

Data Mining: Clustering and Prediction

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

- What is data mining? It could 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.
- 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. [source: Jiawei Han, Micheline Kamber, and Jian Pei. Data Mining: Concepts and Techniques, 3rd edition, Morgan Kaufmann, 2011]

Example Applications

- Classification, prediction
 - Will a customer's review of a product be affected by the earlier reviews?
 - Is the incoming credit card transaction legitimate or fraudulent?
 - What is the probability of observing bird species X, given time, location, weather, climate, human population features, and habitat features?
 - Identify sub-atomic particles from measurements of energy levels in a collider.
 - Predict the income from an ad shown by a search engine for a given keyword search query.
 - Is this email spam or not?
 - How likely is this bank customer to repay the mortgage?
- Clustering
 - What are the key customer groups and what characterizes each group?
 - What are the main categories of users of an online game?
 - Identify groups of viruses with genetic similarity, which are different from other groups of viruses.
- Graph mining
 - How does information spread in a social network?
 - Which people are likely to collaborate in the near future?
 - Which communities do users belong to?
- Association rules
 - What products do people tend to purchase together?

Typical Data Mining Steps: 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.
- For data mining and its results to be meaningful, the data miner has to understand the domain and the given data. This usually requires close collaboration with a domain expert.
 - Understanding the data requires knowing the attributes, their types, and meaning. This includes peculiarities like the encoding of missing data.
 - Summary statistics such as min, max, mean, standard deviation, and quantiles provide valuable insight about the data distribution.
 - Histograms summarize one- and multidimensional distributions, helping discover skew and correlations.
 - 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 have to be explored jointly, and which can be studied separately.

Typical Data Mining Steps: Data Preparation (Cont.)

- 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 is often needed to address 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 an attribute like age for which it is known that negative values are invalid.
 - Data often needs to be integrated from different sources. In addition to joining the data, this can require non-trivial entity identification. For example, an author might appear as Amy Smith, A. Smith, or Amy B. Smith in different publications.
 - Depending on the problem and data mining method, it might be necessary or beneficial to apply data reduction and transformation such as normalization, Principal Components Analysis, Wavelet transform, Fourier transform, or attribute removal.

Typical Data Mining Steps: Mining the Data

- After the data is properly prepared, 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 data. Model quality is evaluated on a separate test set. Then the model is used on new inputs to predict the desired output.
 - Popular techniques are decision tree, Random Forest, SVM, artificial neural network, Bayesian approaches, regression, and boosting.
 - Clustering algorithms such as K-means, hierarchical clustering, and density-based clustering are used to identify groups of similar entities that are different from other groups.
 - Frequent pattern mining is usually concerned with identifying frequent itemsets, association rules, and frequent sequences.

Typical Data Mining Steps: 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, there are many possible post-processing tasks:
 - 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. Data analysts need 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, and (3) compact rules explaining the predictions.
- 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.



Example

- We take a closer look at Prof. Riedewald's [Scolopax](#) system for analysis of big observational data.
- We applied it to a large high-dimensional data set containing reports of bird sightings in North America.
- MapReduce is used to train models for predicting the probability of observing a species given input features such as location, time, habitat properties, climate features, human population properties and so on.
- Users access it through a standard Web browser. Requests are essentially search queries for interesting patterns. These queries are also processed on a MapReduce cluster. Results are managed in HBase.

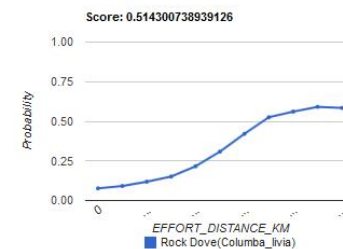
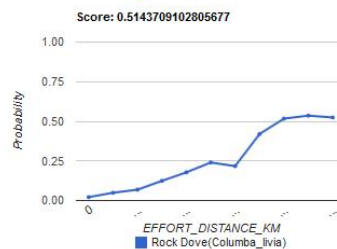
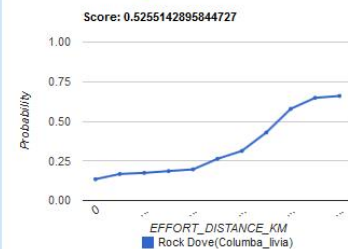
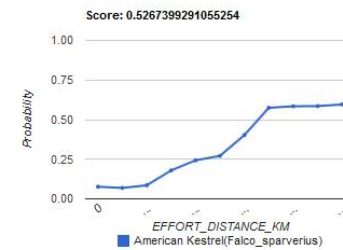
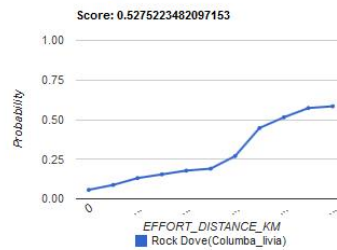
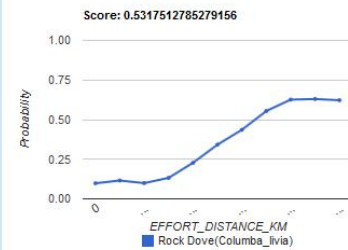
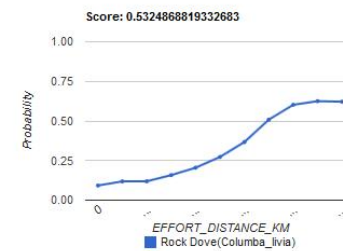
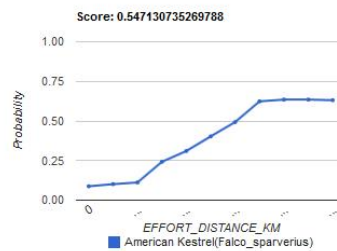
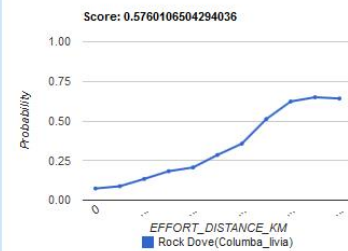
Strong Trend Search

- In the example on the next page, Scolopax was used to identify attributes that potentially have a **strong effect** on the species observation probability.
 - Each plot summarizes the effect of the attribute shown on the x-axis.
 - This effect was computed separately for different attributes, different species, and different geographical regions.
- The summaries are presented to the user ranked by the strength of the potential effect on the species observation probability. Since the plots are managed in HBase, the user can then interactively browse the result or filter based on properties such as the attribute, species, or region of interest.



