

# Nearest neighbor methods

## Lecture 10

David Sontag  
New York University

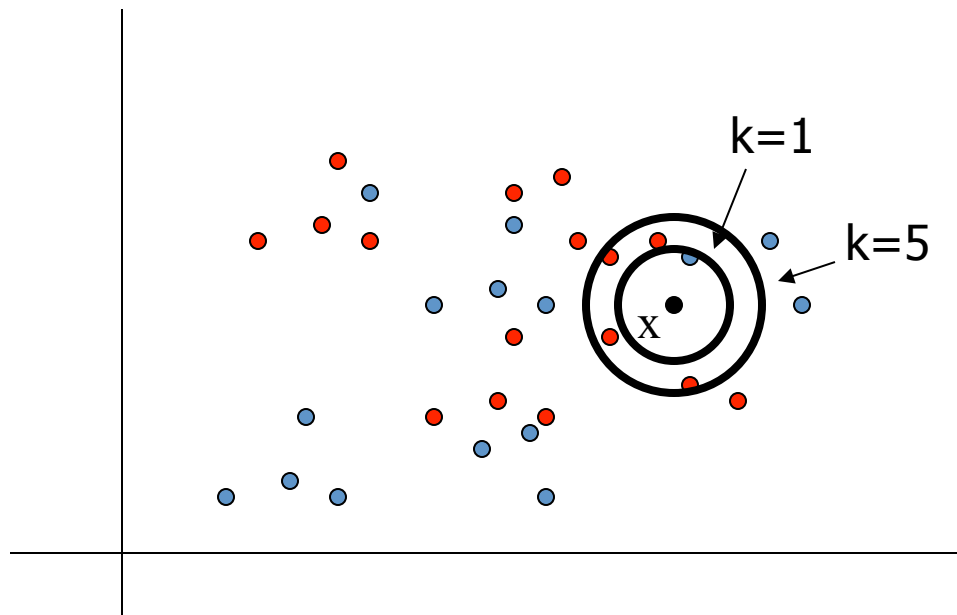
Slides adapted from Vibhav Gogate, Carlos Guestrin,  
Mehryar Mohri, & Luke Zettlemoyer

# Nearest Neighbor Algorithm

- Learning Algorithm:
  - Store training examples
- Prediction Algorithm:
  - To classify a new example  $\mathbf{x}$  by finding the training example  $(\mathbf{x}^i, y^i)$  that is *nearest* to  $\mathbf{x}$
  - Guess the class  $y = y^i$

# K-Nearest Neighbor Methods

- To classify a new input vector  $x$ , examine the  $k$ -closest training data points to  $x$  and assign the object to the most frequently occurring class



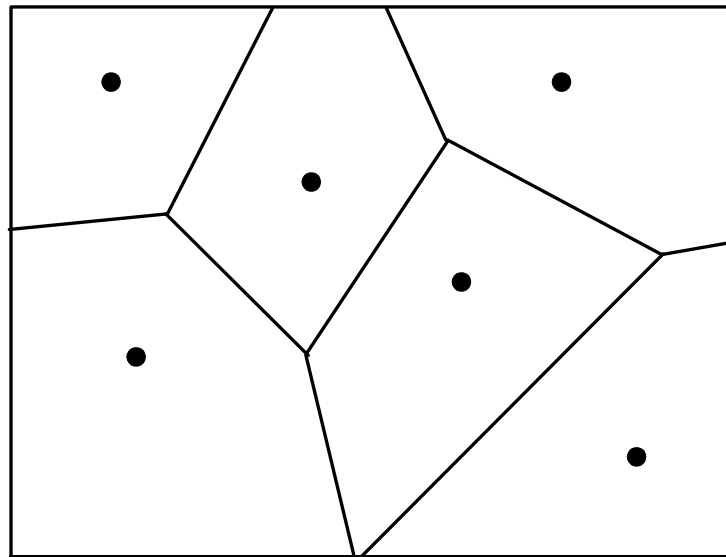
Regression:  
average of all values from the

common values for  $k$ : 3, 5

# Decision Boundaries

- The nearest neighbor algorithm does not explicitly compute decision boundaries. However, the decision boundaries form a subset of the Voronoi diagram for the training data.

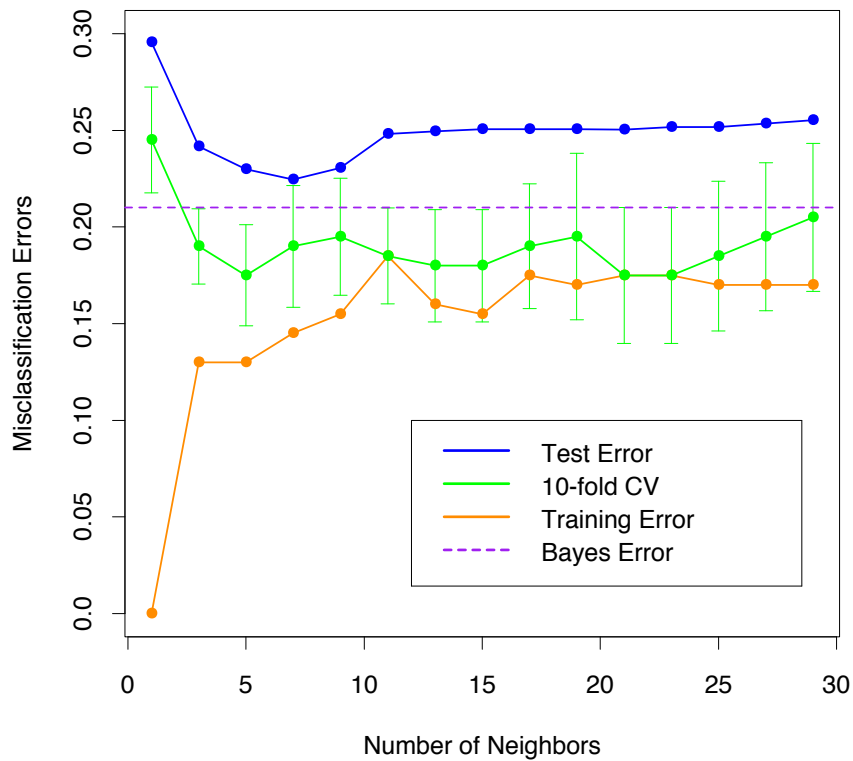
*1-NN Decision Surface*



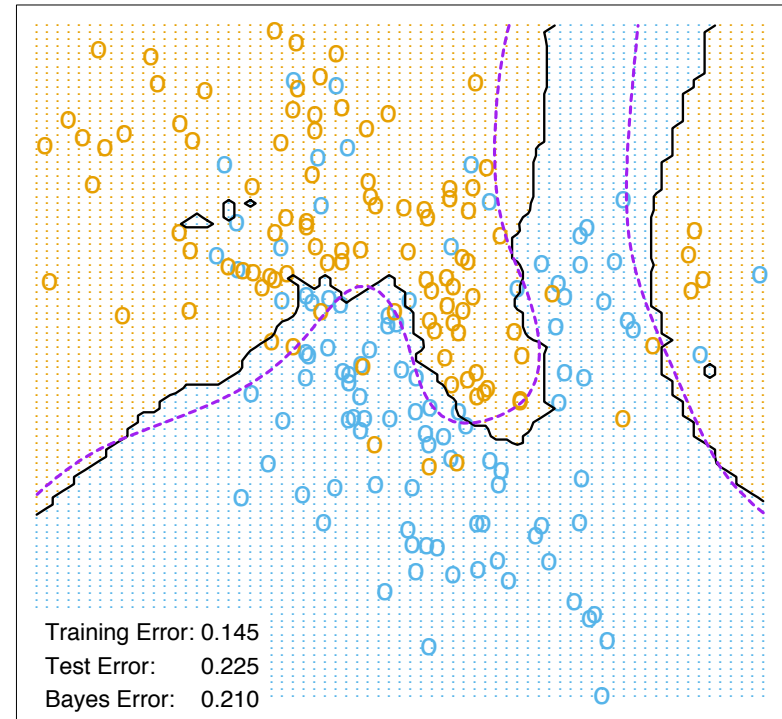
- The more examples that are stored, the more complex the decision boundaries can become

# Example results for k-NN

How to choose K ?



7-Nearest Neighbors



[Figures from Hastie and Tibshirani, Chapter 13]

# Nearest Neighbor

## When to Consider

- Instance map to points in  $R^n$
- Less than 20 attributes per instance
- Lots of training data

## Advantages

- Training is very fast
- Learn complex target functions
- Do not lose information

## Disadvantages

- Slow at query time
- Easily fooled by irrelevant attributes

# Issues

- Distance measure
  - Most common: Euclidean
- Choosing  $k$ 
  - Increasing  $k$  reduces variance, increases bias
- For high-dimensional space, problem that the nearest neighbor may not be very close at all!
- Memory-based technique. Must make a pass through the data for each classification. This can be prohibitive for large data sets.

## Distance

- Notation: object with  $p$  measurements

$$\mathbf{x}^i = (x_1^i, x_2^i, \dots, x_p^i)$$

- Most common distance metric is *Euclidean* distance:

$$d_E(\mathbf{x}^i, \mathbf{x}^j) = \left( \sum_{k=1}^p (x_k^i - x_k^j)^2 \right)^{\frac{1}{2}}$$

- ED makes sense when different measurements are commensurate; each is variable measured in the same units.
- If the measurements are different, say length and weight, it is not clear.



## Standardization

When variables are not commensurate, we can standardize them by dividing by the sample standard deviation. This makes them all equally important.

The estimate for the standard deviation of  $x_k$  :

$$\hat{\sigma}_k = \left( \frac{1}{n} \sum_{i=1}^n (x_k^i - \bar{x}_k)^2 \right)^{\frac{1}{2}}$$

where  $\bar{x}_k$  is the sample mean:

$$\bar{x}_k = \frac{1}{n} \sum_{i=1}^n x_k^i$$

## Weighted Euclidean distance

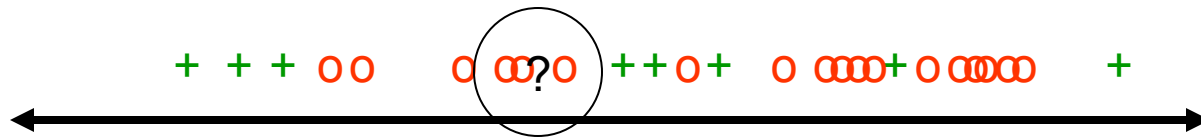
Finally, if we have some idea of the relative importance of each variable, we can weight them:

$$d_{\text{WE}}(i, j) = \left( \sum_{k=1}^p w_k (x_k^i - x_k^j)^2 \right)^{\frac{1}{2}}$$

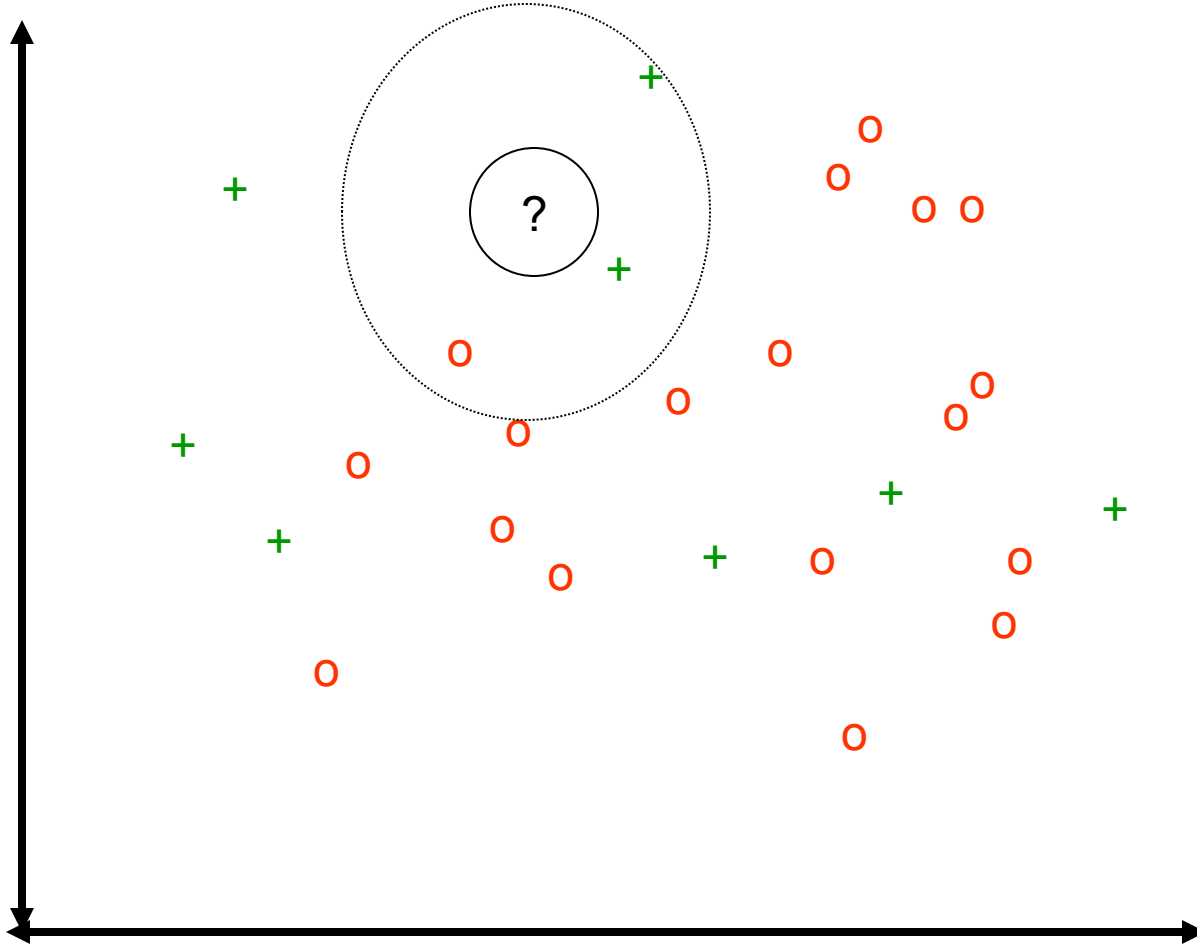
# The Curse of Dimensionality

- Nearest neighbor breaks down in high-dimensional spaces because the “neighborhood” becomes very large.
- Suppose we have 5000 points uniformly distributed in the unit hypercube and we want to apply the 5-nearest neighbor algorithm.
- Suppose our query point is at the origin.
  - 1D –
    - On a one dimensional line, we must go a distance of  $5/5000 = 0.001$  on average to capture the 5 nearest neighbors
  - 2D –
    - In two dimensions, we must go  $\sqrt{0.001}$  to get a square that contains 0.001 of the volume
  - D –
    - In D dimensions, we must go  $(0.001)^{1/D}$

# K-NN and irrelevant features



# K-NN and irrelevant features



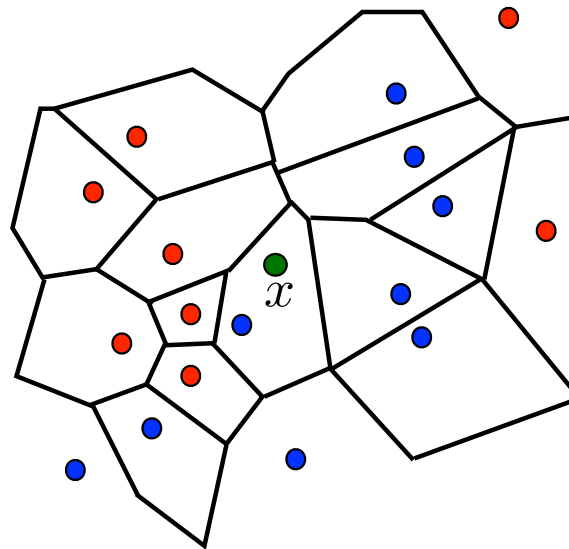
## Nearest neighbor problem

- **Problem:** given sample  $S = ((x_1, y_1), \dots, (x_m, y_m))$ , find the nearest neighbor of test point  $x$ .
- general problem extensively studied in computer science.
- exact vs. approximate algorithms.
- dimensionality  $N$  crucial.
- better algorithms for small intrinsic dimension (e.g., limited doubling dimension).

