samples +
- 98% for training
- 1% for development
- 1% for testing

What if we get large testing error?

- Get more training data → Fixes high variance (overfitting)
- Try smaller set of features → Fixes high variance (overfitting)
- Try getting additional features → Fixes high bias (underfitting)
- Try adding polynomial features → Fixes high bias (underfitting)

  $(x_1^2, x_2^2, x_1x_2, \dots)$
- Try increasing regularization parameter → Fixes high variance (overfitting)
- Try decreasing regularization parameter → Fixes high bias (underfitting)

Linear Regression

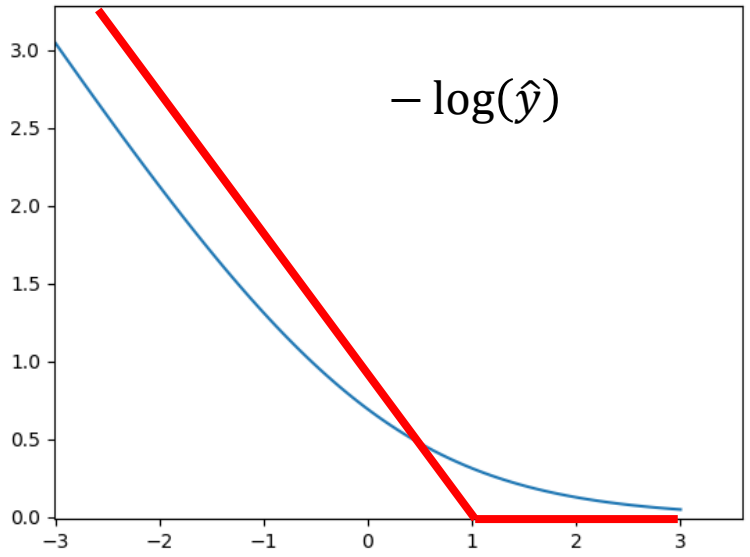
Open Jupyter notebook:

Linear models.ipynb

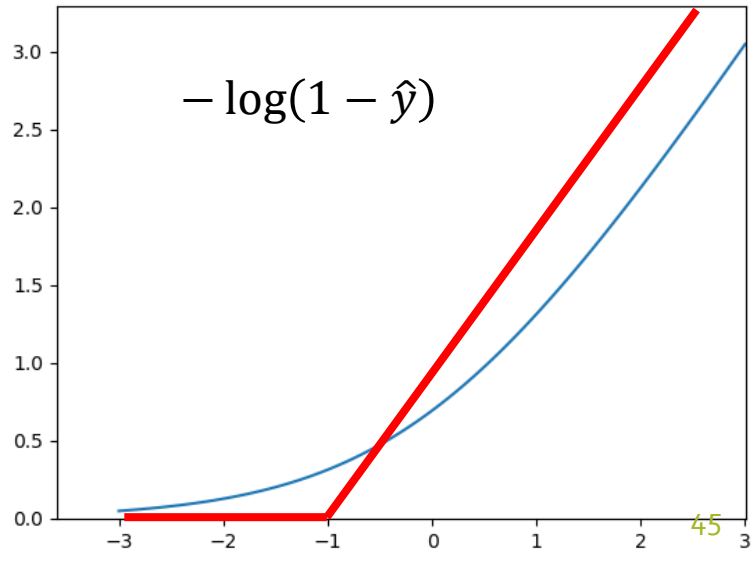
Support Vector Machine

- Alternative view of logistic regression:
- Cost: $-y\log(\hat{y}) - (1-y)\log(1-\hat{y}(x))$, where $\hat{y}(x) = \frac{1}{1+\exp(-\theta^T x)}$

If $y = 1$, we want $\hat{y}(x) \approx 1$, i.e. $\theta^T x \gg 0$



If $y = 0$, we want $\hat{y}(x) \approx 0$, i.e. $\theta^T x \ll 0$



Support Vector Machine

- Logistic Regression:

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\underbrace{(-\log \hat{y}(x^{(i)}))}_{Cost_1(\theta^T x^{(i)})} + (1-y^{(i)})\underbrace{(-\log\left(1-\hat{y}(x^{(i)})\right))}_{Cost_0(\theta^T x^{(i)})}\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

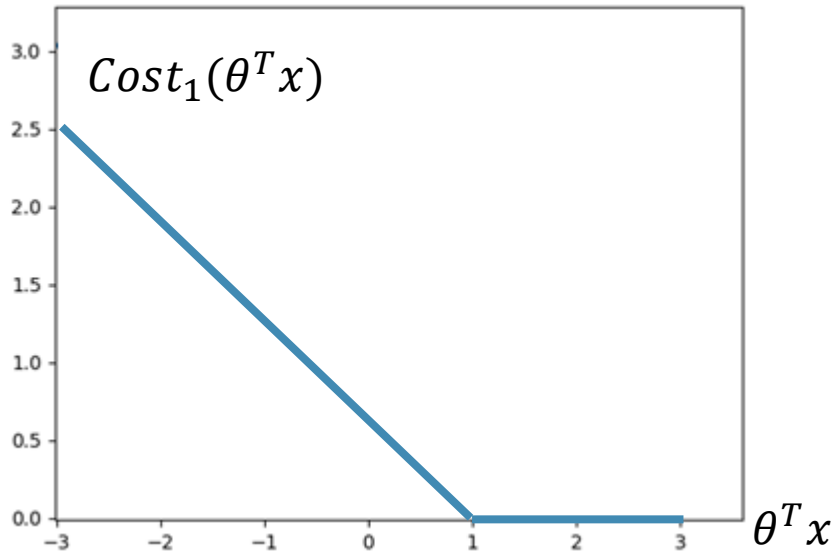
- Support Vector Machine:

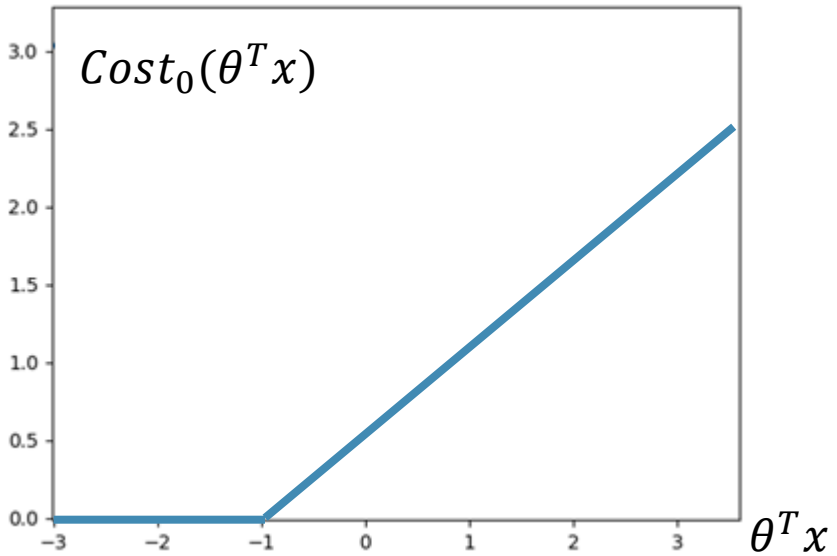
$$\min_\theta \underbrace{\sum_{i=1}^{m} y^{(i)} Cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) Cost_0(\theta^T x^{(i)}))}_{A} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{n}\theta_j^2}_{B} = A + B\lambda$$

$$\equiv \min_\theta CA + B \qquad \text{where} \qquad C = \frac{1}{\lambda}$$

Support Vector Machine

$$\min_{\theta} C \sum_{i=1}^{m} y^{(i)} Cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) Cost_0(\theta^T x^{(i)})) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$





If $y = 1$, we want $\theta^T x \geq 1$ (not just ≥ 0)

If $y = 0$, we want $\theta^T x \leq -1$ (not just ≤ 0)

SVM as Large Margin Classifier

- SVM decision boundary:
- For large C

$$\min_{\theta} C \sum_{i=1}^{m} \overbrace{y^{(i)} Cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) Cost_0(\theta^T x^{(i)})}^{0} + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

- Then the optimization problem would change to:

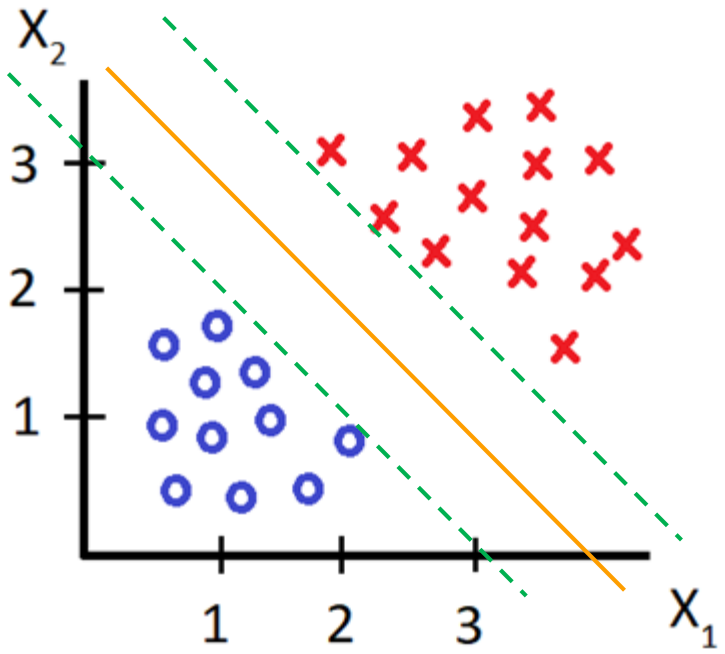
$$\min_{\theta} \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

S.T $\quad \theta^T x^{(i)} \geq 1 \quad$ if $\quad y^{(i)} = 1$

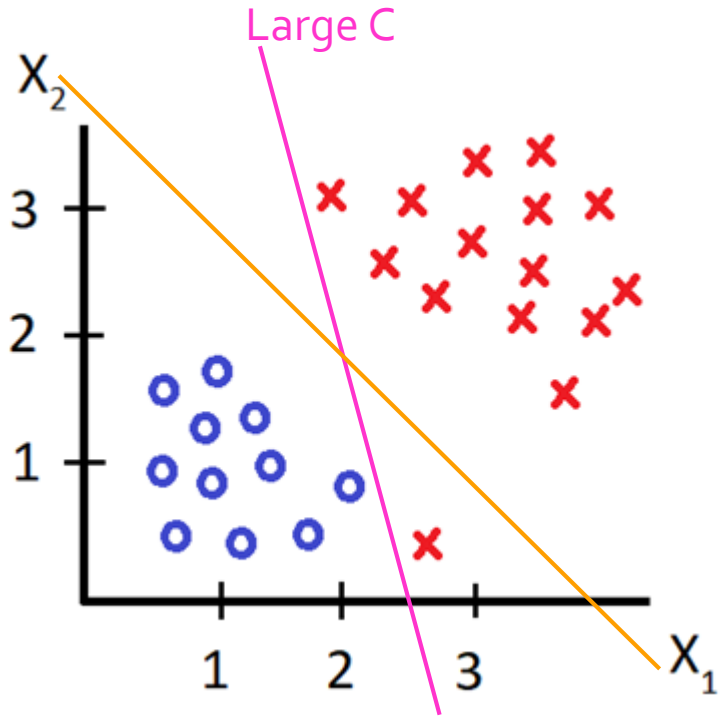
$\theta^T x^{(i)} \leq -1 \quad$ if $\quad y^{(i)} = 0$

SVM as Large Margin Classifier

- Linearly separable case:



- In the presence of outlier:



Example:

- Consider the training set to the right, where "X" denotes positive examples ($y = 1$) and "O" denotes negative examples ($y = 0$). Suppose you train an SVM (which will predict 1 when $\theta_0 + \theta_1 x_1 + \theta_2 x_2 \geq 0$). What values might the SVM give for $\theta_0, \theta_1, and\ \theta_2$?