► Settings

► Filter Results



Dynamic Pattern Discovery

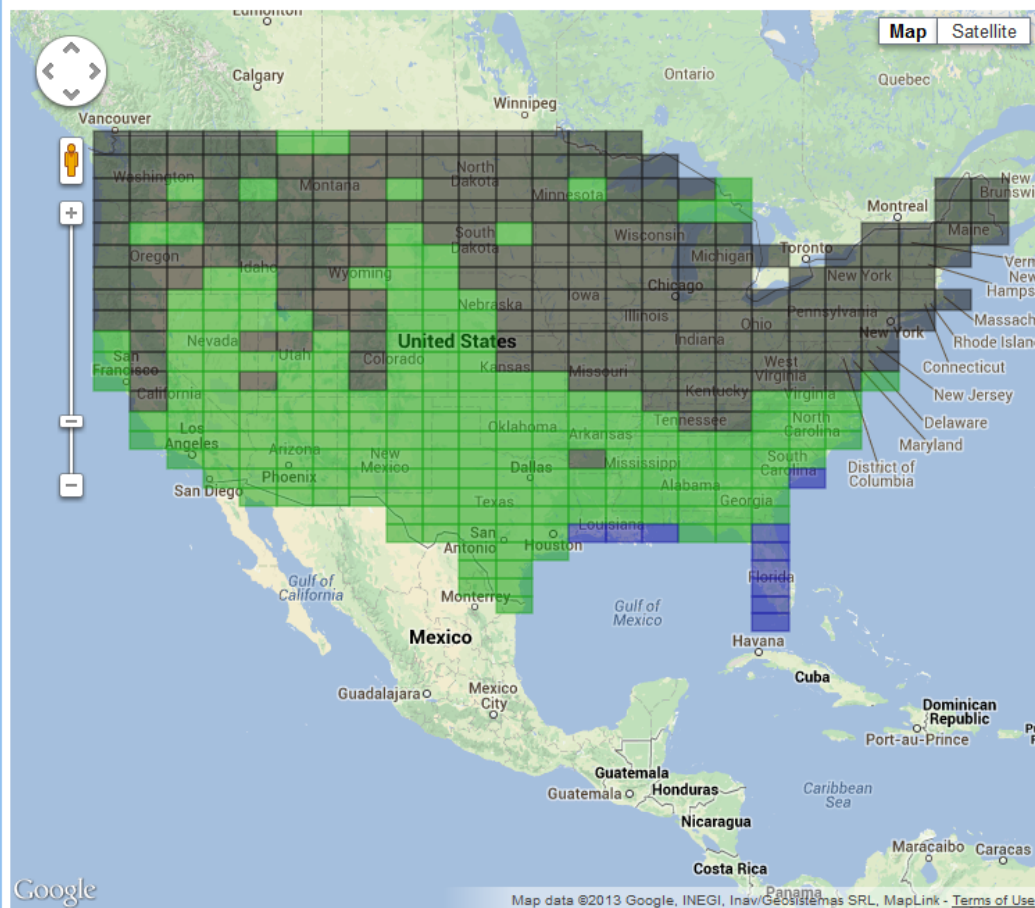
- In the next example, Scolopax also uses MapReduce to identify **clusters** of related summaries. Here clustering was applied to the annual trajectory of a species in different regions.
 - The trajectory is defined by the overall probability of observing the species in different months or weeks of the year.
- The clusters are purely based on trajectory similarity and do not take geographical location into account. Hence if the identified clusters still line up with large geographic regions, then the user can be confident that the pattern is likely caused by some biological process. In the example, the three clusters show the following likely migration behavior:
 - Purple: The tree swallow spends the winter in the southern US, in particular in Florida, the Gulf coast, and South Carolina.
 - Green: In spring and fall the tree swallow migrates north, crossing a horizontal band that ranges from California to the Carolinas.
 - Black: The tree swallow spends the summer in the northern US.

<< Previous

Next >>

Species: *Tachycineta_bicolor*

Summary Attribute: MONTH



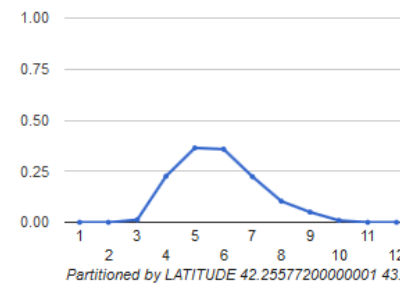
Predefined Queries

TreeSwallow_US25by25_3clusters
 EastWoodPewee_US25by25_3clusters
 EastBlue_EastWoodPewee_25by25_4cluster
 All_10by20_4clusters

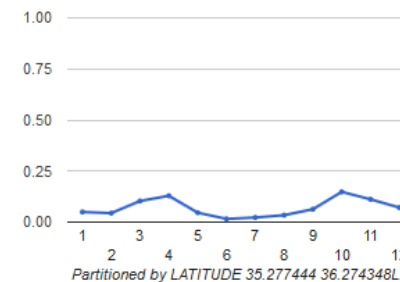
Add Query

Delete Query

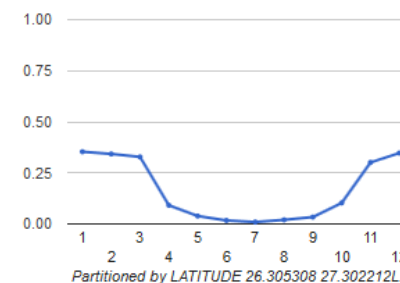
Plot



Plot



Plot



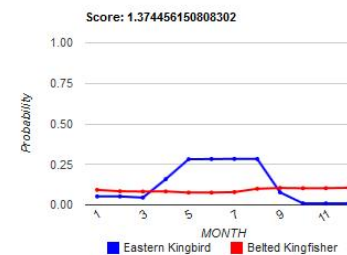
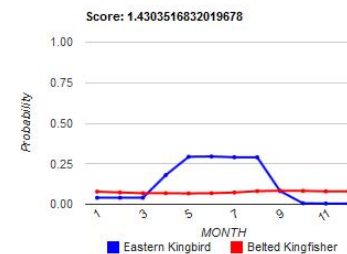
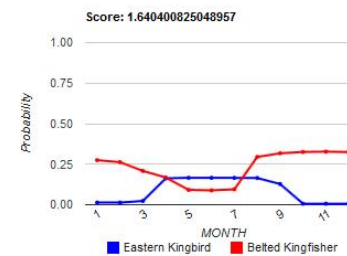
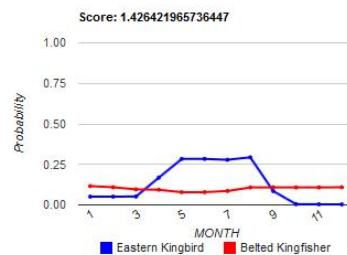
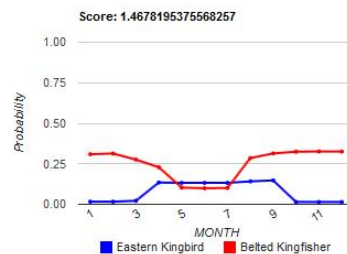
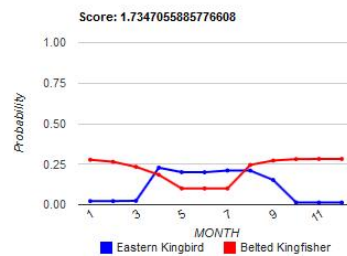
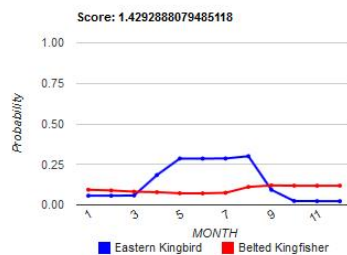
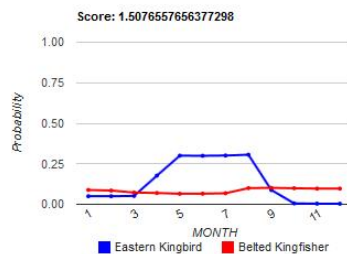
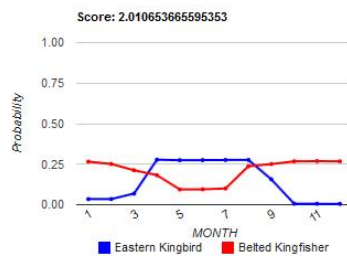
(Anti-) Correlation Search

- In the last example, Scolopax relies on MapReduce **join** algorithms developed by Prof. Riedewald's group to identify relationships between species.
- The example result is for a join query that searches for pairs of plots with the following properties: the plots are on different species (eastern kingbird and belted kingfisher in the example), but on the same attribute of interest (month in the example) and the same geographical region (the red box on the map highlights one of them for the example).
- The join results are then **ranked** based on the dissimilarity of the two plots.
 - Note the interesting opposite trends in several of the top-ranked results. Seeing such a result helps the ornithologist identify possible hypotheses about migration patterns or habitat competitions.



