

Data Mining: Clustering and Prediction

Diego Rivera Correa

Slides owned by Mirek Riedewald



This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

Key Learning Goals

- What is the difference between supervised and unsupervised learning?
- Give an example for a supervised learning algorithm.
- Give an example for an unsupervised learning algorithm.

Key Learning Goals

- Write the pseudo-code for K-means clustering in MapReduce.
- Will the Spark implementation of K-means be significantly faster than the MapReduce implementation? Justify your answer.
- For decision-tree training, explain the parallel counting algorithm for a given set of possible split point candidates using an example.

Introduction

- Data mining can be concisely characterized as “statistics plus computers.”
- Its goals are similar to those of statistical analysis, but the availability of massive computing power enabled novel types of automatic algorithms.

Introduction

- Han et al. point out that the goal of data mining is the extraction of **interesting** (non-trivial, implicit, previously unknown and potentially useful) **patterns** or **knowledge** from a huge amount of data.

Introduction

- Intuitive example: A grocery store has millions of receipts. A human cannot manually inspect them all, but a data mining algorithm can discover that customers who buy tortilla chips often also buy salsa.
- That pattern can support store layout, coupons, or recommendations.

Example Applications

- **Classification/prediction:** Used when there is a target outcome.
- **Intuitive examples:**
 - Is this credit-card transaction fraud?
 - Is this email spam?
 - What is the probability of seeing a bird species under certain weather and habitat conditions?
 - What income will an ad generate for a search query?

Example Applications

- **Clustering:** Used when we want natural groups.
 - **Example:** Group online game players by behavior: casual players, competitive players, social players, and high-spending players.
- **Graph mining:** Used when relationships matter.
 - **Example:** In a social network, predict which people are likely to collaborate or become friends.
- **Association rules:** Used to find items or events that frequently occur together.
 - **Example:** People who buy diapers may also buy baby wipes.

Typical Data Mining Steps

- The three essential phases of the data mining lifecycle include:
 - **Data preparation**
 - **Mining the data**
 - **Post-processing**

Typical Data Mining Steps:

1. Data Preparation

- In practice, the preparation phase can often take much longer and require more resources than the actual mining phase.
- There are two major preparation steps: (1) understanding data and domain, and (2) cleaning and pre-processing the data.

Typical Data Mining Steps:

1. Data Preparation

- Understanding the domain: You need to know what the attributes mean.
 - This generally requires collaboration with a domain expert.
- **Example:** Suppose a bird-sighting dataset has attributes like latitude, longitude, month, temperature, habitat type, and observer effort.
- You must know what each means before modeling. Otherwise, you may misinterpret the result.

Typical Data Mining Steps:

1. Data Preparation

- Understanding attribute types: An attribute is a property or feature of an object.
- **Example:** For a person, attributes might include age, income, hair color, education level, and ZIP code.

Typical Data Mining Steps:

1. Data Preparation

- **Numeric:** Have meaningful arithmetic.
 - Examples: age, income, temperature
- **Categorical/nominal:** Labels with no natural order.
 - Examples: hair color, country, species
- **Ordinal:** Ordered, but differences are not necessarily equal.
 - Examples: star rating, military rank
- **Binary:** Special case of categorical with two values.

Typical Data Mining Steps:

1. Data Preparation

- Further insights about relationships between different attributes can be obtained from data summaries such as scatterplots and correlation coefficients.
- Knowledge about statistical interactions between attributes (called variables in statistics) helps the data miner decide which attributes should be explored jointly, and which can be studied separately.

Typical Data Mining Steps:

1. Data Preparation

- **Correlation example:** If temperature and ice-cream sales rise together, they have positive correlation.
 - But correlation does not prove causation.
- **Important intuition:** Some attributes should be studied together because they interact.
- **Example:** Weather may affect bird sightings differently depending on the season. Temperature alone may not explain much; temperature plus month may explain more.

Typical Data Mining Steps:

1. Data Preparation

- Once the data and problem are sufficiently understood, usually the data needs to be **cleaned and pre-processed** before data mining can commence.
- Data cleaning often addresses noise and missing values. A common data-cleaning challenge is to fix the encoding of missing values. Sometimes there exists an explicit NULL value, in other cases it might be encoded as -99 (e.g., for age)

Typical Data Mining Steps:

1. Data Preparation

- Data often needs to be integrated from multiple sources. In addition to joining the data, this may require entity resolution.
 - For example, an author might appear as Amy Smith, A. Smith, or Amy B. Smith in different contexts.
- Depending on problem and data mining method, it might be necessary or beneficial to apply data reduction and transformation such as normalization, Fourier transform, or attribute removal.

Typical Data Mining Steps:

2. Mining the Data

- After proper data preparation, data mining techniques extract the desired information and patterns.
- For classification and prediction problems, first a **model** is trained on a subset of the given *labeled* data. Model quality is evaluated on a separate test set. Then the model is used on new inputs to predict the desired output.

Typical Data Mining Steps:

2. Mining the Data

- Clustering algorithms such as K-means, and hierarchical clustering are used to identify groups of similar entities that are different from other groups.
- Frequent-pattern mining is concerned with identifying frequent itemsets, association rules, and frequent sequences.

Typical Data Mining Steps:

3. Post-Processing

- A data mining model on its own usually is not intelligible. Hence additional tools are used to determine and present what was learned from the data. For classification and prediction, what are some of the possible post-processing tasks?

Typical Data Mining Steps:

3. Post-Processing

- To evaluate the quality of a prediction model, its **accuracy** or error rate on data representing future input needs to be determined.
- A classification or prediction model usually is a complex “blackbox” function that returns some output for a given input.

Typical Data Mining Steps:

3. Post-Processing

- Data analysts want to understand the big picture of what the model has learnt. This usually involves identifying
 - (1) the most important variables and their effect on the output.
 - (2) patterns indicating variable interactions.
 - (3) compact rules explaining the predictions.

Typical Data Mining Steps:

3. Post-Processing

- **Example:** mortgage repayment prediction model. The model takes inputs about a bank customer with attributes **credit score, annual income, debt-to-income ratio, employment history, previous missed payments**.
- The model predicts: “How likely is this customer to repay the mortgage?”

Typical Data Mining Steps:

3. Post-Processing

- (1) the most important variables and their effect on the output.
- The model may learn that **credit score**, **debt-to-income ratio**, and **employment history** are the most important variables.
 - A higher credit score may increase repayment probability.
 - A higher debt-to-income ratio may decrease repayment probability.
 - Longer stable employment may increase repayment probability.

Typical Data Mining Steps:

3. Post-Processing

- (2) patterns indicating variable interactions.
- Income alone may not always predict repayment. A customer with high income but also very high debt may still be risky. So the model may learn an interaction: **income helps only when debt-to-income ratio is not too high.**

Typical Data Mining Steps:

3. Post-Processing

- (3) compact rules explaining the predictions.
- **Example rule:** If credit score > 740 and debt-to-income ratio $< 30\%$, predict high repayment probability.
- **Another rule:** If credit score < 600 and debt-to-income ratio $> 45\%$, predict low repayment probability.

Typical Data Mining Steps:

3. Post-Processing

- Suppose our given customer has:
 - Credit score = 760
 - Annual income = 95k
 - Debt-to-income ratio = 22%
 - Employment history = 8 years
 - Missed payments = 0
- What should the model predict?

Typical Data Mining Steps:

3. Post-Processing

- Suppose our given customer has:
 - Credit score = 760
 - Annual income = 95k
 - Debt-to-income ratio = 22%
 - Employment history = 8 years
 - Missed payments = 0
- What should the model predict? **High probability of repayment.**

Typical Data Mining Steps:

3. Post-Processing

- For clustering, analysts verify if the grouping makes sense based on their domain expertise. Usually this also involves finding compact labels or descriptions to express what the entities in the same cluster have in common.
- Visualization of results and patterns found is a powerful tool for humans to gain insights from data mining.

Parallel Data Mining

- Many mature and feature-rich data mining libraries and products are available. Unfortunately, most data mining solutions are not designed for execution in large distributed systems. This often leaves only the following 3 options:
 1. Use an existing, but often limited, library of distributed data mining solutions, e.g., Apache Mahout or Spark ML.

Parallel Data Mining

2. Design your own distributed version of a data mining algorithm and implemented it in MapReduce or Spark. This tends to be non-trivial, as many data mining algorithms are sophisticated and rely heavily on in-memory computation for performance.

3. Leverage existing mature libraries that were written for in-memory execution on a single machine by using them in a larger ensemble that can be trained and managed in a distributed system.

Let us look at two popular types of data mining and machine learning: unsupervised learning and supervised learning.

We then start with clustering, which is an unsupervised learning approach.

Supervised

- **Supervised learning** uses labeled examples. The algorithm learns from inputs where the correct answer is already known.
- **Example:** You train a spam detector using emails already labeled as “spam” or “not spam.” Later, the model predicts whether a new email is spam.

Supervised

- Common supervised tasks include classification, where we predict a category: "Is this transaction fraudulent or legitimate?"
- We can also predict a numeric or probability value: "How likely is a customer to repay a mortgage?"

Unsupervised

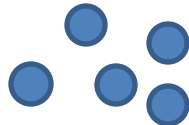
- **Unsupervised learning** uses data without labels. The algorithm tries to find structure by itself.
- **Example:** You give customer purchase histories to an algorithm without telling it the customer types. It may discover groups such as “budget shoppers,” “organic-product buyers,” and “bulk buyers.”
- **Clustering** is the main unsupervised method introduced in these slides.

Clustering

- Clustering is one of the most popular data mining approaches in practice, because it automatically detects “natural” groups or communities in big data.
- These clusters could be the end-result. Or they could be used to improve other data mining steps by customizing those steps depending on the cluster membership of an object of interest.

Clustering

- In general, a cluster is a collection of data objects. The goal of clustering typically is to identify clusters such that objects are **similar** to one another within the same cluster, but dissimilar to the objects in other clusters.
- Clustering does not require any training data with known cluster membership. Hence it is considered an **unsupervised** learning method.



Three clusters based on the Euclidean distances between objects.

Example for Clustering Applications

- Suppose customers are represented by two attributes: yearly spending and number of purchases.

Customer	Yearly Spending	Purchases per Year
A	\$120	3
B	\$150	4
C	\$1,200	25
D	\$1,350	28
E	\$5,000	80
F	\$5,300	85

Example for Clustering Applications

Cluster	Customers	Interpretation
Cluster 1	A, B	Low-spending occasional customers
Cluster 2	C, D	Frequent moderate spenders
Cluster 3	E, F	High-value loyal customers

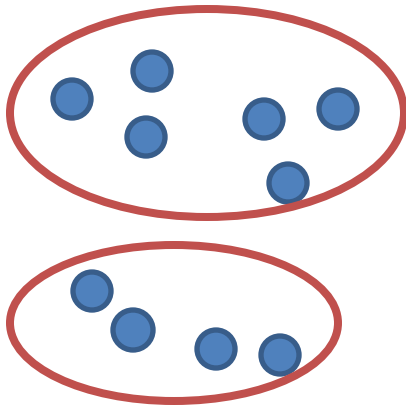
- No one labels these groups beforehand. The algorithm discovers them.

Example for Clustering Applications

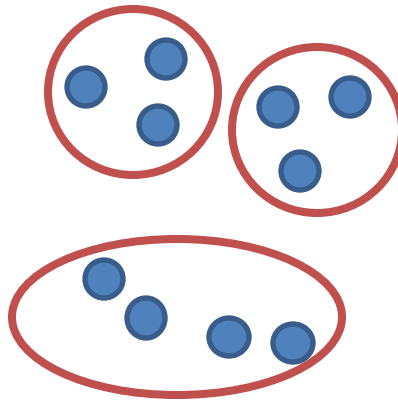
- Clustering is useful because it helps us summarize large datasets.
- Instead of analyzing thousands or millions of individual records, we can analyze a smaller number of groups.

Ambiguity of Clusters

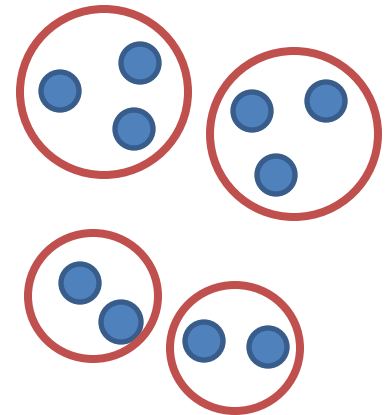
- The notion of what constitutes a cluster is subjective and depends on the preferences of the user.



2 clusters?



3 clusters?



4 clusters?

Distance and Similarity

- Clustering is inherently determined by the choice of **distance** or **similarity** measure.
- Objects in the same cluster should have small distance from each other. Objects in different clusters should have larger distance.

Distance and Similarity

Customer	Yearly Spending	Purchases per Year
A	\$100	2
B	\$120	3
C	\$4,000	70

- Customer A and B are similar. Their spending and purchase counts are close.
- Customer C is very different. C spends much more and buys much more often.

Distance and Similarity

- The choice of distance or similarity measure depends on user preferences.
 - Document similarity is usually measured based on the terms they contain.
 - The Minkowski distance is a popular family of distance measures.

Minkowski Distance

- The Minkowski distance is a general family of distance measures for numeric data.
- For clustering, each object is treated like a point with multiple attributes. It calculates the similarity between points by aggregating the absolute differences across all dimensions.
- The two common Minkowski distances are:
 - Manhattan distance
 - Euclidean distance

Minkowski Distance

- Consider a data set $\{x(1), x(2), \dots, x(n)\}$ of n d -dimensional data points, i.e., each $x(i)$ is a d -dimensional vector $(x_1(i), x_2(i), \dots, x_d(i))$.
- For $q > 0$, the **Minkowski distance** between any two data points $x(i)$ and $x(j)$ is defined as
$$\left(\sum_{k=1}^d |x_k(i) - x_k(j)|^q\right)^{1/q}$$
- For $q = 1$, it is called the Manhattan distance.
- For $q = 2$, it is called the Euclidean distance.