## Efficient Indexing: N=2

### ■ Algorithm:

- compute Voronoi diagram in  $O(m \log m)$ .
- **point location** data structure to determine NN.
- complexity:  $O(m)$  space,  $O(\log m)$  time.



## KNN Advantages

- Easy to program
- No optimization or training required
- Classification accuracy can be very good; can outperform more complex models



# CMSC 478

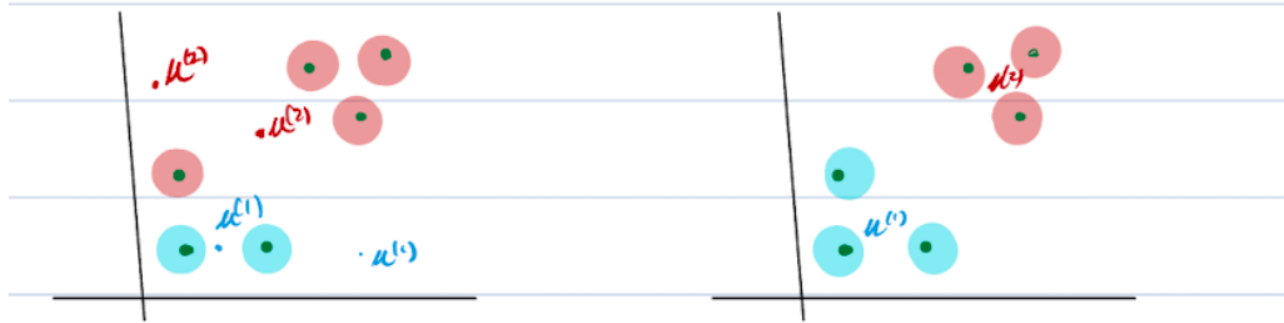
## Unsupervised Learning

### K-means Clustering

KMA Solaiman

[ksolaima@umbc.edu](mailto:ksolaima@umbc.edu)

## How do we find these clusters? (Iterative Approach)



- ▶ (Randomly) Initialize Centers  $\mu^{(1)}$  and  $\mu^{(2)}$ .
- ▶ Assign each point,  $x^{(i)}$ , to closest cluster

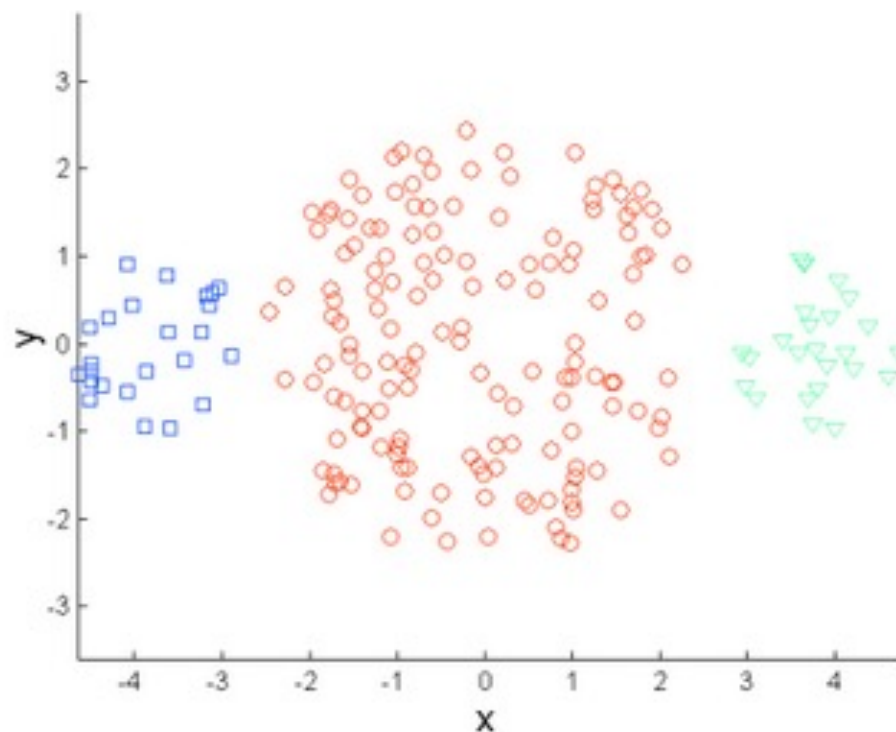
$$C^{(i)} = \operatorname{argmin}_{j=1,\dots,k} \|\mu^{(j)} - x^{(i)}\|^2 \text{ for } i = 1, \dots, n$$

- ▶ Compute new center of each cluster:

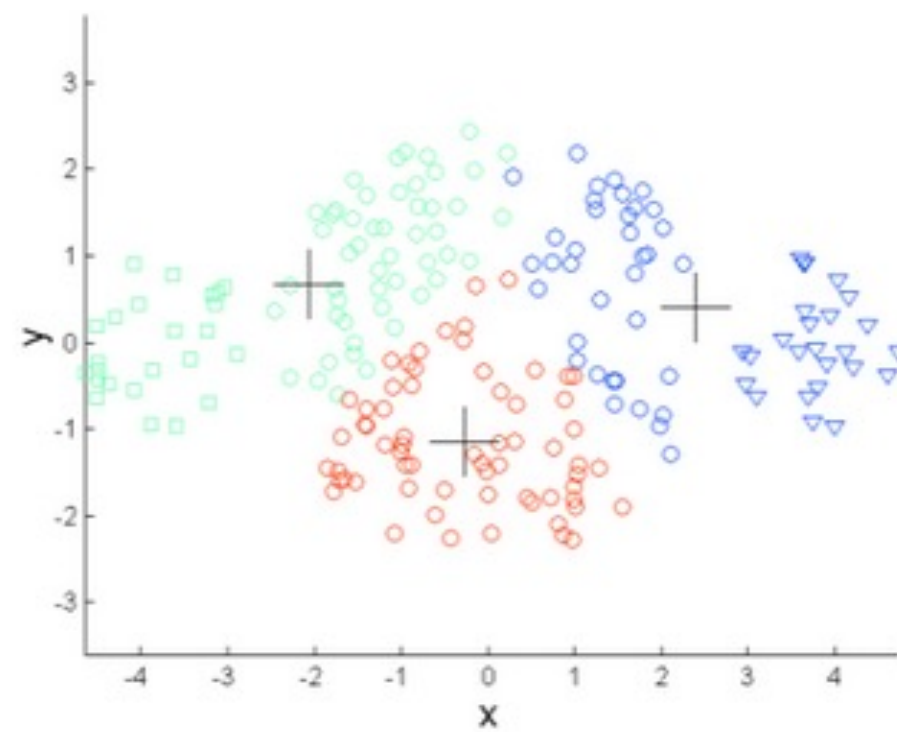
$$\mu^{(j)} = \frac{1}{|\Omega_j|} \sum_{i \in \Omega_j} x^{(i)} \text{ where } \Omega_j = \{i : C^{(i)} = j\}$$

Repeat until clusters stay the same!

# Different number of clusters

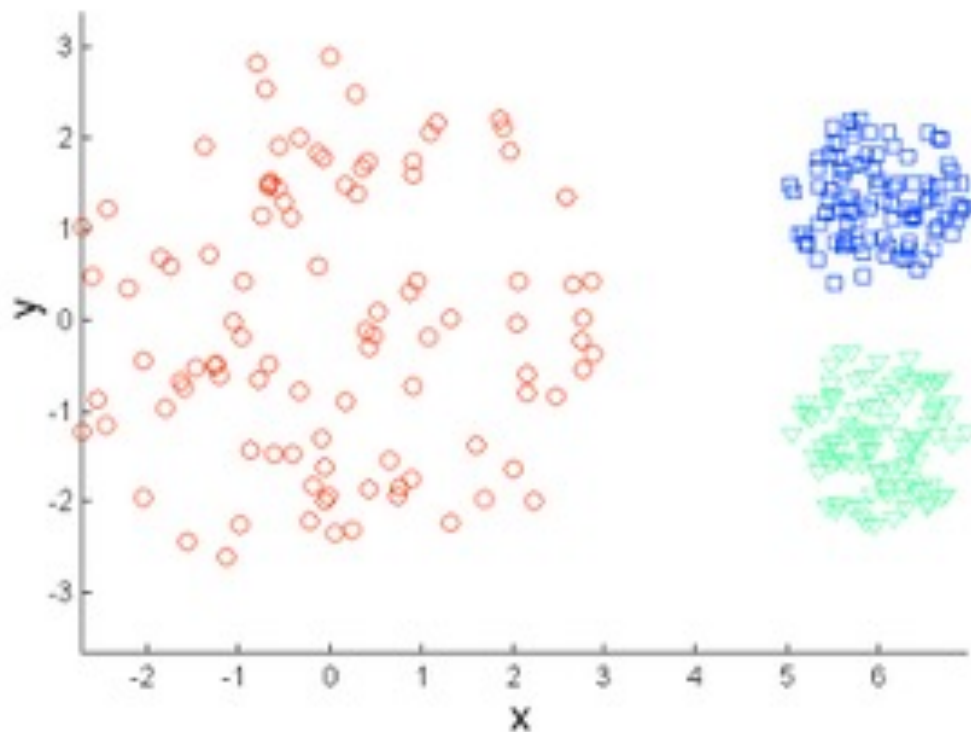


Original Points

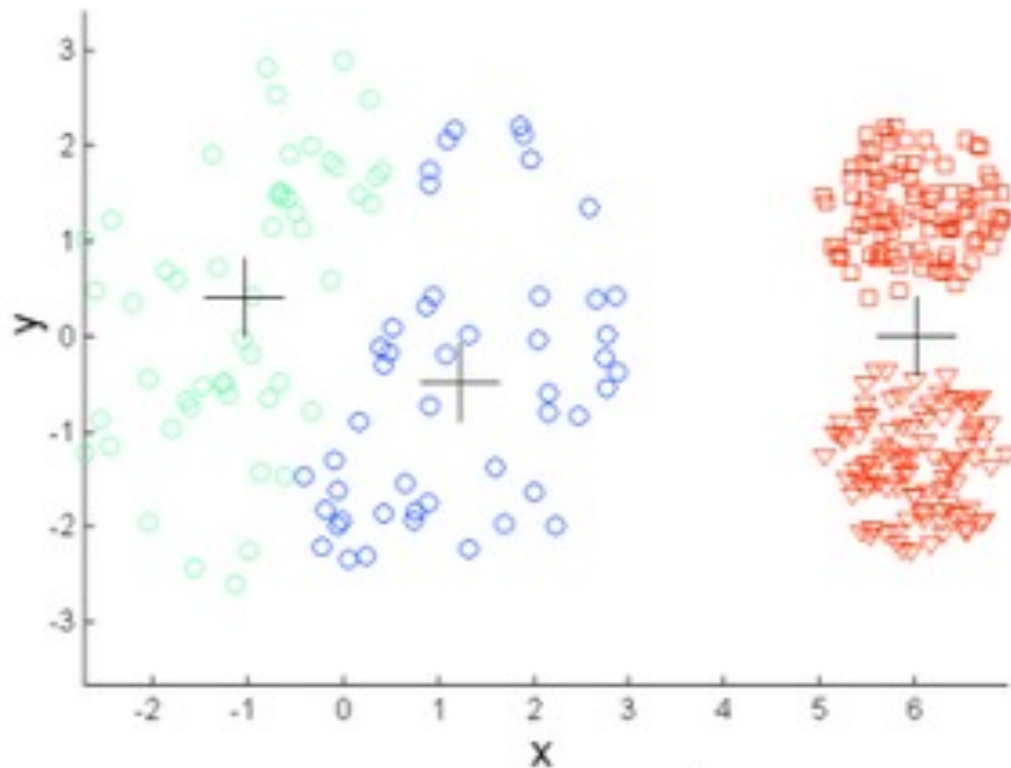


K-means ( $k = 3$ )

# Different Densities



Original Points



K-means ( $k = 3$ )

# Choosing K?

- # of clusters
- Cluster centers
  - K-means++
- Sensitivity to outliers
  - identify and handle outliers before applying k-means clustering
  - removing them, transforming them, or using a robust variant of k-means clustering that is less sensitive to the presence of outliers

# K-means++

- Compute Density Estimation
- Assign centroids based on that

# K-means++

- Compute Density Estimation
- Assign centroids based on that



# K-means++

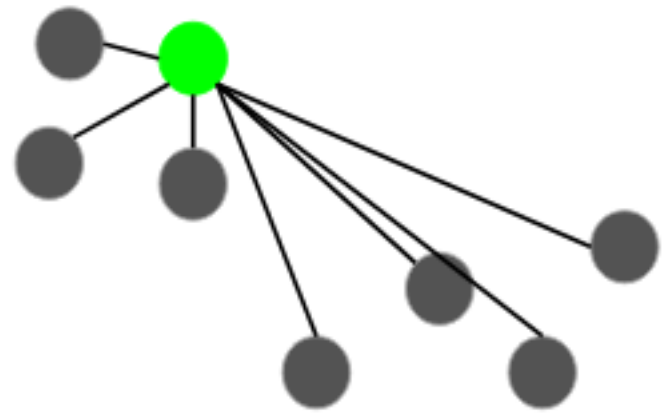
- Compute Density Estimation
- Assign centroids based on that
- 3 clusters