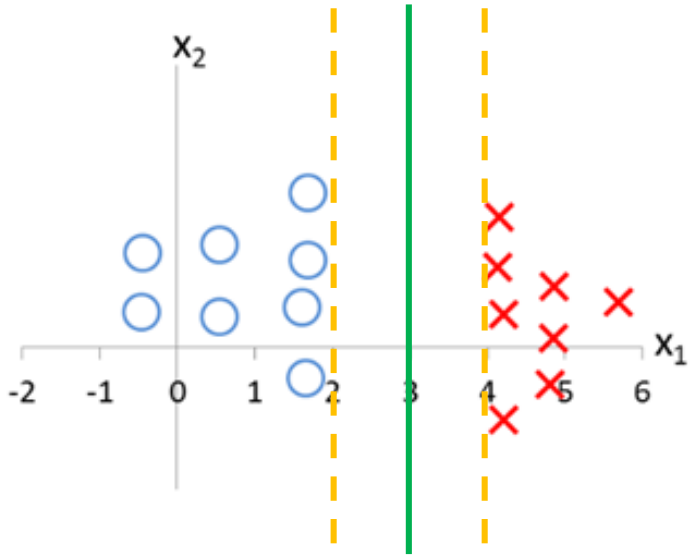
To find support vectors, find the boundaries with:

$\theta^T x \geq 1$ and $\theta^T x \leq -1$

$\theta_0 = -3,$

$\theta_1 = 1,$

$\theta_2 = 0$



Kernels for nonlinear decision boundary

- The nonlinear decision boundary will be:

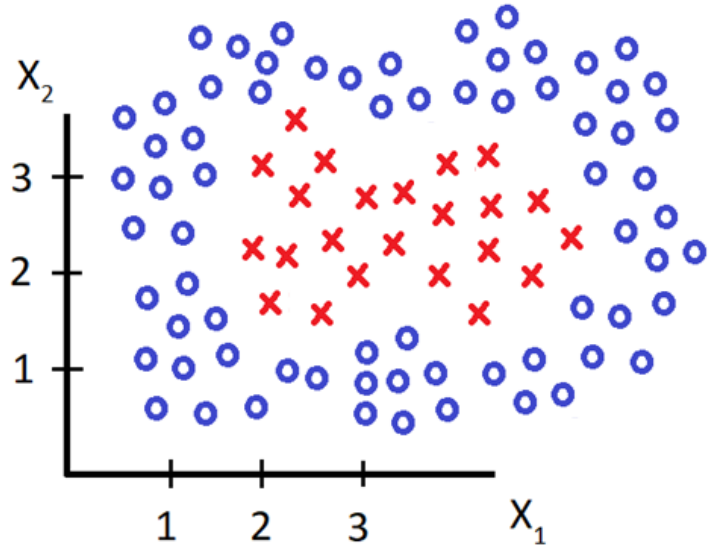
$y = 1$ if $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \cdots \geq 0$

$y = 0$ if $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \cdots < 0$

But we can consider:

$f_1 = x_1, f_2 = x_2, f_3 = x_1 x_2, f_4 = x_1^2, f_5 = x_2^2, \ldots$

Is there a better choice of features?



Kernel as similarity function

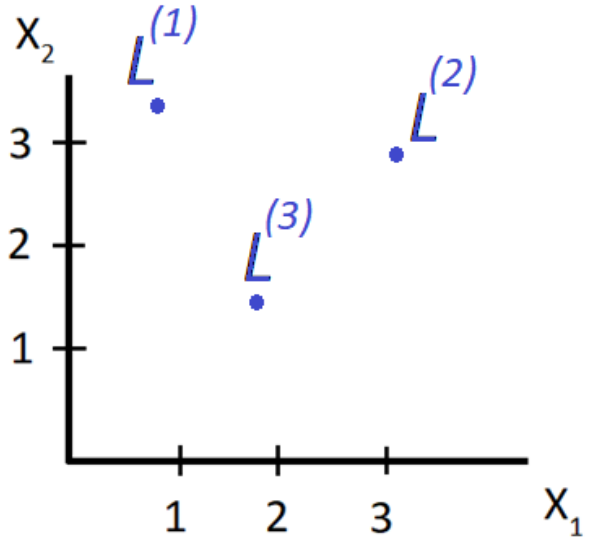
- We can choose some landmarks and compare the similarity of new data with them.
- We can have different similarity measures.

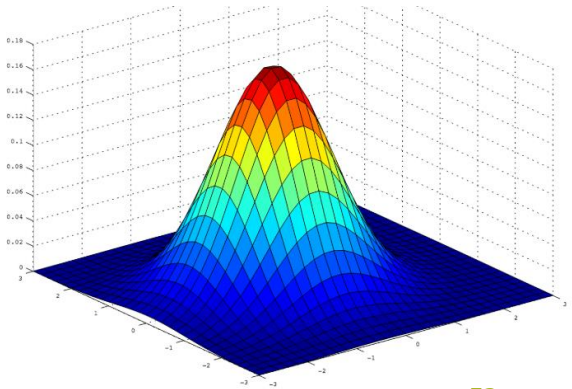
$$f_1 = similarity(x, L^{(1)}) = \exp(-\frac{\|x - L^{(1)}\|^2}{2\sigma^2})$$

$$f_2 = similarity(x, L^{(2)}) = \exp(-\frac{\|x - L^{(2)}\|^2}{2\sigma^2})$$

$$f_3 = similarity(x, L^{(3)}) = \exp(-\frac{\|x - L^{(3)}\|^2}{2\sigma^2})$$

Gaussian Kernel





- If $x \approx L^{(1)}$: $f_1 \approx 1, \quad f_2 \approx 0, \quad f_3 \approx 0$

Example:

- Assume:

$y = 1$ if $\theta^T f = \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

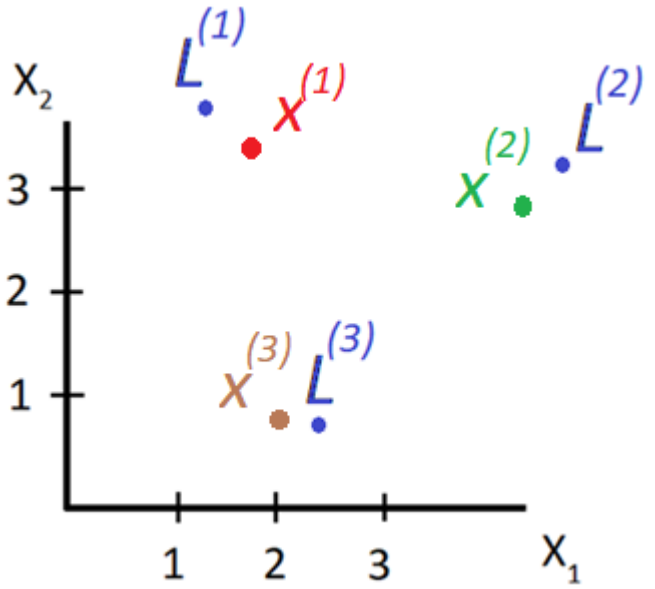
$y = 0$ Otherwise

$\theta_0 = -0.5, \ \theta_1 = 1, \ \theta_2 = 1, \ \theta_3 = 0$

At $x^{(1)}$: $f_1 \approx 1, f_2 \approx 0, f_3 \approx 0 \rightarrow \theta^T f = 0.5 > 0$ Class 1

At $x^{(2)}$: $f_1 \approx 0, f_2 \approx 1, f_3 \approx 0 \rightarrow \theta^T f = 0.5 > 0$ Class 1

At $x^{(3)}$: $f_1 \approx 0, f_2 \approx 0, f_3 \approx 1 \rightarrow \theta^T f = -0.5 < 0$ Class 0



Support Vector Machine (SVM)

Open Jupyter notebook:

SVM.ipynb

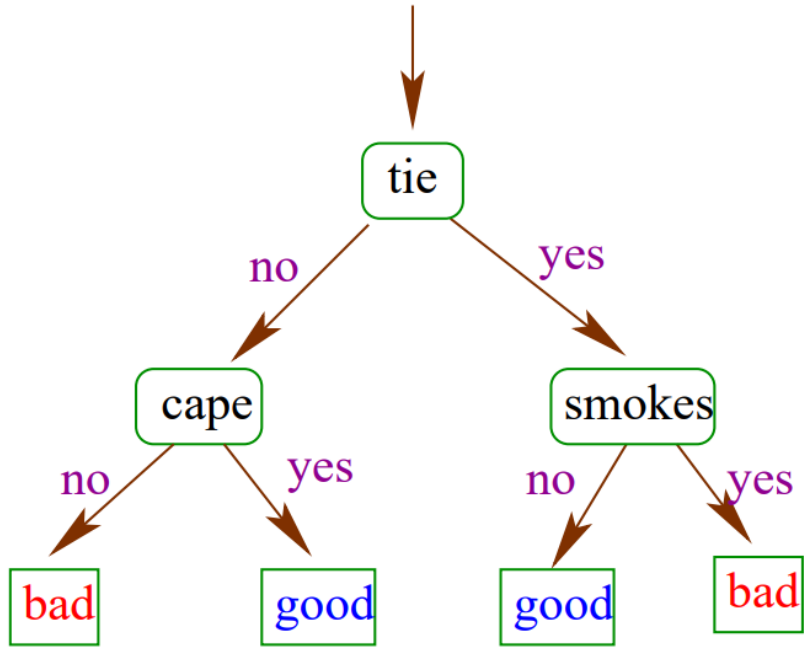
Decision Tree Classifier

- Example: Good or Bad people based on their appearance

	sex	mask	cape	tie	ears	smokes	class
			training data				
batman	male	yes	yes	no	yes	no	Good
robin	male	yes	yes	no	no	no	Good
alfred	male	no	no	yes	no	no	Good
penguin	male	no	no	yes	no	yes	Bad
catwoman	female	yes	no	no	yes	no	Bad
joker	male	no	no	no	no	no	Bad
			test data				
batgirl	female	yes	yes	no	yes	no	??
riddler	male	yes	no	no	no	no	??

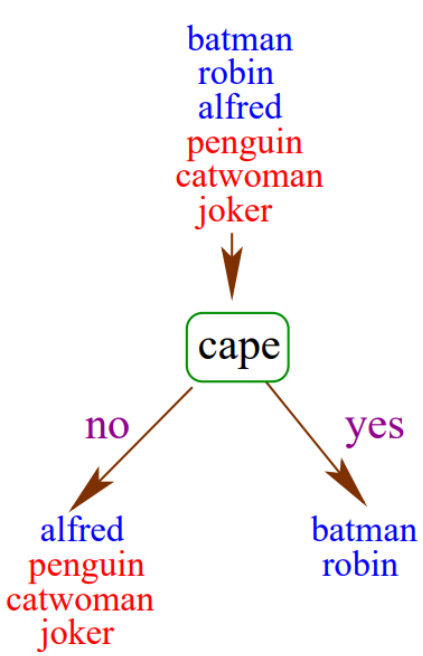
Decision Tree Classifier

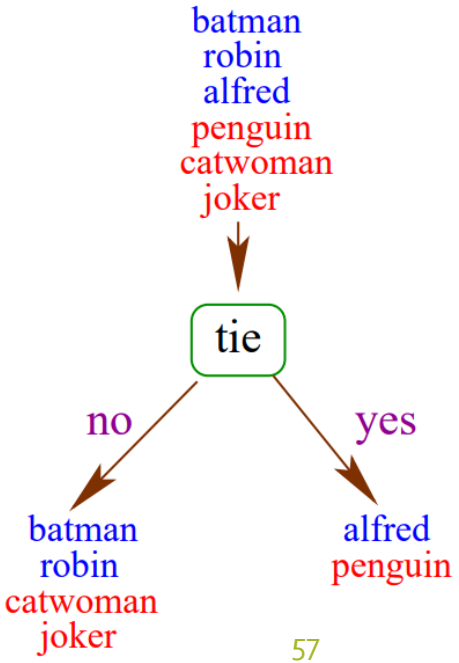
	sex	mask	cape	tie	ears	smokes	class
			training data				
batman	male	yes	yes	no	yes	no	Good
robin	male	yes	yes	no	no	no	Good
alfred	male	no	no	yes	no	no	Good
penguin	male	no	no	yes	no	yes	Bad
catwoman	female	yes	no	no	yes	no	Bad
joker	male	no	no	no	no	no	Bad
			test data				
batgirl	female	yes	yes	no	yes	no	??
riddler	male	yes	no	no	no	no	??