Intuition for the Manhattan distance: It is the distance of walking from $x(i)$ to $x(j)$ when walking on a rectangular grid of roads—like in Manhattan, New York.

Challenges of Distance Computation

- Since the distance or similarity measure strongly affects the clustering, it should be chosen carefully. Several challenges often must be addressed:
 - Curse of dimensionality
 - Diverse attribute domains
 - Categorical and ordinal attributes
- We discuss each challenge next.

Curse of Dimensionality

- The **curse of dimensionality** happens when we use many attributes, especially if many of them are noisy or irrelevant.
- Distance is computed by combining differences across all attributes. If we include too many weak or noisy attributes, the important signal can get buried.
 - To address this problem, remove any attribute that is known to be very noisy or not interesting.

Curse of Dimensionality

- Depending on the attributes considered for the distance computation, the clustering might change significantly.
- If it is not known which attributes are of most interest, try different subsets and determine experimentally for which of them good clusters are found.

Diverse Attribute Domains

- An attribute with a large domain often dominates the overall distance.
 - Consider Amy, Bill, and Carla with (age, income) combinations (25, \$50K), (58, \$51K), and (27, \$52K).
 - Amy and Carla have very similar age and income, while Bill has similar income, but belongs to a different generation.
 - However, the absolute age difference is in the 30s, while the income difference is in the 1000s, dominating the overall distance.

Diverse Attribute Domains

- Because income is measured in dollars, the income difference dominates the distance.
- The algorithm may think Amy is closer to Bill than to Carla because Bill's income is closer, even though Amy and Carla are much closer in age.
- The algorithm may give too much importance to attributes with large numeric ranges.
 - This does not mean the attribute is truly more important. It only means the units are larger.

Diverse Attribute Domains

- This problem can be addressed by **scaling**.
- Scaling: Differences in attribute domains can be addressed by normalizing each domain to $[0,1]$, using a linear transformation.

Diverse Attribute Domains

- Suppose we measure income in thousands of dollars instead of dollars.
- This means that our records post-scaling are:
 - Amy (25, 50)
 - Bill (58, 51)
 - Carla (27, 52)
- How do the distances look like now?

Categorical Attributes

- Categorical attributes encode concepts that have no natural notion of **order** or **numerical difference**, making it difficult to choose a distance measure.
 - What is a meaningful distance between brown and black hair color, and how does it compare to the distance between brown and blonde?
- The distance for a categorical attribute could be set to 0 if the values agree, and to 1 otherwise.

Categorical Attributes

- Alternatively, a categorical attribute can be transformed to a set of binary attributes. For each possible value a binary attribute is created that is set to 1 if the categorical attribute has this value, and to 0 otherwise. This is called **one-hot** encoding.
 - Consider hair color with values blonde, red, brown, and black. In the transformed data, 4-valued attribute hair color is replaced by 4 binary attributes blonde, red, brown, and black. For a blonde person, they are set to blonde=1, red=0, brown=0, and black=0.

Categorical Attributes

Person	Hair Color
A	blonde
B	brown
C	black

- After one-hot encoding:

Person	blonde	brown	black
A	1	0	0
B	0	1	0
C	0	0	1

Categorical Attributes

- One-hot encoding avoids fake ordering, but it increases the number of dimensions.
- If a categorical attribute has 100 possible values, one-hot encoding creates 100 binary attributes.
- That can contribute to the curse of dimensionality.

Ordinal Attributes

- Ordinal attributes have a natural notion of order, but **not of numerical difference**. A typical example are ranks in the army and star ratings for products.
 - A product rating of 4 stars is better than 3 stars, but is a 2-star product twice as good as a 1-star product?
- Would different evaluators agree how to quantify the improvement reflected by one additional star?

Ordinal Attributes

- An ordinal attribute could be treated as a categorical attribute for distance computation. However, this ignores the additional knowledge about the ordering.
- To preserve the ordering, the values of the ordinal attribute can be mapped to values in range $[0,1]$.
 - For example, product rating k between 0 and 5 stars can be mapped to $k/5$. This mapping is based on implicit assumptions about the differences and ratios between ratings; it needs to be approached with care.

Next we explore algorithms that find clusters.

K-means Clustering

- K-means is one of the most popular clustering algorithms. It is comparably simple; therefore it serves as a perfect example for an algorithm that we can implement from scratch for parallel execution.
- K-means has solid mathematical foundations, with a well-defined optimization goal.

K-means Clustering

- In addition to a distance measure between objects, the user specifies parameter K , the number of clusters to be found.
- Starting from an initial configuration of K cluster centers, the algorithm iteratively adjusts the centers to improve the clustering.

What is a Cluster Center?

- K-means uses the centroid of a set of objects as their cluster center. Given a set C of objects, their centroid m is defined as $m = \frac{\sum_{x \in C} x}{|C|}$, i.e., m 's value in dimension i is the average of the values in the i -th dimension over all objects in C .
 - Consider a set $C = \{(1,1), (2,4), (6,4)\}$ of two-dimensional data points. Its centroid is point $((1+2+6)/3, (1+4+4)/3) = (3,3)$.
- As the example shows, the centroid does not need to be in C .
- To compute the centroid, **addition** and **division** need to be defined for the dimensions. This is guaranteed when working in a vector space. Categorical and ordinal attributes need to be transformed to numerical values.

K-means Algorithm

- The algorithm pseudo-code is shown below.
- For an example execution, see the next page.

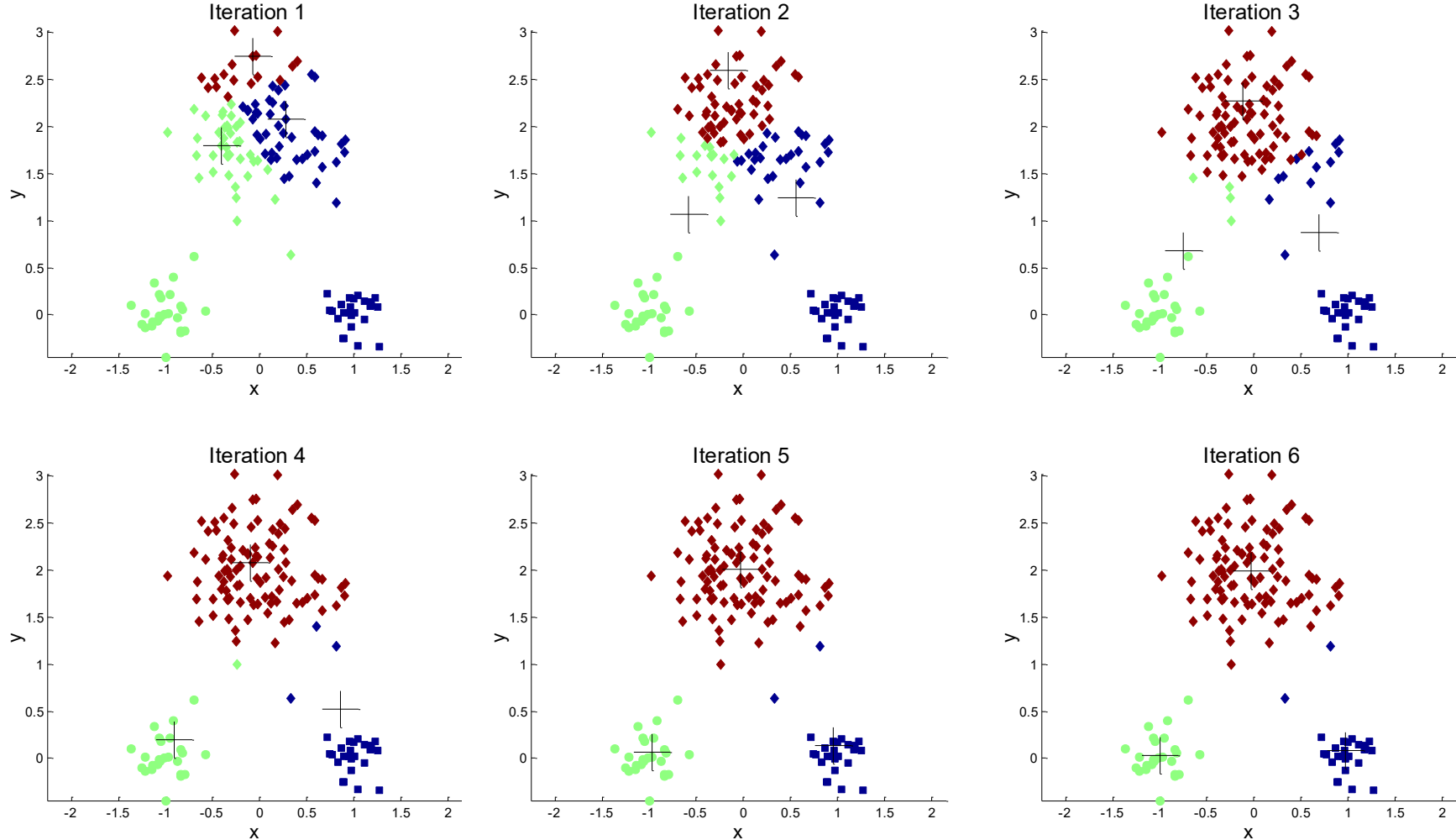
```
// Input: desired number of clusters K, data set D
K-means( K, D ) {

    Centroids = choose K initial centroids

    Repeat until centroids do not change {
        For each x in D do
            Assign x to the nearest centroid in Centroids

        For each C in Centroids do
            Re-compute C based on all data objects assigned to it
    }
}
```

The algorithm performs multiple iterations. In each iteration, first each data object is assigned to the nearest center, then the centers are updated based on this object assignment.



Convergence of K-means: The example shows how cluster centers are adjusted in each iteration. Colors indicate cluster membership for the input data points. Despite starting with all three centers in the big group, K-means automatically pushes the centers out to line up with the “natural” clusters. [from “Introduction to Data Mining” by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]

K-means Details

- K-means is comparably fast with complexity $O(n * K * I * d)$. Here n denotes the number of input objects, I the number of iterations, and d the number of dimensions (i.e., attributes of the objects).
- Implementing K-means requires two crucial design decisions: selection of the **initial centers** and choice of **distance measure**.

K-means Details

- The initial centers affect the final clustering found. Unfortunately, no existing algorithm guarantees to pick initial centers that result in the best clustering possible.
- A common approach relies on random restarts: Select the initial K centers randomly from the input data and run K-means until convergence.
 - Then repeat for another set of K randomly selected centers, until satisfied with the quality of the best clustering found.

Evaluating Cluster Quality

- A good clustering has points that are **close to their own centroid**. We measure this using **Sum of Squared Error (SSE)**:

$\sum_{i=1}^K \sum_{x \in C_i} \text{dist}^2(m_i, x)$. Here C_i denotes a cluster and m_i its centroid.

- **Lower SSE means tighter clusters** for the same value of K . But SSE usually decreases as K increases, so **do not compare SSE across different values of K directly**.

K-means Convergence

- Will SSE improve in each iteration: Yes, until convergence.
 - Given K cluster centers, K-means assigns each object to the nearest center. If it re-assigns an object x from one cluster to another, x 's distance to its cluster center must have decreased, reducing SSE.
 - Given K clusters, K-means updates the centers. It is easy to show that the centroid has a smaller SSE than any other possible center choice. Hence if a cluster centroid is updated, SSE must have decreased.

K-means Convergence

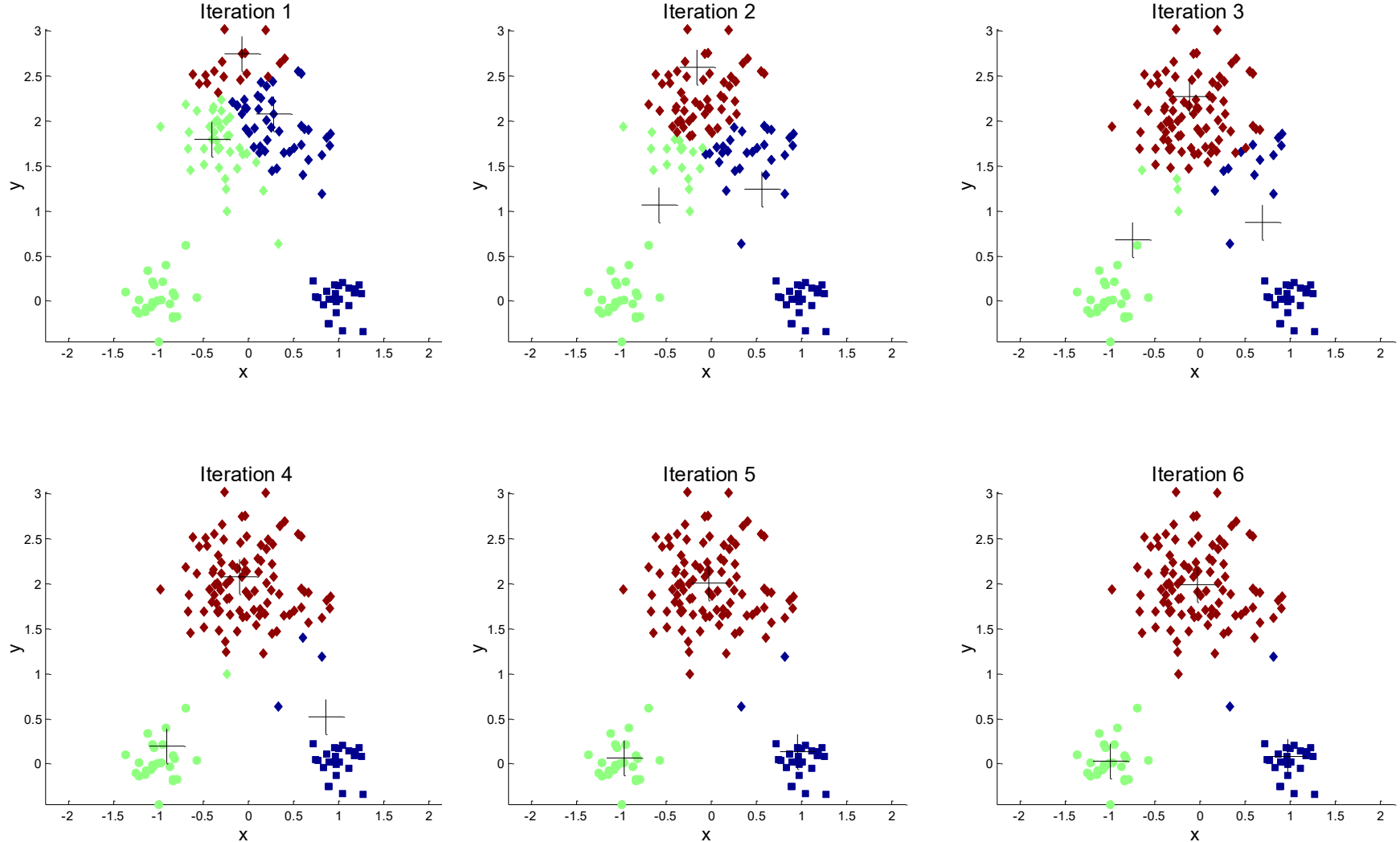
- Will K-means converge: Yes.
 - K-means iterates until the clustering does not change any more. In each iteration, SSE decreases.
 - Since SSE decreases in each iteration, K-means must have reached a new partitioning it has not explored before. So, if it tried to go on forever, it would eventually run out of configurations.