Random Pick



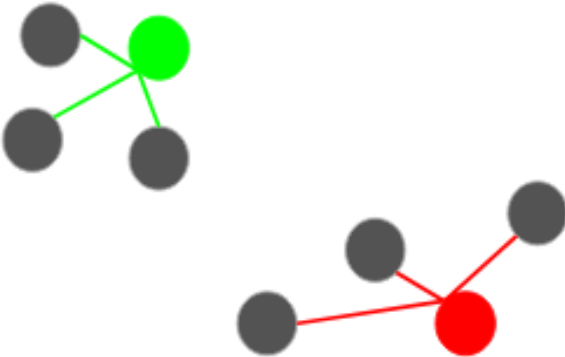
Calculate  $D(x)$



Largest  $D(x)^2$

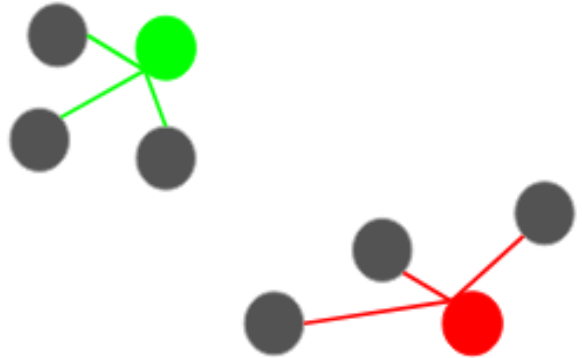


Largest  $D(x)^2$





Largest  $D(x)^2$

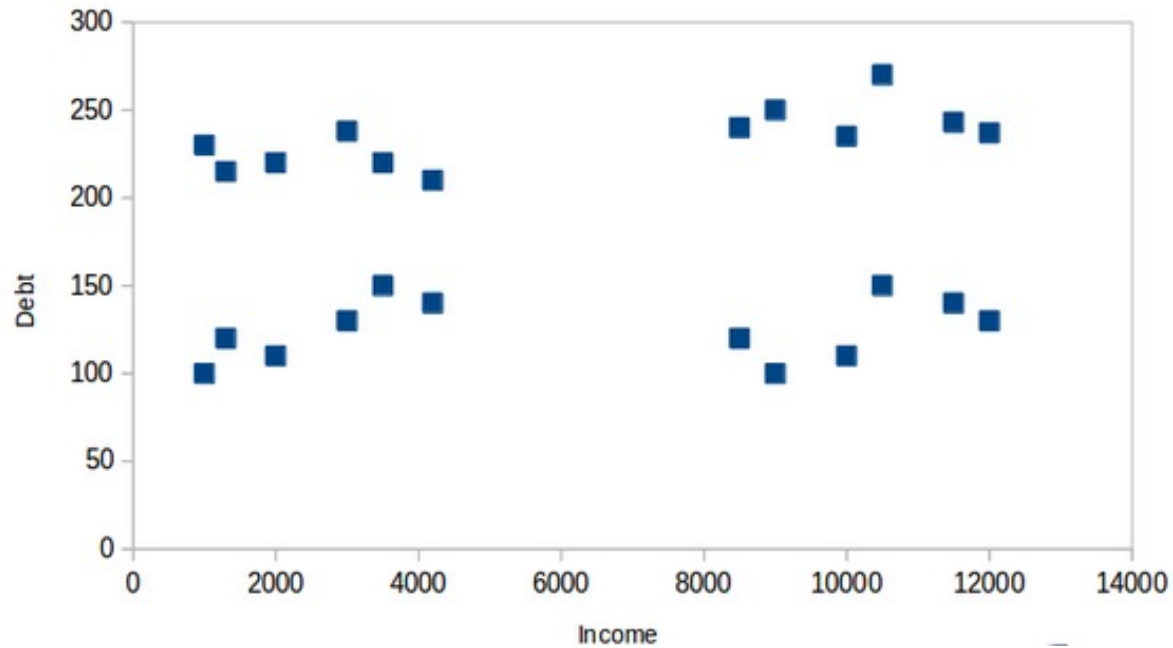


Largest  $D(x)^2$   
from both center

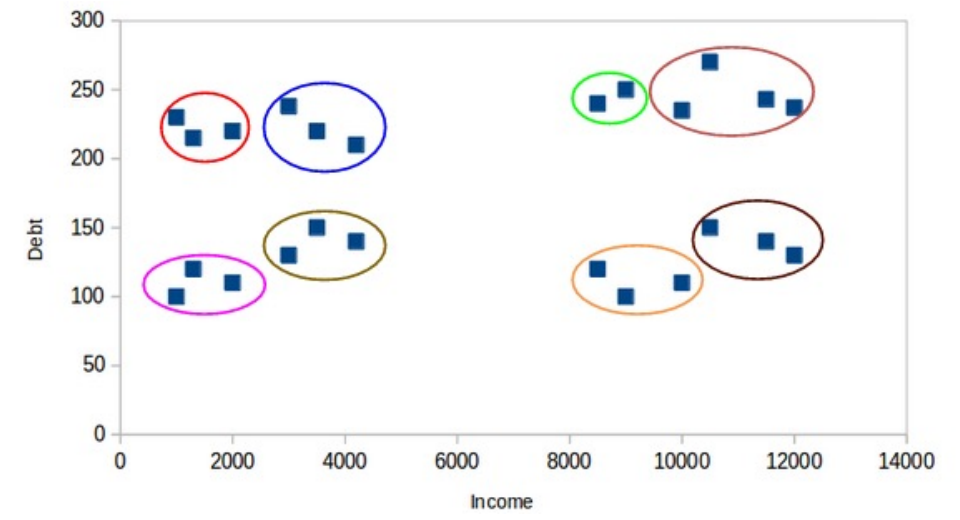
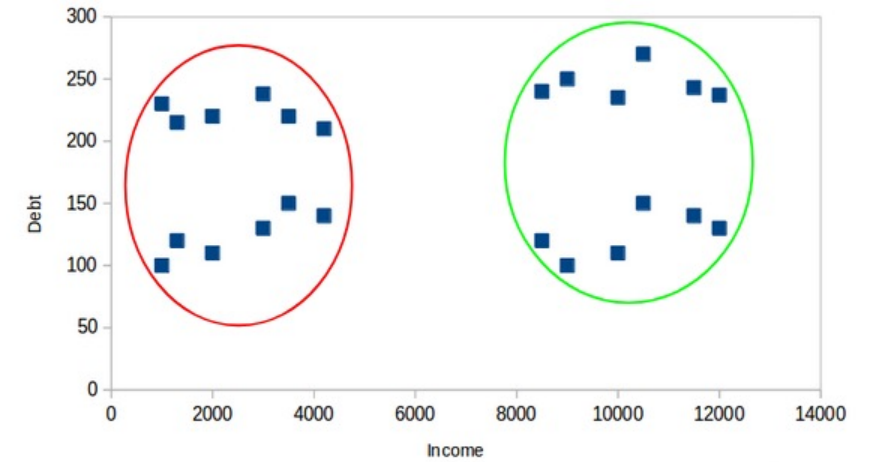
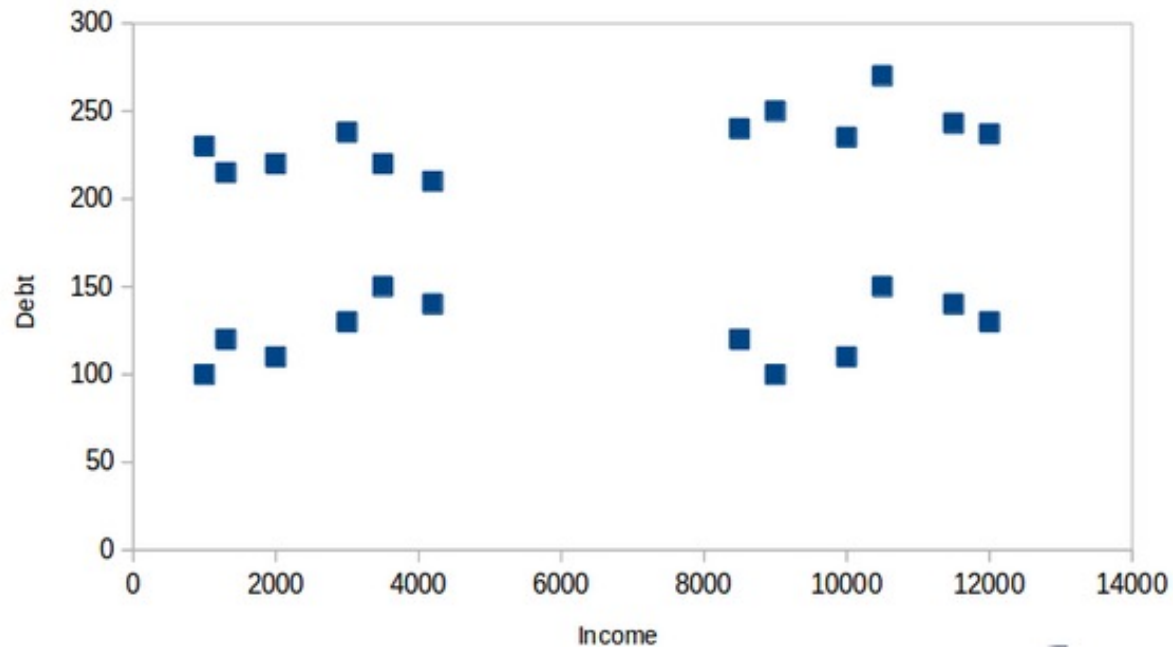


- Steps to Initialize the Centroids Using K-Means++
  1. The first cluster is chosen uniformly at random from the data points we want to cluster. This is similar to what we do in K-Means, but instead of randomly picking all the centroids, we just pick one centroid here
  2. Next, we compute the distance ( $D(x)$ ) of each data point ( $x$ ) from the cluster center that has already been chosen
  3. Then, choose the new cluster center from the data points with the probability of  $x$  being proportional to  $(D(x))^2$
  4. We then repeat steps 2 and 3 until  $k$  clusters have been chosen

# How to Choose the Right Number of Clusters?

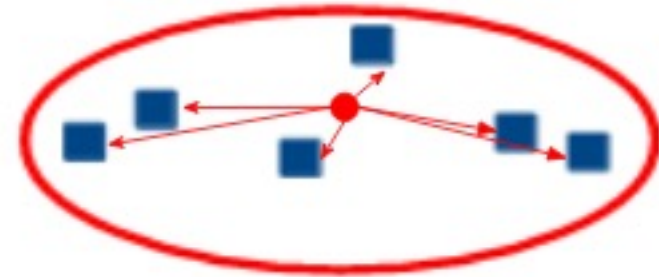


# How to Choose the Right Number of Clusters?



# Evaluation Metrics

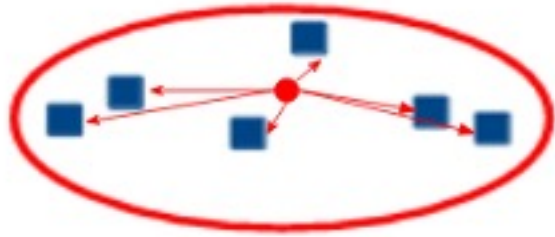
- Inertia
  - sum of distances of all the points within a cluster from the centroid of that cluster.
  - lesser the inertia value, the better our clusters are.
- Silhouette Score
  - high silhouette score = clusters are well separated
  - 0 = overlapping clusters,
  - negative score suggests poor clustering solutions.
  - For each data,
$$s = (b - a) / \max(a, b)$$
    - 'a' is the average distance within the cluster, 'b' is the average distance to the nearest cluster, and 'max(a, b)' is the maximum of 'a' and 'b'
  - Mean for all points



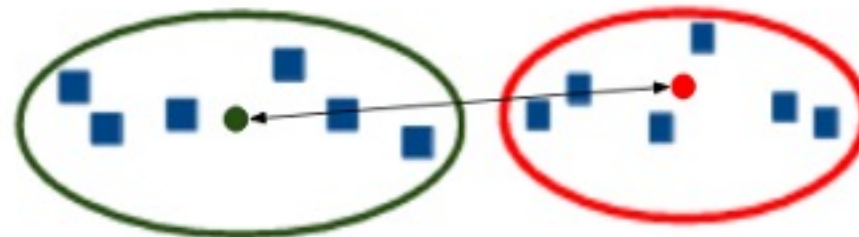
Intra cluster distance



- Dunn index



Intra cluster distance



Inter cluster distance

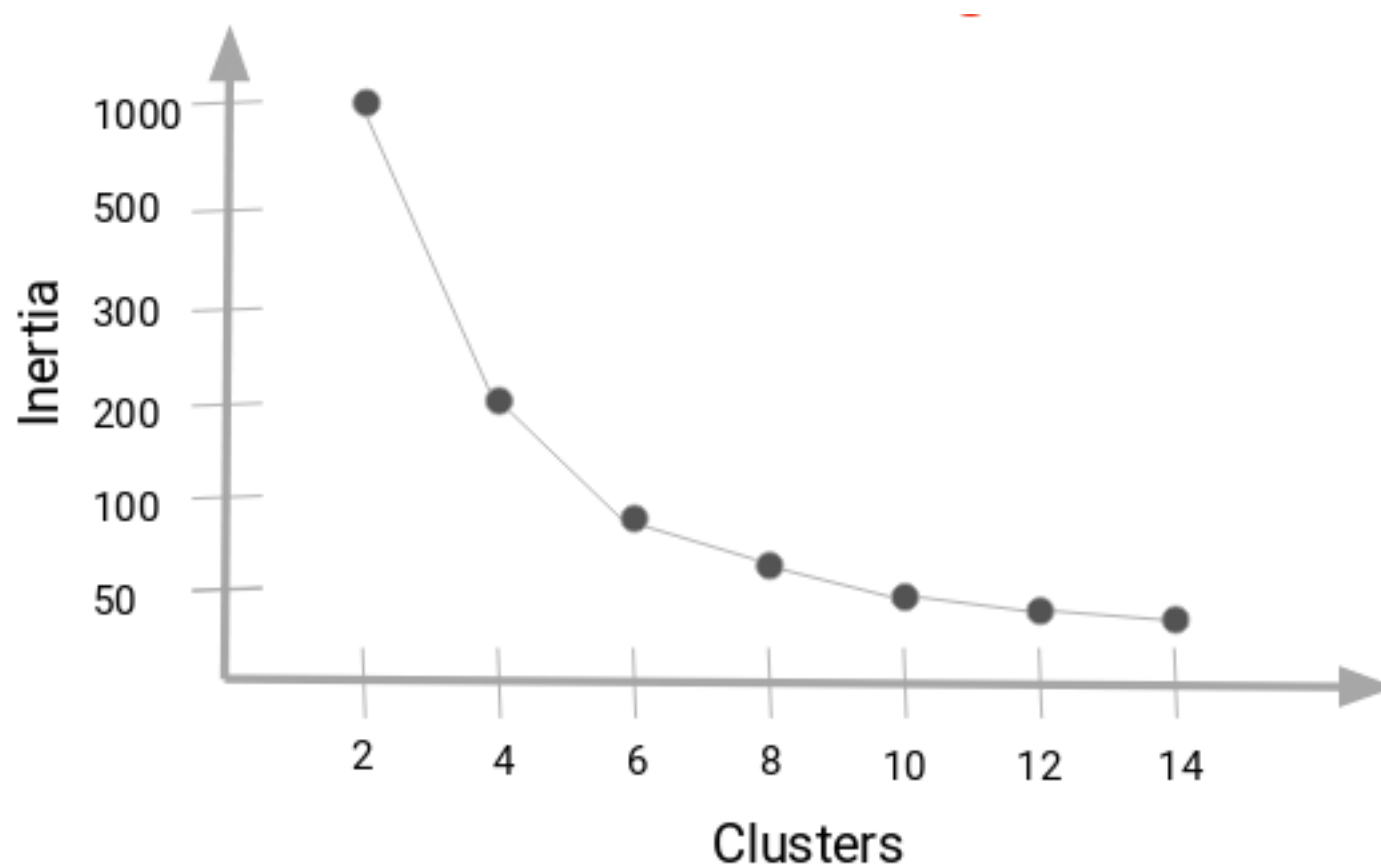
$$\text{Dunn Index} = \frac{\text{min(Inter cluster distance)}}{\text{max(Intra cluster distance)}}$$