How to Build Decision Tree

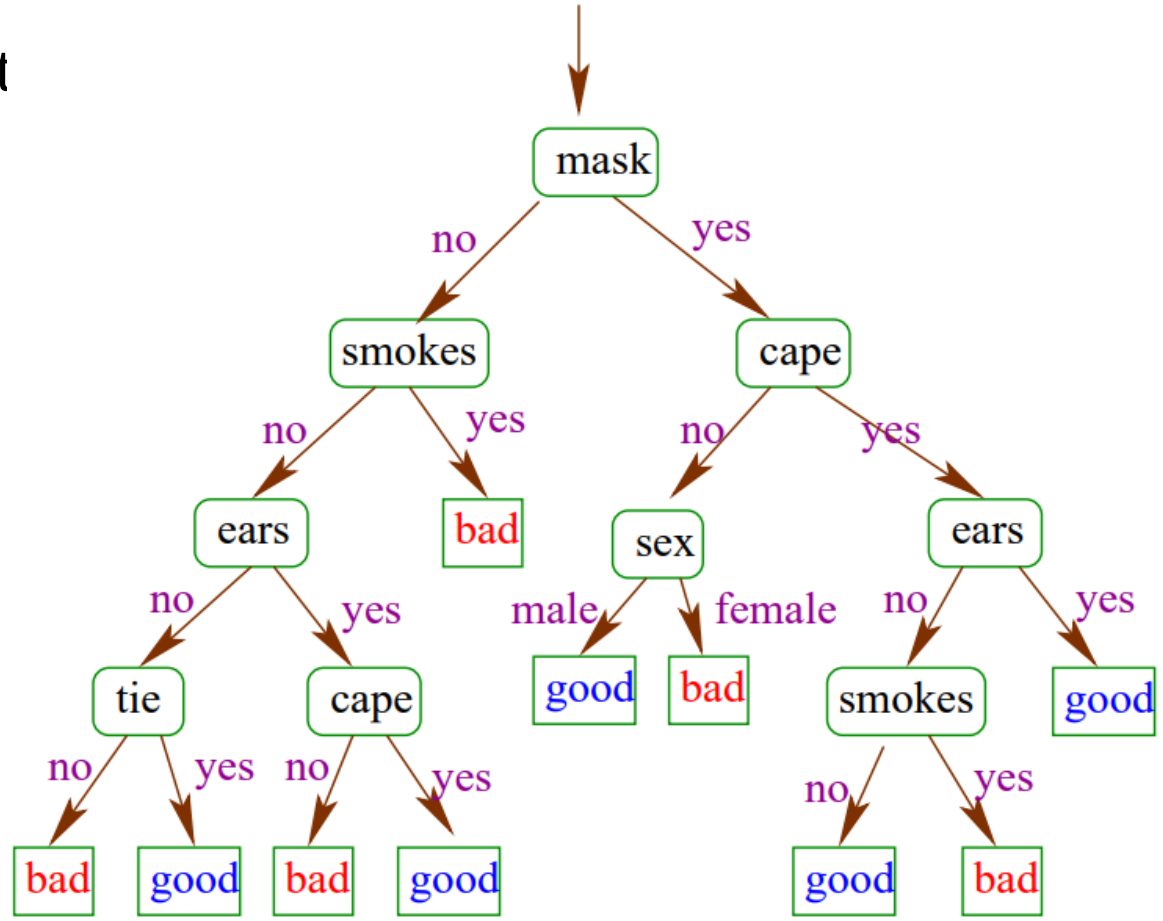
- Choose rule to split on
- Divide data using splitting rule into disjoint subsets
- Repeat recursively for each subset
- Stop when leaves are (almost) “pure”



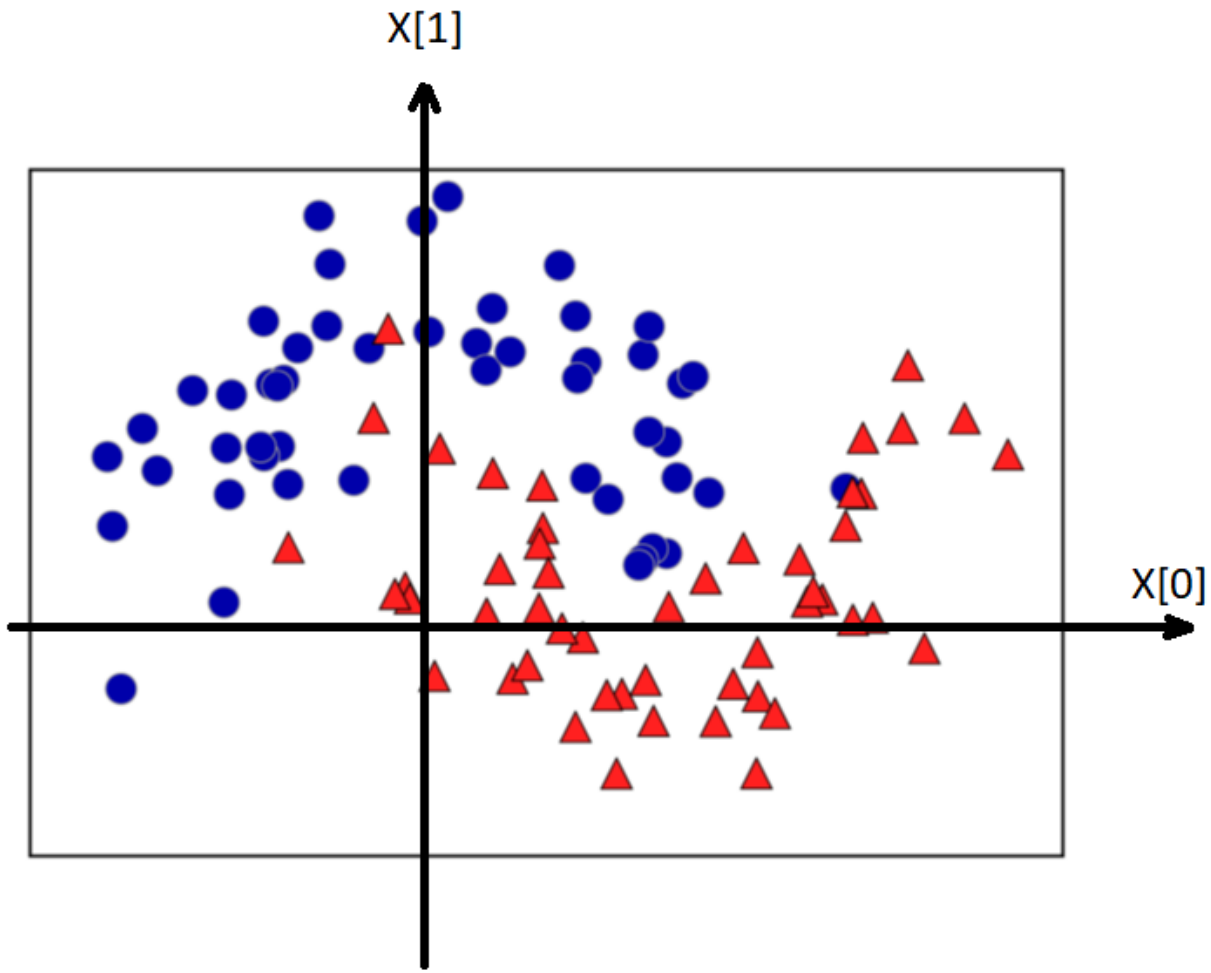


Possible Classifier

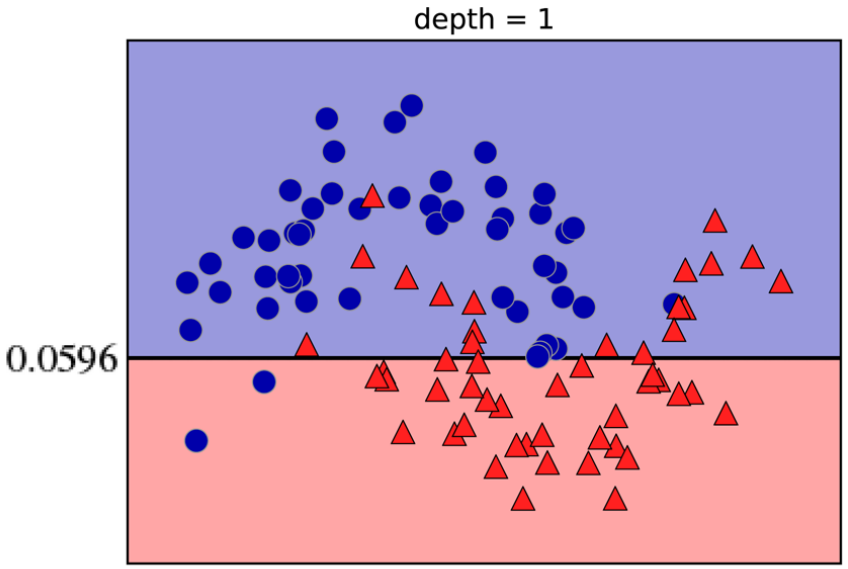
- Perfectly classifies the training dat
- BUT, intuitively overly complex

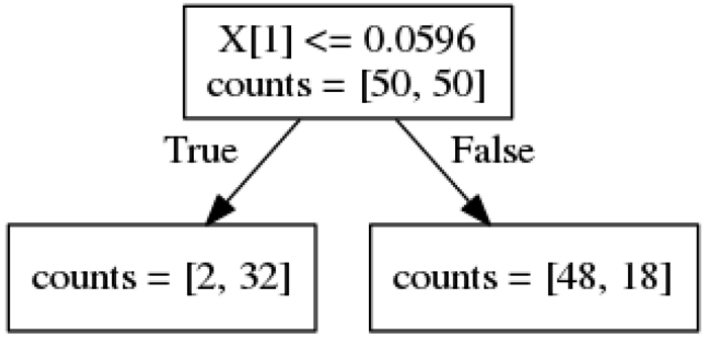


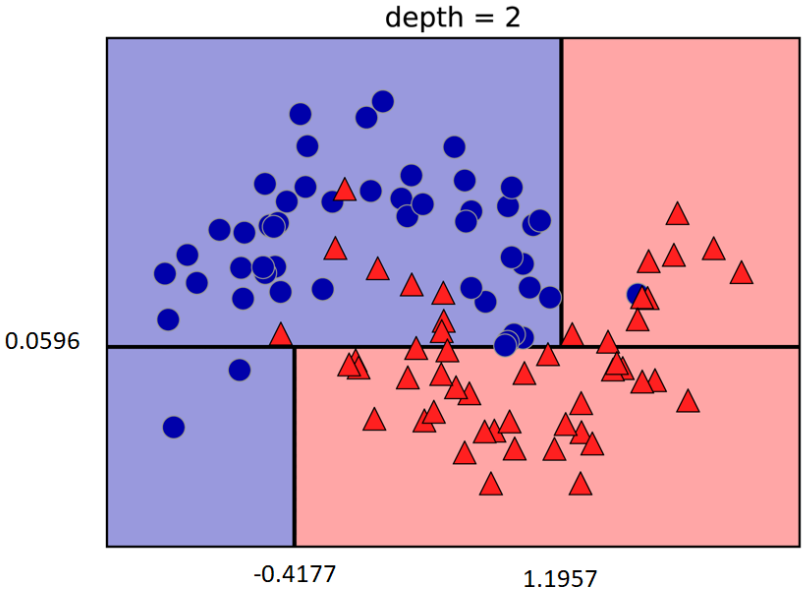
Decision Tree Classifier on Two-moons data

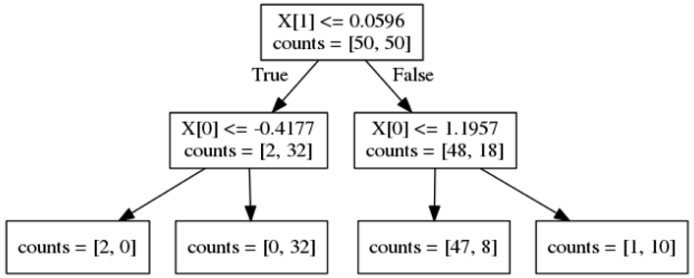


Decision Boundaries of depths 1 and 2





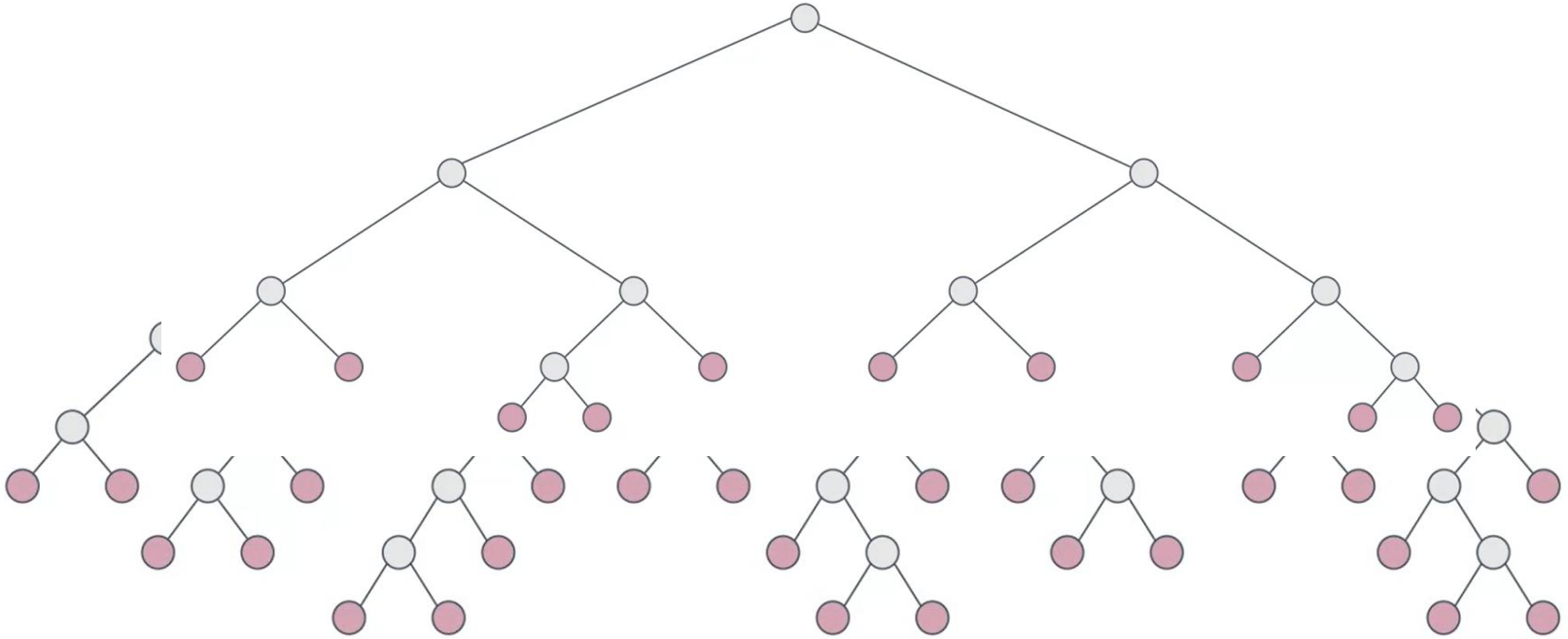




Decision Tree in Python

```
from sklearn import tree

classifier = tree.DecisionTreeClassifier(max_leaf_nodes=None, min_samples_leaf=1, max_depth=None)
```