Optimality

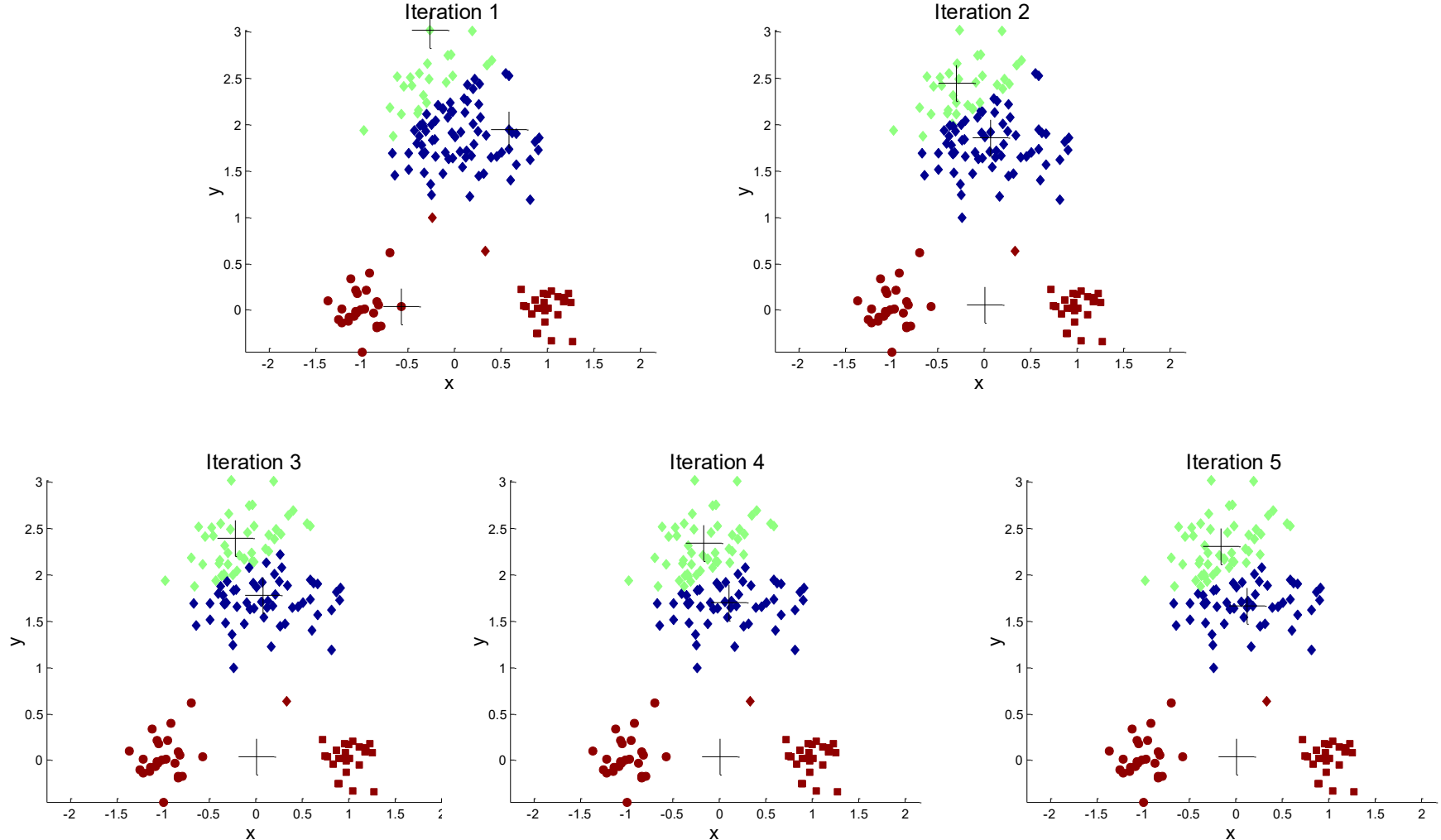
- The previous discussion showed that each iteration of K-means improves SSE until it converges.
- Unfortunately, this does not mean that it will always find the **optimal** clustering, i.e., the clustering with the lowest possible SSE for the given data.

Optimality

- More formally, K-means is guaranteed to converge to a **local** optimum.
- Since there are potentially many such local optima, it is not guaranteed to converge to the **global** optimum.
- The choice of initial cluster centers determines to which local optimum K-means will converge. Let's look at some examples.



In this example, the initial selection of cluster centers appears poor, because all three centers are in the same “natural” cluster. However, iteration by iteration the centers move in the right direction. [from “Introduction to Data Mining” by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]



In this example, the initial selection of cluster centers appears more reasonable as all centers are fairly far apart. Unfortunately, K-means gets trapped in a local optimum that does not correspond to the “natural” clusters. [from “Introduction to Data Mining” by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]

Selecting Initial Cluster Centers

- Since the choice of initial cluster centers determines to which local optimum K-means converges, researchers have explored how to find a good initial configuration.
- Unfortunately, there is no general solution for this problem.

Selecting Initial Cluster Centers

- Some heuristics attempt to place each initial cluster center in a different “natural” cluster of the data.
- The probability of finding such a configuration through random selection is extremely low.
 - Hence it is often attempted to place initial centers far apart from each other. This is not guaranteed to work well either.

Selecting Initial Cluster Centers

- Even if K-means is lucky in choosing “good” initial cluster centers in different natural clusters, it might converge to a “bad” clustering as we will discuss next.

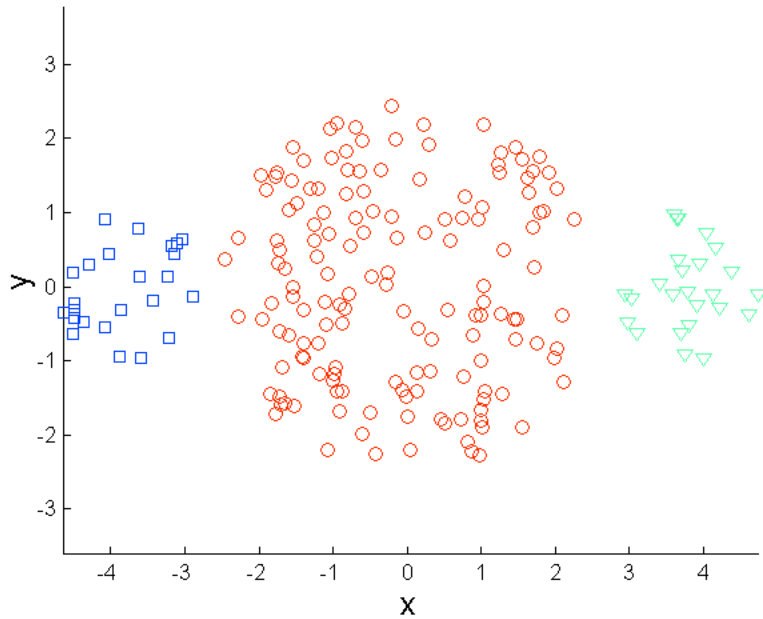
Limitations of K-means

- Assume the user was able to choose the perfect value of K , the right distance measure, and the best initial cluster centers.
- Would this guarantee that a good clustering is found?

Limitations of K-means

- Unfortunately, there are inherent limitations of K-means that prevent it from finding the “natural” clusters under certain circumstances.
- In particular, K-means has problems when clusters are of differing **sizes**, **densities**, or have **non-globular** shapes—as we illustrate with examples below.

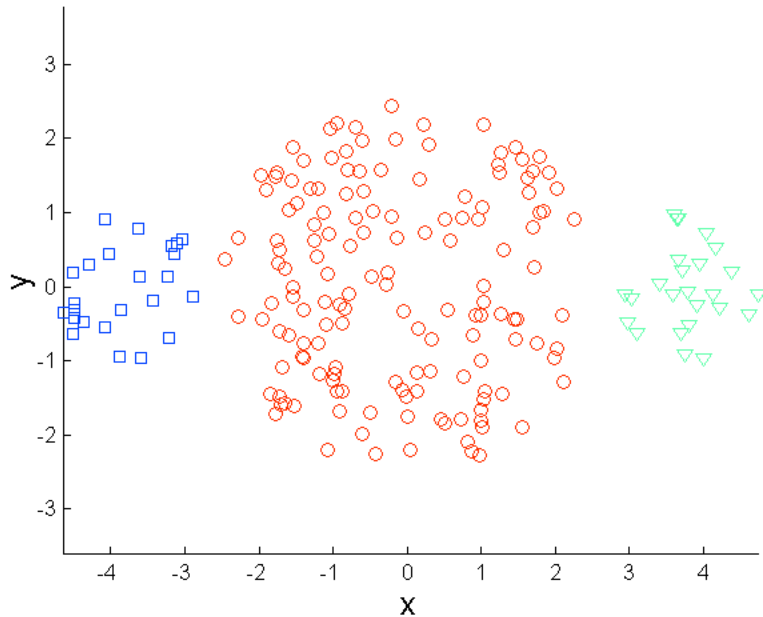
Differing Sizes



Natural clustering: Intuitively, many users would like to discover the three clusters indicated by point shape and color.

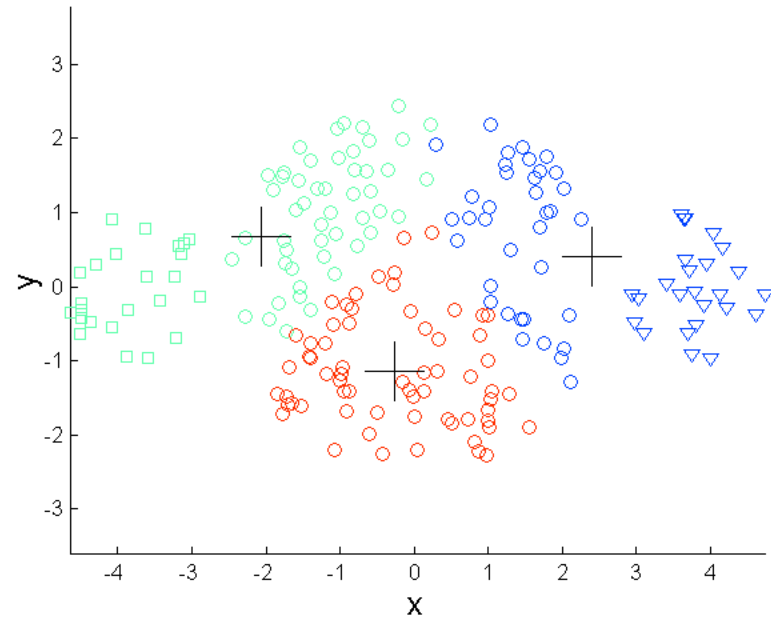
[from "Introduction to Data Mining" by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]

Differing Sizes



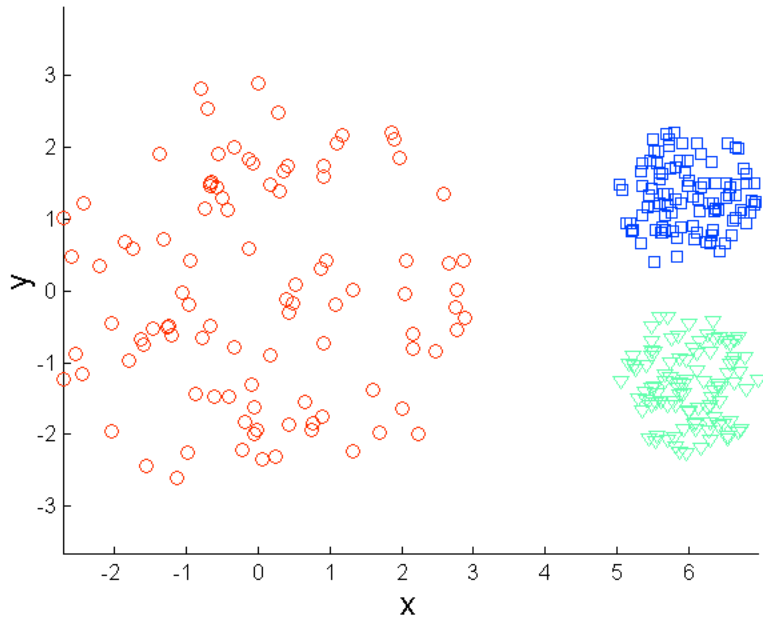
Natural clustering: Intuitively, many users would like to discover the three clusters indicated by point shape and color.

[from "Introduction to Data Mining" by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]



Optimal K-means result: The large natural cluster contains many more points than the small ones. Due to the different diameters of the natural clusters, K-means cannot find them. Even if it started with the perfect cluster centers, it would converge to a clustering as shown, because that minimizes SSE. For instance, note that points on the left fringe of the large cluster are closer to the center of the left small cluster than to the center of the large cluster.