Settings

Filter Results



Important Note about Discovery

- Thanks to abundant data and powerful algorithms and hardware, it is now feasible to explore a huge space of possible patterns.
- This increases the risk of discovering **spurious** patterns, i.e., patterns that arose due to idiosyncrasies or coincidences in the data sample, but are not representative of the true distribution.
 - For example, if we measure the ratios between width and height of buildings in Boston, we may find one whose proportions almost exactly mirror the ratio of distances between some planets. However, it is unlikely that the architect was indeed influenced by such celestial relationships...
- Scolopax will deliver both real and spurious patterns. It is then up to the domain experts to either (1) apply statistical approaches to limit the probability of false discoveries or (2) collect new data samples specifically designed to verify or refute a hypothesis derived from a discovered pattern.

Parallel Data Mining

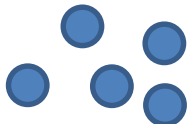
- Many mature and feature-rich data mining libraries and products are available. This includes the R system and the Weka open-source Java library. Unfortunately, most data mining solutions are not designed for execution in distributed systems. This often leaves only the following three options:
 1. Use an existing, but often limited, library of distributed data mining solutions, e.g., Apache Mahout or Spark ML.
 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 fairly complicated 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. Weka, which is open-source and written in Java, presents itself as an obvious choice for use in Hadoop.

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

We start with clustering, an unsupervised learning approach.

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 these steps depending on the cluster membership of an object of interest.
- 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.



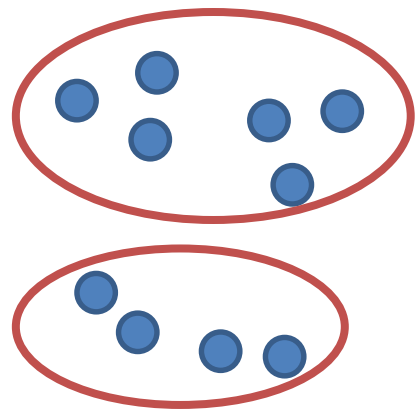
Three clusters based on the Euclidean distances between objects.

Examples of Clustering Applications

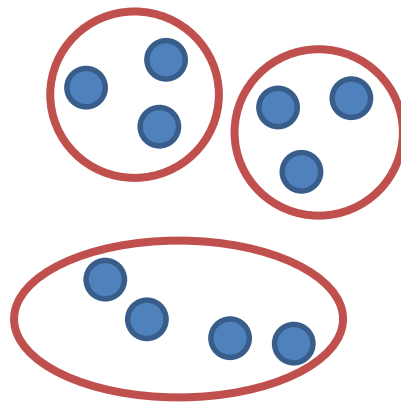
- Marketing: Discover distinct customer groups based on the products people purchase. Then use this knowledge for targeted marketing.
- Land use: Identify areas of similar land use in an earth observation database.
- City-planning: Identify groups of houses according to house type, value, and location
- Bird studies: Find species with similar migration behavior.

2.2 Ambiguity of Clusters

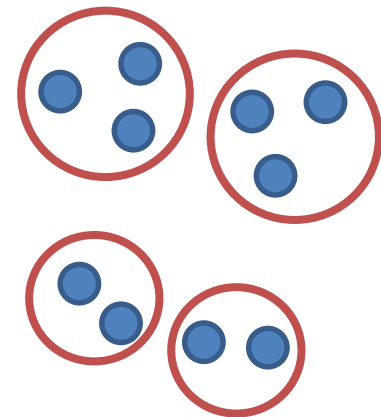
- The notion of what constitutes a cluster is subjective and depends on the preferences of the user.



2 clusters.



3 clusters



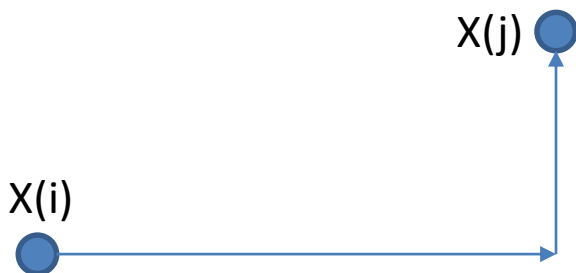
4 clusters

Distance and Similarity

- Clustering is inherently determined by the choice of **distance** or **similarity** measure.
 - Consider a data set of bird sighting reports. If the distance is measured based on latitude and longitude, then observations made in similar locations will be clustered together. The same data set could instead be clustered based on the weather on the observation day. Now a cluster might contain reports from far-away locations that had similar weather.
- 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

- 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
$$\sqrt[q]{|x_1(i) - x_1(j)|^q + |x_2(i) - x_2(j)|^q + \dots + |x_d(i) - x_d(j)|^q}$$
- For $q = 1$, it is called the *Manhattan distance*.
- For $q = 2$, it is called the *Euclidean distance*.



Intuition for 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 has to be chosen carefully. Several challenges often have to be addressed:
 - Curse of dimensionality
 - Diverse attribute domains
 - Categorical and ordinal attributes
- We discuss each challenge next.

Curse of Dimensionality

- The Minkowski distance between two objects is an aggregate of the differences in the individual dimensions. The more dimensions, the smaller the relative impact of an individual dimension. In particular, if there are many noisy attributes, they can bury the distance signal in the combined noise. Other distance measures suffer from similar issues.
 - To address this problem, remove any attribute that is known to be very noisy or not interesting.
- 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.
- This problem can be addressed by **scaling** and **weighting**.
 - **Scaling**: Differences in attribute domains can be addressed by normalizing each domain to $[0,1]$, using a linear transformation. Alternatively, a logarithmic transformation compresses differences between large values more than for small ones.
 - **Weighting**: Each attribute could be weighted according to its importance. Then the distance is defined as a weighted sum over the contributions in the individual dimensions.
- Choosing the appropriate transformation or weights requires expert knowledge and involves trial-and-error.

Categorical and Ordinal Attributes

- **Categorical attributes:** These attributes encode concepts that have no natural notion of **order** or **numerical difference**.
 - For categorical attributes, it is often difficult to choose the appropriate distance measure.
 - Consider hair color. What is a meaningful distance between brown and black, and how does it compare to the distance between red and blonde?
 - In some cases one can define specialized measures, e.g., based on genetic similarity for bird species.
 - In general, the difference for a categorical attribute could be measured by setting it to 0 if the values agree, and to 1 otherwise.
 - For example, two people with the same hair color have difference zero in the hair color dimension, while two people with different hair color have difference 1.
 - Alternatively, a categorical attribute can be transformed to a set of binary attributes. For each possible value of the categorical attribute, a corresponding binary attribute is created, which 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 possible 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, these attributes are set to blonde=1, red=0, brown=0, and black=0.

Categorical and Ordinal Attributes (Cont.)

- **Ordinal attributes:** Ordinal attributes have a natural notion of order, but not numerical difference. A typical example are ranks in the army and star-ratings for products. It is clear that a product rating of 4 stars is better than 3 stars, but it is not clear if a star is a meaningful measure of the difference.
 - 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 between ratings and hence needs to be approached with care.

Next we take a look at 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 one can implement from scratch for parallel execution.
- K-means has solid mathematical foundations, with a [well-defined optimization goal](#).
- In addition to a distance measure between objects, the user only 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 have to be transformed to numerical values.