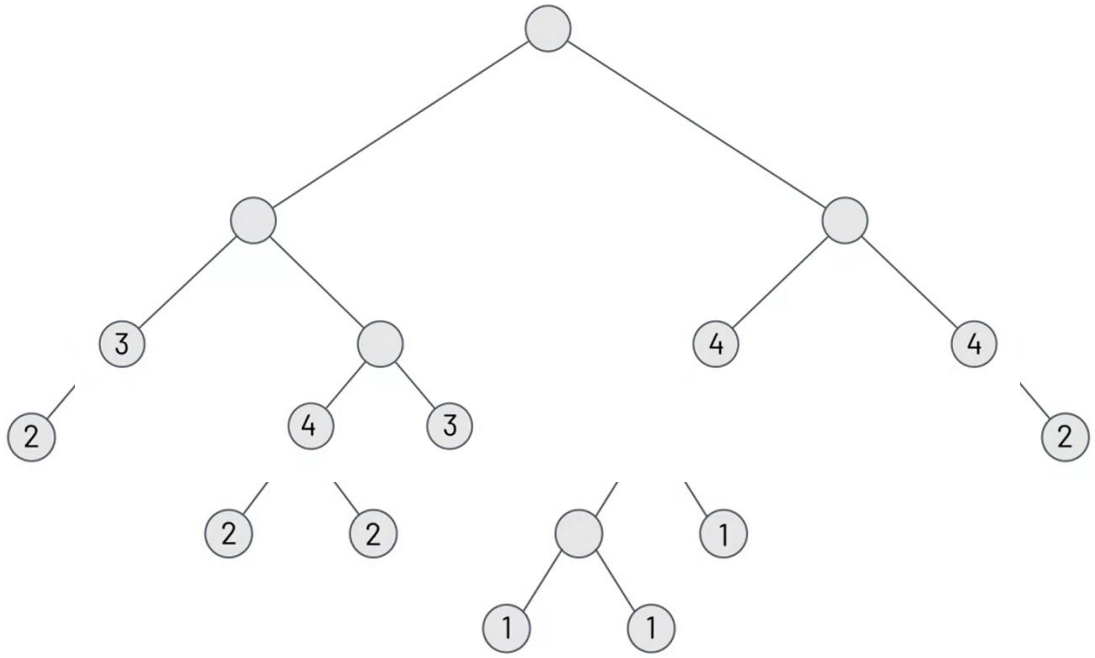


max_leaf_nodes=10

Decision Tree in Python

```
from sklearn import tree

classifier = tree.DecisionTreeClassifier(max_leaf_nodes=None, min_samples_leaf=1, max_depth=None)
```

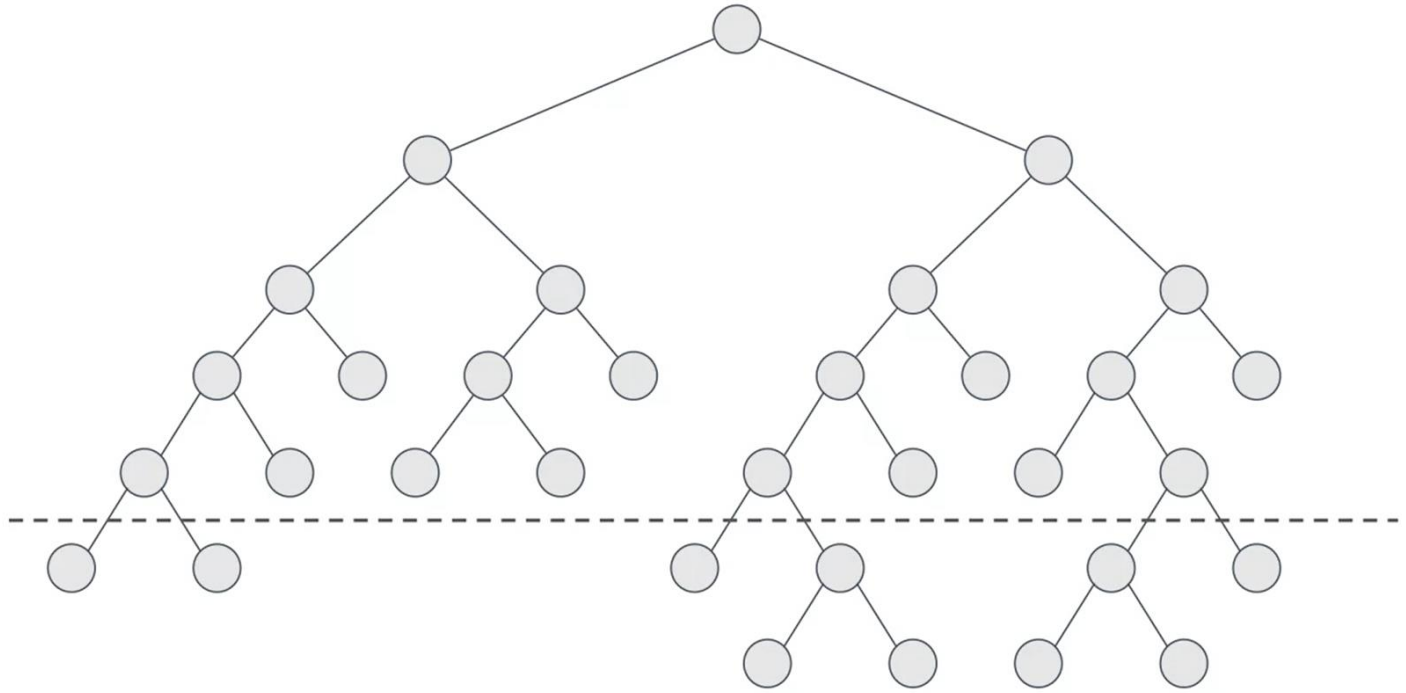


min_samples_leaf=3

Decision Tree in Python

```
from sklearn import tree

classifier = tree.DecisionTreeClassifier(max_leaf_nodes=None, min_samples_leaf=1, max_depth=None)
```



max_depth=4

Decision Trees Pros and Cons

- Pros:

  - the resulting model can easily be visualized and understood by nonexperts (at least for smaller trees)

  - the algorithms are completely invariant to scaling of the data

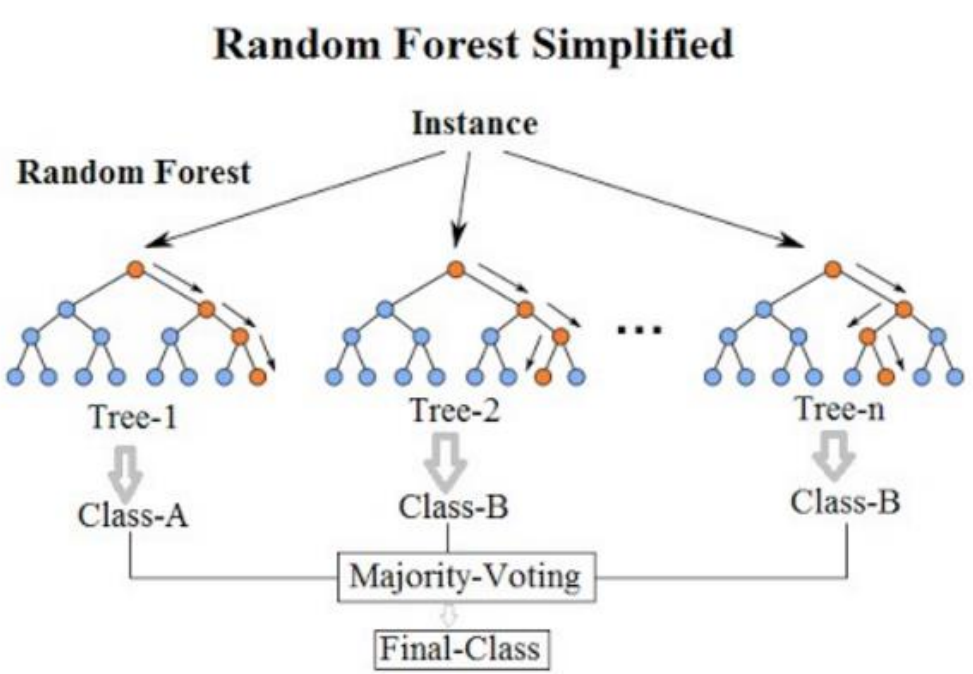
- Cons:

  - they tend to overfit and provide poor generalization performance.

  - the model does not have robustness and may change drastically with small change in the data

Random Forest

- To mitigate the limitation of the Decision Tree, Random Forest is used as an ensemble.
- The input of each tree is sampled data from the original dataset.
- Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction



Decision Tree Classifier

Open Jupyter notebook:

Decision Trees.ipynb

K-Nearest Neighbors (KNN) Classifier

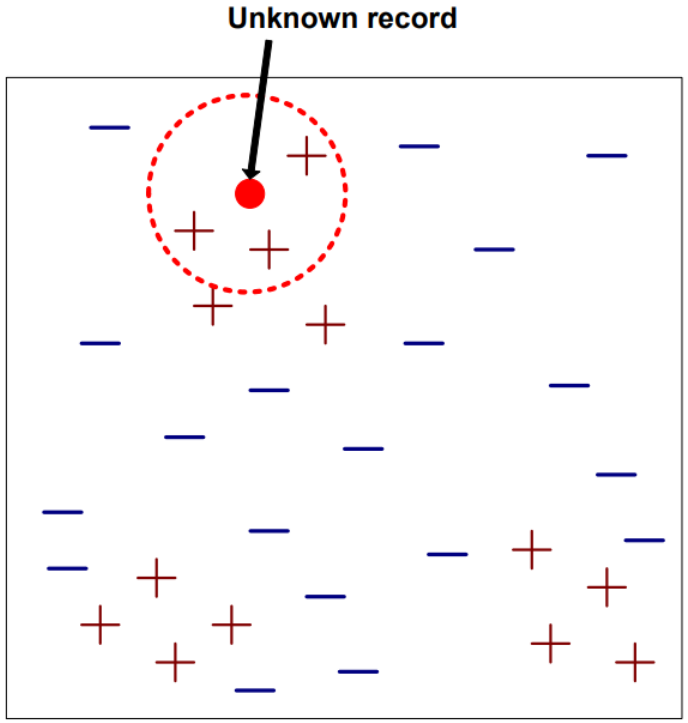
- **Basic ideas:**

➢ *k*-NN stores all samples and classifies based on similarity measure (distance)

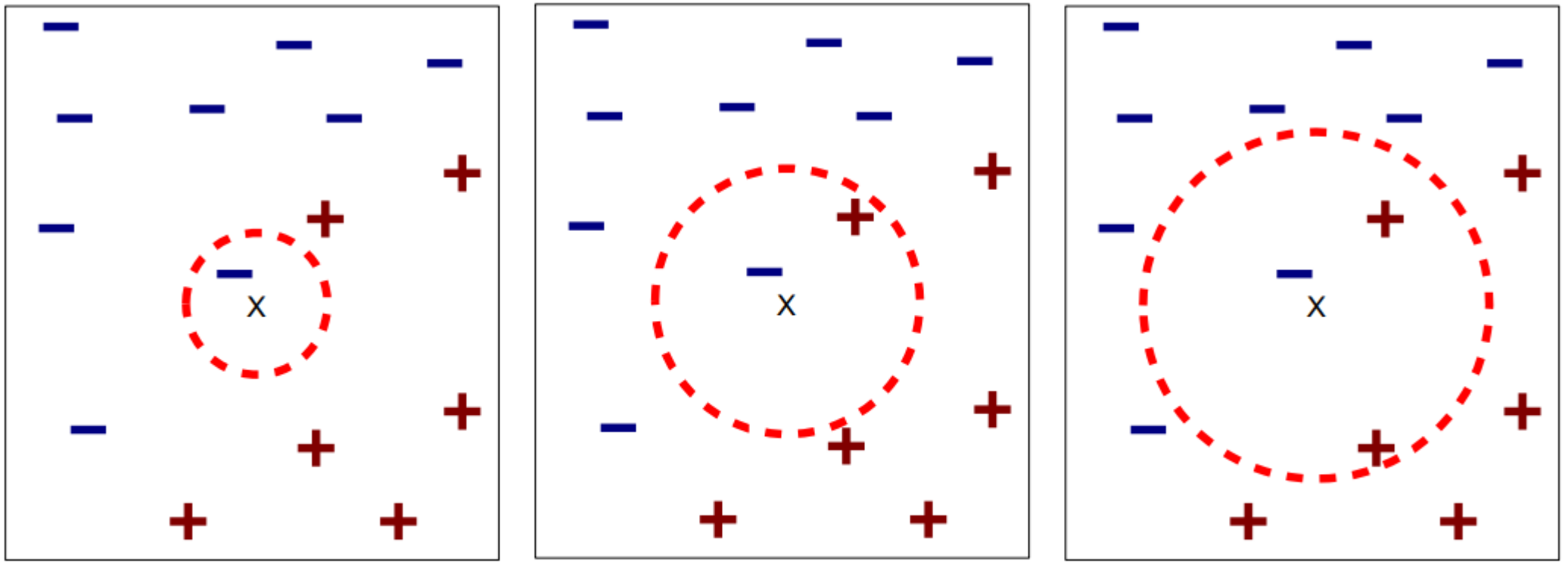
➢ The *k*-NN classification rule is to assign to a test sample the majority category label of its *k* nearest training samples