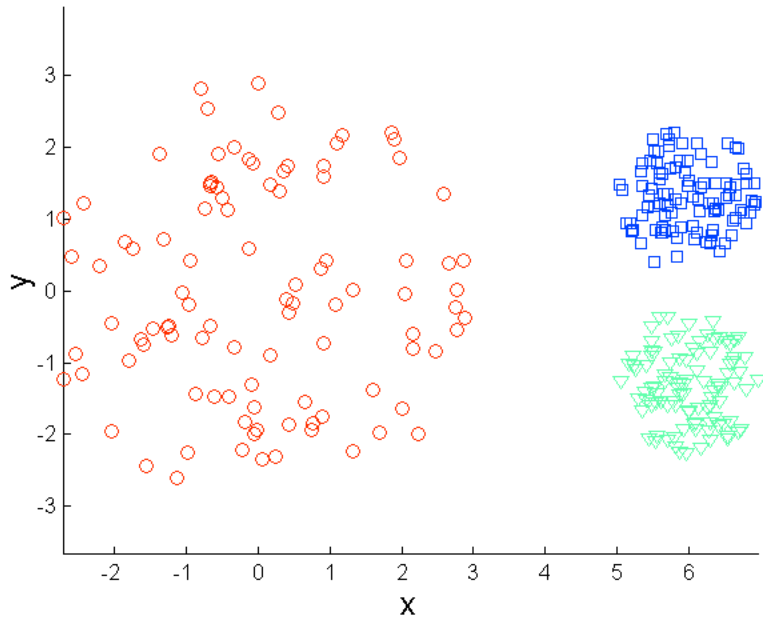
Differing Density



Natural clustering: Intuitively, many users would like to discover the three clusters indicated by point shape and color.

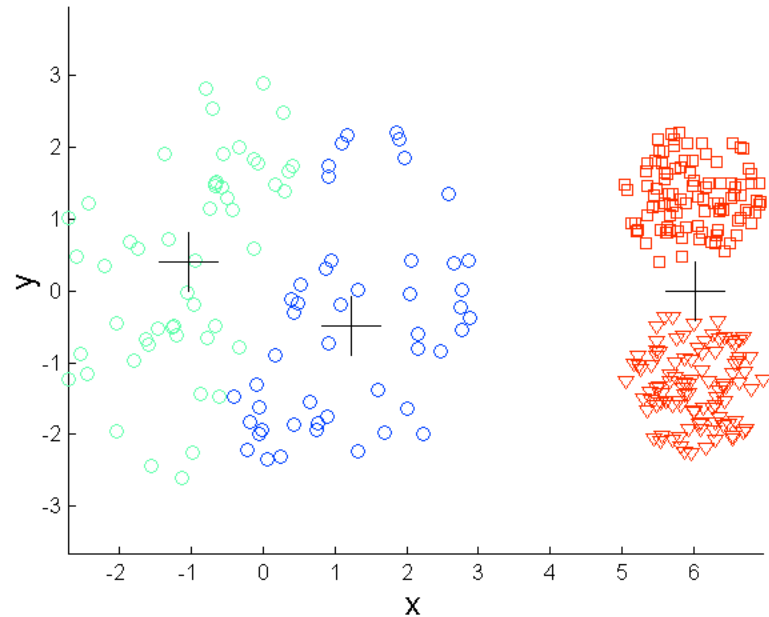
[from "Introduction to Data Mining" by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]

Differing Density



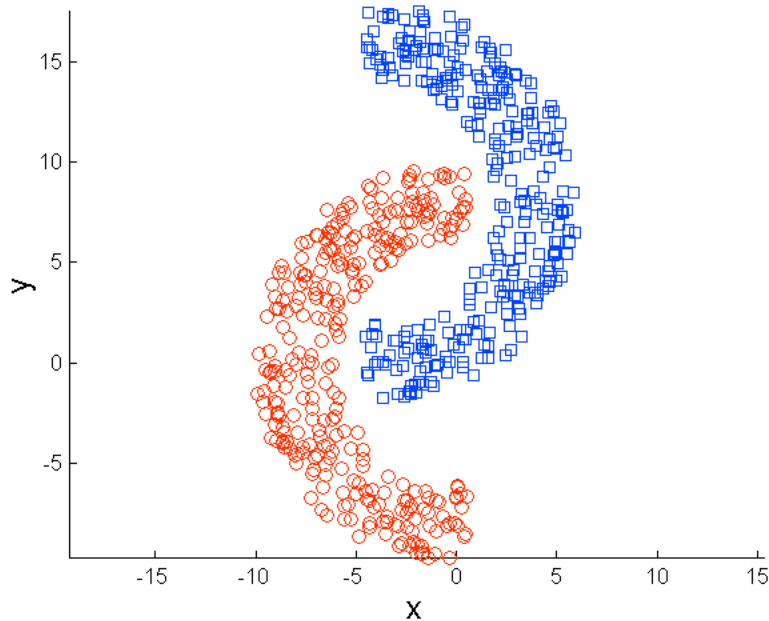
Natural clustering: Intuitively, many users would like to discover the three clusters indicated by point shape and color.

[from "Introduction to Data Mining" by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]



Optimal K-means result: In this example, each of the natural clusters has about the same number of points. However, due to their different densities, the large natural cluster has a much larger diameter than the other two. This results in the same problem as for the previous example with differing sizes.

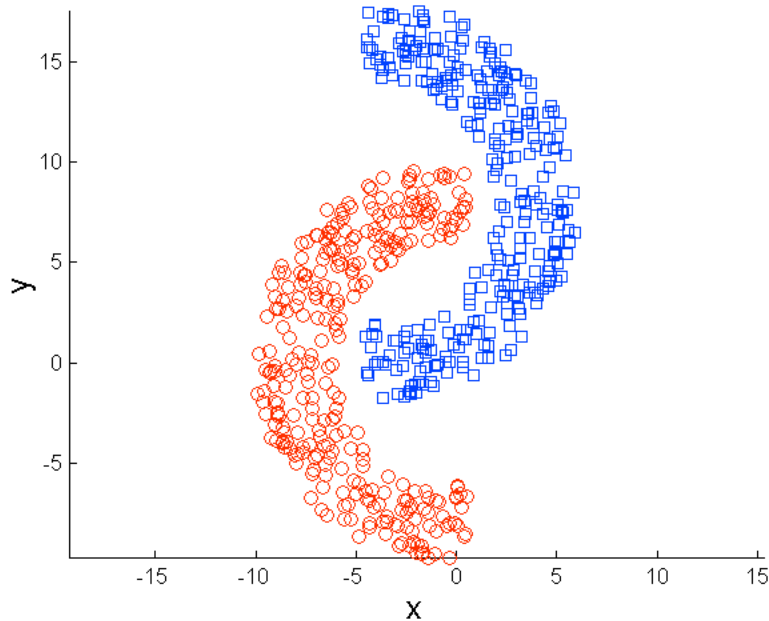
Non-globular Shapes



Natural clustering: Intuitively, many users would like to discover the two clusters indicated by point shape and color.

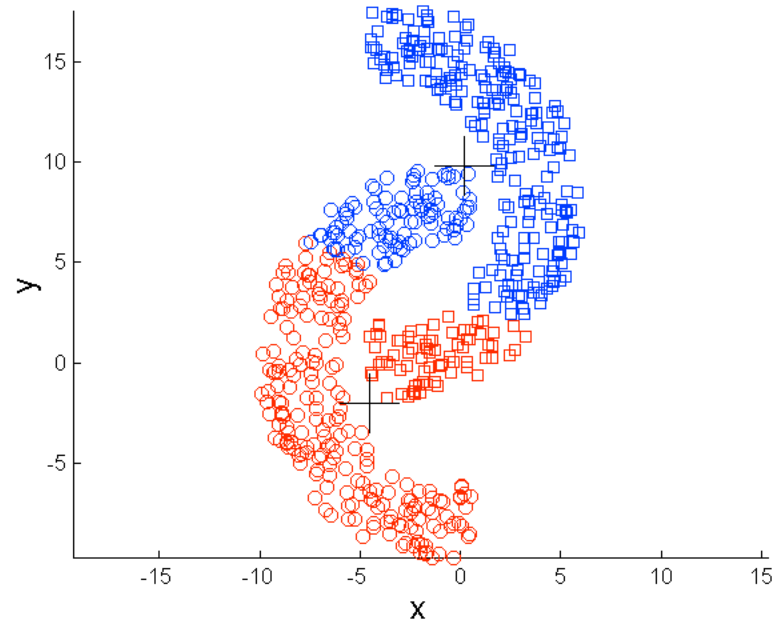
[from "Introduction to Data Mining" by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]

Non-globular Shapes



Natural clustering: Intuitively, many users would like to discover the two clusters indicated by point shape and color.

[from "Introduction to Data Mining" by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]



Optimal K-means result: Due to the elongated shape, some points of the left natural cluster are closer to the right natural cluster than to their own center. Again, even if K-means started with the perfect cluster centers, it would still converge to a clustering as shown, because that minimizes SSE.

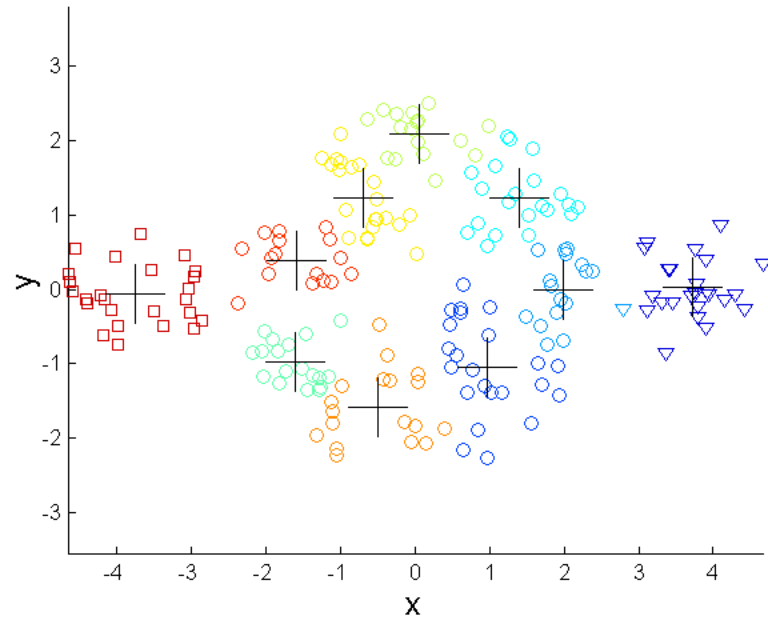
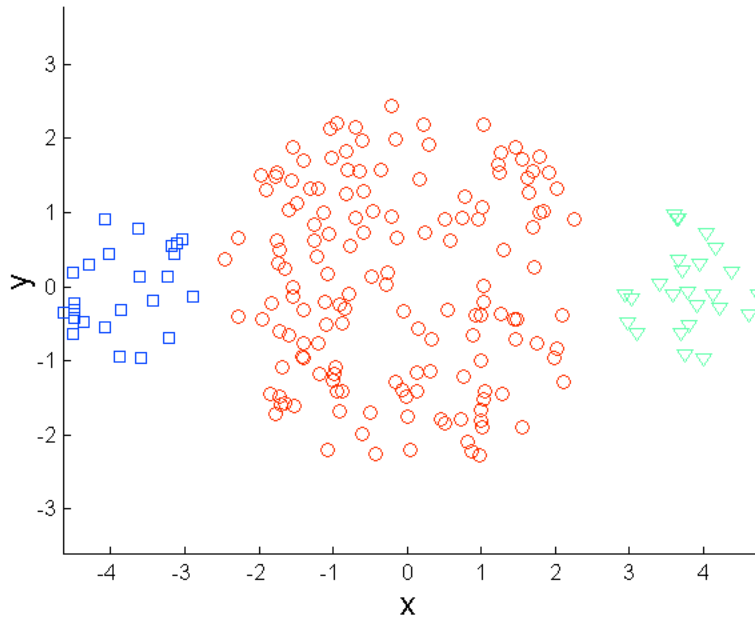
Addressing K-means Limitations

- There exist solutions for the inherent problems of K-means.
- Choose a **different algorithm**:
 - Each clustering algorithm is designed for a certain intuitive notion of a cluster. For instance, if the user is looking for clusters based on point density, not based on minimizing SSE, then they should consider an algorithm designed for finding density-based clusters.