Clusters are far apart

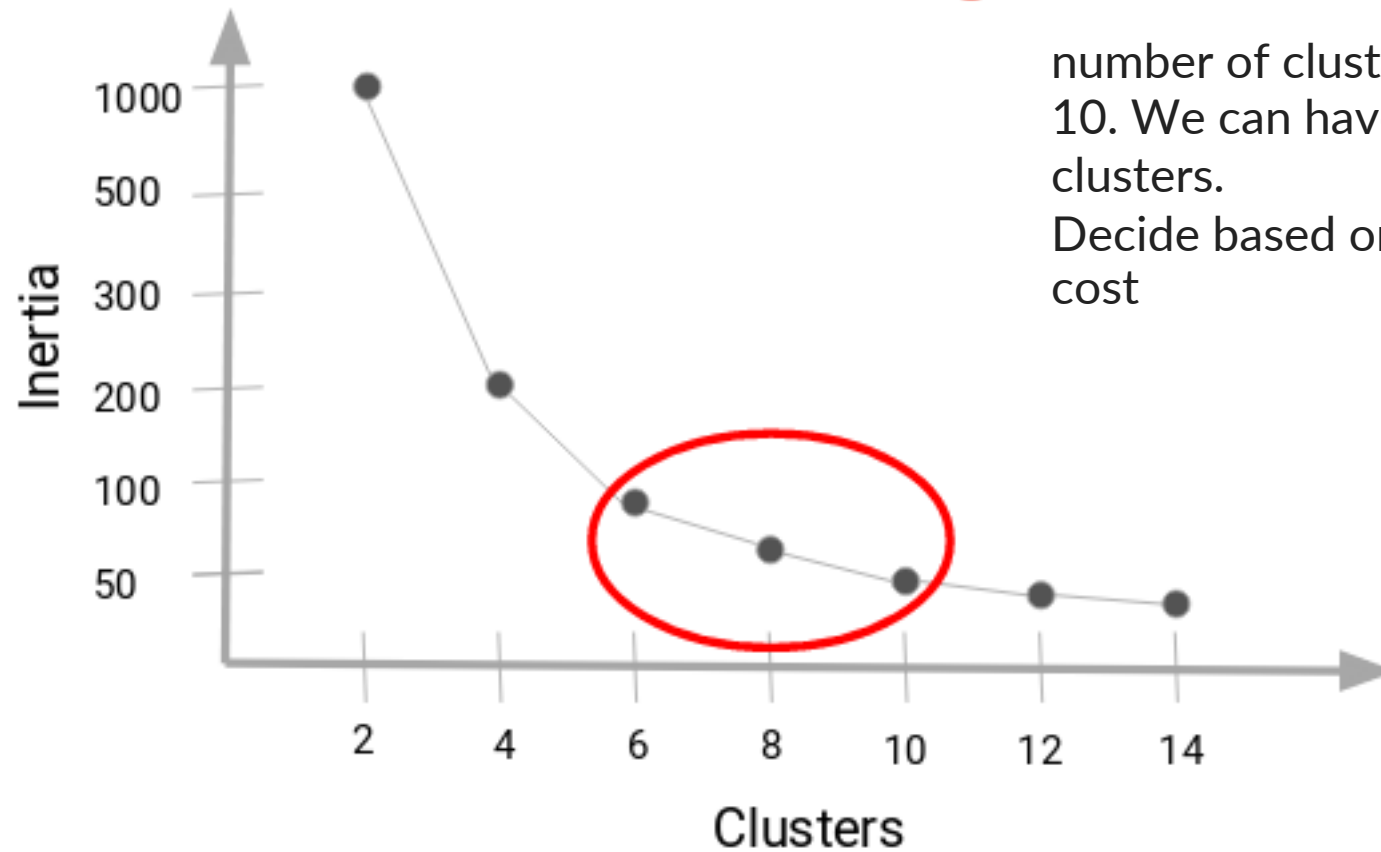
$$\text{Dunn Index} = \frac{\text{min(Inter cluster distance)}}{\text{max(Intra cluster distance)}}$$

Clusters are compact

# Empirical Choice of K



# Empirical Choice of K



number of clusters between 6 and 10. We can have 7, 8, or even 9 clusters.  
Decide based on computational cost

# Unsupervised Learning: PCA and ICA

KMA Solaiman

Adapted from Chris Ré and  
Zilinkas

# Covariance

covariance: how (linearly) correlated are variables

$$\sigma_{ij} = \frac{1}{N-1} \sum_{k=1}^N (x_{ki} - \mu_i)(x_{kj} - \mu_j)$$

covariance of variables  $i$  and  $j$

Mean of variable  $i$

Mean of variable  $j$

Value of variable  $i$  in object  $k$

Value of variable  $j$  in object  $k$

# Covariance

covariance: how (linearly) correlated are variables

$$\sigma_{ij} = \frac{1}{N-1} \sum_{k=1}^N (x_{ki} - \mu_i)(x_{kj} - \mu_j)$$

covariance of variables  $i$  and  $j$

Mean of variable  $i$

Mean of variable  $j$

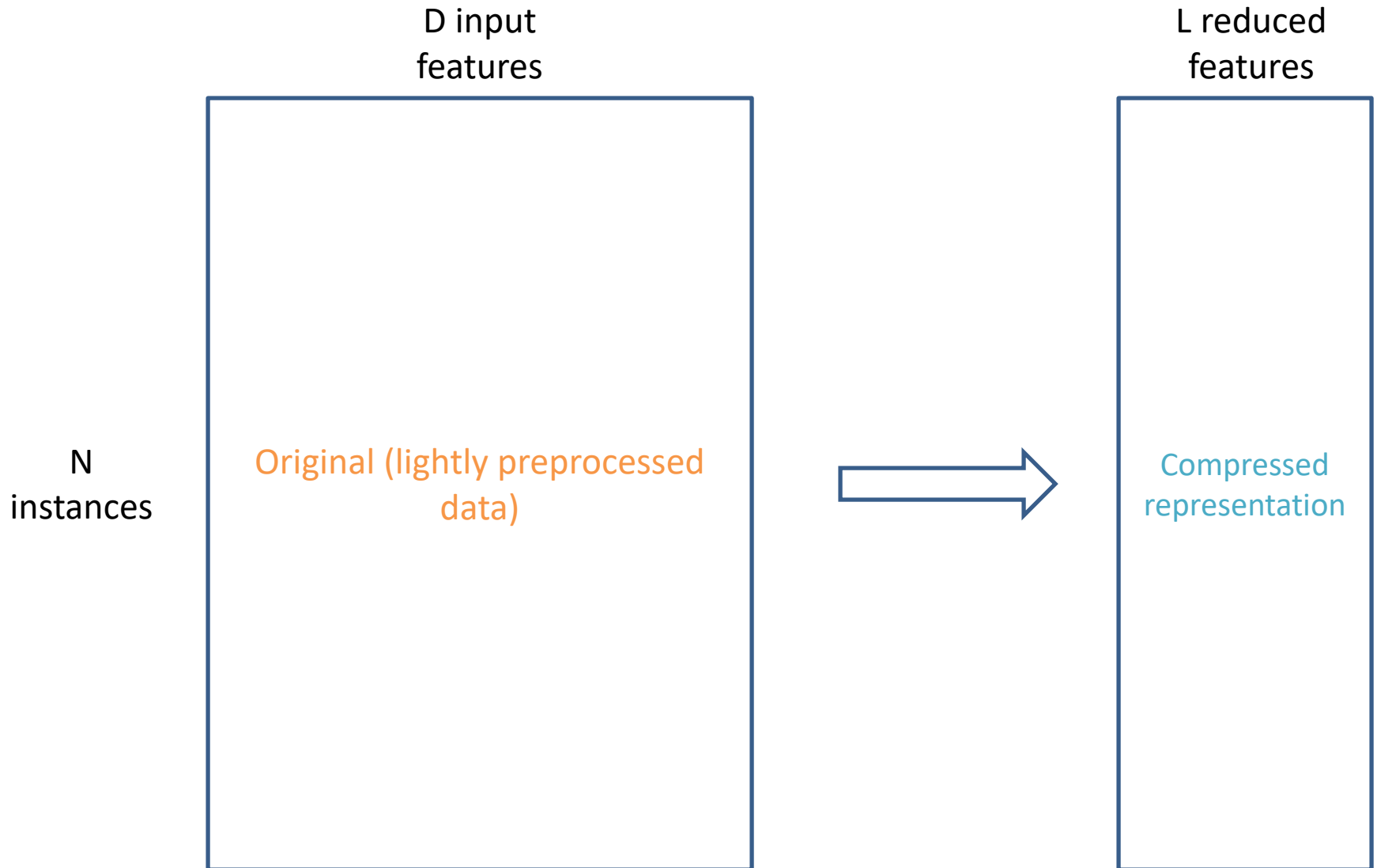
Value of variable  $i$  in object  $k$

Value of variable  $j$  in object  $k$

$$\sigma_{ij} = \sigma_{ji}$$

$$\Sigma = \begin{pmatrix} \sigma_{11} & \cdots & \sigma_{1K} \\ \vdots & \ddots & \vdots \\ \sigma_{K1} & \cdots & \sigma_{KK} \end{pmatrix}$$

# Dimensionality Reduction



# Dimensionality Reduction

clarity of representation vs. ease of understanding

oversimplification: loss of important or relevant  
information



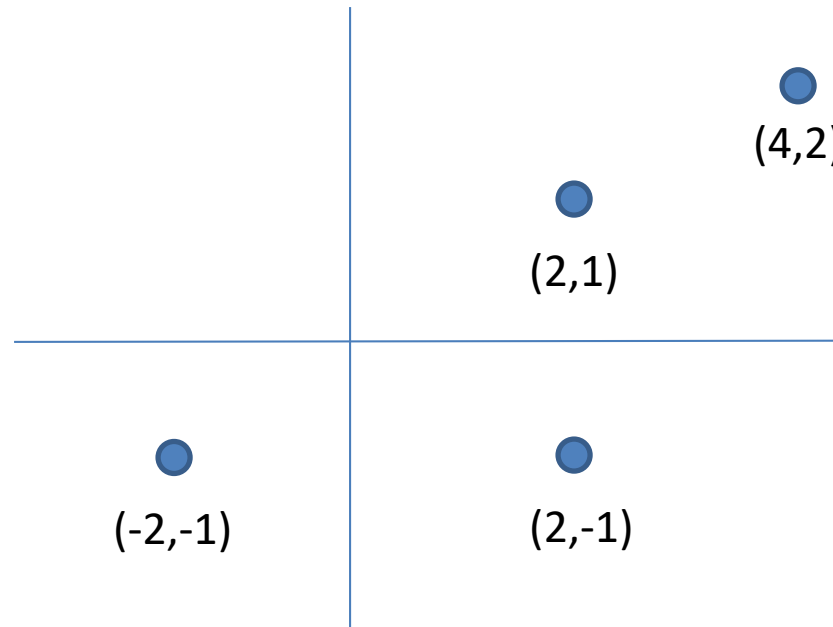
# Why “maximize” the variance?

How can we efficiently summarize? We maximize the variance within our summarization

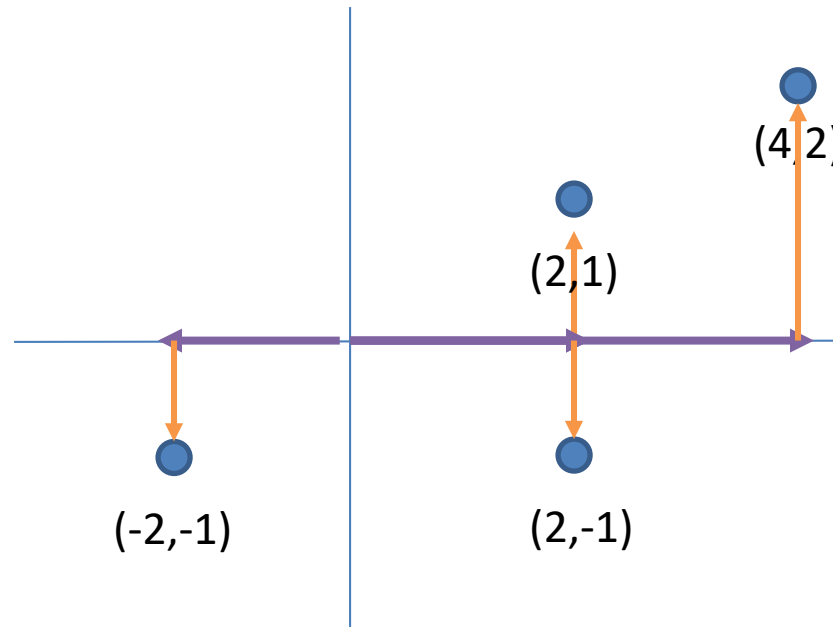
We don't increase the variance in the dataset

How can we capture the most information with the fewest number of axes?

# Summarizing Redundant Information

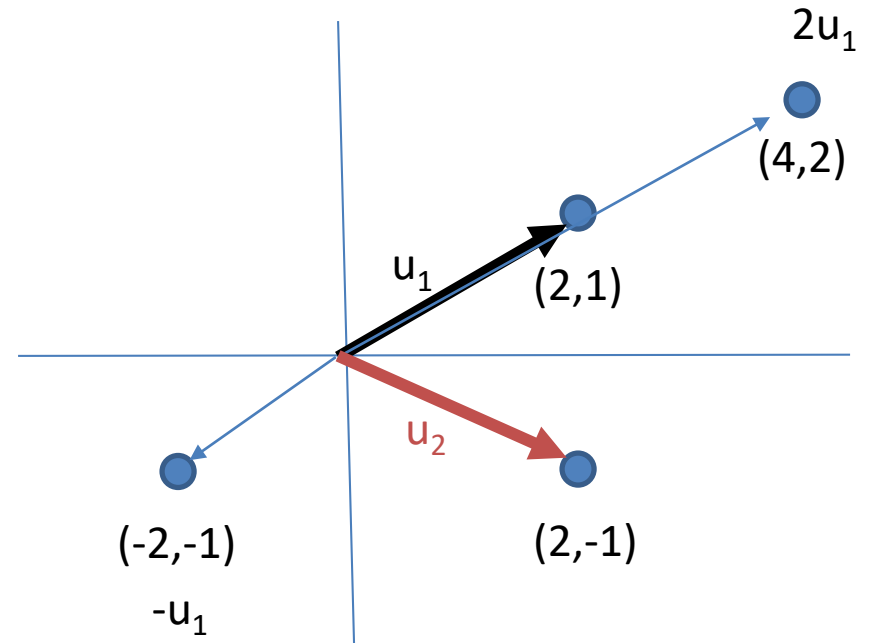
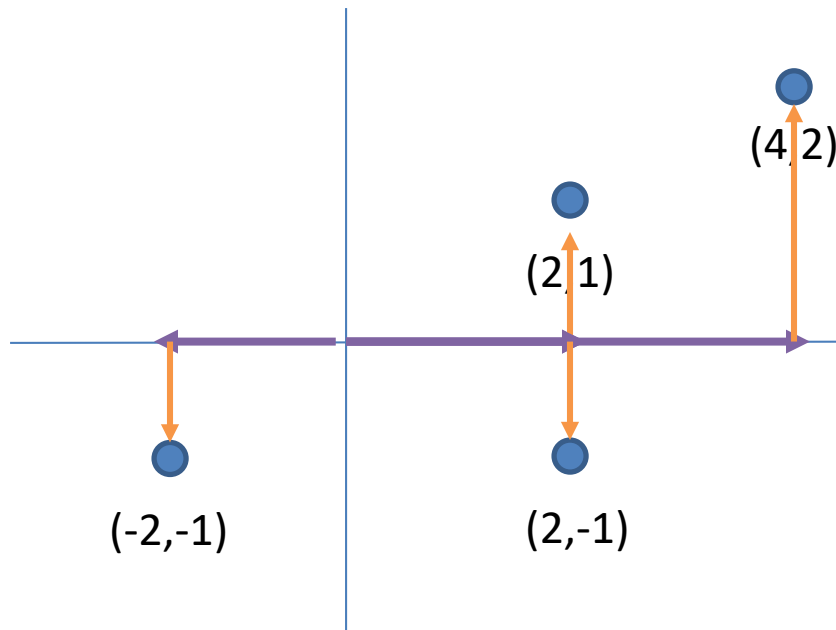