➢ In practice, *k* is usually chosen to be odd, so as to avoid ties

➢ The *k* = 1 rule is generally called the nearest-neighbor classification rule



Definition of nearest neighbor

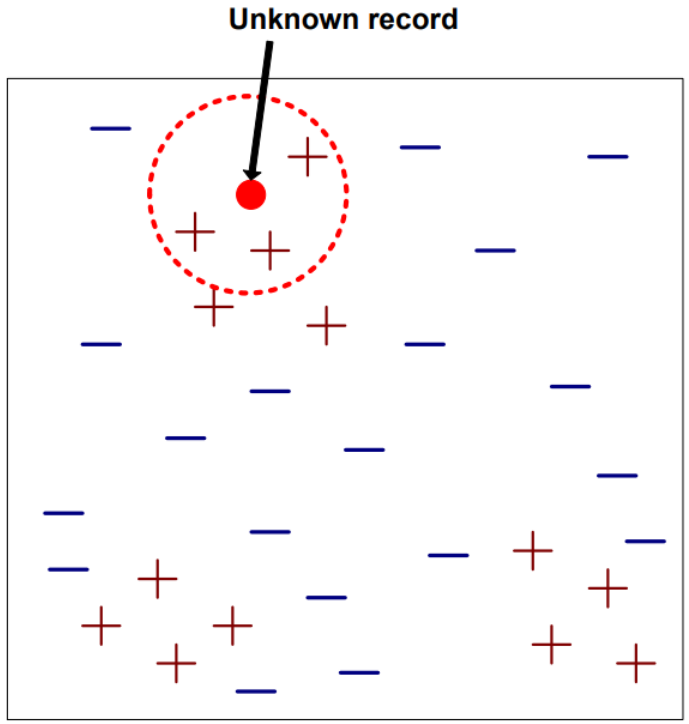


(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

Large *k* vs. small *k*

- Large *k*:
  - Distant neighbors are included
  - New samples are most likely in the same class
  - Most prone to underfitting (high bias)

- Small *k* :
  - Very close neighbors included
  - Highly variant due to small changes in the samples
  - Most prone to overfitting (high variance)



K-Nearest Neighbors Distance functions

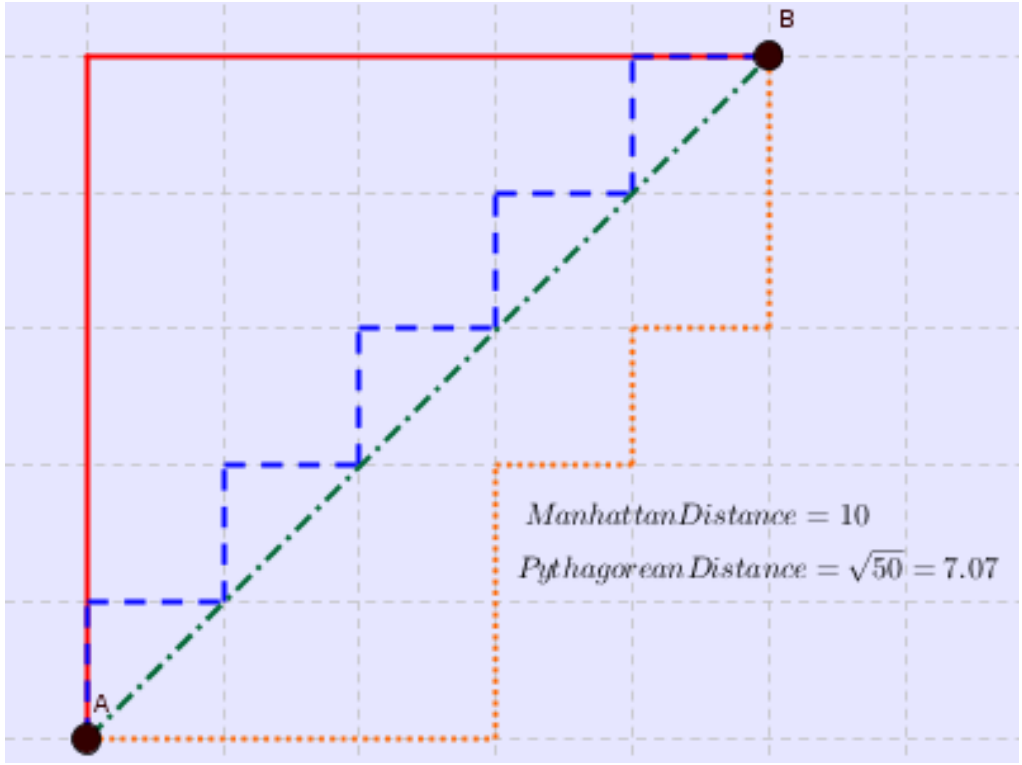
- Different distances computed between two points:

  – Euclidean distance $d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$

  – Manhatten distance $d(p,q) = \sum_i |p_i - q_i|$

  – q norm distance $d(p,q) = (\sum_i |p_i - q_i|^q)^{1/q}$

Distances (Manhattan vs. Euclidean)



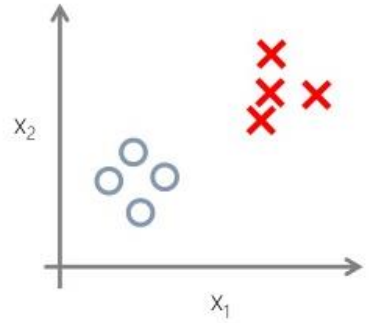
K-Nearest Neighbors

Open Jupyter notebook:

KNN.ipynb

Supervised vs. Unsupervised

Supervised Learning

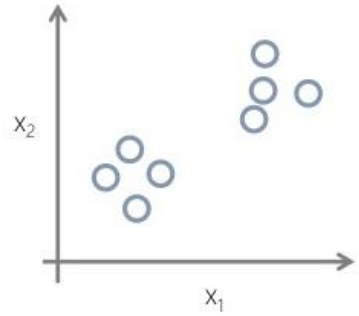


Training set has labels:

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \dots, (x^{(m)}, y^{(m)})\}$$

The objective is to differentiate classes and classify new sample points.

Unsupervised Learning



Training set does NOT have any label:

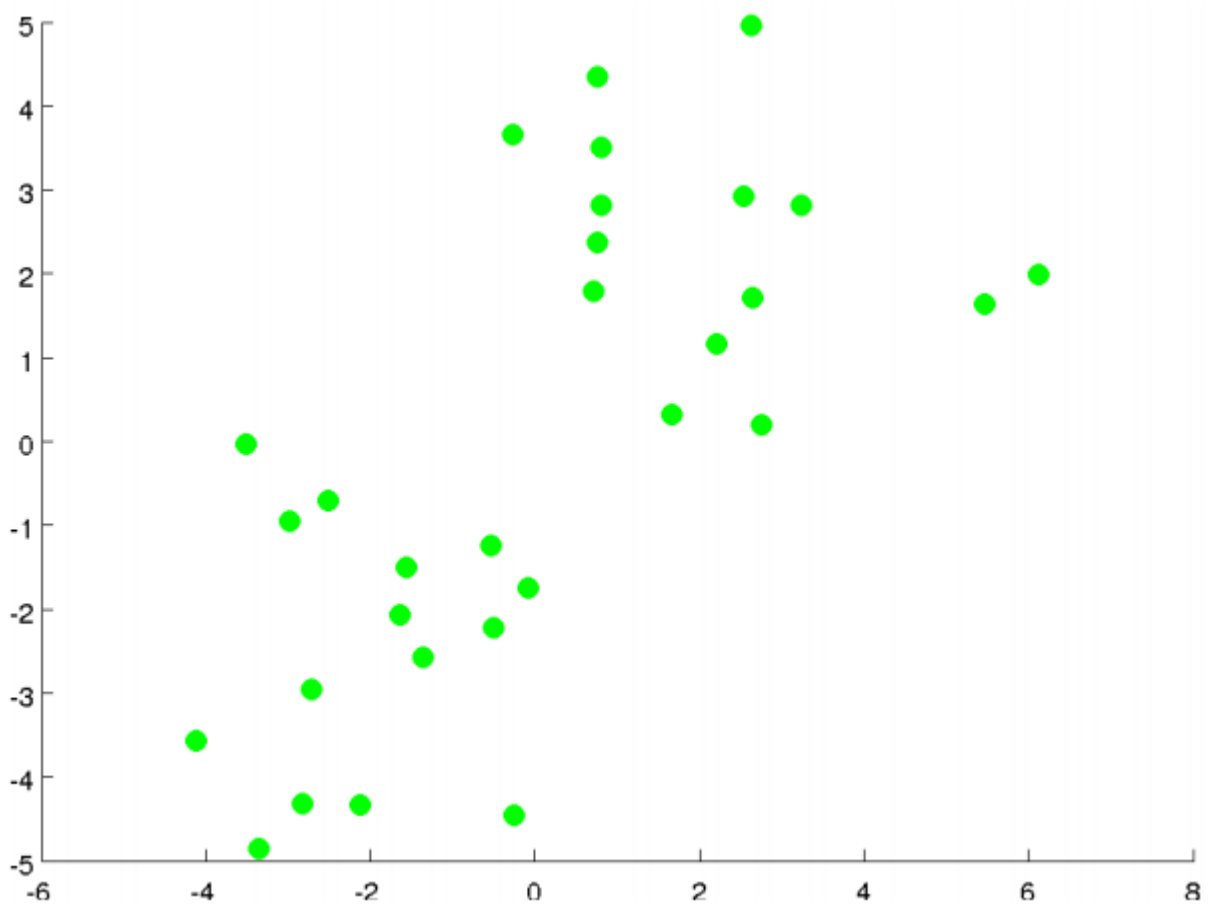
$$\{x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(m)}\}$$

The objective is to find clusters.

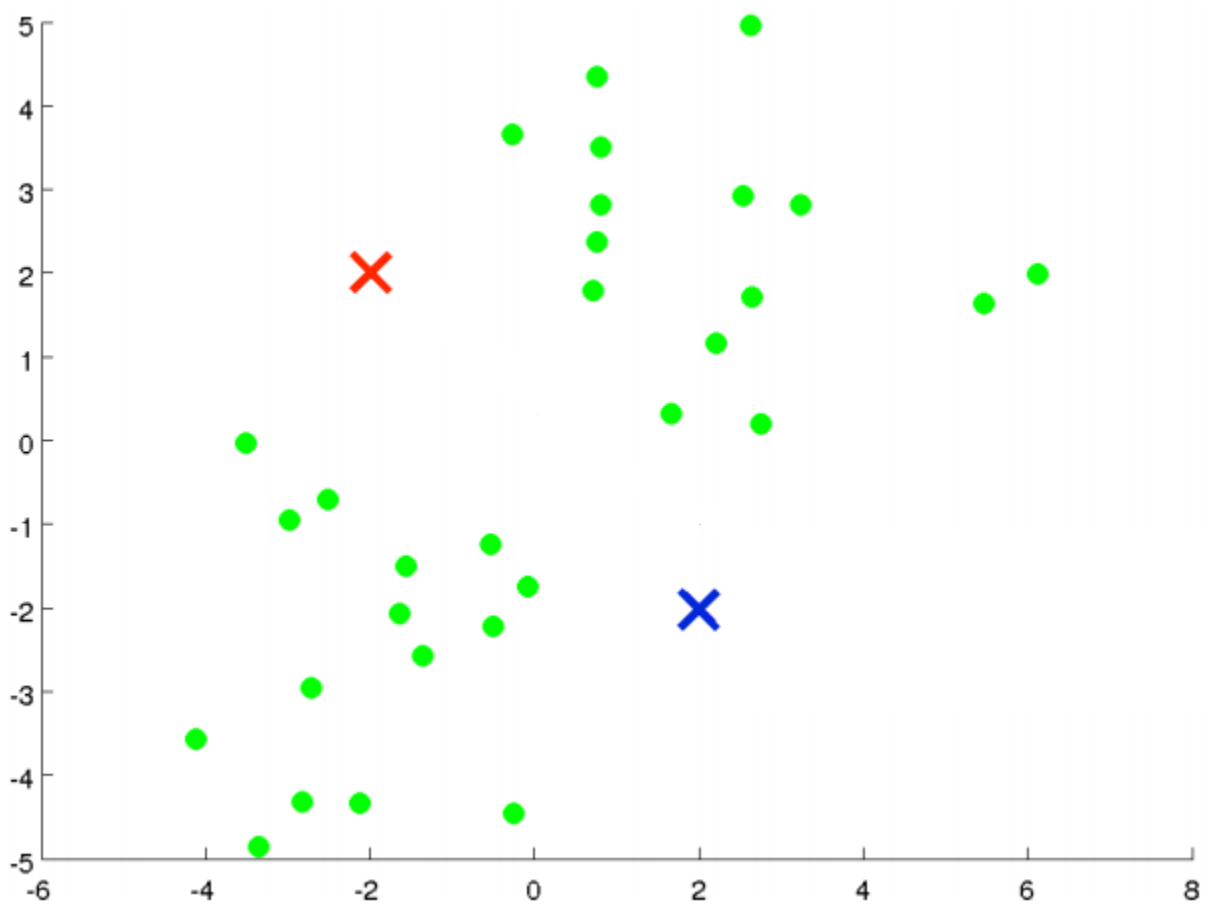
Unsupervised learning

- In unsupervised learning, the training set is of the form $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}\}$ without labels $y^{(i)}$.
- In unsupervised learning, you are given an unlabelled dataset and are asked to find "structure" in the data.
- Clustering is an example of unsupervised learning.
- Clustering is NOT the only unsupervised learning algorithm.