Addressing K-means Limitations

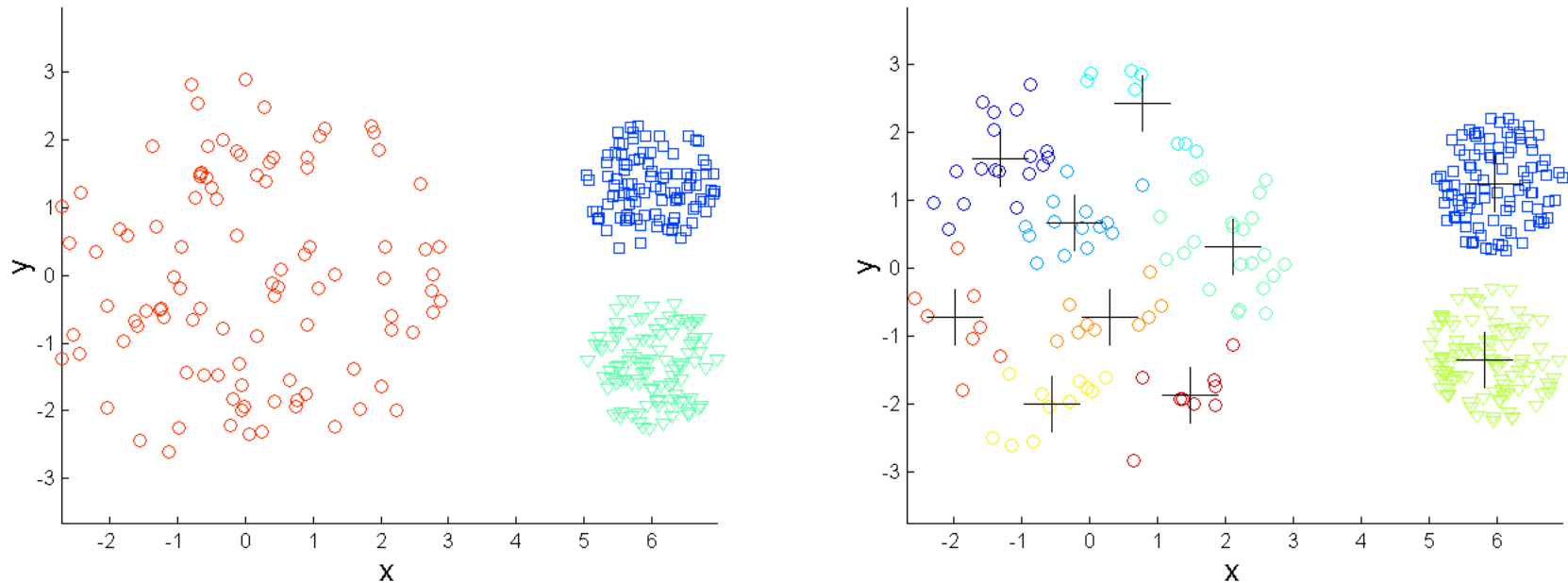
- Fix K-means through **post-processing**:
 - Instead of using the desired K , the user can run K-means for some $K' > K$. Then the K' clusters found are post-processed by a combination of the following operations:
 - Eliminate small clusters that may represent outliers.
 - Split very large clusters.
 - Merge clusters that are close to each other.

Fixing the Differing-Size Problem



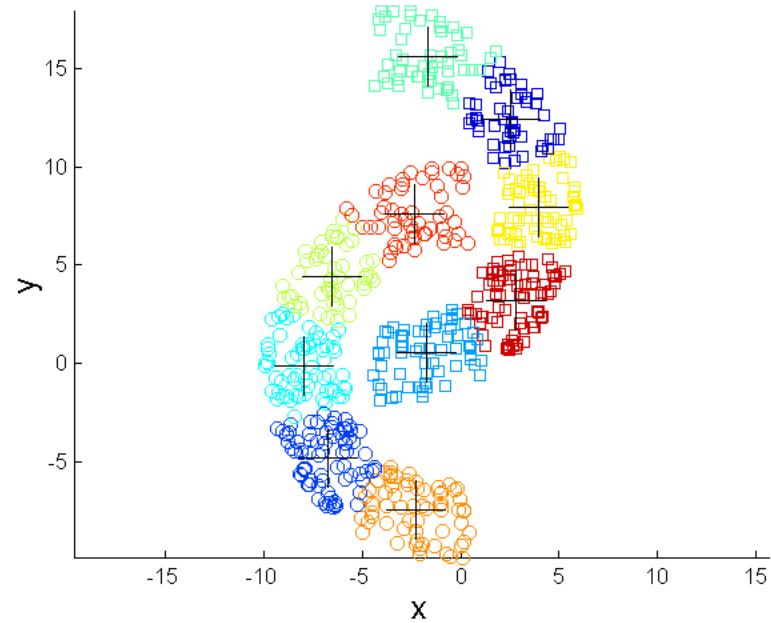
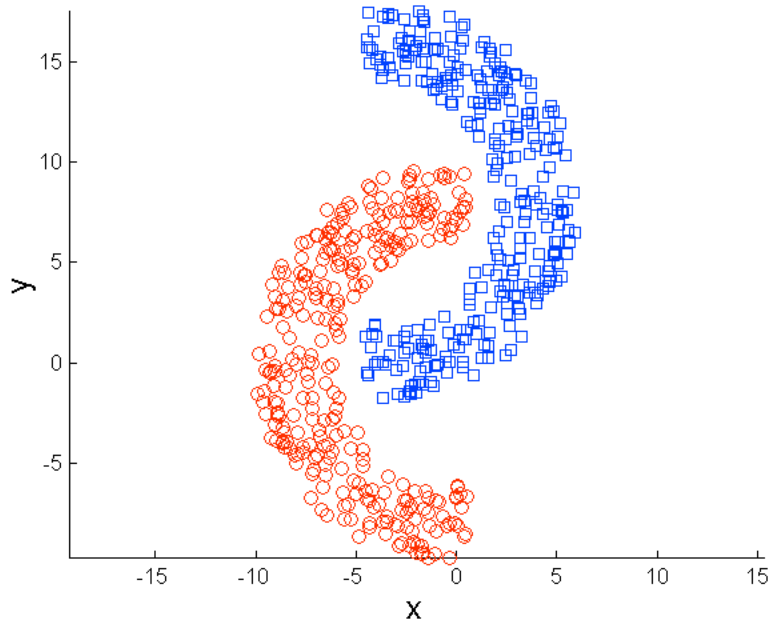
The K-means run for $K'=10$ found 10 small clusters, each part of one of the three natural clusters. An appropriate post-processing algorithm might be able to combine the 8 small clusters in the middle to form the large natural cluster.

Fixing the Differing-Density Problem



The K-means run for $K'=10$ found 10 small clusters, each part of one of the three natural clusters. An appropriate post-processing algorithm might be able to combine the 8 small low-density clusters on the left to form the large natural cluster.

Fixing the Shape Problem

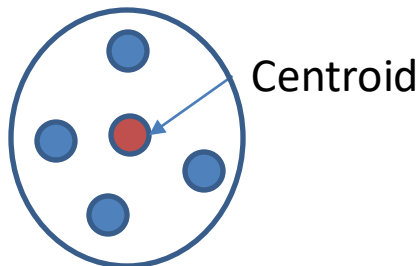


The K-means run for $K'=10$ found 10 small clusters, each part of one of the two natural clusters. An appropriate post-processing algorithm might be able to combine each group of five adjacent small clusters into the corresponding large natural cluster.

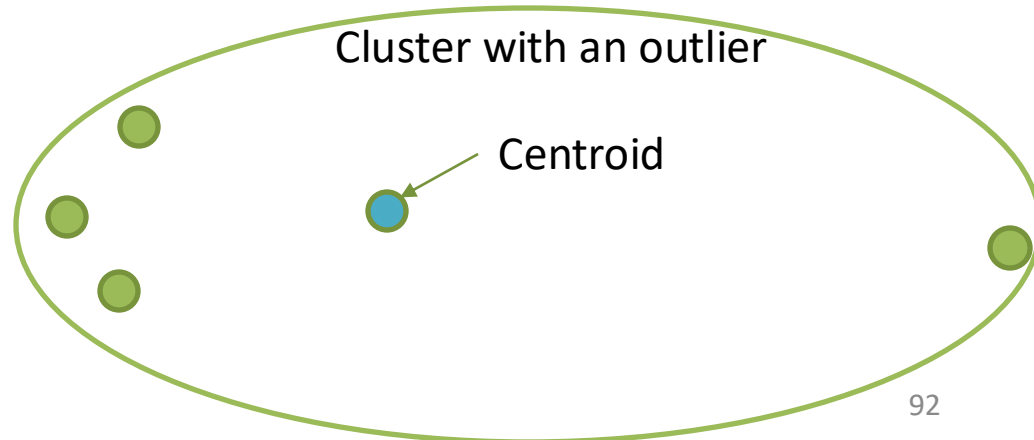
K-Means and Outliers

- K-means is sensitive to outliers, because each centroid is an average of the cluster members.
- Even a single outlier with a very large value in a dimension can dominate the average.

Cluster without outliers



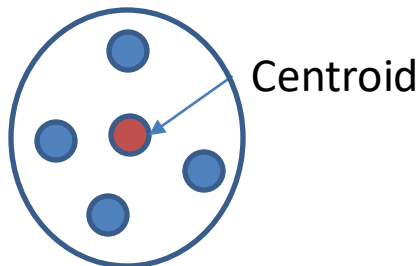
Cluster with an outlier



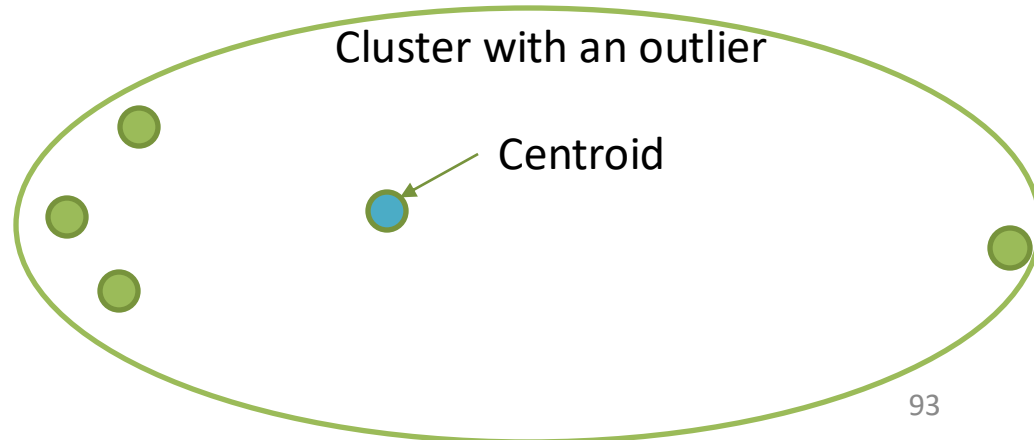
K-Means and Outliers

- This problem is addressed by the **K-medoids** algorithm. Instead of the centroid, it represents a cluster by its medoid, which is the *most centrally located real object* in a cluster.
 - The algorithm works like K-means, but finding the medoid is computationally more expensive than finding the centroid.

Cluster without outliers



Cluster with an outlier



Distributed K-Means

- Like other iterative algorithms, e.g., for graph processing, we distinguish between immutable and evolving data.
- The former should be loaded once and then kept in memory, while the latter potentially needs to be shuffled across the network.
- For clustering, the set of input objects is fixed. Each task receives a partition of this dataset.

Distributed K-Means

- Cluster centers can change in each iteration. When centers change, membership of objects in clusters might also change.
- Which data structures do we need to represent the immutable and evolving data?
 - We store the data objects in `Objects` and the cluster centers in `Centers`.

Distributed K-Means (Cont.)

- Each iteration consists of two rounds.
- Round 1:
 - Each task receives a partition of $\mathcal{O}$ bjects. Spark can re-use these partitions in later iterations.
 - We broadcast $\mathcal{C}$ enters to all tasks.
 - The task assigns each object in the partition to the closest center.

Distributed K-Means (Cont.)

- Round 2:
 - Each task receives a group of objects that were assigned to the same center.
 - The task computes the new center for the group by averaging the coordinates of all objects in the group.
- The output of round 2 is the new set of centers.

K-Means in MapReduce

- We implement round 1 as Map and round 2 as Reduce. Centers is broadcast using the file cache. Mappers must read their chunk of Objects anew in every iteration.
- The driver program repeatedly calls this program, putting the new Centers in the file cache after each iteration.

```
Class Mapper {  
  Centroids // Array containing the K cluster centers  
  
  setup()  
    Centroids = read centroids from file cache  
  
  map( object o ) {  
    closestCenter = Centroids[0]  
    minDist = dist(closestCenter, o)  
  
    for i=1 to k-1 {  
      if (dist(Centroids[i], o) < minDist)  
        closestCenter = Centroids[i]  
        minDist = dist(Centroids[i], o)  
    }  
    emit( closestCenter, o )  
  }  
}
```

```
reduce( center, [o1, o2,...]) {  
  for each object o in inputList {  
    update object count  
    update coordinate-sum for each dimension  
  }  
  
  newCentroid = compute the average  
                 for each dimension  
  emit ( newCentroid )  
}
```

Algorithm Analysis

- In each iteration, the entire Objects file is transferred to Mappers, then from Mappers to Reducers.
- Reducers write out the (usually small) file with the new centroids, which is broadcast to all Mappers in the next iteration.

Algorithm Analysis

- Like for the iterative graph algorithms, performance suffers from MapReduce's inability to exploit the repetitive structure of the computation.

K-Means in Spark

- Spark's ability to maintain data in memory in RDDs or DataSets enables the desired implementation where the object set is kept across iterations.
- If we implement the algorithm from scratch, we need to:
 - Broadcast the new centers to all Executors using `broadcast()` or create a Scala collection that is automatically sent to all tasks.
 - Keep `Objects` in an RDD or DataSet. Use `cache()` or `persist()` to tell Spark to keep it in memory as much as possible.
 - Use `map` on `Objects`, calling a function that for object `o` emits pair `(nearestCenterID, o)`.
 - Group by `nearestCenterID` and aggregate each group to compute the average in each dimension.
- For real code (from the Spark 2.4.0 distribution) look at <http://khoury.northeastern.edu/home/mirek/code/SparkKMeans.scala>
- Or we can just use machine learning library Spark MLlib.