K-means Algorithm

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

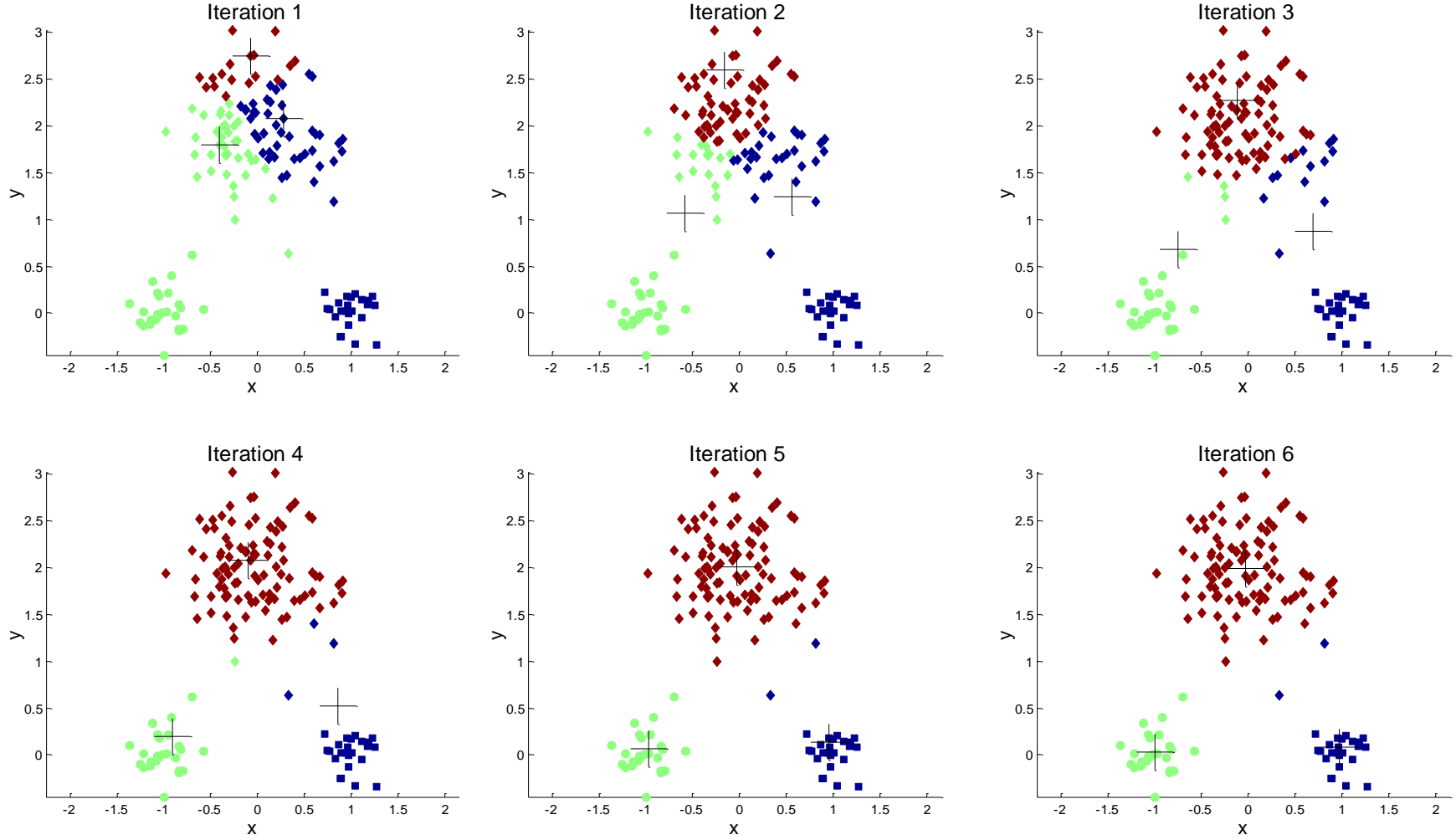
```
// Input: desired number of clusters K, data set D
K-means( K, D ) {

    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
            Recompute 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**.
 - The choice of initial centers affects the final clustering found. Unfortunately, there is no algorithm that is guaranteed to pick good initial centers that result in the best clustering possible. Many heuristics exist, e.g., some trying to place the initial centers far away from each other to improve the probability of each turning into the centroid of a different “natural” cluster.
 - A commonly used 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.
 - For vector spaces, the Euclidean distance is commonly used. Depending on the application one can choose other measures such as cosine similarity or a correlation coefficient.

Evaluating Cluster Quality

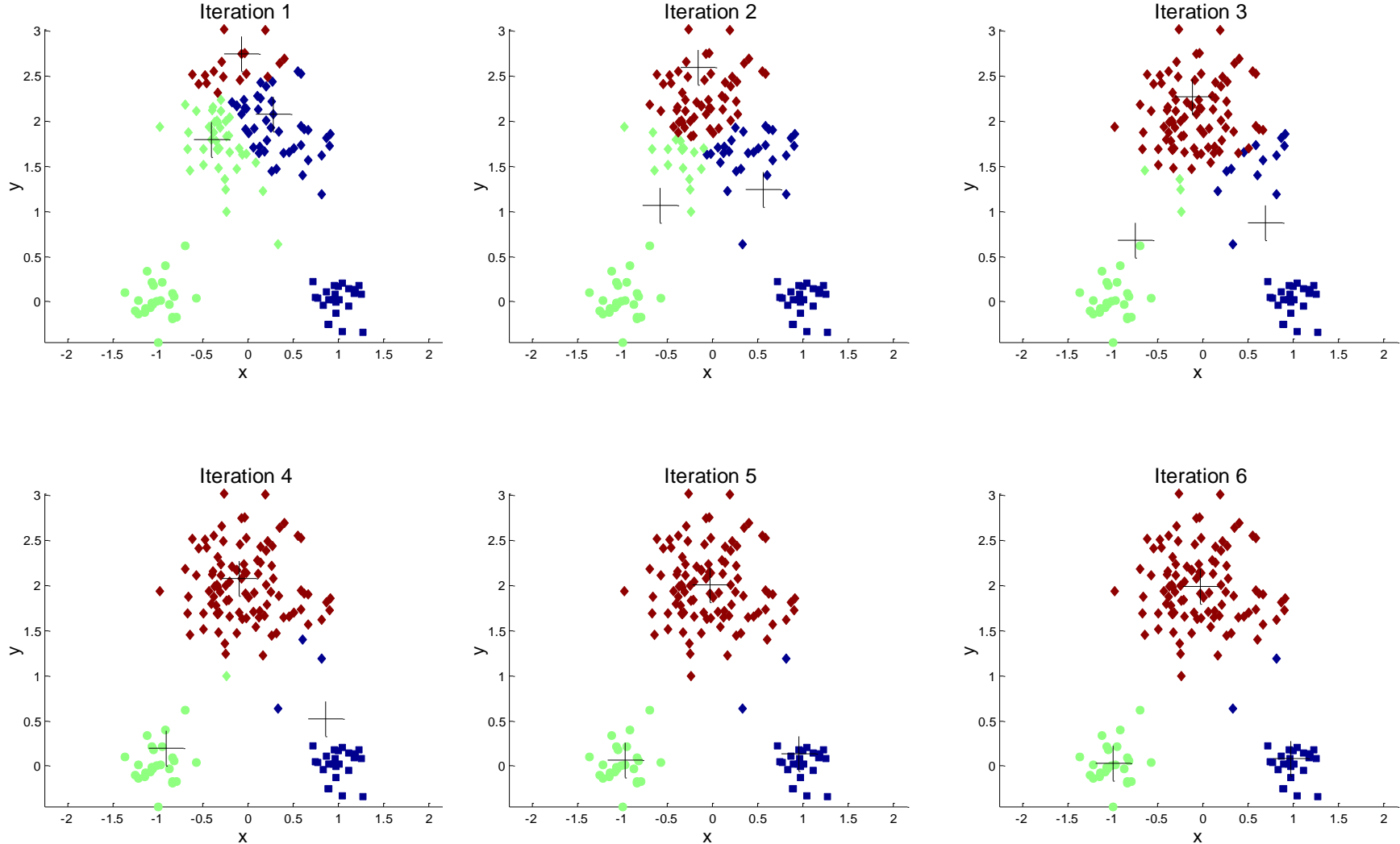
- For clustering as an unsupervised learning technique, there is no **ground truth** about clusters and cluster membership. How can we evaluate the quality of a clustering under these circumstances?
- In K-means, the centroid of a cluster represents all objects in the cluster. If an object is far from the centroid, then the centroid is a poor representative for it. Hence the quality of a K-means clustering is evaluated based on the “errors” made by representing individual objects with the corresponding cluster centroid. A common error measure in vector space is the **Sum of Squared Error (SSE)**, defined as $\sum_{i=1}^K \sum_{x \in C_i} \text{dist}^2(m_i, x)$. Here C_i denotes a cluster and m_i its centroid.
- The goal of K-means is to minimize SSE for a given K.
 - In general, a larger K will result in lower SSE. Hence it is not meaningful to use SSE for comparing clusterings with different K.