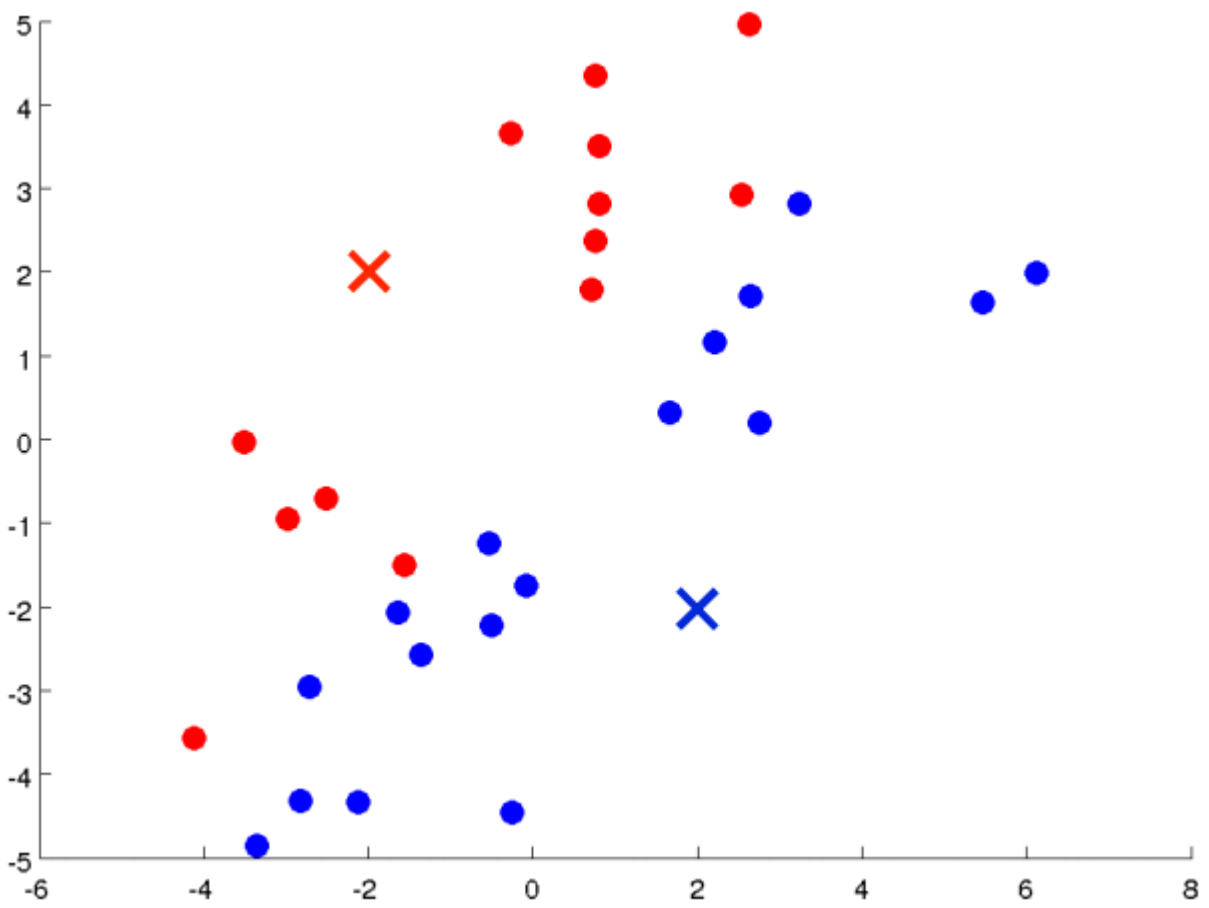
K-means



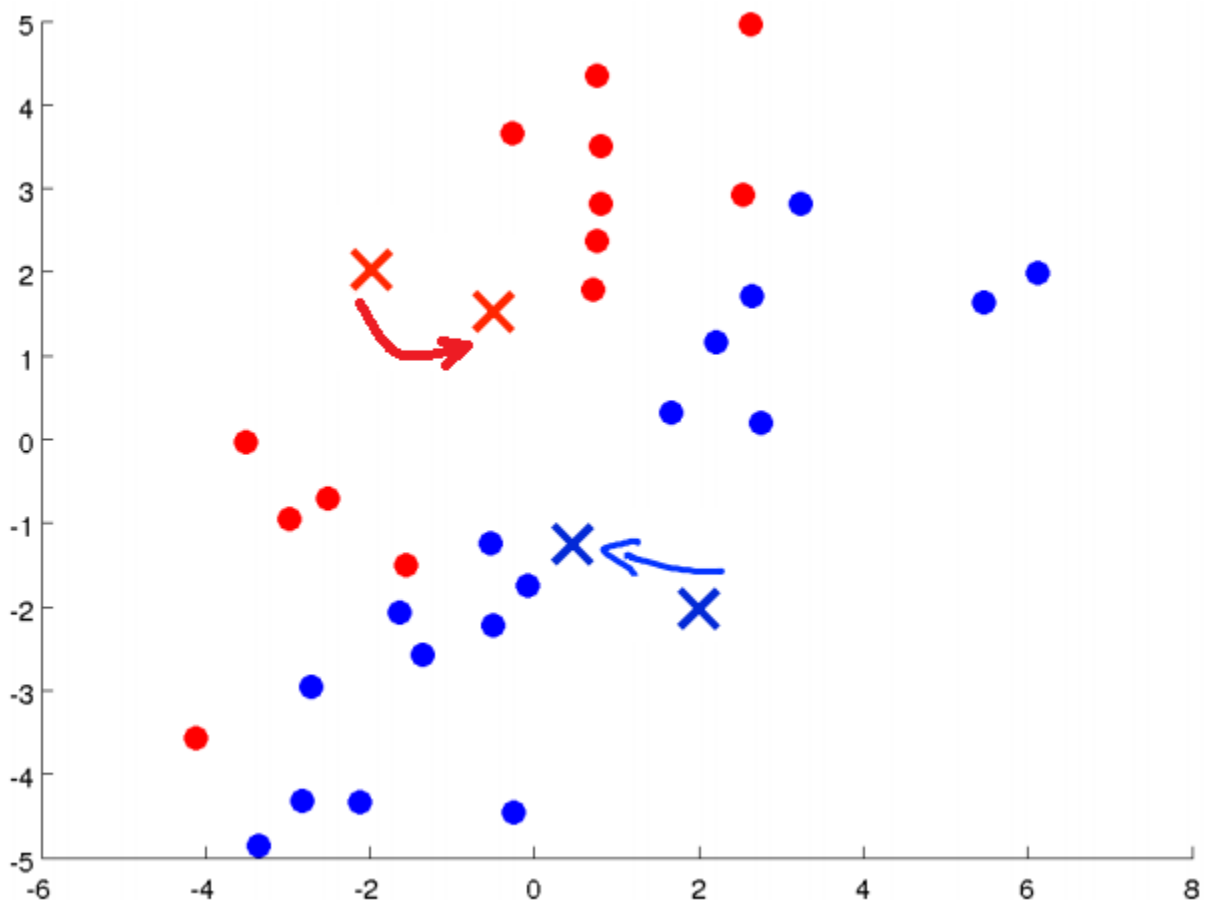
K-means



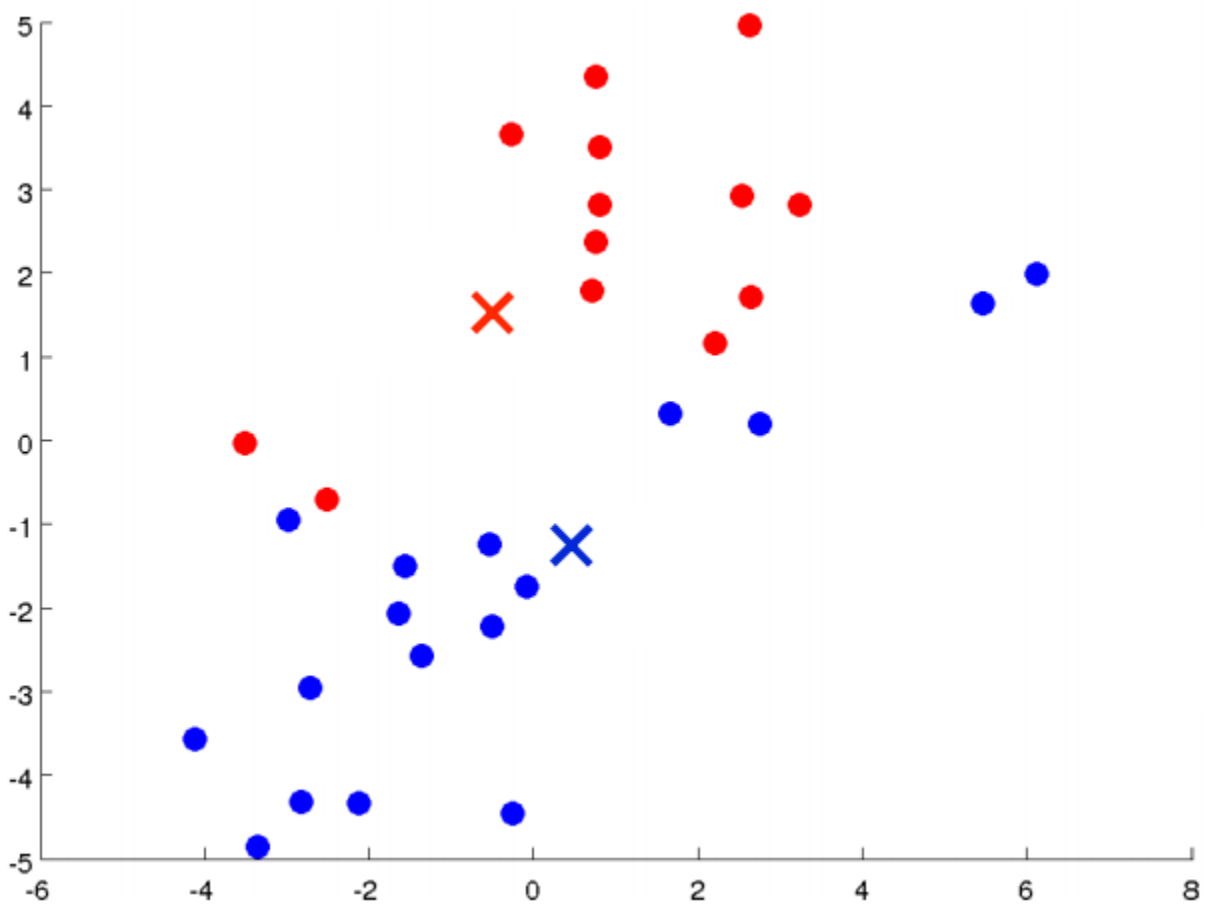
K-means



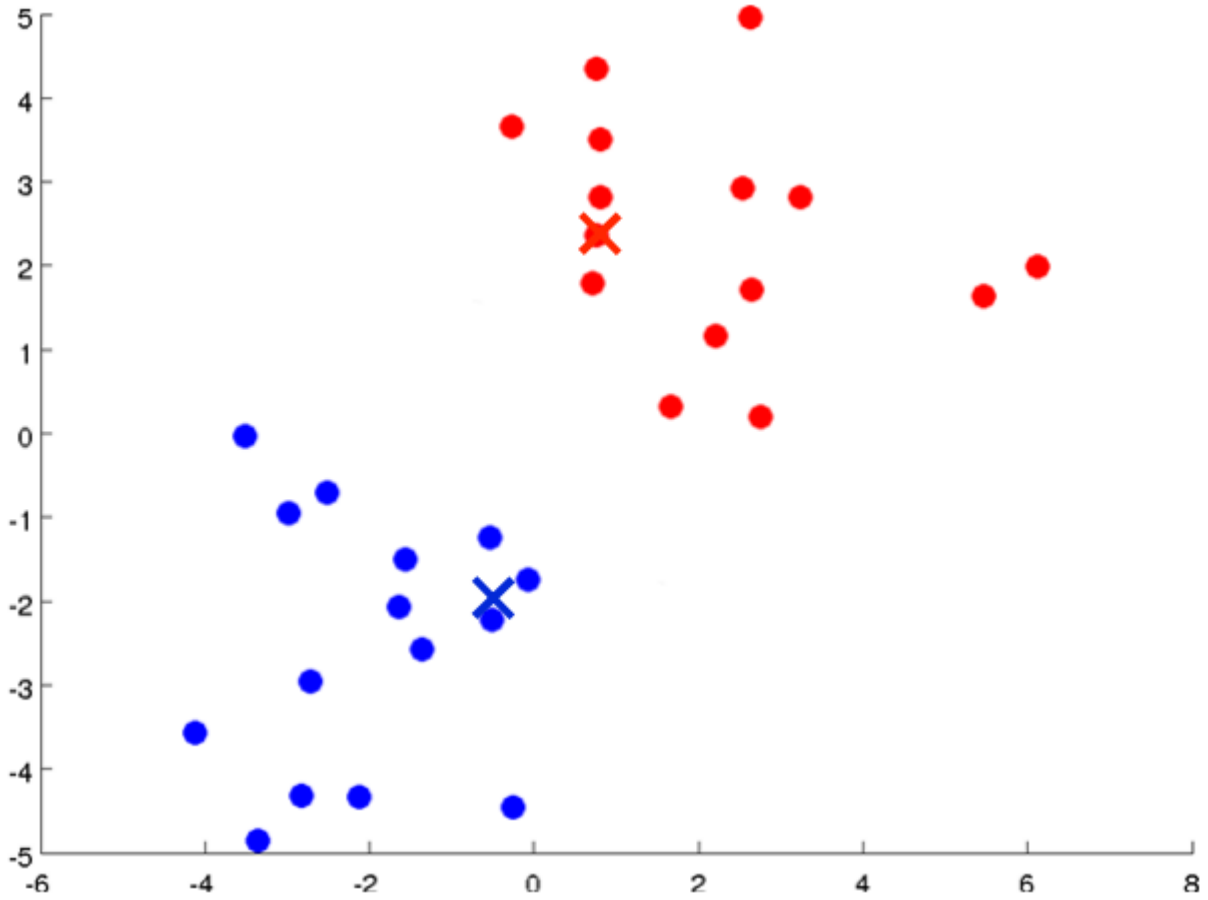
K-means



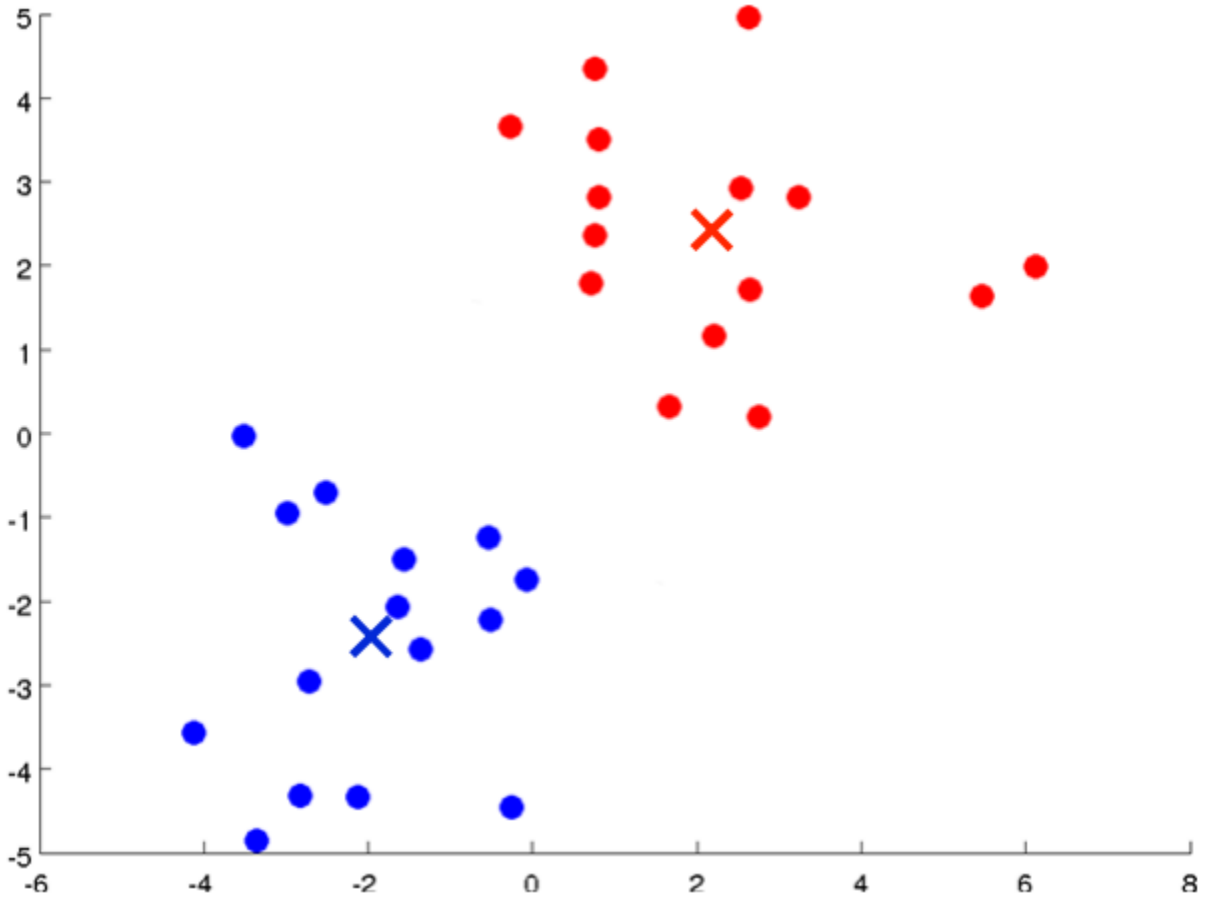
K-means



K-means



K-means



K-means algorithm

- Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K$:
- Repeat {

  For $i = 1$ to m:

  $C^{(i)}$ = index (from 1 to K) of cluster centroid closest to $x^{(i)}$

  For $k = 1$ to K:

  μ_k = average of points assigned to cluster k }

For example:

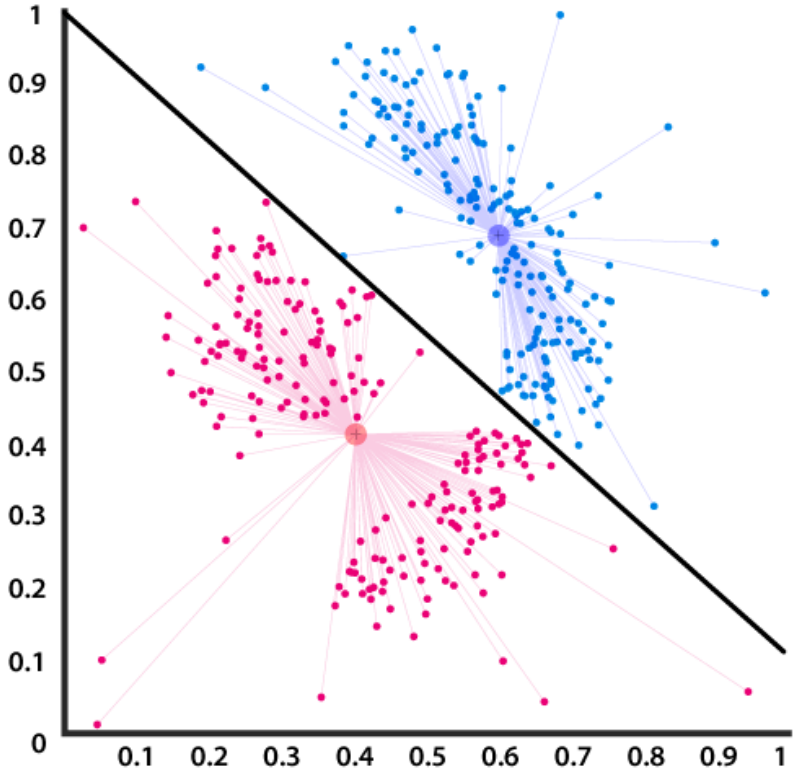
For four points $x^{(1)}, x^{(4)}, x^{(7)}, x^{(10)}$ in the same cluster #2, the index of cluster centroid for all of them is as $C^{(1)} = 2,\ C^{(4)} = 2,\ C^{(7)} = 2,\ C^{(10)} = 2,$ and