K-Means in MLlib with RDD

(from Spark 2.3.2 Documentation)

```
import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
import org.apache.spark.mllib.linalg.Vectors

// Load and parse the data
val data = sc.textFile("data/mllib/kmeans_data.txt")
val parsedData = data.map(s => Vectors.dense(s.split(' ').map(_.toDouble))).cache()

// Cluster the data into two classes using KMeans
val numClusters = 2
val numIterations = 20
val clusters = KMeans.train(parsedData, numClusters, numIterations)

// Evaluate clustering by computing Within Set Sum of Squared Errors
val WSSSE = clusters.computeCost(parsedData)
println(s"Within Set Sum of Squared Errors = $WSSSE")

// Save and load model
clusters.save(sc, "target/org/apache/spark/KMeansExample/KMeansModel")
val sameModel = KMeansModel.load(sc, "target/org/apache/spark/KMeansExample/KMeansModel")
```

K-Means in Mllib With DataSet

(from Spark 2.3.2 Documentation)

```
import org.apache.spark.ml.clustering.KMeans
import org.apache.spark.ml.evaluation.ClusteringEvaluator

// Loads data.
val dataset = spark.read.format("libsvm").load("data/mllib/sample_kmeans_data.txt")

// Trains a k-means model.
val kmeans = new KMeans().setK(2).setSeed(1L)
val model = kmeans.fit(dataset)

// Make predictions
val predictions = model.transform(dataset)

// Evaluate clustering by computing Silhouette score
val evaluator = new ClusteringEvaluator()

val silhouette = evaluator.evaluate(predictions)
println(s"Silhouette with squared euclidean distance = $silhouette")

// Shows the result.
println("Cluster Centers: ")
model.clusterCenters.foreach(println)
```

Notes and Calendar

9	June 30, July 2	Data Mining 2 (Ensembles)	Data Mining 2	Homework 4 Due
10	July 7, July 9	Intelligent Partitioning	Intelligent Partitioning	
11	July 14, July 16	Lineage, Spark SQL	More About Spark	
12	no class on July 21 for exam preparation, July 23			Exam
13	July 28, July 30	Exam Solutions, Spark Stream, CAP Theorem		Begin Project
14	Aug 4, Aug 6	Topics TBD	TBD	Project Due
15	Aug 11, Aug 13	Topics TBD	TBD	Project Presentations

Notes and Calendar

9	June 30, July 2	Data Mining 2 (Ensembles)	Data Mining 2	Homework 4 Due
10	July 7, July 9	Intelligent Partitioning	Intelligent Partitioning	
11	July 14, July 16	Lineage, Spark SQL	More About Spark	
12	no class on July 21 for exam preparation, July 23			Exam
13	July 28, July 30	Exam Solutions, Spark Stream, CAP Theorem		Begin Project
14	Aug 4, Aug 6	Topics TBD	TBD	Project Due
15	Aug 11, Aug 13	Topics TBD	TBD	Project Presentations

Notes and Calendar

- The practice exam is mostly done...
 - This will be released on THU July 16 (I think)
 - We will discuss it on TUE July 21
- Please use this as a chance to simulate the test environment (and the practice the 30% rule as well)

Notes and Calendar

9	June 30, July 2	Data Mining 2 (Ensembles)	Data Mining 2	Homework 4 Due
10	July 7, July 9	Intelligent Partitioning	Intelligent Partitioning	
11	July 14, July 16	Lineage, Spark SQL	More About Spark	
12	no class on July 21 for exam preparation, July 23			Exam
13	July 28, July 30	Exam Solutions, Spark Stream, CAP Theorem		Begin Project
14	Aug 4, Aug 6	Topics TBD	TBD	Project Due
15	Aug 11, Aug 13	Topics TBD	TBD	Project Presentations

Final Project

- There is a document on Canvas that covers some sample projects.
- I suggest to start forming teams. As for choosing a topic, let's wait until we cover next week's topic.

Final Project

- Proposal to change the following:
 - Project Intermediate Report is due SUN AUG 2.
 - I will post a report template.
 - Instead of having in-person presentations, the groups can record their presentations and submit them via Gradescope.
 - TENTATIVE: I think we can give you guys until AUG 14 (FRI) to submit the final report and the video presentations.

Final Project

- There is a document on Canvas that covers some sample projects.
- I suggest to start forming teams. As for choosing a topic, let's wait until we cover next week's topic.
- Proposal to change the following:
 - Project Intermediate Report is due SUN AUG 2.

Now we change gears and look at supervised learning, in particular classification and prediction (a.k.a. regression).

Classification and Prediction

- Classification and prediction are among the most common data mining tasks in practice. The goal is to predict some output of interest for a given input record, for instance:
 - Predict if somebody is likely to repay a mortgage.
 - Predict if a credit-card transaction is fraudulent.
 - Predict if the customer will purchase the product.

Classification and Prediction

- Formally, consider a data set with attributes $X_1, \dots, X_d$, and Y . From this data, a model is trained, which is a function $f: (X_1, \dots, X_d) \rightarrow Y$. This function can then be used to predict the unknown output y for a given input record $(x_1, \dots, x_d)$.
- For **classification** problems, Y is a discrete attribute, called the class label. For **prediction** problems, Y is a continuous attribute.

Classification Example: Labeled Data

- Given a data set of recent graduates, NEU wants to predict for each graduating senior if they will receive a job offer within a year of graduation (column “Job Offer?”).

Training Data

Name	Age	GPA	Major	Job Offer?
Joe	24	3.7	CS	Yes
Amy	28	3.9	CS	Yes
Joe	29	3.3	ECE	Yes
Bill	24	3.1	Bio	No
Beth	22	3.8	Art	No

Test Data

Name	Age	GPA	Major	Job Offer?
Mary	23	4.0	CS	Yes
Joe	24	3.9	Hist	No
Amy	25	3.6	CS	Yes

Classification Example: Induction

- **Induction** refers to the process of training (or “fitting”) a model that captures the relationship between input attributes (name, age, GPA, major) and output (job offer).

Name	Age	GPA	Major	Job Offer?
Joe	24	3.7	CS	Yes
Amy	28	3.9	CS	Yes
Joe	29	3.3	ECE	Yes
Bill	24	3.1	Bio	No
Beth	22	3.8	Art	No

Model 1: IF (major = CS OR major = ECE)
THEN job = Yes; ELSE job = No

Model 2: IF (name = Joe OR name = Amy)
THEN job = Yes; ELSE job = No

Classification Example: Induction

- Many different models can be fit to the same training data, some more realistic (Model 1) than others (Model 2). It usually is not that obvious to identify the more realistic model.
 - Models may pick up correlations that are not causal relationships, e.g., the name of a person is not causing the person to get a job offer. This is an idiosyncrasy of the given data.
 - Larger training data protects better against such spurious relationships, e.g., when a Joe or Amy without job offer appear in the data, but it does not eliminate it.

Name	Age	GPA	Major	Job Offer?
Joe	24	3.7	CS	Yes
Amy	28	3.9	CS	Yes
Joe	29	3.3	ECE	Yes
Bill	24	3.1	Bio	No
Beth	22	3.8	Art	No

Model 1: IF (major = CS OR major = ECE)
THEN job = Yes; ELSE job = No

Model 2: IF (name = Joe OR name = Amy)
THEN job = Yes; ELSE job = No

Classification Example: Deduction

- **Deduction** refers to the process of using the model to make predictions.
- The more meaningful Model 1 gets more of the test records right than Model 2. The fraction of correctly classified test records is called the **accuracy** of the model.
- To evaluate a model realistically, the test records should not be used for training and should be drawn from the same distribution as future records for which the model will be used to make a prediction.

Name	Age	GPA	Major	Job Offer?
Mary	23	4.0	CS	Yes
Joe	24	3.9	Hist	No
Amy	25	3.6	CS	Yes

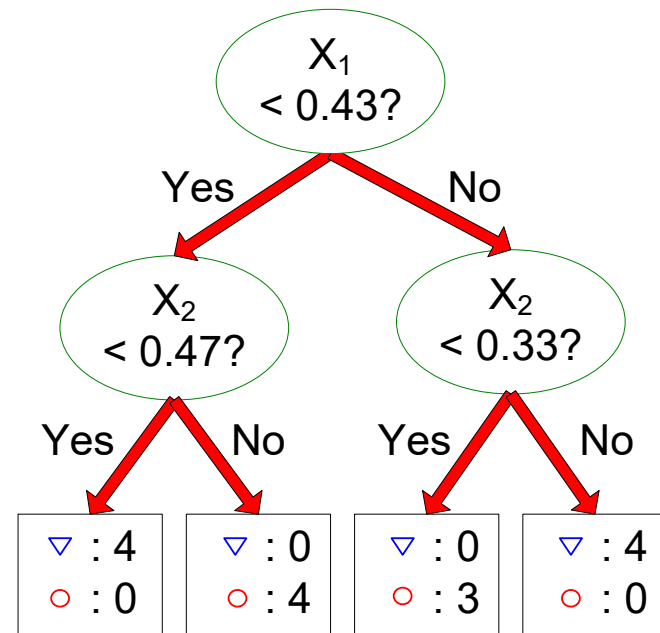
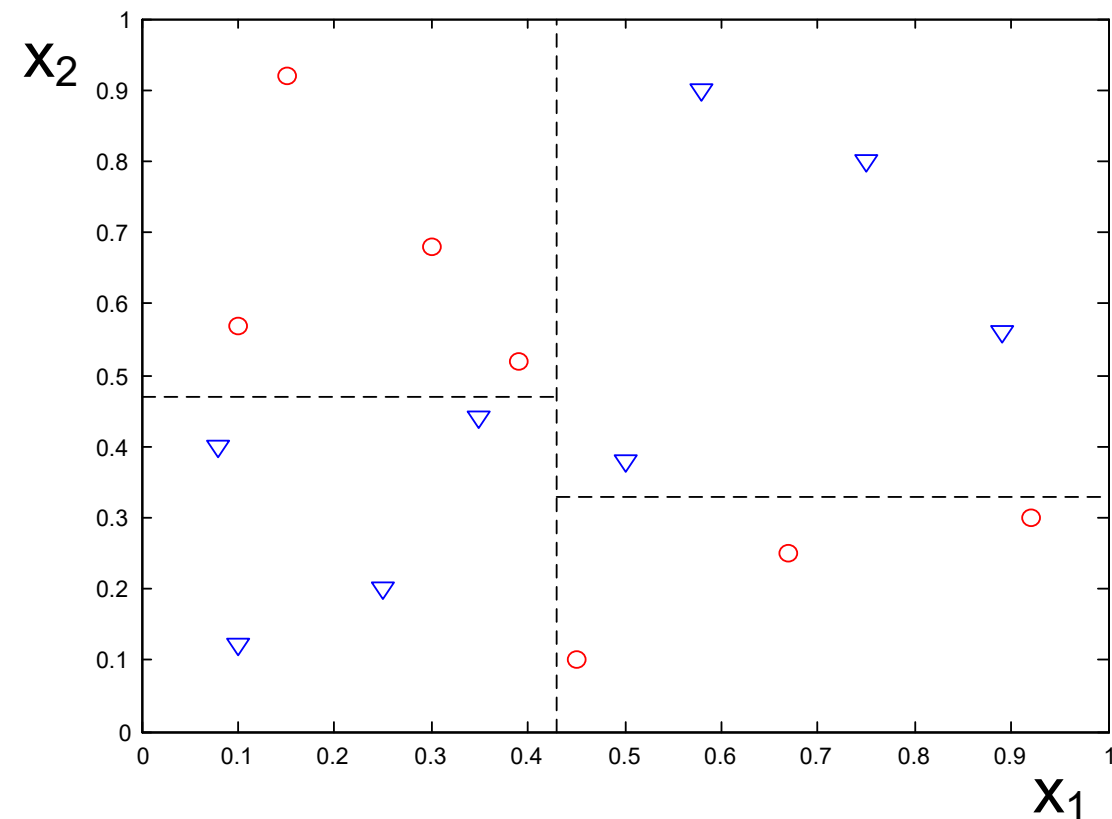
Model 1: IF (major = CS OR major = ECE)
THEN job = Yes; ELSE job = No

Model 2: IF (name = Joe OR name = Amy)
THEN job = Yes; ELSE job = No

Model 1 prediction	Model 2 prediction
Yes	No
No	Yes
Yes	Yes

Decision Trees

- Decision trees are a popular technique for classification problems.
- A decision tree splits the data space recursively in order to separate the different classes from each other as much as possible.
- The nodes of the tree contain split attributes that guide the search to the appropriate leaf when making a prediction.



The decision tree on the right defines partitions of the data space. The goal of these partitions is to have “pure” leaves, i.e., have ideally only members of a single class in a leaf.

The *decision boundary* is the border between two neighboring regions of different classes. For trees that split on a single attribute at a time, the decision boundary is parallel to the axes.

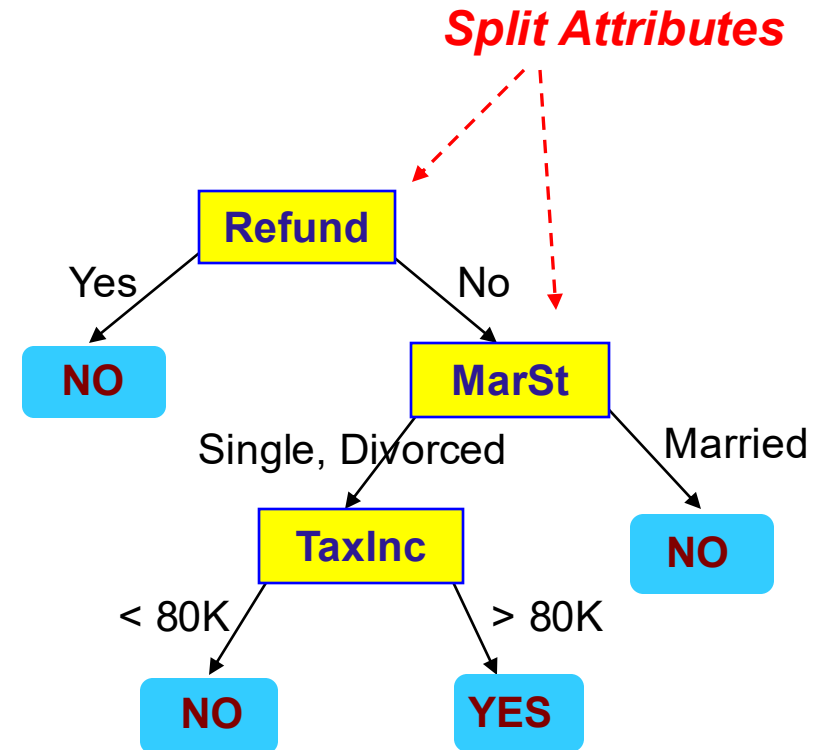
Tree Uniqueness

- For a given data set, there are usually many structurally different trees that achieve similar accuracy on the training data.
- For **consistent** data, there is more than one tree structure that perfectly represents the entire training data set.
 - Data is consistent if it does not contain two records that have the same input values, but different output.