K-means Convergence

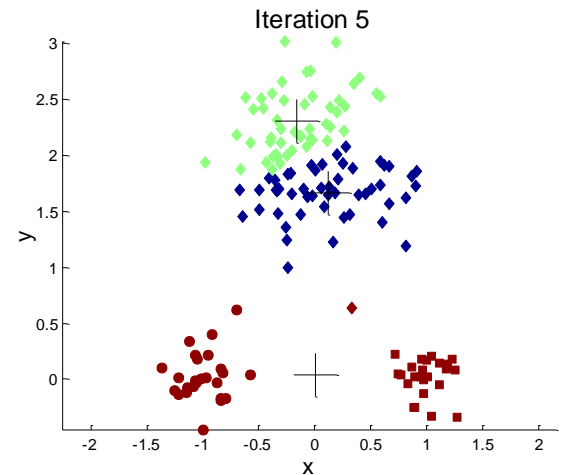
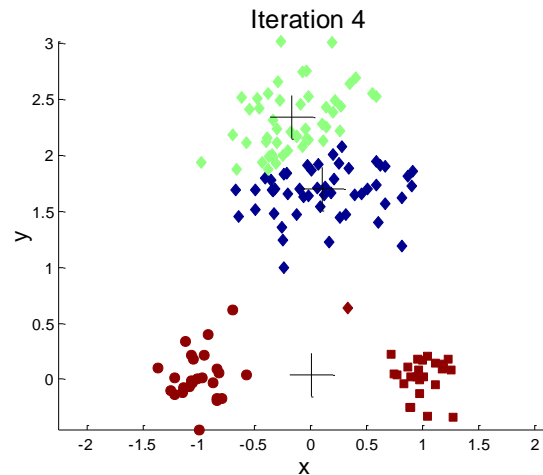
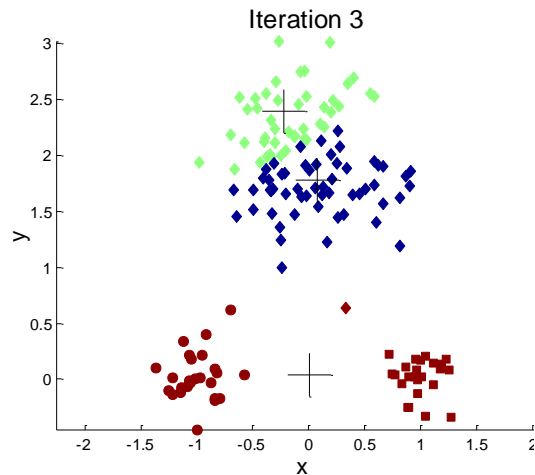
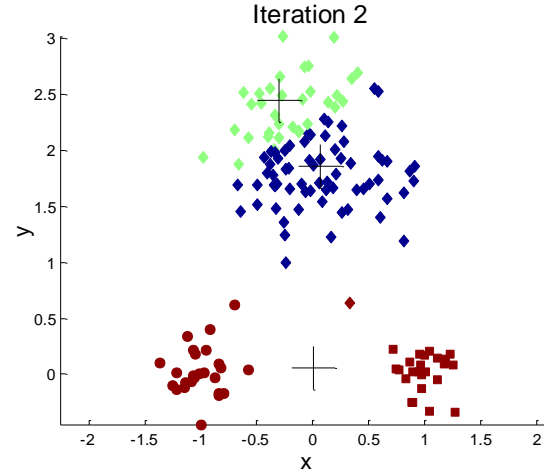
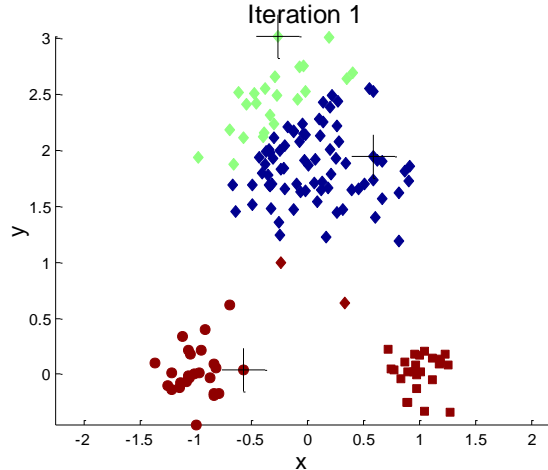
- K-means keeps iterating until the cluster centers stabilize. This raises two questions: (1) will SSE improve in each iteration and (2) will the algorithm always converge?
- Will SSE improve in each iteration: Yes, until convergence.
 - Given K centers, K-means assigns each object to the nearest center. If K-means re-assigns an object x from one cluster to another, that means x now has a smaller distance to its cluster center. Hence the corresponding squared term became smaller, reducing SSE.
 - Given K clusters, K-means updates the centroids. It is easy to show that the centroid has a smaller SSE for that cluster than any other possible center. Hence if any cluster centroid is updated, SSE must have decreased again.
- Will K-means converge: Yes.
 - There is a finite number of possible partitionings of n objects into K partitions.
 - 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.
- 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 located 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 fairly 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]

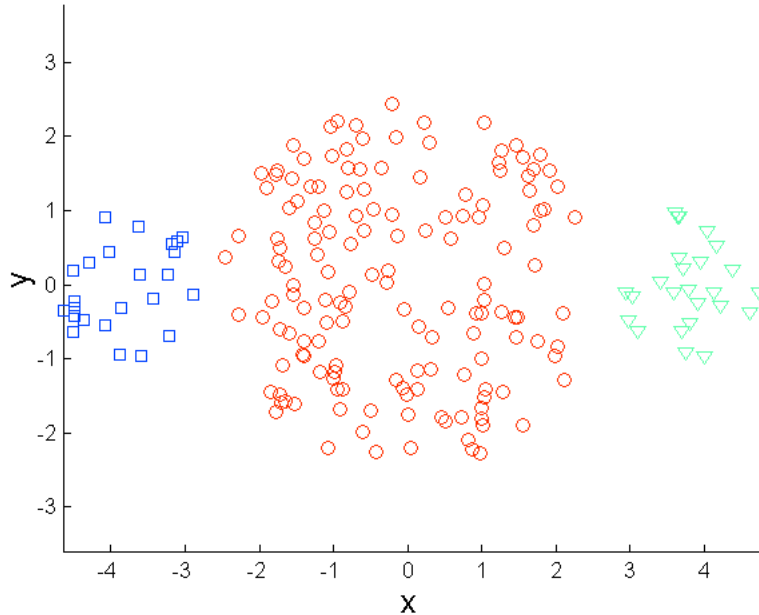
Selecting Initial Cluster Centers

- Since the choice of initial cluster centers determines to which local optimum K-means converges, researchers have explored how to choose a good initial configuration. Unfortunately, there is no general solution for this problem.
- 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 center far apart from each other. This is not guaranteed to work well either.
- Another option is to use a different clustering algorithm that does not need such an initial configuration, e.g., hierarchical clustering. The clusters found by such an algorithm could inform the choice of initial centers for K-means.