$$\mu_2 = \frac{1}{4}(x^{(1)} + x^{(4)} + x^{(7)} + x^{(10)})$$

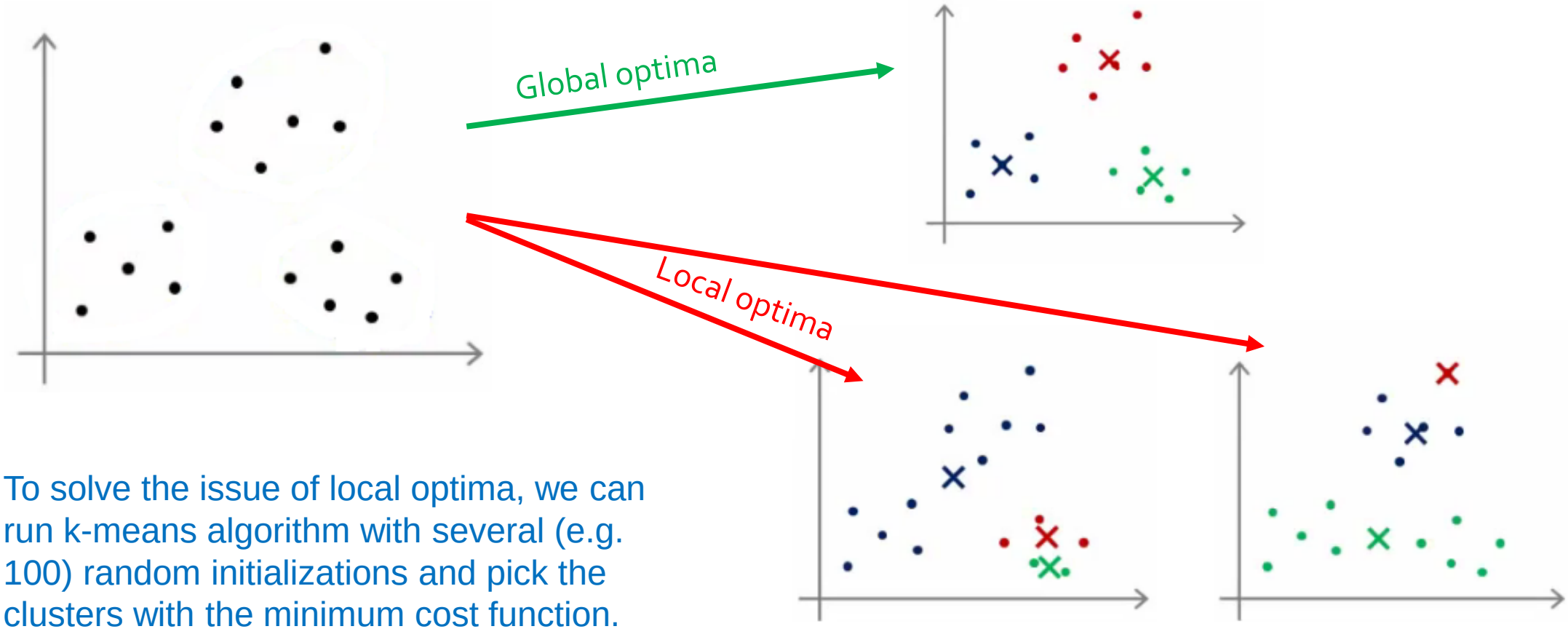
K-means objective function

- $C^{(i)}$ = index (from 1 to K) of cluster centroid where $x^{(i)}$ is assigned
- μ_k = cluster centroid k ($\mu_k \in \mathbb{R}^n$)
- $\mu_{c^{(i)}}$ = centroid of cluster to which $x^{(i)}$ is assigned
- Objective function:

$$J\left(C^{(1)}, \ldots, C^{(m)}, \mu_1, \ldots, \mu_K\right) = \frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - \mu_{c^{(i)}}\right\|^2$$



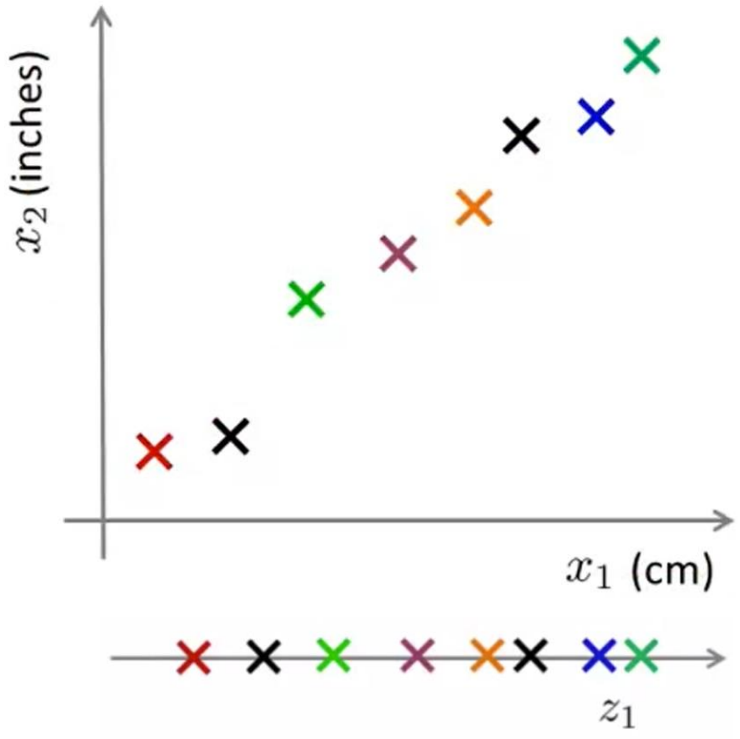
K-means weakness (local optima)



To solve the issue of local optima, we can run k-means algorithm with several (e.g. 100) random initializations and pick the clusters with the minimum cost function.