# Summarizing Redundant Information



$$(2,1) = 2*(1,0) + 1*(0,1)$$

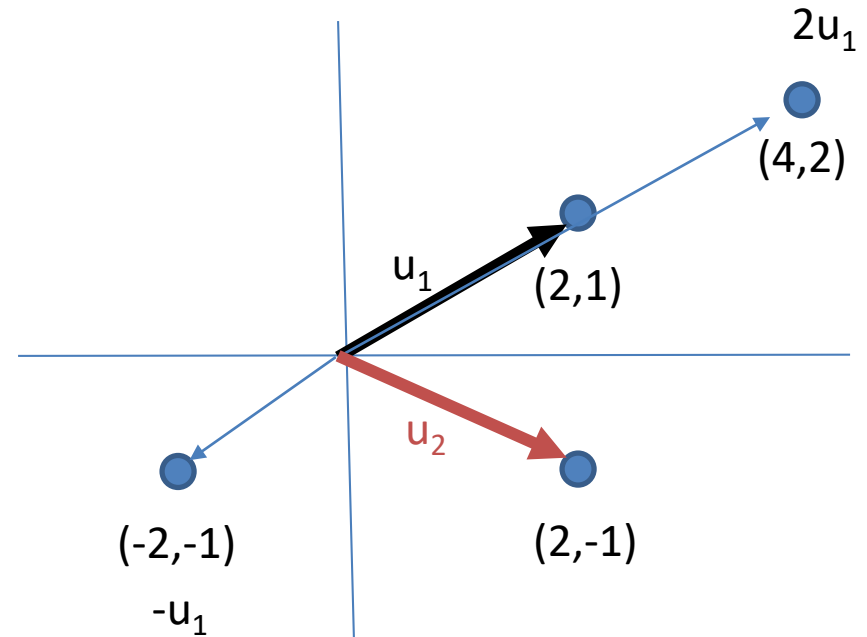
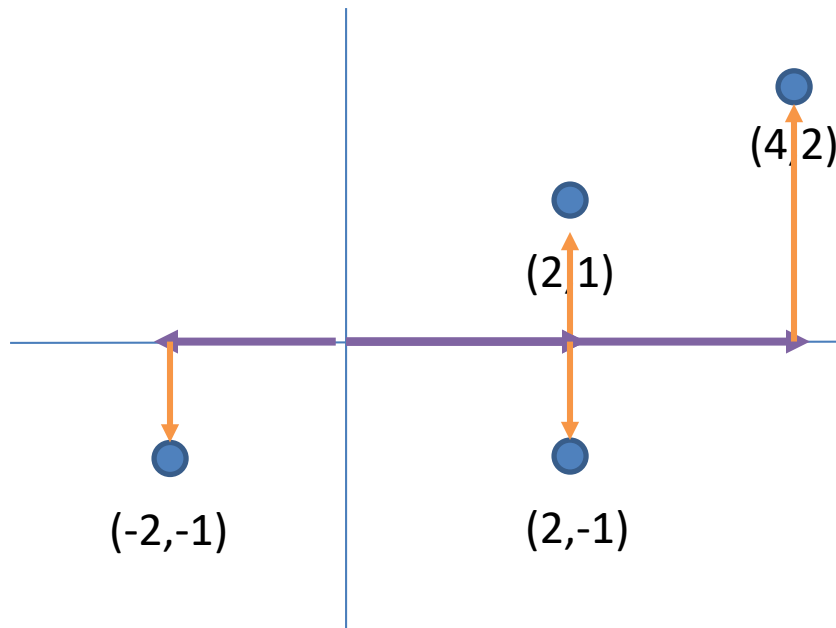
# Summarizing Redundant Information



$$(2, 1) = 1 \cdot (2, 1) + 0 \cdot (2, -1)$$

$$(4, 2) = 2 \cdot (2, 1) + 0 \cdot (2, -1)$$

# Summarizing Redundant Information



$$(2,1) = 1 \cdot (2,1) + 0 \cdot (2,-1)$$
$$(4,2) = 2 \cdot (2,1) + 0 \cdot (2,-1)$$

(Is it the most general? These vectors aren't orthogonal)

# Preprocessing

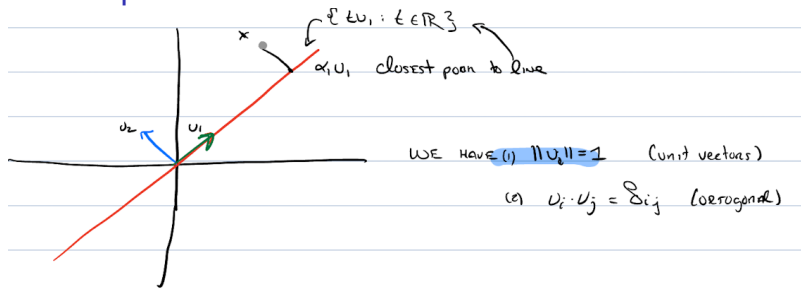
Given  $x^{(1)}, \dots, x^{(n)} \in \mathbb{R}^d$  we preprocess:

- ▶ **Center the data**  $x^{(i)} \mapsto x^{(i)} - \mu$
- ▶ **Rescale the data** May need to rescale components, e.g., “Feet per gallon” v. “Miles per Gallon”

$$x^{(i)} \mapsto \frac{x^{(i)} - \mu}{\sigma}.$$

We will assume from now on that the data is preprocessed.

# PCA As Optimization



How do you find the closest point to the line?

$$\begin{aligned}\alpha_1 &= \operatorname{argmin}_{\alpha} \|x - \alpha u_1\|^2 \\ &= \operatorname{argmin}_{\alpha} \|x\|^2 + \alpha^2 \|u_1\|^2 - 2\alpha u_1^T x\end{aligned}$$

Then, differentiate wrt  $\alpha$ , set to 0, and use  $\|u_1\|^2$ , which leads to:

$$2\alpha - 2u_1^T x = 0 \implies \alpha = u_1^T x.$$

# Finding PCA

There are two ways you can find PCA:

- ▶ Maximize the projected subspace of the data. (we see more)

$$\max_{u \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n (u \cdot x^{(i)})^2.$$

- ▶ Minimize the residual

$$\min_{u \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n (x^{(i)} - u \cdot x^{(i)})^2.$$

We need to recall some more linear algebra to solve this.



## Back to PCA!

$$\max_{u \in \mathbb{R}^d: \|u\|^2=1} \frac{1}{n} \sum_{i=1}^n (u \cdot x^{(i)})^2$$

We can write:

$$\frac{1}{n} \sum_{i=1}^n (u \cdot x^{(i)})^2 = \frac{1}{n} \sum_{i=1}^n u^T x^{(i)} (x^{(i)})^T u = u^T \left( \underbrace{\frac{1}{n} \sum_{i=1}^n x^{(i)} (x^{(i)})^T}_C \right) u.$$

$C$  is the covariance of the data, since we subtracted the mean.

The first eigenvector of the data's covariance matrix is the principal component

## More PCA

- ▶ **Multiple Dimensions** What if we want multiple dimensions?  
We keep the top- $k$ .

$$\max_{U \in \mathbb{R}^{k \times d}: UU^T = I_k} \frac{1}{n} \sum_{u=1}^n \|Ux^{(i)}\|^2.$$

- ▶ **Reduce dimensionality.** How do we represent data with just those  $k < d$  scalars  $\alpha_j$  for  $j = 1, \dots, k$

$$x = \alpha_1 u_1 + \alpha_2 u_2 + \dots + \alpha_d u_d \text{ keep only } (\alpha_1, \dots, \alpha_k)$$

- ▶ Lurking instability: what if  $\lambda_j = \lambda_{j+1}$ ?
- ▶ **Choose  $k$ ?** One approach is “amount of explained variance”

$$\frac{\sum_{j=1}^k \lambda_j}{\sum_{i=1}^n \lambda_i} \geq 0.9 \text{ note } \text{tr}(C) = \sum_{i=1}^n C_{i,i} = \sum_{i=1}^n \lambda_i$$

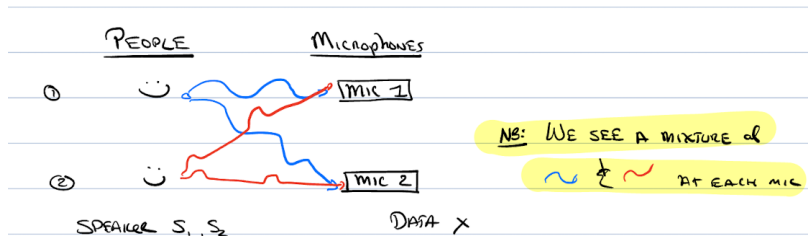
Recall  $\lambda_j \geq 0$  since  $C$  is a covariance matrix.

# Recap of PCA

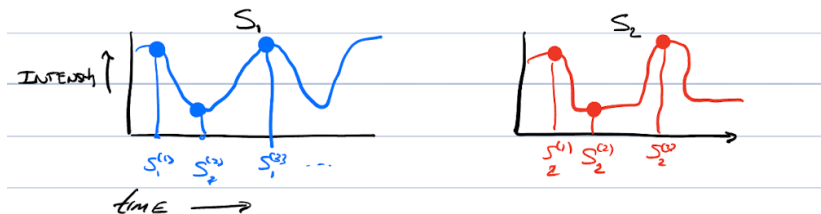
- ▶ Project the data onto a subspace: Find the subspace that captures as much of the data as possible (or doesn't explain the least amount).
- ▶ Dimensionality reduction and visualization
- ▶ Note: The preprocessing (especially centering) featured in our interpretation.

# Independent Component Analysis

# Cocktail Party Problem



# The Data



$S_j^{(t)}$  is the intensity at time  $t$  from speaker  $j$ .

We do **not** observe  $S^{(t)}$  directly, only  $x^{(t)}$  the microphones.

Our model is.

$$x_j^{(t)} = a_{j,1}S_1^{(t)} + a_{j,2}S_2^{(t)}.$$

“Microphone  $j$  at time  $t$  ( $x_j^{(t)}$ ) receives a mixture of speaker 1 at time  $t$  ( $S_1^{(t)}$ ) and speaker 2 at time  $t$  ( $S_2^{(t)}$ ).”

# **DECISION TREES & RANDOM FORESTS**

# ID3 / C4.5 / J48 Algorithm

- Greedy algorithm for decision tree construction developed by [Ross Quinlan](#) 1987-1993
- Top-down construction of tree by recursively selecting ***best attribute*** to use at current node
  - Once attribute selected for current node, generate child nodes, one for each possible attribute value
  - Partition examples using values of attribute, & assign these subsets of examples to the child nodes
  - Repeat for each child node until examples associated with a node are all positive or negative



# Choosing best attribute



- **Key problem:** choose attribute to split given set of examples
- Possibilities for choosing attribute:
  - **Random:** Select one at random
  - **Least-values:** one with smallest # of possible values
  - **Most-values:** one with largest # of possible values
  - **Max-gain:** one with largest expected information gain
  - **Gini impurity:** one with smallest gini impurity value
- The last two measure the **homogeneity** of the target variable within the subsets
- The ID3 and C4.5 algorithms uses **max-gain**

# A Simple Example

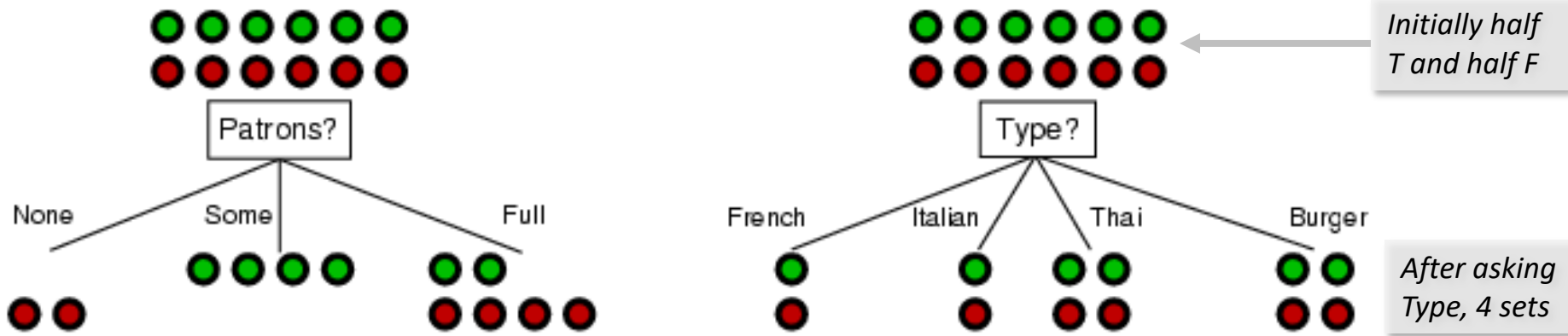
For this data, is it better to start the tree by asking about the restaurant **type** or its current **number of patrons**?

Example	Attributes										Target <i>Wait</i>
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>	
$X_1$	T	F	F	T	Some	\$\$\$	F	T	French	0-10	T
$X_2$	T	F	F	T	Full	\$	F	F	Thai	30-60	F
$X_3$	F	T	F	F	Some	\$	F	F	Burger	0-10	T
$X_4$	T	F	T	T	Full	\$	F	F	Thai	10-30	T
$X_5$	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
$X_6$	F	T	F	T	Some	\$\$	T	T	Italian	0-10	T
$X_7$	F	T	F	F	None	\$	T	F	Burger	0-10	F
$X_8$	F	F	F	T	Some	\$\$	T	T	Thai	0-10	T
$X_9$	F	T	T	F	Full	\$	T	F	Burger	>60	F
$X_{10}$	T	T	T	T	Full	\$\$\$	F	T	Italian	10-30	F
$X_{11}$	F	F	F	F	None	\$	F	F	Thai	0-10	F
$X_{12}$	T	T	T	T	Full	\$	F	F	Burger	30-60	T

# Choosing an attribute



**Idea:** good attribute choice splits examples into subsets that are as close to *all of one type* as possible, e.g., for binary attributes, all T or all F

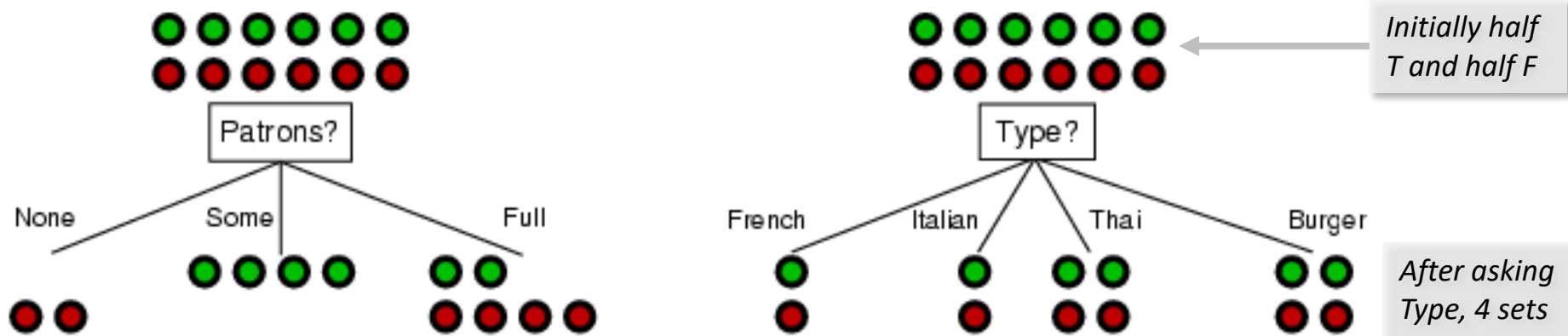


Which is better: asking about *Patrons* or *Type*?