- Even if K-means is lucky in choosing “good” initial cluster centers in different natural clusters, it might converge to a “bad” clustering as will be discussed soon.

Limitations of K-means

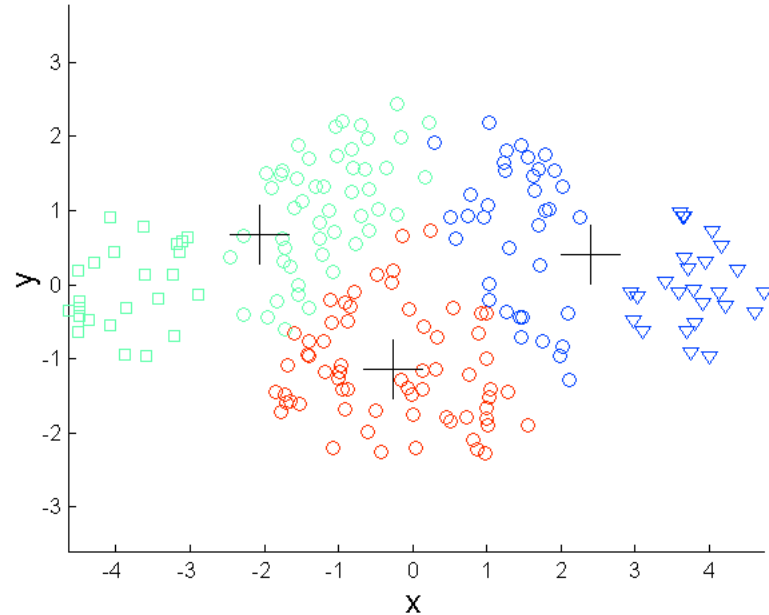
- Assume the user was able to choose the perfect value of K , the right distance measure, and the best initial cluster centers. Would this guarantee that a good clustering is found?
- Unfortunately, there are inherent limitations of K-means that prevent it from finding the “natural” clusters under certain circumstances.
- In particular, K-means has problems when clusters are of differing **sizes**, **densities**, or have **non-globular** shapes—as we discuss next.

Differing Sizes



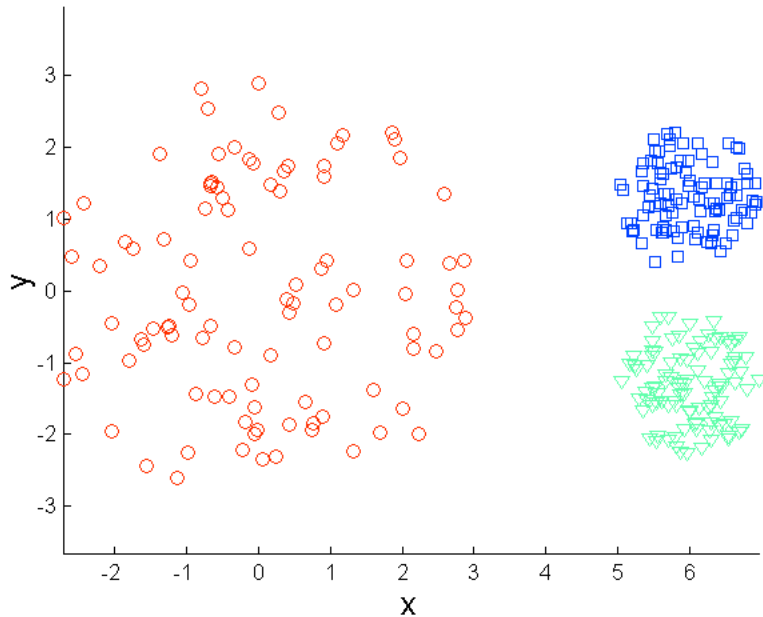
Natural clustering: Intuitively, many users would like to discover the three clusters indicated by point shape and color.

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



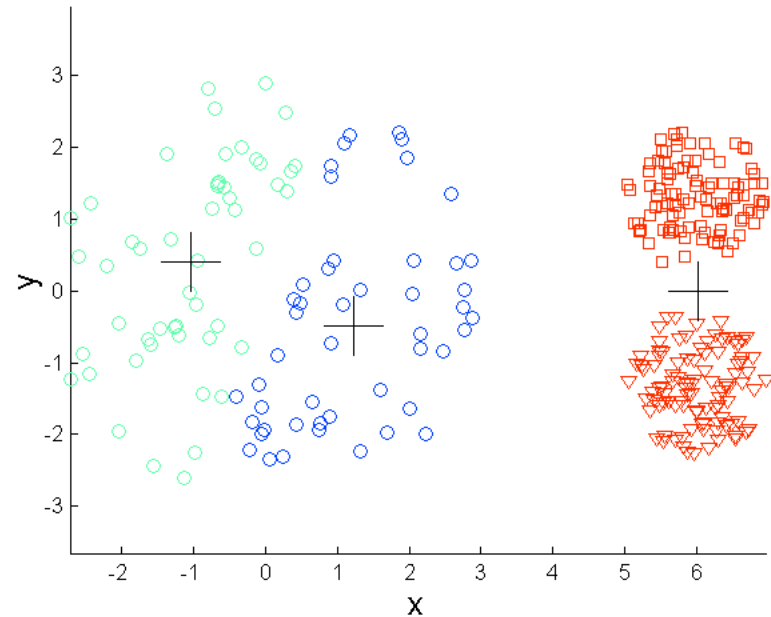
Optimal K-means result: The large natural cluster contains many more points than the small ones. Due to the different diameters of the natural clusters, K-means cannot find them. Even if it started with the perfect cluster centers, it would converge to a clustering as shown, because it minimizes SSE.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



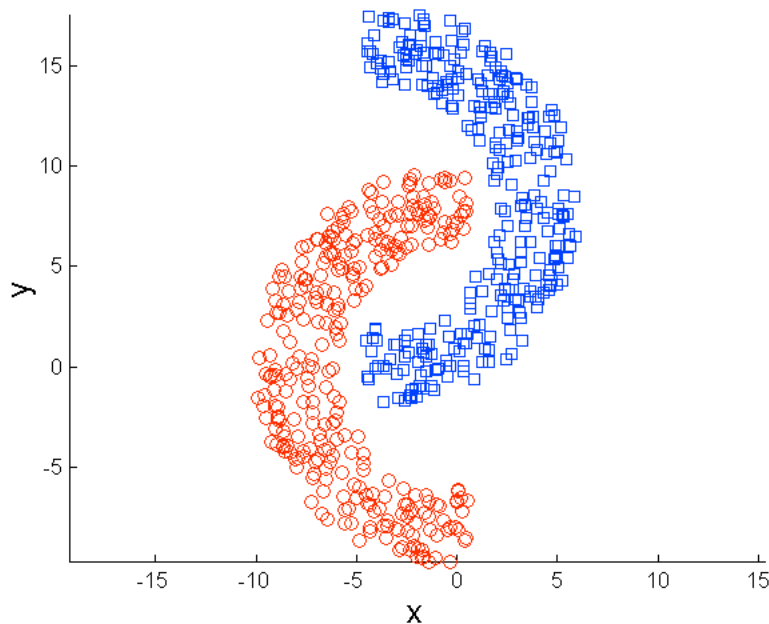
Natural clustering: Intuitively, many users would like to discover the three clusters indicated by point shape and color.

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



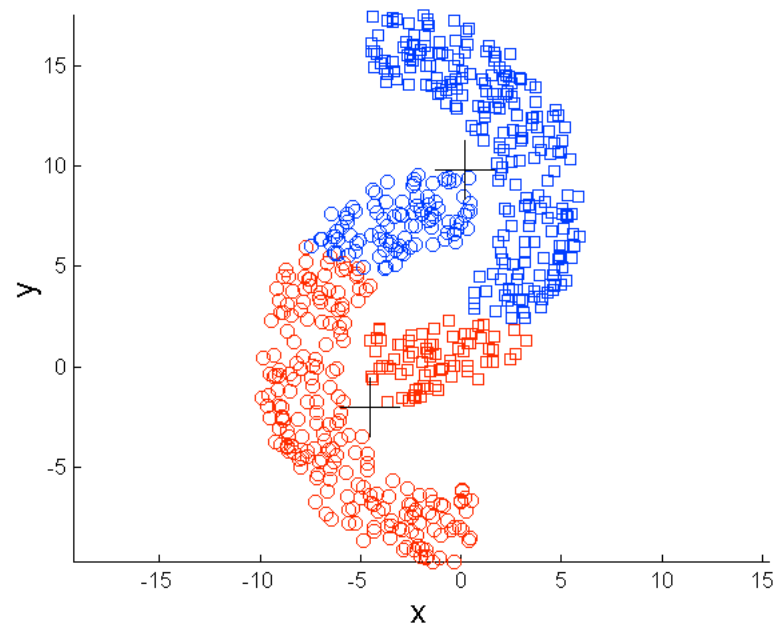
Optimal K-means result: In this example, each of the natural clusters has about the same number of points. However, due to their different densities, the large natural cluster has a much larger diameter than the other two. This results in the same problem as for the previous example with differing sizes.