Decision Tree That Exactly Represents the Training Data

<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

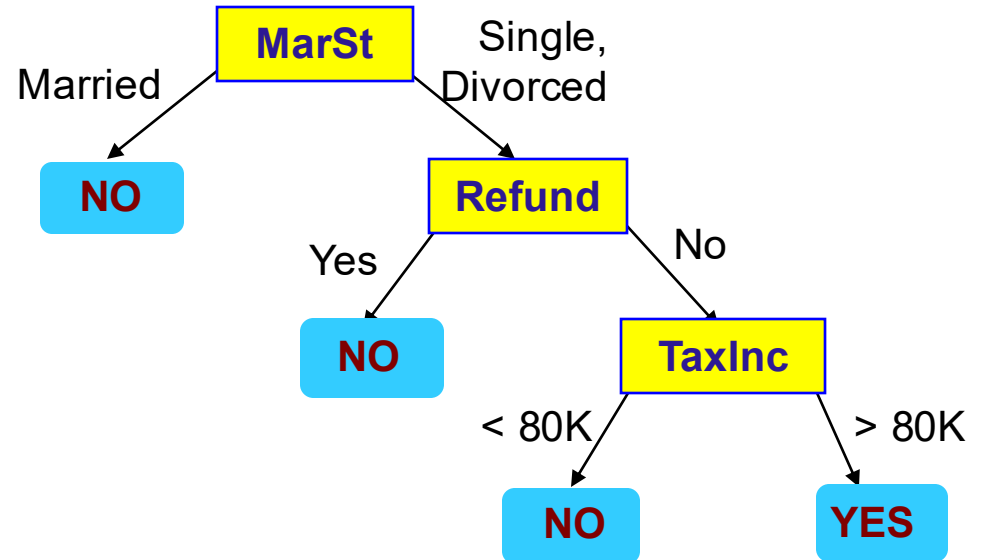
Training Data



Model

A Different Decision Tree That Exactly Represents the Same Training Data.

<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

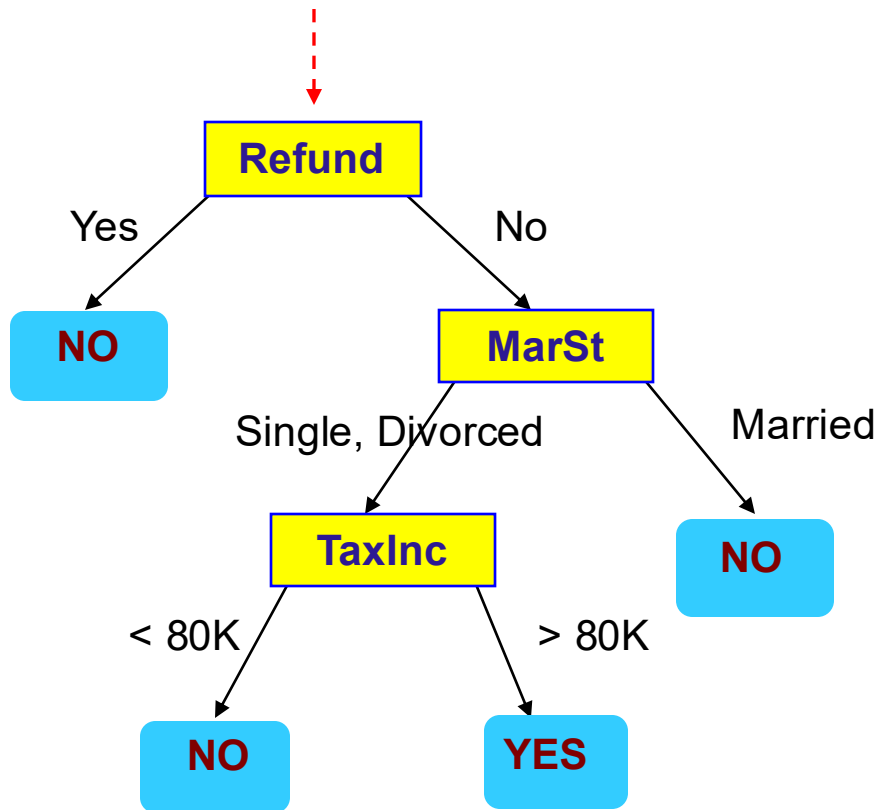


[Example source: "Introduction to Data Mining" by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]

Making Predictions

- Given a record containing only values for the input attributes, the tree will find the data partition this record falls into. It will then return a class value based on the training records in that partition.
- Tree traversal starts at the root, following the pointer to the next node based on the input value for the node's split attribute.

Start from the root of the tree



Test Data

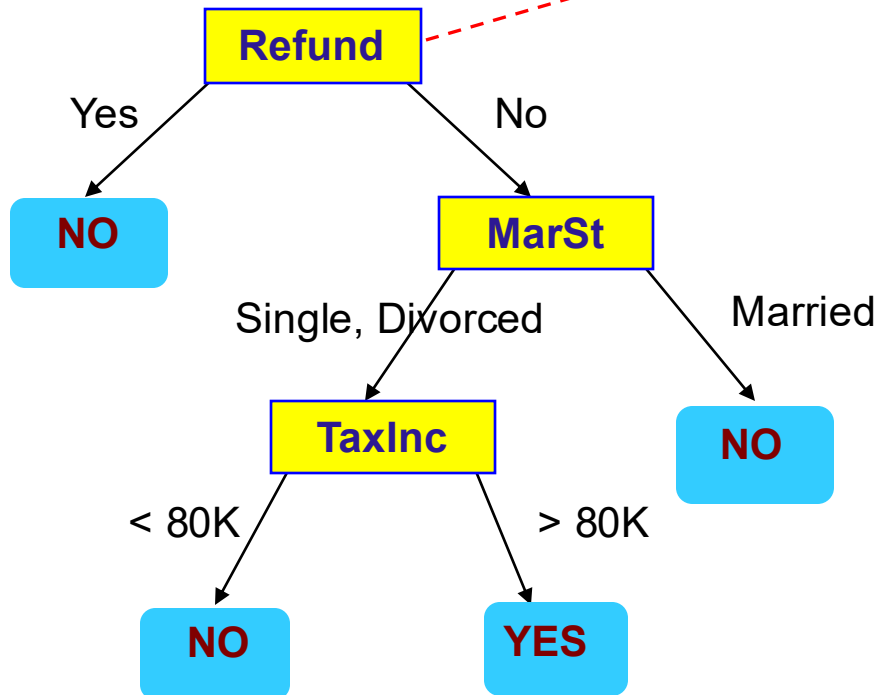
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

[Example source: "Introduction to Data Mining" by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]

Check the Refund value to determine which pointer to follow.

Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

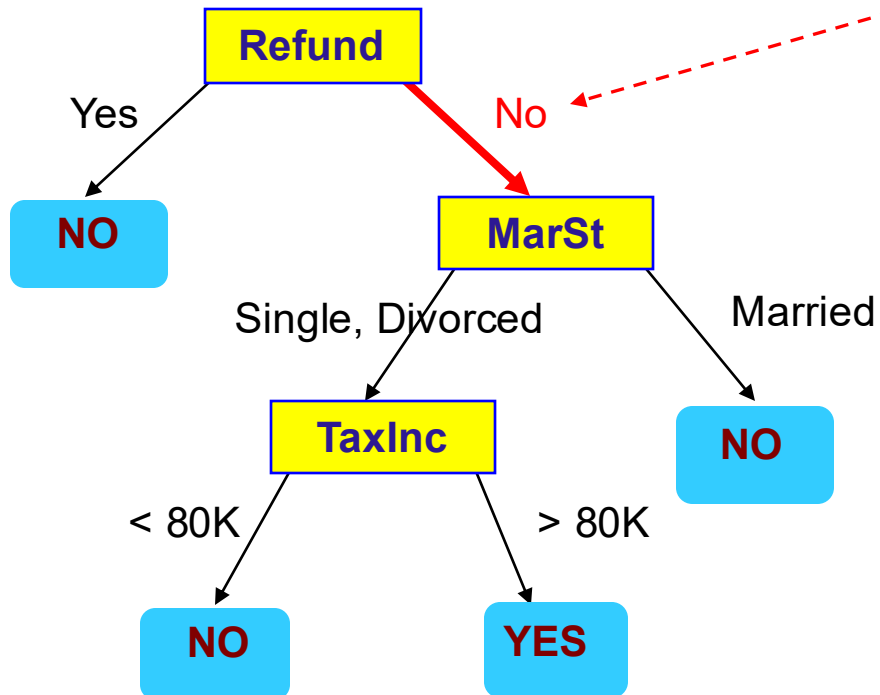


[Example source: "Introduction to Data Mining" by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]

Refund = No, therefore follow the right pointer.

Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

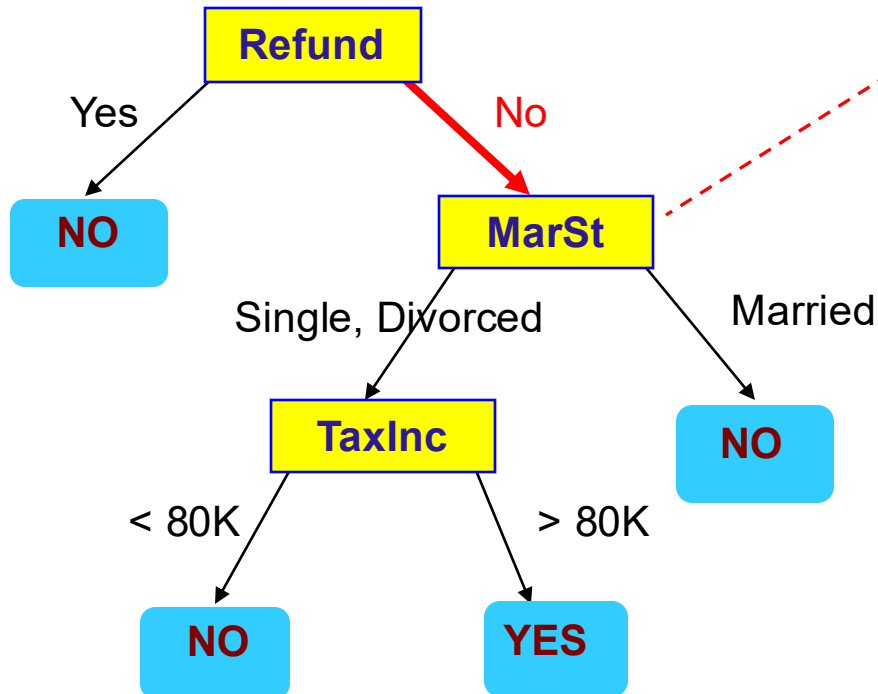


[Example source: "Introduction to Data Mining" by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]

Check the Marital Status value to determine which pointer to follow.

Test Data

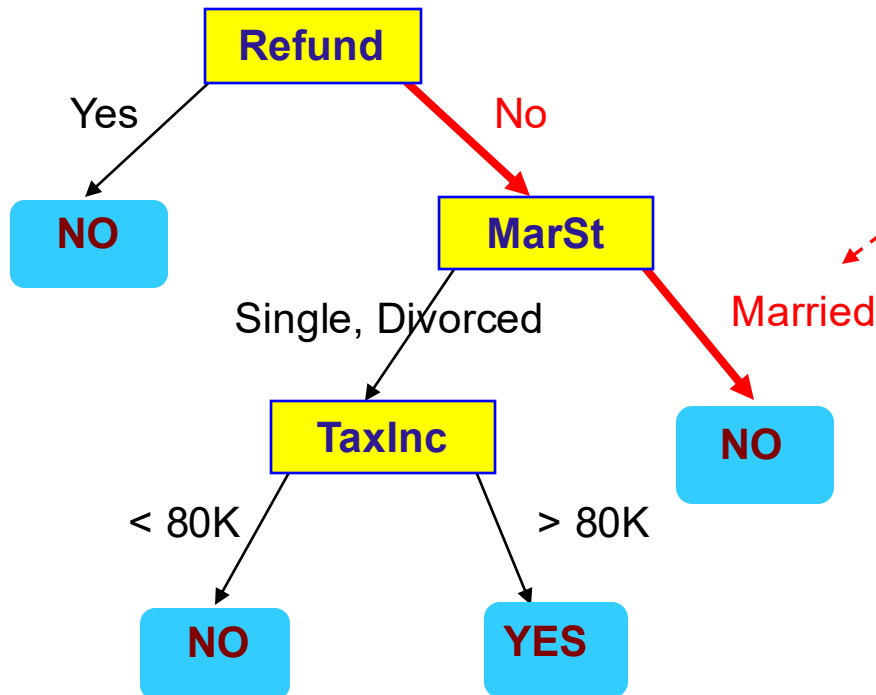
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



[Example source: "Introduction to Data Mining" by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]

Test Data

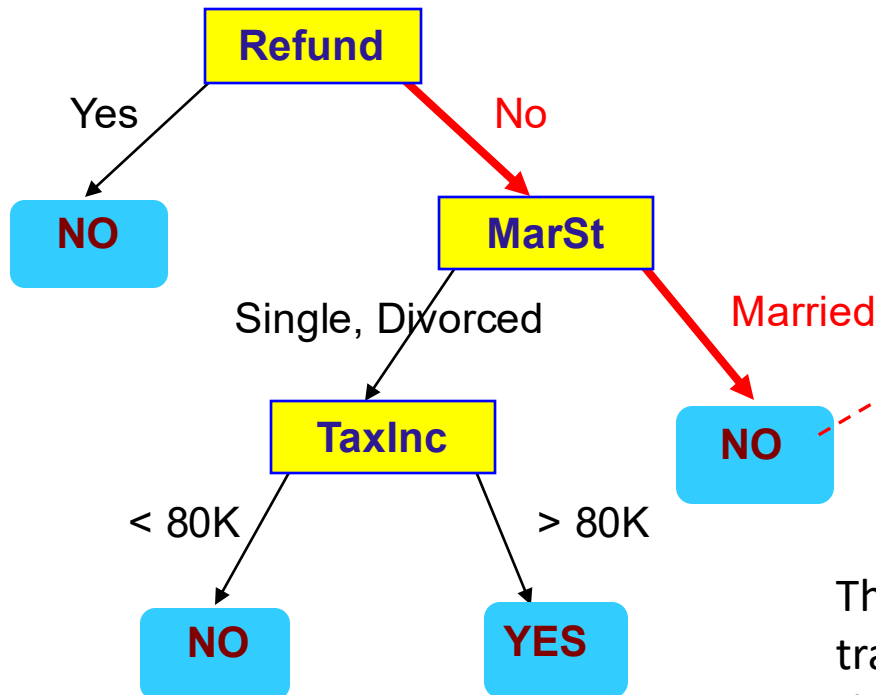
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Follow the corresponding pointer to the leaf node.

Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



The reached leaf node only contains training records with Cheat = No. Hence the tree will return Cheat = No as the predicted class.

Parallel Decision Trees

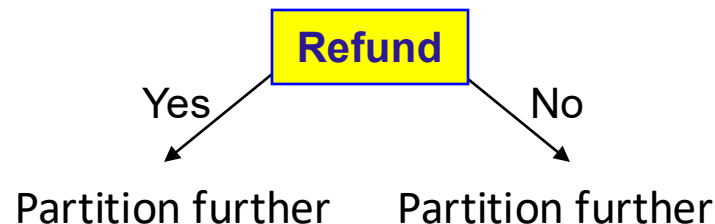
- For classification and prediction techniques in general, and trees in particular, both induction (model training) and deduction (model use) are potential targets for parallelization.
- Due to their regular structure, trees are easier to parallelize than many other classification techniques.

Parallel Decision Tree Induction

- Tree induction starts with all training records at the root, recursively partitioning the data until a stopping condition is met. For each node, the **split predicate**, i.e., the condition determining how to partition the data, is selected based on how well it 'purified' the partitions.
- Typical stopping conditions are the node is pure, no valid/useful split remains, or the partition is too small to split reliably.

Parallel Decision Tree Induction

- Trees offer two obvious opportunities for parallel training:
 - (1) A tree node partitions the data space, e.g., records with Refund = Yes go one way, those with Refund = No another. The split decisions in one subtree are independent of those in the other, therefore they can be performed in parallel without communication.



Parallel Decision Tree Induction

- Trees offer two obvious opportunities for parallel training:
 - (2) To find a node's split predicate, each attribute and its possible split values are explored. The score for each of these candidate predicates can be computed independently, but afterward coordination is required to select the split predicate with the highest score.

// This program assumes that each node performs a *binary* split into “left” and “right” partition

```
map( nodeID, listOfRecords ) {  
    leftPartition = {}; rightPartition = {}
```

```
    // Find best split predicate for the node
```

```
    for each attribute A
```

```
        for each possible split point S
```

```
            computeScore( listOfRecords, A, S )
```

```
            Keep track of the (A, S) pair with the highest score
```

```
    // Let (bestA, bestS) be the winning split predicate. Partition the data according to the split predicate.
```

```
    for each record r in listOfRecords
```

```
        if satisfiesPredicate( r , (bestA, bestS) )
```

```
            leftPartition.add( r )
```

```
        else
```

```
            rightPartition.add( r )
```

```
    // Write the node information containing nodeID, split predicate, and child node IDs to file
```

```
    treeFile.write( nodeID, (bestA, bestS), newID(nodeID, left), newID(nodeID, right) )
```

```
    // Emit the two partitions for further splitting in the next iteration
```

```
    if not stoppingConditionMet( leftPartition )
```

```
        emit( newID(nodeID, left), leftPartition )
```

```
    if not stoppingConditionMet( rightPartition )
```

```
        emit( newID(nodeID, right), rightPartition )
```

```
}
```

This algorithm parallelizes computation by training different subtrees concurrently.

MapReduce Program Discussion

- The program uses a breadth-first approach to train the tree layer-by-layer in each iteration.
- Initially the input is the root-node ID and the entire training set. A single Map function call finds the split predicate for the root and partitions the data accordingly. In the next iteration, there are two Map calls—one for each child node of the root. And so on.

MapReduce Program Discussion

- As the computation proceeds, initially the number of different partitions increases, resulting in greater possible parallelism.
- At some point splitting in some branches ends as the stopping condition is met. Hence at some point there will be fewer Map function calls as the tree grows deeper.
- In the beginning each iteration reads the entire input data and writes out the re-partitioned input. As nodes meet the stopping condition, their data records will not be transferred any more, slowly decreasing data transfer as the computation winds down.

Improvements

- In the Map function, finding the best split predicate involves repeated reading of the input records—at least once for each attribute. This will be expensive when the data does not fit in memory.
- To reduce cost, one can work with a **random sample**, which usually provides a good approximate score and hence results in splits of similar quality.

Improvements

- For data sets with a huge number of attributes, one should consider parallelizing the process of finding the best split attribute for a node.

Improvements

- However, if different split attributes for the same node are explored in different Map tasks, then Reduce will be needed to find the winner and perform the data partitioning.
- Once the training data at a node fit in memory, the entire subtree could be trained in the map function.

Counting for Split Finding

- To find the best split point for attribute A, the data at the current subtree root is sorted on A. then all possible “middle” points between consecutive A-values are explored to find the one with the highest score.
 - The score is determined by a purity measure

Training data

A	B	Label
2	50	+
5	10	-
1	30	+
3	20	-

Counting for Split Finding

- To find the best split point for attribute A, the data at the current subtree root is sorted on A. then all possible “middle” points between consecutive A-values are explored to find the one with the highest score.
 - The score is determined by a purity measure

Training data

A	B	Label
2	50	+
5	10	-
1	30	+
3	20	-

Training data sorted on A

1	2	3	5
+	+	-	-

Counting for Split Finding

- To find the best split point for attribute A, the data at the current subtree root is sorted on A. then all possible “middle” points between consecutive A-values are explored to find the one with the highest score.
 - The score is determined by a purity measure

Training data

A	B	Label
2	50	+
5	10	-
1	30	+
3	20	-

Training data sorted on A

1	2	3	5
+	+	-	-

Split point candidate	1.5
Number of + and - cases going left	+: 1 -: 0
Number of + and - cases going right	+: 1 -: 2

Counting for Split Finding

- To find the best split point for attribute A, the data at the current subtree root is sorted on A. then all possible “middle” points between consecutive A-values are explored to find the one with the highest score.
 - The score is determined by a purity measure

Training data

A	B	Label
2	50	+
5	10	-
1	30	+
3	20	-

Training data sorted on A

1	2	3	5
+	+	-	-

Split point candidate	1.5	2.5
Number of + and - cases going left	+: 1 -: 0	+: 2 -: 0
Number of + and - cases going right	+: 1 -: 2	+: 0 -: 2

Counting for Split Finding

- To find the best split point for attribute A, the data at the current subtree root is sorted on A. then all possible “middle” points between consecutive A-values are explored to find the one with the highest score.
 - The score is determined by a purity measure

Training data

A	B	Label
2	50	+
5	10	-
1	30	+
3	20	-

Training data sorted on A

1	2	3	5
+	+	-	-

Split point candidate	1.5	2.5	4
Number of + and - cases going left	+: 1 -: 0	+: 2 -: 0	+: 2 -: 1
Number of + and - cases going right	+: 1 -: 2	+: 0 -: 2	+: 0 -: 1

Counting for Split Finding

- To find the best split point for attribute A, the data at the current subtree root is sorted on A. then all possible “middle” points between consecutive A-values are explored to find the one with the highest score.
 - The score is determined by a purity measure

Training data

A	B	Label
2	50	+
5	10	-
1	30	+
3	20	-

Split point 2.5 is the winner, because it perfectly separates classes + and -.

Training data sorted on A

1	2	3	5
+	+	-	-

Split point candidate	1.5	2.5	4
Number of + and - cases going left	+: 1 -: 0	+: 2 -: 0	+: 2 -: 1
Number of + and minus cases going right	+: 1 -: 2	+: 0 -: 2	+: 0 -: 1

Parallel Counting for Split Finding

- Can we count the number of + and – cases going left versus right for all possible split points in **parallel**?
- We do not want to copy all data records to all tasks. However, if a task only receives some of the records, then it cannot even determine possible split points.

Parallel Counting for Split Finding

- In the example, assume task 0 only receives the records with A-values 1 and 3. Then the middle point would be 2, not 1.5. Intuitively, task 0 cannot determine the correct split candidates, because it is missing some of the data.
- How could we solve this?

Parallel Counting with Predefined Split Points

- What if each task knew all possible split points from the beginning? If all tasks count the left and right class distribution for the same split candidates, then it is easy to find the total counts.
 - In the example, assume each task was told to check split points 1.5 and 3.5.

Round 1, task 0

Training data partition	A	B	Label
	2	50	+
	5	10	-

Round 1, task 1

Training data partition	A	B	Label
	1	30	+
	3	20	-

Parallel Counting with Predefined Split Points

- What if each task knew all possible split points from the beginning? If all tasks count the left and right class distribution for the same split candidates, then it is easy to find the total counts.
 - In the example, assume each task was told to check split points 1.5 and 3.5.

Round 1, task 0

Training data partition	A	B	Label
	2	50	+
	5	10	-

Output:

1.5: left(+: 0, -: 0), right(+: 1, -: 1)

3.5: left(+: 1, -: 0), right(+: 0, -: 1)

Round 1, task 1

Training data partition	A	B	Label
	1	30	+
	3	20	-

Parallel Counting with Predefined Split Points

- What if each task knew all possible split points from the beginning? If all tasks count the left and right class distribution for the same split candidates, then it is easy to find the total counts.
 - In the example, assume each task was told to check split points 1.5 and 3.5.

Round 1, task 0

Training data partition	A	B	Label
	2	50	+
	5	10	-

Output:

1.5: left(+: 0, -: 0), right(+: 1, -: 1)

3.5: left(+: 1, -: 0), right(+: 0, -: 1)

Round 1, task 1

Training data partition	A	B	Label
	1	30	+
	3	20	-

Output:

1.5: left(+: 1, -: 0), right(+: 0, -: 1)

3.5: left(+: 1, -: 1), right(+: 0, -: 0)

Parallel Counting with Predefined Split Points

- What if each task knew all possible split points from the beginning? If all tasks count the left and right class distribution for the same split candidates, then it is easy to find the total counts.
 - In the example, assume each task was told to check split points 1.5 and 3.5.

Round 1, task 0

Training data partition	A	B	Label
	2	50	+
	5	10	-

Output:

1.5: left(+: 0, -: 0), right(+: 1, -: 1)

3.5: left(+: 1, -: 0), right(+: 0, -: 1)

Round 1, task 1

Training data partition	A	B	Label
	1	30	+
	3	20	-

Output:

1.5: left(+: 1, -: 0), right(+: 0, -: 1)

3.5: left(+: 1, -: 1), right(+: 0, -: 0)

Round 2 then groups by split point and aggregates the counts as:

1.5: left(+: 1, -: 0), right(+: 1, -: 2)

3.5: left(+: 2, -: 1), right(+: 0, -: 1)

How Well Does This Work?

- Data can be partitioned in any way, without duplication, and both round 1 and round 2 parallelize well.
- However, if round 2 has multiple tasks, then a third round is needed to find the split with the best score.

How Well Does This Work?

- How do we **select the split-point candidates** for an attribute A?
 - We can pick evenly-spaced values from A's domain. More candidates result in higher computation cost but increase the probability of finding a good split point. In the example, the optimal split is missed, because no candidate between 3 and 5 was tried.
 - To not miss the optimal splits, we could sort the input on A and use all middle values between consecutive A-values. This adds extra sort cost and often produces too many candidates, resulting in high cost during the parallel counting phase.

Parallel Decision Tree Deduction

- For a single test record, parallelizing deduction will usually not be efficient. Even for large trees, it would be faster to access nodes in a file instead of running multiple MapReduce iterations.
- Parallel deduction becomes viable if predictions are made for a huge set of test records. The tree model can broadcast to all worker machines; then each Map call independently computes the prediction for a test record.

```
// The file with the decision tree is made available through the file cache
Class Mapper {
    tree

    setup()
        tree = load tree from file cache

    map( test record r ) {
        output = tree.makePrediction( r )
        emit( r, output)
    }
}
```

Summary

- Data mining techniques are crucial tools for discovering hidden patterns in big data. Unfortunately, many techniques are complex.
 - Available implementations often are highly optimized for centralized in-memory processing.
- For some of the simpler techniques such as K-means clustering and decision trees, parallel implementations can be designed with reasonable effort.
- As a fallback, when the input data is too big to fit in memory, create an in-memory sample and then apply an existing centralized implementation. While not perfect, this often gives a good approximation of the desired results.
 - And it can be parallelized easily by exploring different samples and/or model parameter settings on different worker machines.

References

- Data mining textbook: Jiawei Han, Micheline Kamber, and Jian Pei. Data Mining: Concepts and Techniques, 3rd edition, Morgan Kaufmann, 2011
- Biswanath Panda and Joshua S. Herbach and Sugato Basu and Roberto J. Bayardo. PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce. Proc. Int. Conf. on Very Large Data Bases (VLDB), 2009
 - https://scholar.google.com/scholar?cluster=11753975382054642310&hl=en&as_sdt=0,22

Exam Practice Time!

Exam Practice Time!

Boston, also known as “Titletown”, is home to some of the most successful sports teams in the US. However, when Tom Brady, David Ortiz, and Jayson Tatum advocate for it, they feel disrespected by the New York media who claim that New York has won more titles. Luckily, our 3 friends know that you are taking CS 6240, and need your Spark savvy to help determine the “winningest city”.

You are given three pair RDDs: *Champs* with key champID and value object (year, teamID, sport) reports for each championship the year, winning team, and sport, e.g., (NBA24, (2024, Celtics, basketball)). *Team* has key teamID and value object (name, stadium, capacity, sport), e.g., (Celtics, (Boston Celtics, TD Garden, 20K, basketball)). And *City* has the city name as key and the list of nearby stadiums as the value, e.g., (Boston, (Fenway Park, TD Garden, Gillette Stadium)).