# Choosing an attribute

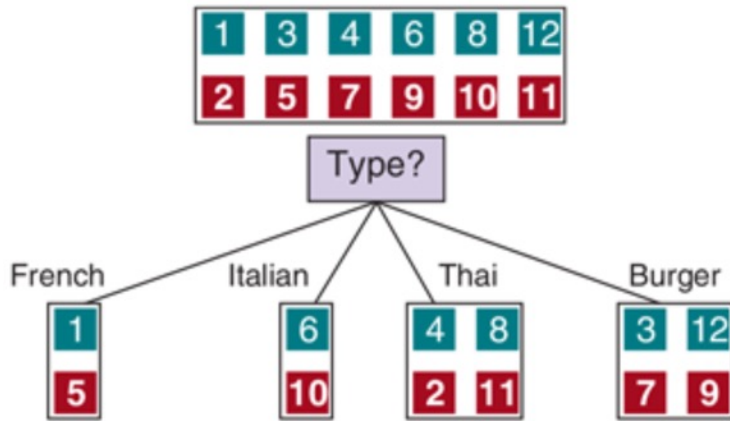


**Idea:** good attribute choice splits examples into subsets that are as close to *all of one type* as possible, e.g., for binary attributes, all T or all F

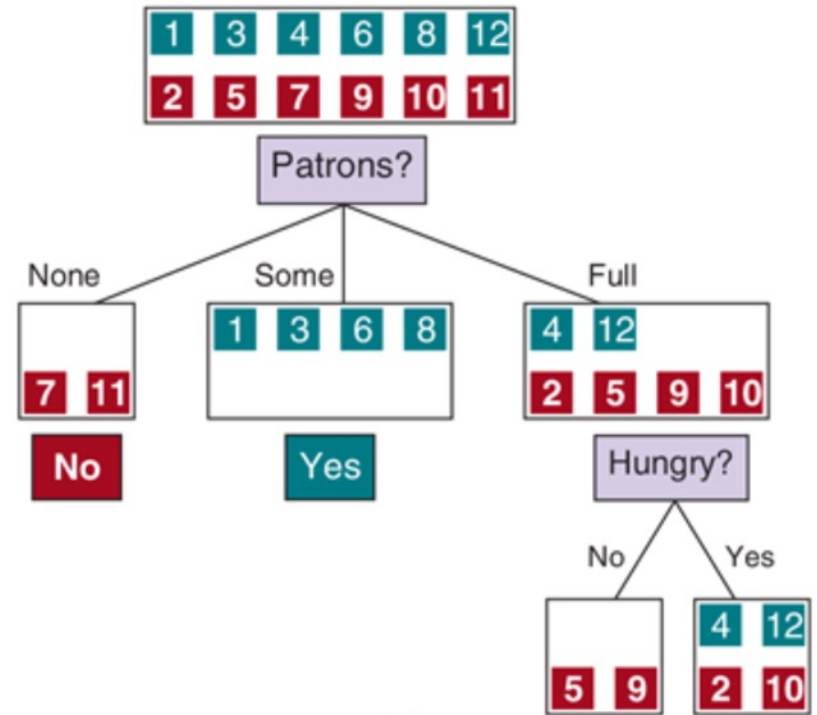


- **Patrons:** for six examples we know the answer and for six we can predict with probability 0.67
- **Type:** our prediction is no better than chance (0.50)

# Choosing Patrons yields more information



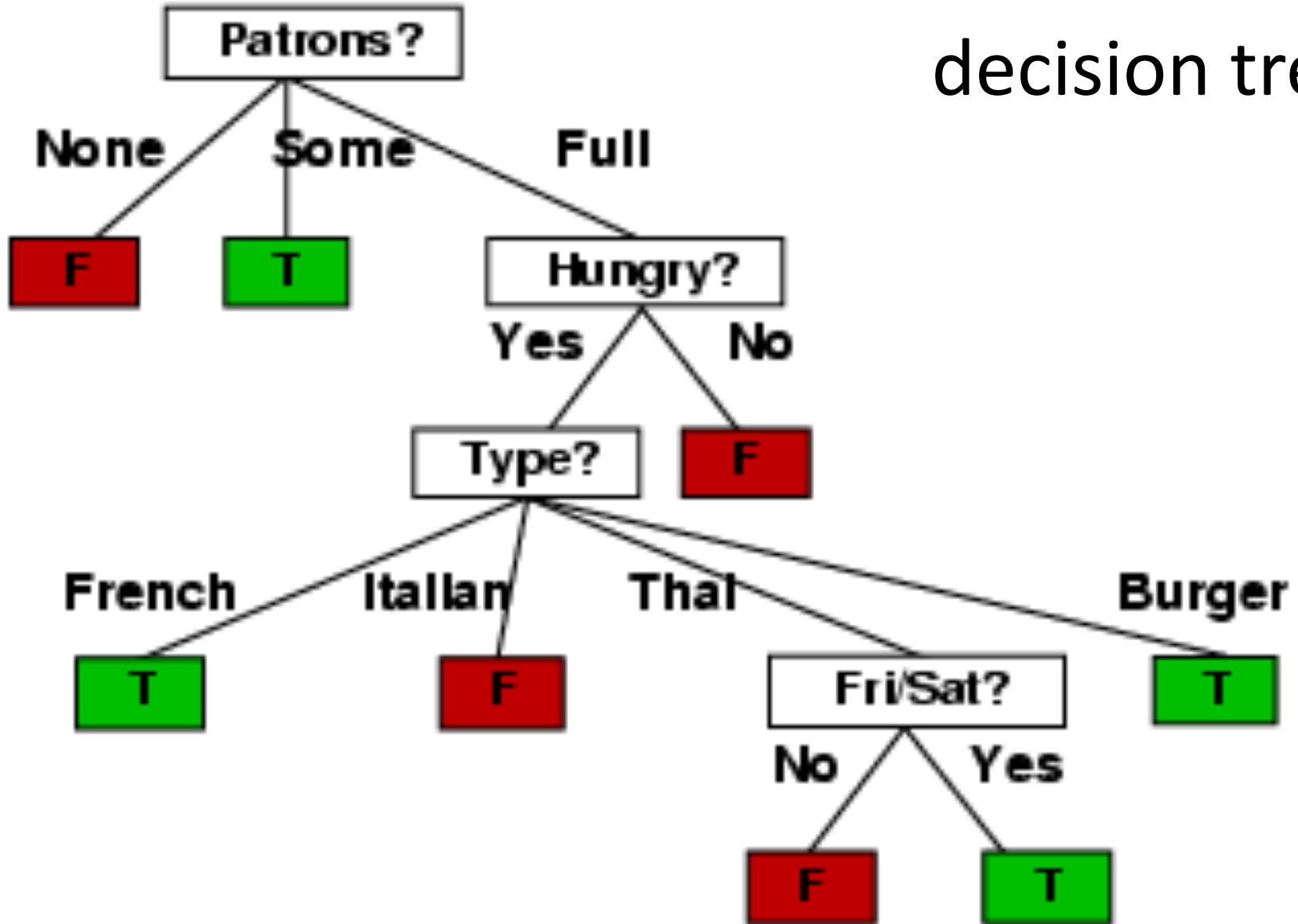
(a)



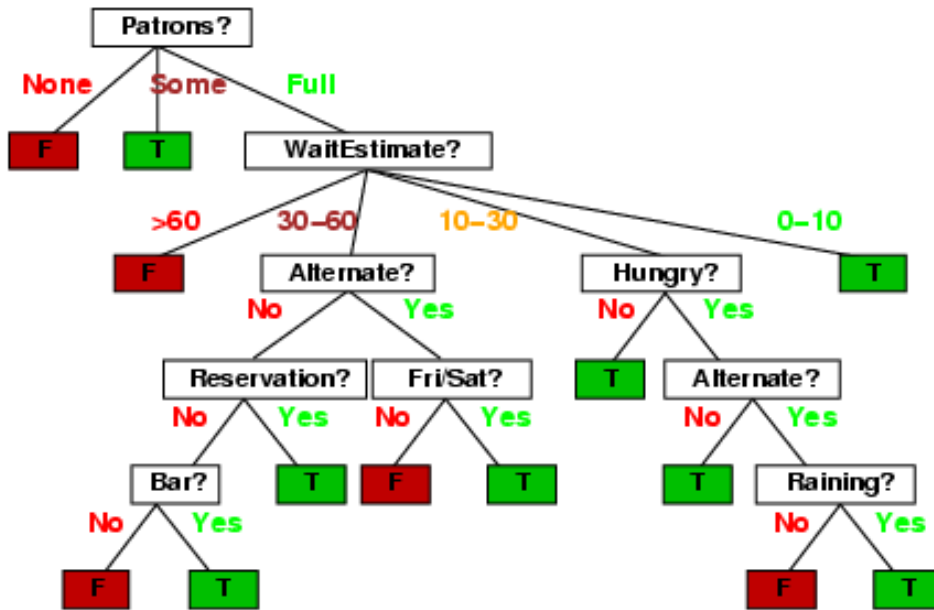
(b)

The ID3 algorithm used this to decide what attribute to ask about next when building a decision tree

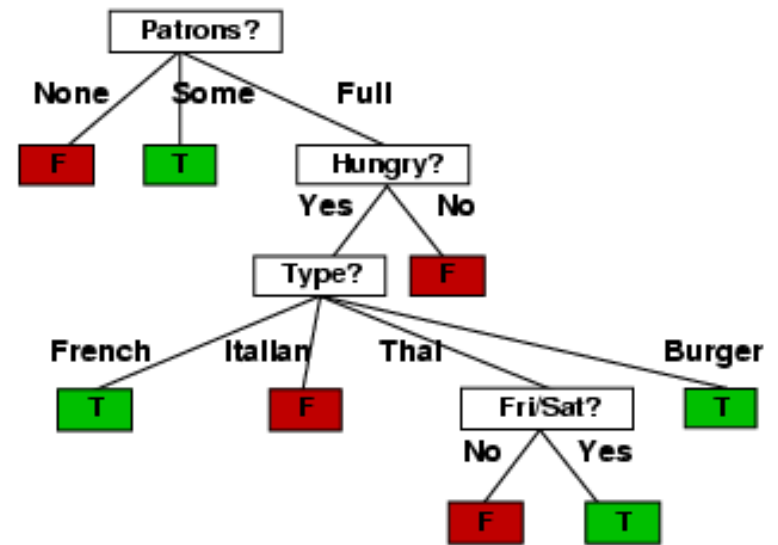
# ID3-induced decision tree



# Compare the two Decision Trees



Human-generated decision tree



ID3-generated decision tree

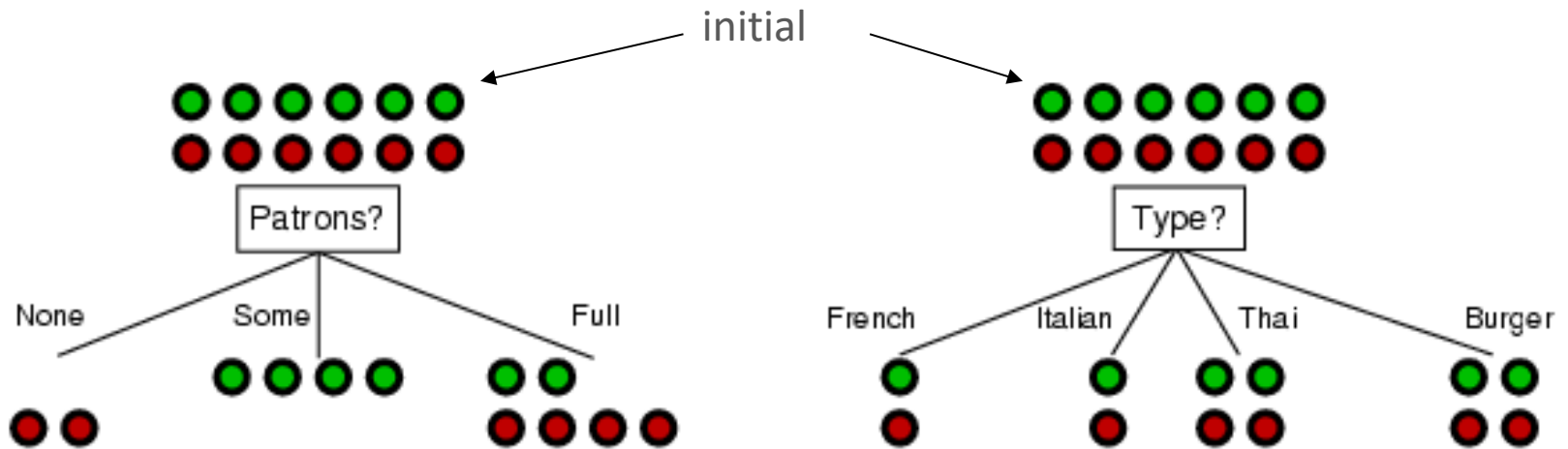
- Intuitively, ID3 tree looks better: it's shallower and has fewer nodes
- ID3 uses **information theory** to decide which question is best to ask next

# Information gain in knowing an attribute

- **Gain(X,T) = Info(T) - Info(X,T)** is difference of
  - **Info(T)**: info needed to identify T's class
  - **Info(X,T)**: info needed to identify T's class after attribute X's value known
- This is gain in information due to knowing value of attribute X
- Used to rank attributes and build DT where each node uses attribute with greatest gain of those not yet considered in path from root
- goal: **create small DTs** to minimize questions



# Information Gain



- Initially half of examples are stay and half **leave**
- After knowing Type?, still half are stay and half **leave**  
We are no wiser for knowing Type 😞
- After knowing Patrons?, we know the class for six and know a likely class for the other six  
We've learned something, but need more info if Patrons=Full 😊