2.10.3 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]

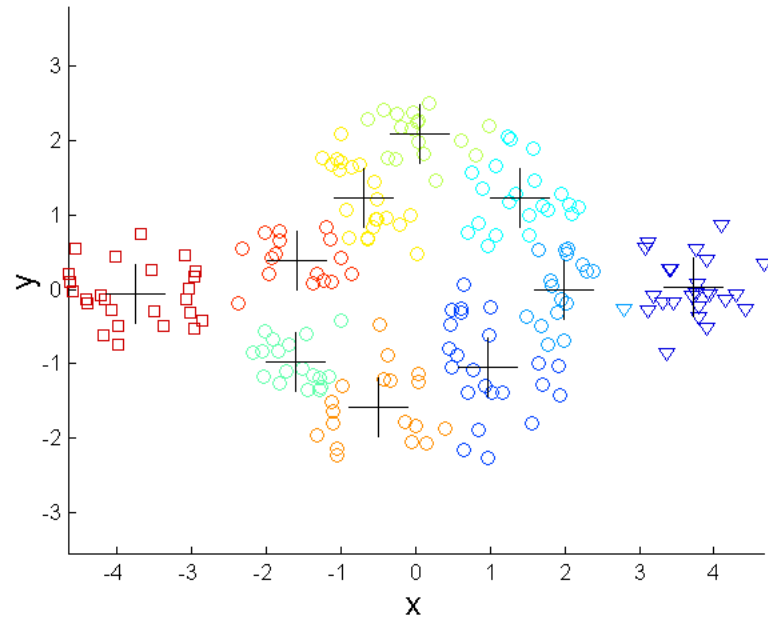
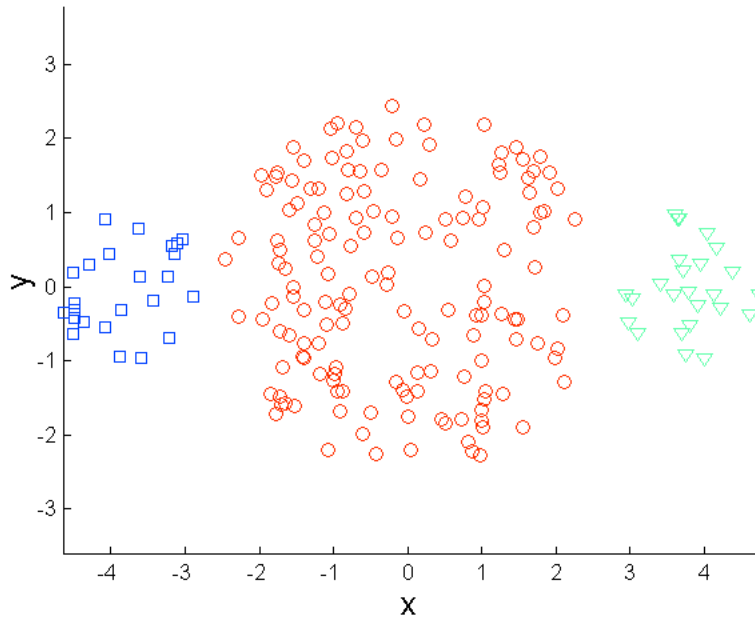


Optimal K-means result: Due to the elongated shape, some points of the left natural cluster are closer to the right natural cluster than to their own center. Again, even if K-means started with the perfect cluster centers, it would converge to a clustering as shown, because it minimizes SSE.

Addressing K-means Limitations

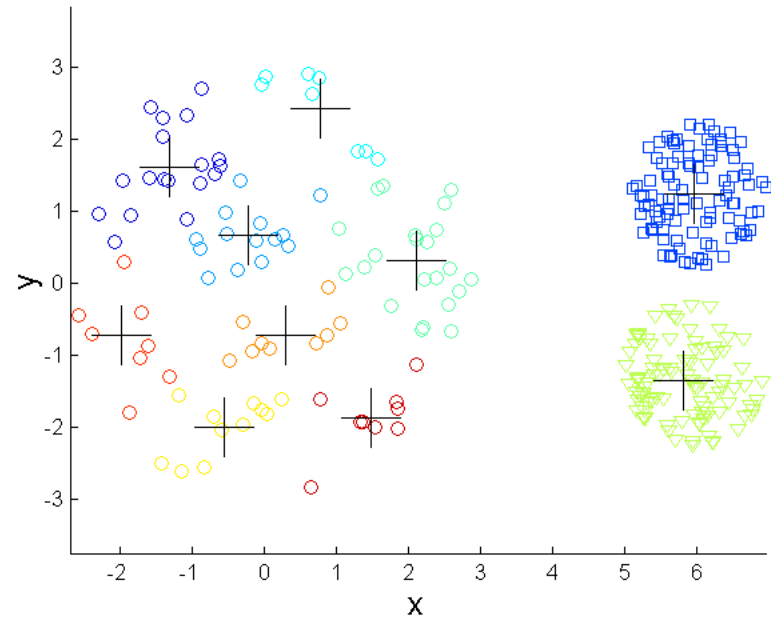
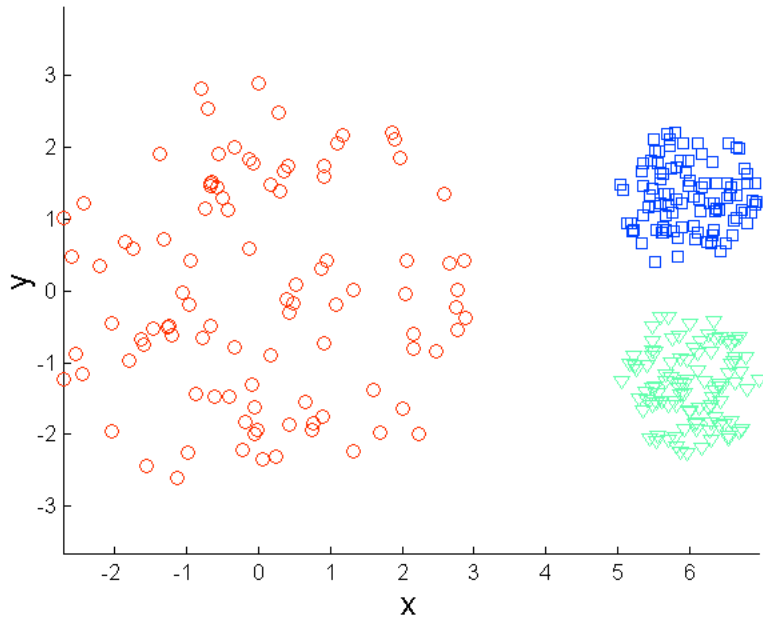
- There exist solutions for the inherent problems of K-means.
- Choose a different algorithm:
 - Each clustering algorithm is designed for a certain intuitive notion of a cluster. For instance, if the user is looking for clusters based on point density, not based on minimizing SSE, then she should consider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



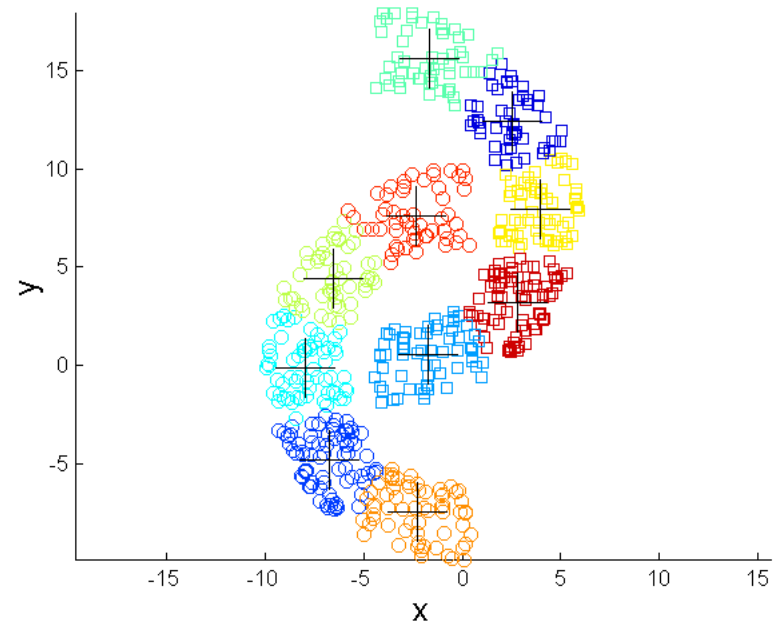
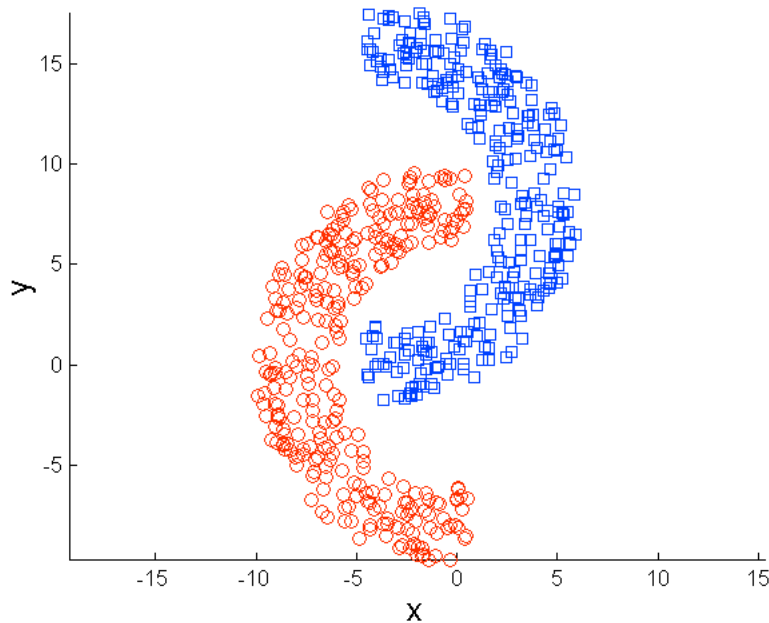
The K-means run for $K'=10$ found 10 small clusters, each part of one of the three natural clusters. An appropriate post-processing algorithm might be able to combine the 8 small clusters in the middle to form the large natural cluster.

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

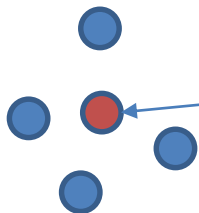


The K-means run for $K'=10$ found 10 small clusters, each part of one of the two natural clusters. An appropriate post-processing algorithm might be able to combine each group of five adjacent small clusters into the corresponding large natural cluster.

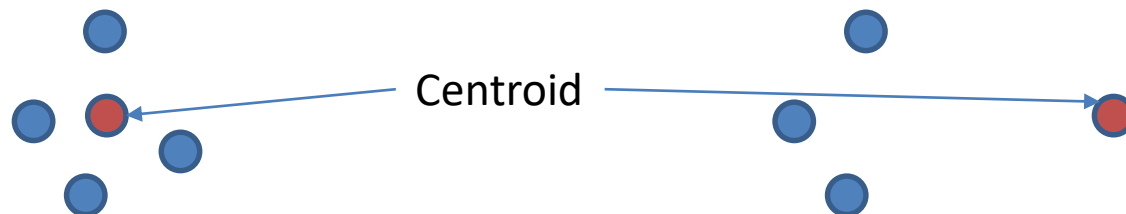
K-Means and Outliers

- 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.
- This problem is addressed by the **K-medoids** algorithm. Instead of the centroid, this algorithm represents a cluster by its medoid, which is the *most centrally located real object* in a cluster.