Principal Component Analysis

- Data compression



Reduce dimension from 2D to 1D

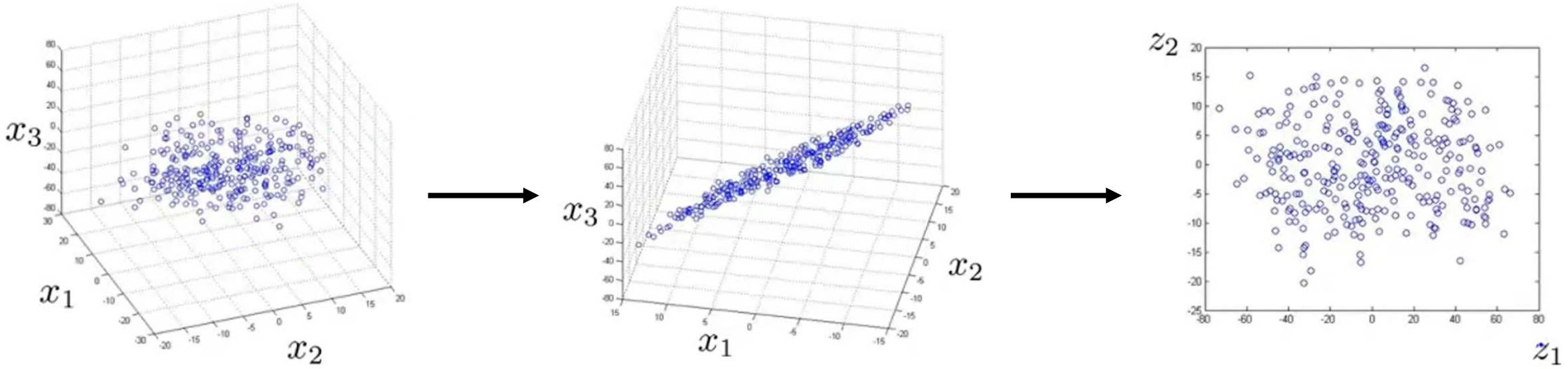
$$x^{(1)} = \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} \in \mathbb{R}^2 \longrightarrow z^{(1)} \in \mathbb{R}$$

$$x^{(2)} = \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} \in \mathbb{R}^2 \longrightarrow z^{(2)} \in \mathbb{R}$$

Benefits:

1- Less memory usage
2- Less run time of algorithms

Principal Component Analysis

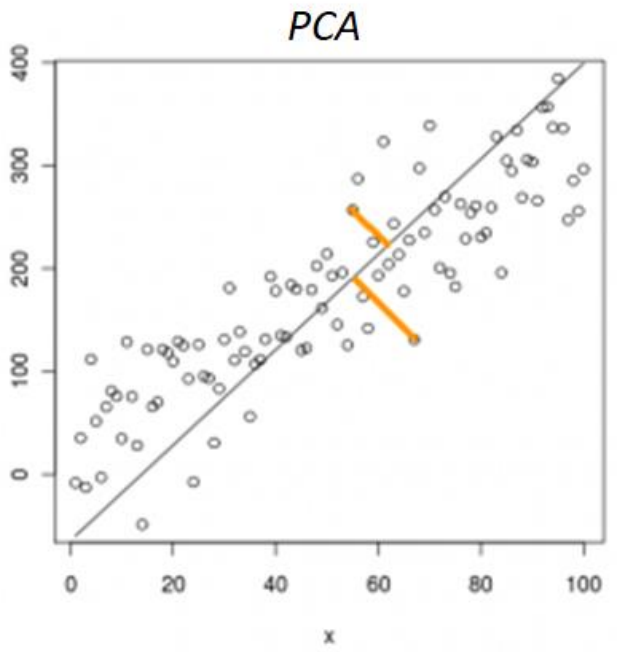


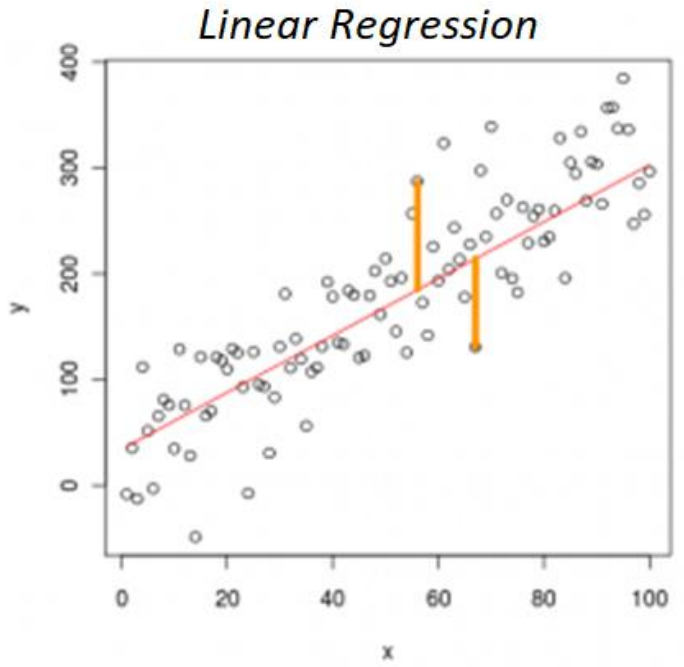
$x^{(i)} \in \mathbb{R}^3$

$z^{(i)} \in \mathbb{R}^2$

Principal Component Analysis

- How to find the lower dimension space?





But PCA is not linear regression

PCA Algorithm

1. Get the training data: $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$
2. Data preprocessing by feature scaling
3. Compute the covariance matrix: $\Gamma \in \mathbb{R}^{n \times n}$

$$\Gamma = \frac{1}{m} \sum_{i=1}^{m} \left(x^{(i)}\right)\left(x^{(i)}\right)^T$$

4. Compute eigenvectors of matrix Γ using singular value decomposition

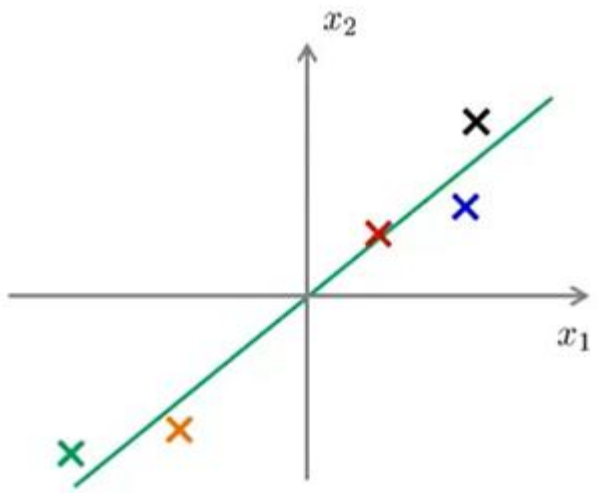
$$[U, S, V] = \text{svd}[\Gamma]$$

5. Select the first k columns of U.

$$U = \underbrace{\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \ldots & u^{(n)} \\ | & | & & | \end{bmatrix}}_{k} \in \mathbb{R}^{n \times n}$$

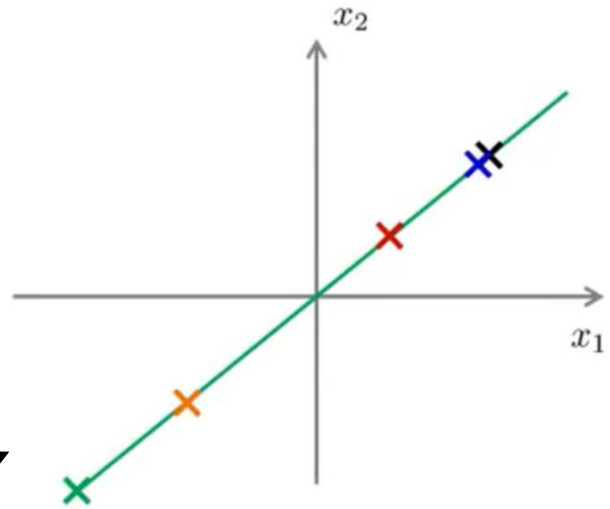
6. Compute $z = U_{reduce}^T x$

Reconstruction from reduced dimension



$z = U_{reduce}^T x$

$z \in \mathbb{R}$



$x_{approx} = U_{reduce} z$

$x_{approx} \in \mathbb{R}^2$

How to choose the number of principal components, k?

- Typically, k is the smallest value by which 99% of variance is retained.

$$[U, S, V] = \text{svd}(\Gamma)$$

where S is diagonal,

$$S = \begin{bmatrix} s_{11} & \overset{k}{\cdots} & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & s_{nn} \end{bmatrix}$$

then, find the smallest k by which

$$\frac{\sum_{i=1}^{k} s_{ii}}{\sum_{i=1}^{n} s_{ii}} \geq 0.99$$

Application of PCA

Original dataset: $\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)$

Extract unlabeled dataset and use PCA:

$$x^{(1)}, x^{(2)}, \ldots, x^{(m)} \in \mathbb{R}^n \xrightarrow{PCA} z^{(1)}, z^{(2)}, \ldots, z^{(m)} \in \mathbb{R}^k$$

New training set will be

$$\left(z^{(1)}, y^{(1)}\right), \left(z^{(2)}, y^{(2)}\right), \ldots, \left(z^{(m)}, y^{(m)}\right)$$

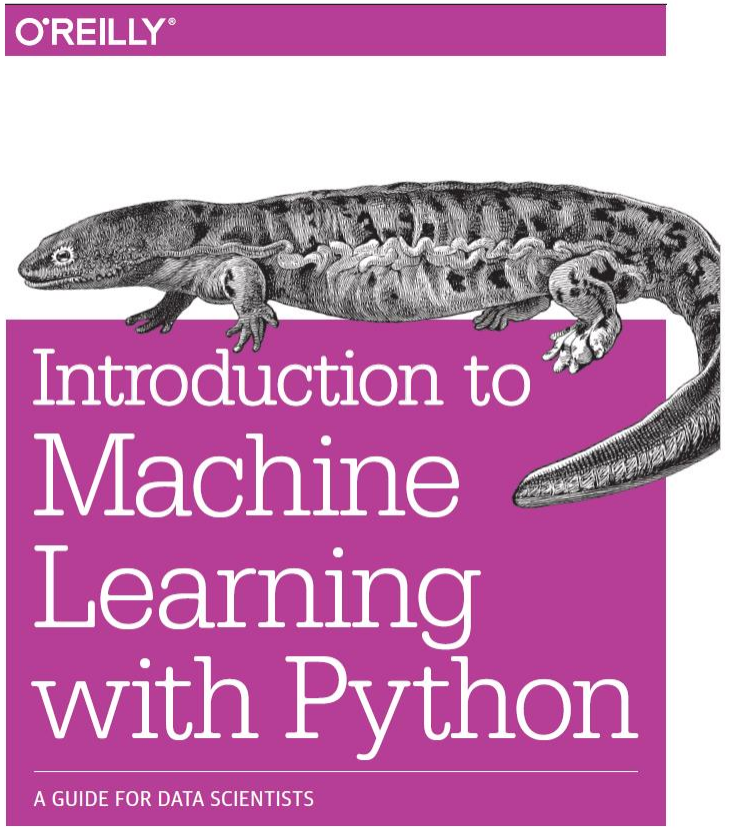
Now, we can use the new training set for training regressors or classifiers, but note that: mapping from $x^{(i)}$ to $z^{(i)}$ should be defined on the training set only. We use the same training mapping for validation and test sets.

K-means and PCA

Open Jupyter notebook:

Unsupervised learning.ipynb

Reference for ML



Acknowledgement

Special thanks to
MEGSA and CMEGSA

Survey

http://bit.ly/2THsQHh