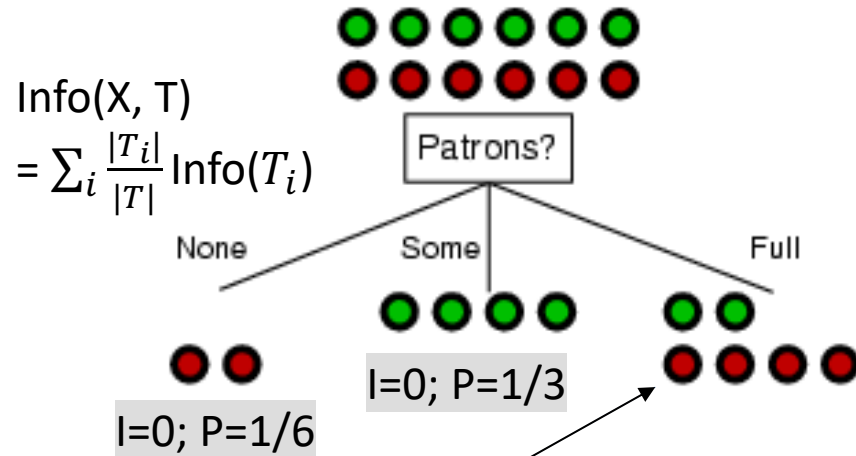
# Information Gain



$$I = \text{Info}(T)$$

$$= - \sum_c \widehat{p}_c \log_2 \widehat{p}_c$$

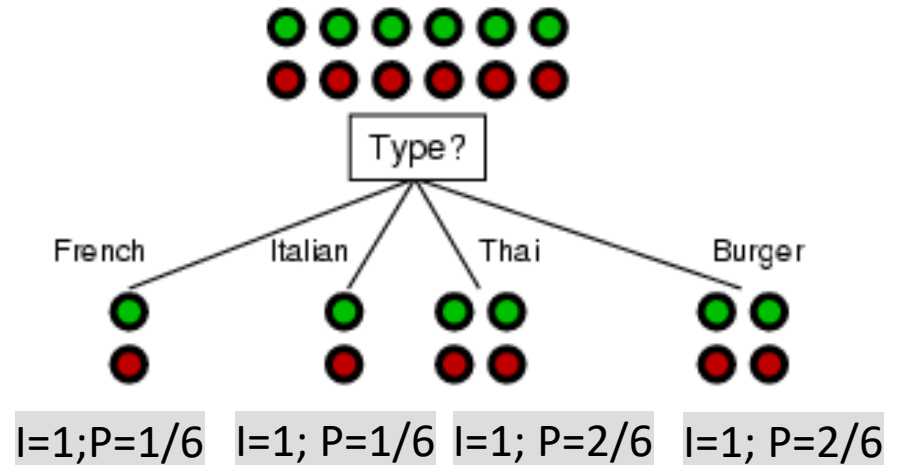
$$I = -(.5 * \log_2(.5) + .5 * \log_2(.5)) = 0.5 + 0.5 \Rightarrow 1.0$$



$$I = -(1/3 * \log_2(1/3) + 2/3 * \log_2(2/3)),$$

$$P = 6/12 = 1/2 \Rightarrow 0.91/2 = 0.46$$

Information gain =  $1 - 0.46 \Rightarrow 0.54$



$$I = 6/6 * 1 \Rightarrow 1.0$$

Information gain =  $1 - 1 \Rightarrow 0.0$

- **Information gain** for asking **Patrons** = **0.54**, for asking **Type** = **0**
- Note: If only one of the N categories has any instances, the information entropy is always 0

# Extensions of ID3

- Using other selection metric gain ratios, e.g., [gini impurity metric](#)
- Handle real-valued data
- Noisy data and overfitting
- Generation of rules
- Setting parameters
- Cross-validation for experimental validation of performance
- **C4.5**: extension of ID3 accounting for unavailable values, continuous attribute value ranges, pruning of decision trees, rule derivation, etc.

# Real-valued data?

- Many ML systems work only on **nominal data**
- We often classify data into one of 4 basic types:
  - **Nominal data** is named, e.g., representing restaurant type as Thai, French, Italian, Burger
  - **Ordinal data** has a well-defined sequence: small, medium, large
  - **Discrete data** is easily represented by integers
  - **Continuous data** is captured by **real** numbers
- There are others, like intervals: age 0-3, 3-5, ...
- Handling some types may need new techniques

# Real-valued => Nominal Data

For ML systems that expect nominal data:

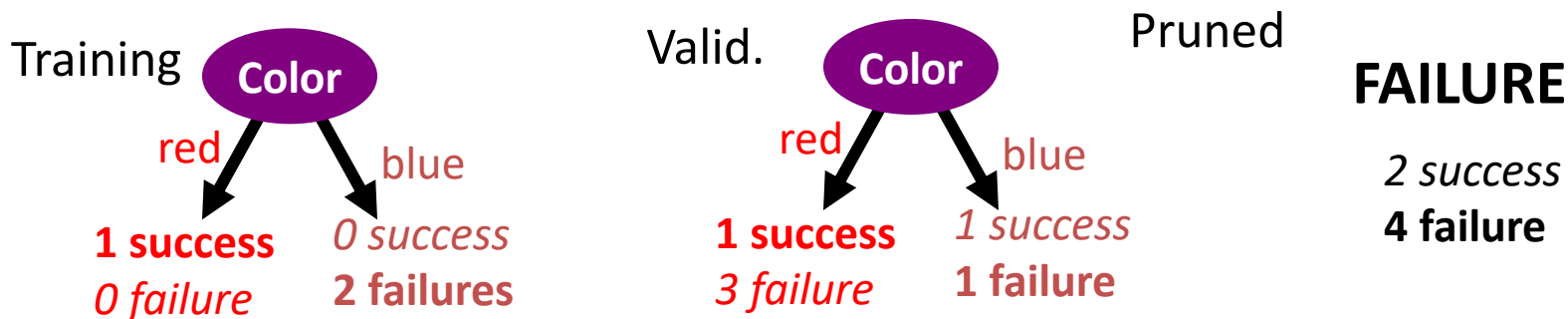
- Select thresholds defining intervals so each becomes a discrete value of attribute
- Use heuristics: e.g., always divide into quartiles
- Use domain knowledge: e.g., divide age into infant (0-2), toddler (2-5), school-aged (5-8)
- Or treat this as another learning problem
  - Try different ways to discretize continuous variable; see which yield better results w.r.t. some metric
  - E.g., try midpoint between every pair of values

# Avoiding Overfitting

- Remove obviously **irrelevant features**
  - E.g., remove ‘year observed’, ‘month observed’, ‘day observed’, ‘observer name’ from the attributes used
- Get **more training data**
- **Pruning** lower nodes in a decision tree
  - E.g., if info. gain of best attribute at a node is below a threshold, stop and make this node a leaf rather than generating children nodes

# Pruning decision trees

- Pruning a decision tree is done by replacing a whole subtree by a leaf node
- Replacement takes place if the expected error rate in the subtree is greater than in the single leaf, e.g.,
  - **Training data:** 1 training red success and 2 training blue failures
  - **Validation data:** 3 red failures and one blue success
  - Consider replacing subtree by a single node indicating failure
- After replacement, only 2 errors instead of 4



# Ensembles

Key Idea: “Wisdom of the crowd”

groups of people can often make better decisions than individuals

Apply this to ML

Learn multiple classifiers and combine their predictions



# Combining Multiple Classifiers by Voting

Train several classifiers and take majority of predictions

For regression use mean or median of the predictions

For ranking and collective classification use some form of averaging

A common family of approaches is called **bagging**

# Bagging: Split the Data

Option 1: Split the data into  $K$  pieces and train a classifier on each

Q: What can go wrong with option 1?

# Bagging: Split the Data

Option 1: Split the data into  $K$  pieces and train a classifier on each

Q: What can go wrong with option 1?

A: Small sample → poor performance

# Bagging: Split the Data

Option 1: Split the data into  $K$  pieces and train a classifier on each

Q: What can go wrong with option 1?

A: Small sample → poor performance

Option 2: Bootstrap aggregation (bagging)  
resampling

# Bagging: Split the Data

Option 1: Split the data into K pieces and train a classifier on each

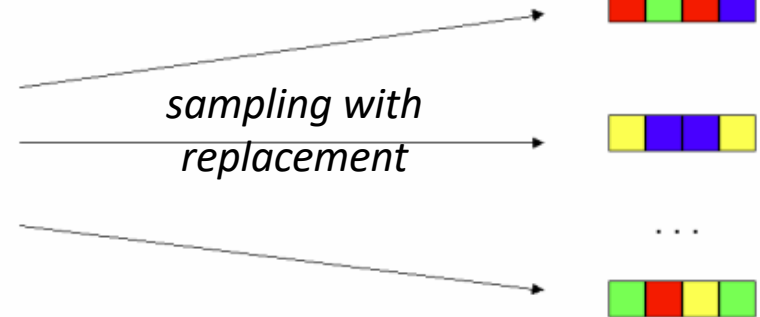
Q: What can go wrong with option 1?

A: Small sample  $\rightarrow$  poor performance

Option 2: Bootstrap aggregation (bagging) resampling

Obtain datasets  $D_1, D_2, \dots, D_N$  using bootstrap resampling from  $D$

Given a dataset  $D...$



get new datasets  $\hat{D}$  by random sampling with replacement from  $D$

# Bagging: Split the Data

Option 1: Split the data into K pieces and train a classifier on each

Q: What can go wrong with option 1?

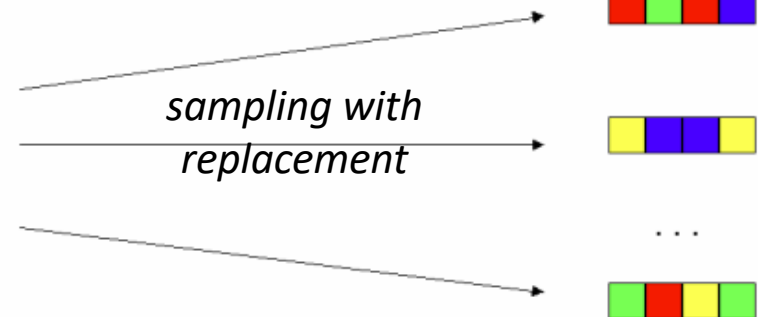
A: Small sample  $\rightarrow$  poor performance

Option 2: Bootstrap aggregation (bagging) resampling

Obtain datasets  $D_1, D_2, \dots, D_N$  using bootstrap resampling from  $D$

Train classifiers on each dataset and average their predictions

Given a dataset  $D...$



get new datasets  $\hat{D}$  by random sampling with replacement from  $D$

# Bagging Decision Trees

How would it work?

# Bagging Decision Trees

How would it work?

Bootstrap sample  $S$  samples  $\{(X_1, Y_1), \dots, (X_S, Y_S)\}$

Train a tree  $t_s$  on  $(X_s, Y_s)$

At test time:  $\hat{y} = \text{avg}(t_1(x), \dots, t_S(x))$



# Random Forests

Bagging trees with one modification

At each split point, choose a **random subset of features** of size **k** and pick the best among these

Train decision trees of depth **d**

Average results from multiple randomly trained trees

Q: What's the difference between bagging decision trees and random forests?

# Summary: decision tree learning

- Still widely used learning methods in practice for problems with relatively **few features**
- Strengths
  - **Fast and easy** to implement
  - **Simple model**: translate to a set of rules
  - **Useful**: empirically valid in many commercial products
  - **Robust**: handles noisy data
  - **Explainable**: easy for people to understand
- Weaknesses
  - Large decision trees may be hard to understand
  - Requires fixed-length feature vectors
  - Non-incremental, adding one new feature requires rebuilding entire tree

# Random Forests

Bagging trees with one modification

At each split point, choose a random subset of features of size **k** and pick the best among these

Train decision trees of depth **d**

Average results from multiple randomly trained trees

Q: What's the difference between bagging decision trees and random forests?

A: Bagging → highly correlated trees (reuse good features)

# CMSC 478: Reinforcement Learning

# Markov Decision Process: Formalizing Reinforcement Learning

Markov Decision  
Process:

$$(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$$

set of possible actions      state-action transition distribution  
set of possible states      reward of (state, action) pairs      discount factor

Start in initial state  $s_0$

for  $t = 1$  to ...:

choose action  $a_t$

“move” to next state  $s_t \sim \pi(\cdot | s_{t-1}, a_t)$

get reward  $r_t = \mathcal{R}(s_t, a_t)$

objective: maximize  
discounted reward

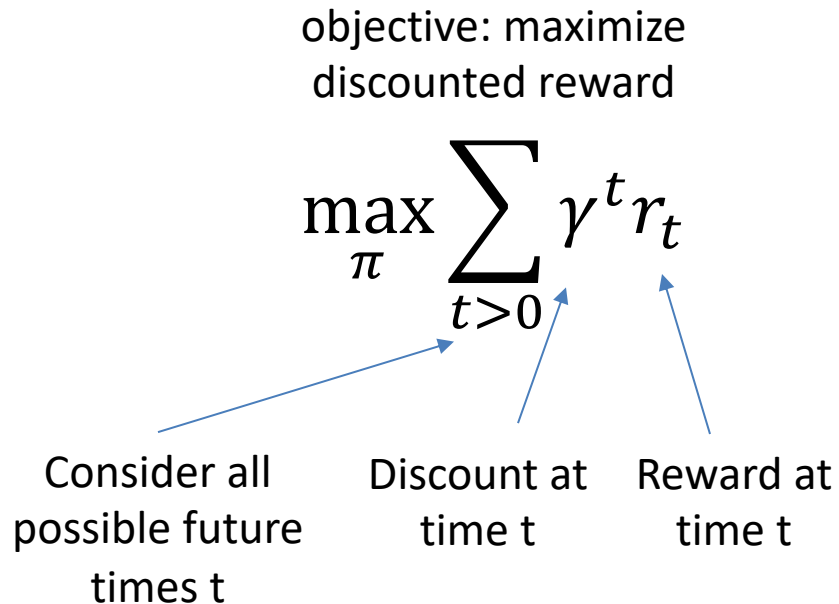
$$\max_{\pi} \sum_{t>0} \gamma^t r_t$$

Consider all  
possible future  
times  $t$

Discount at  
time  $t$

Reward at  
time  $t$

# Example of Discounted Reward



- If the discount factor  $\gamma = 0.8$  then reward  $0.8^0 r_0 + 0.8^1 r_1 + 0.8^2 r_2 + 0.8^3 r_3 + \dots + 0.8^n r_n + \dots$
- Allows you to consider all possible rewards in the future but preferring current vs. future self

# Markov Decision Process: Formalizing Reinforcement Learning

Markov Decision  
Process:

$$(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$$

set of possible actions      state-action transition distribution  
set of possible states      reward of (state, action) pairs      discount factor

Start in initial state  $s_0$

for  $t = 1$  to ...:

choose action  $a_t$

“move” to next state  $s_t \sim \pi(\cdot | s_{t-1}, a_t)$

get reward  $r_t = \mathcal{R}(s_t, a_t)$

objective: maximize  
discounted reward

$$\max_{\pi} \sum_{t>0} \gamma^t r_t$$

“solution”: the policy  $\pi^*$  that maximizes the  
expected (average) time-discounted reward

# Markov Decision Process: Formalizing Reinforcement Learning

Markov Decision  
Process:

$$(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$$

set of possible actions      state-action transition distribution  
set of possible states      reward of (state, action) pairs      discount factor

Start in initial state  $s_0$

for  $t = 1$  to ...:

choose action  $a_t$

“move” to next state  $s_t \sim \pi(\cdot | s_{t-1}, a_t)$

get reward  $r_t = \mathcal{R}(s_t, a_t)$

objective: maximize  
discounted reward



$$\max_{\pi} \sum_{t>0} \gamma^t r_t$$

$$\text{“solution” } \pi^* = \operatorname{argmax}_{\pi} \mathbb{E} \left[ \sum_{t>0} \gamma^t r_t ; \pi \right]$$



# Dynamic programming

use value functions to structure the search for good policies

 policy evaluation: compute  $V^\pi$  from  $\pi$   
policy improvement: improve  $\pi$  based on  $V^\pi$  

start with an arbitrary policy

repeat evaluation/improvement until convergence

# Optimal Policy

3	→	→	→	+1
2	↑		↑	-1
1	↑	←	←	←
	1	2	3	4

- A **policy**  $\Pi$  is a complete strategy that tells the agent what action to take in each state.
- The **optimal policy**  $\Pi^*$  is the policy that yields the highest expected utility for every state in the environment.

This problem is called a Markov Decision Problem (MDP)

How to compute  $\Pi^*$ ?

# Defining Value Function

- Problem:
  - When making a decision, we only know the reward so far, and the possible actions
  - We've defined value function retroactively (i.e., the value function/utility of a history/sequence of states is known *once we finish it*)
  - What is the value function of a particular ***state*** in the middle of decision making?
  - Need to compute ***expected value function*** of possible future histories/states

# Defining Value Function

$$V^\pi(s) = \mathbb{E} [R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots \mid s_0 = s, \pi].$$

$V^\pi(s)$  is simply the expected sum of discounted rewards upon starting in state  $s$ , and taking actions according to  $\pi$ .<sup>1</sup>

Given a fixed policy  $\pi$ , its value function  $V^\pi$  satisfies the **Bellman equations**:

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} P_{s\pi(s)}(s') V^\pi(s').$$

- What is the value function of a particular ***state*** in the middle of decision making?
- Need to compute ***expected value function*** of possible future histories/states

# Value Iteration

---

## Algorithm 4 Value Iteration

---

- 1: For each state  $s$ , initialize  $V(s) := 0$ .
- 2: **for** until convergence **do**
- 3:     For every state, update

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s'). \quad (15.4)$$

---

3			+1	
2			-1	
1				
	1	2	3	4

# Value Iteration

---

## Algorithm 4 Value Iteration

---

- 1: For each state  $s$ , initialize  $V(s) := 0$ .
- 2: **for** until convergence **do**
- 3:     For every state, update

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s'). \quad (15.4)$$


---

3	0.812 →	0.868 →	???	+1
2	0.762 ↑		0.660 ↑	-1
1	0.705 ↑	0.655 ←	0.611 ←	0.388 ←
	1	2	3	4

# Value Iteration

---

## Algorithm 4 Value Iteration

---

- 1: For each state  $s$ , initialize  $V(s) := 0$ .
- 2: **for** until convergence **do**
- 3:     For every state, update

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s'). \quad (15.4)$$


---

3	0.812 →	0.868 →	???	+1
2	0.762 ↑		0.660 ↑	-1
1	0.705 ↑	0.655 ←	0.611 ←	0.388 ←
	1	2	3	4

EXERCISE: What is  $V^*([3,3])$  (assuming that the other  $V^*$  are as shown)?

# Value Iteration

---

## Algorithm 4 Value Iteration

---

- 1: For each state  $s$ , initialize  $V(s) := 0$ .
- 2: **for** until convergence **do**
- 3:     For every state, update

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s'). \quad (15.4)$$


---

3	0.812 →	0.868 →	???	+1
2	0.762 ↑		0.660 ↑	-1
1	0.705 ↑	0.655 ←	0.611 ←	0.388 ←
	1	2	3	4

EXERCISE: What is  $V^*([3,3])$  (assuming that the other  $V^*$  are as shown)?



# Value Iteration

---

## Algorithm 4 Value Iteration

---

- 1: For each state  $s$ , initialize  $V(s) := 0$ .
- 2: **for** until convergence **do**
- 3:     For every state, update

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s'). \quad (15.4)$$


---

3	<b>0.812</b> →	<b>0.868</b> →	→	+1
2	<b>0.762</b> ↑		<b>0.660</b> ↑	-1
1	<b>0.705</b> ↑	<b>0.655</b> ←	<b>0.611</b> ←	<b>0.388</b> ←
	1	2	3	4

From (3, 3), 3 options: (3, 2), (4, 3), (3, 4) => but there is no (3,4) but wall, so bounced off and remains at (3, 3)

# Value Iteration

---

## Algorithm 4 Value Iteration

---

- 1: For each state  $s$ , initialize  $V(s) := 0$ .
- 2: **for** until convergence **do**
- 3:     For every state, update

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s'). \quad (15.4)$$


---

3	0.812 →	0.868 →	→	+1
2	0.762 ↑		0.660 ↑	-1
1	0.705 ↑	0.655 ←	0.611 ←	0.388 ←
	1	2	3	4

$$V^*_{3,3} = R_{3,3} + [P_{3,2} V^*_{3,2} + P_{3,3} V^*_{3,3} + P_{4,3} V^*_{4,3}]$$

From (3, 3), 3 options: (3, 2), (4, 3), (3, 4) => but there is no (3,4) but wall, so bounced off and remains at (3, 3)

# Value Iteration

---

## Algorithm 4 Value Iteration

---

- 1: For each state  $s$ , initialize  $V(s) := 0$ .
- 2: **for** until convergence **do**
- 3:     For every state, update

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s'). \quad (15.4)$$


---

3	0.812 →	0.868 →	→	+1
2	0.762 ↑		0.660 ↑	-1
1	0.705 ↑	0.655 ←	0.611 ←	0.388 ←
	1	2	3	4

$$\begin{aligned}
 V^*_{3,3} &= R_{3,3} + \\
 & [P_{3,2} V^*_{3,2} + P_{3,3} V^*_{3,3} + P_{4,3} V^*_{4,3}] \\
 & = -0.04 + \\
 & [0.1 * 0.660 + 0.1 * 0.918 + 0.8 * 1]
 \end{aligned}$$

From (3, 3), 3 options: (3, 2), (4, 3),  
 (3, 4) => but there is no (3,4) but wall, so  
 bounced off and remains at (3, 3)

# Value Iteration

---

## Algorithm 4 Value Iteration

---

- 1: For each state  $s$ , initialize  $V(s) := 0$ .
- 2: **for** until convergence **do**
- 3:     For every state, update

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s'). \quad (15.4)$$


---

3	0.812 →	0.868 →	.918 →	+1
2	0.762 ↑		0.660 ↑	-1
1	0.705 ↑	0.655 ←	0.611 ←	0.388 ←
	1	2	3	4

$$\begin{aligned}
 V^*_{3,3} &= R_{3,3} + \\
 & [P_{3,2} V^*_{3,2} + P_{3,3} V^*_{3,3} + P_{4,3} V^*_{4,3}] \\
 & = -0.04 + \\
 & [0.1 * 0.660 + 0.1 * 0.918 + 0.8 * 1]
 \end{aligned}$$

From (3, 3), 3 options: (3, 2), (4, 3), (3, 4) => but there is no (3,4) but wall, so bounced off and remains at (3, 3)

# Value Iteration

---

## Algorithm 4 Value Iteration

---

- 1: For each state  $s$ , initialize  $V(s) := 0$ .
- 2: **for** until convergence **do**
- 3:     For every state, update

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s'). \quad (15.4)$$


---

3	0.812 →	0.868 →	.918 →	+1
2	0.762 ↑		0.660 ↑	-1
1	0.705 ↑	0.655 ←	0.611 ←	0.388 ←
	1	2	3	4

$$\begin{aligned}
 V^*_{3,3} &= R_{3,3} + \\
 & [P_{3,2} V^*_{3,2} + P_{3,3} V^*_{3,3} + P_{4,3} V^*_{4,3}] \\
 & = -0.04 + \\
 & [0.1 * 0.660 + 0.1 * 0.918 + 0.8 * 1]
 \end{aligned}$$

From (3, 3), 3 options: (3, 2), (4, 3), (3, 4) => but there is no (3,4) but wall, so bounced off and remains at (3, 3)



# Value Iteration

In (3, 3), since  $\rightarrow$  action gave us the **maximum expected future reward**, we choose to keep  $\rightarrow$  in our policy. Same thing was done for all states.

3	<b>0.812</b> $\longrightarrow$	<b>0.868</b> $\longrightarrow$	<del>           0.881            0.812            0.675         </del> <b>0.918</b>	<b>+1</b>
2	<b>0.762</b> $\uparrow$		<b>0.660</b> $\uparrow$	<b>-1</b>
1	<b>0.705</b> $\uparrow$	<b>0.655</b> $\longleftarrow$	<b>0.611</b> $\longleftarrow$	<b>0.388</b> $\longleftarrow$
	1	2	3	4

# Optimal Policy

$$\pi^*(s) = \arg \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s').$$

3	<b>0.812</b> →	<b>0.868</b> →	→	+1
2	<b>0.762</b> ↑		<b>0.660</b> ↑	-1
1	<b>0.705</b> ↑	<b>0.655</b> ←	<b>0.611</b> ←	<b>0.388</b> ←
	1	2	3	4

Whichever is higher becomes next action for (3, 1)

# Optimal Policy

$$\pi^*(s) = \arg \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s').$$

$$\begin{aligned} \pi^*_{3,1} \text{ being } (\leftarrow) = & \\ & P_{\text{up}} V^*_{1,2} + P_{\text{left}} V^*_{3,3} \text{ (Bounced off)} + P_{\text{right}} V^*_{3,2} \\ & = 0.8 * 0.655 + 0.1 * 0.611 + 0.1 * 0.66 \end{aligned}$$

3	0.812 →	0.868 →	→	+1
2	0.762 ↑		0.660 ↑	-1
1	0.705 ↑	0.655 ←	0.611 ←	0.388 ←
	1	2	3	4

Whichever is higher becomes next action for (3, 1)



# Optimal Policy

$$\pi^*(s) = \arg \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s').$$

$$\begin{aligned} \pi^*_{3,1} \text{ being } (\leftarrow) = & \\ P_{\text{up}} V^*_{1,2} + P_{\text{left}} V^*_{3,3} \text{ (Bounced off)} + P_{\text{right}} V^*_{3,2} & \\ = 0.8 * 0.655 + 0.1 * 0.611 + 0.1 * 0.66 & \end{aligned}$$

3	0.812 →	0.868 →	.918 →	+1
2	0.762 ↑		0.660 ↑	-1
1	0.705 ↑	0.655 ←	0.611 ←	0.388 ←
	1	2	3	4

Whichever is higher becomes next action for (3, 1)

# Optimal Policy

$$\pi^*(s) = \arg \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s').$$

3	0.812 →	0.868 →	.918 →	+1
2	0.762 ↑		0.660 ↑	-1
1	0.705 ↑	0.655 ←	0.611 ←	0.388 ←
	1	2	3	4

$$\begin{aligned} \pi^*_{3,1} \text{ being } (\leftarrow) = & \\ P_{\text{up}} V^*_{1,2} + P_{\text{left}} V^*_{3,3} \text{ (Bounced off)} + P_{\text{right}} V^*_{3,2} & \\ = 0.8 * 0.655 + 0.1 * 0.611 + 0.1 * 0.66 & \end{aligned}$$

$$\begin{aligned} \pi^*_{3,1} \text{ being } (\uparrow) = & \\ P_{\text{up}} V^*_{3,2} + P_{\text{left}} V^*_{2,1} + P_{\text{right}} V^*_{1,4} & \end{aligned}$$

Whichever is higher becomes next action for (3, 1)

# Policy Iteration

- Pick a policy  $\Pi$  at random
- Repeat:
  - Compute Value function of each state for  $\Pi$

$$V(s) := V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s').$$

- Compute the policy  $\Pi'$  given these value functions

$$\pi'(s) := \arg \max_{a \in A} \sum_{s'} P_{sa}(s') V(s').$$

- If  $\Pi' = \Pi$  then return  $\Pi$

# Policy Iteration

- Pick a policy  $\Pi$  at random
- Repeat:
  - Compute Value function of each state for  $\Pi$

$$V(s) := V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s').$$

- Compute the policy  $\Pi'$  given these value functions

Or solve the set of linear equations:  
(often a sparse system)

$$\pi'(s) := \arg \max_{a \in A} \sum_{s'} P_{sa}(s') V(s').$$

- If  $\Pi' = \Pi$  then return  $\Pi$

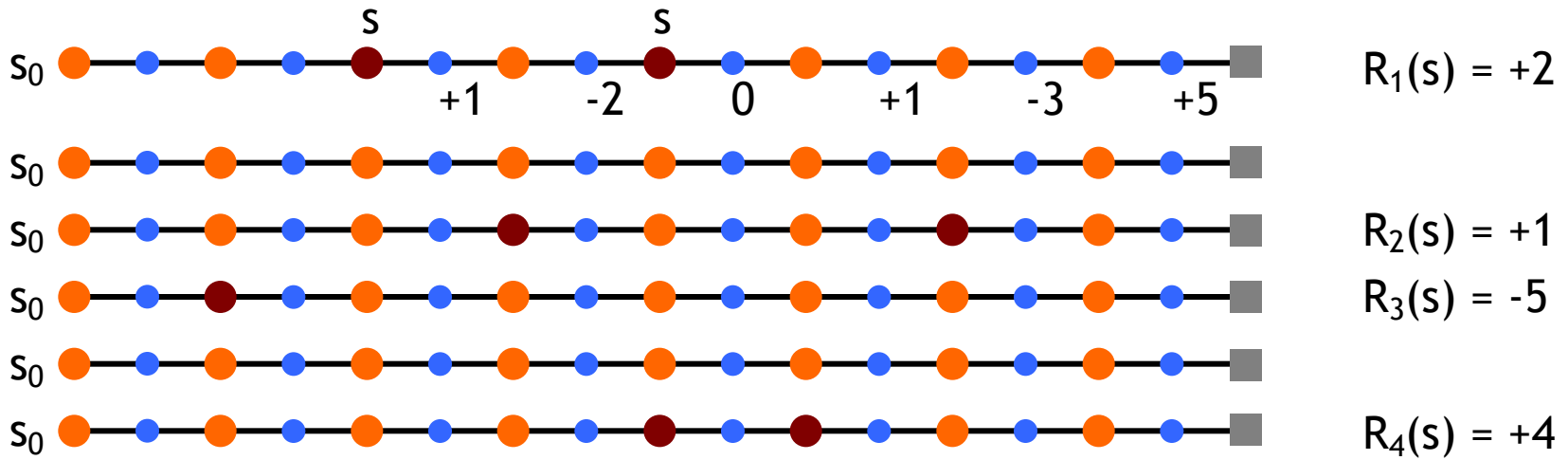
# Monte Carlo policy evaluation

don't need full  
knowledge of  
environment (just  
**(simulated)** experience)

want to estimate  $V^\pi(s)$

expected return starting from  $s$   
and following  $\pi$

estimate as average of  
observed returns in state  $s$



$$V^\pi(s) \approx (2 + 1 - 5 + 4) / 4 = 0.5$$

# Exploration vs. Exploitation

- Problem with naïve reinforcement learning:
  - What action to take?
  - **Best apparent action, based on learning to date** } Exploitation
    - Greedy strategy
    - Often prematurely converges to a suboptimal policy!
  - **Random (or unknown) action** } Exploration
    - Will cover entire state space
    - Very expensive and slow to learn!
    - When to stop being random?
  - Balance exploration (try random actions) with exploitation (use best action so far)

# More on Exploration

- Agent may sometimes choose to explore suboptimal moves in hopes of finding better outcomes
  - Only by visiting all states frequently enough can we guarantee learning the true values of all the states
- When the agent is **learning**, ideal would be to get accurate values for all states
  - Even though that may mean getting a negative outcome
- When agent is **performing**, ideal would be to get optimal outcome
- A learning agent should have an **exploration policy**

# Exploration Policy

- Wacky approach (exploration): act randomly in hopes of eventually exploring entire environment
  - Choose any legal checkers move
- Greedy approach (exploitation): act to maximize utility using current estimate
  - Choose moves that have in the past led to wins
- Reasonable balance: act more wacky (exploratory) when agent has little idea of environment; more greedy when the model is close to correct
  - Suppose you know no checkers strategy?
  - What's the best way to get better?



# Maintaining exploration

key ingredient of RL

deterministic/greedy policy won't explore all actions

don't know anything about the environment at the beginning  
need to try all actions to find the optimal one

maintain exploration

use *soft* policies instead:  $\pi(s,a) > 0$  (for all  $s,a$ )

$\epsilon$ -greedy policy

with probability  $1-\epsilon$  perform the optimal/greedy action  
with probability  $\epsilon$  perform a random action

will keep exploring the environment

slowly move it towards greedy policy:  $\epsilon \rightarrow 0$