 - The algorithm is similar to K-means, but finding the medoid is computationally more expensive than finding the centroid. For an exact solution, the algorithm would try all objects in the cluster to find the one for which the cluster's SSE is minimized. To reduce cost, this can be done for a randomly selected subset.

Cluster without outliers



Cluster with an outlier



Distributed K-Means

- Similar to other iterative algorithms, e.g., for graph processing, we distinguish between static and evolving data. The former should be loaded once and then kept on the workers, while the latter potentially needs to be moved across the network.
 - For clustering, the set of input objects is fixed. Each task receives a partition of this data set.
 - The cluster centers change in each iteration. When centers change, membership of objects in clusters might also change.
- Which data structures do we need to represent the fixed and evolving data?
 - We store the 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, and keeps it for later iterations.
 - We broadcast \mathcal{C} enters to all tasks.
 - The task assigns each object in the partition to the closest center. Then this output is grouped by center ID.
- Round 2:
 - Each task receives a group of objects that were assigned to the same center. (It might receive multiple such groups.)
 - 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 distributed file cache.
 - Unfortunately Mappers have to read their chunk of Objects anew in every iteration.
- The driver program repeatedly calls this program until (almost-) convergence.

```
Class Mapper {  
    Centroids // Array containing the K cluster centers
```

```
    setup()  
        Centroids = read centroids from HDFS file
```

```
    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 sum and count 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. Like for the iterative graph analysis algorithms, performance suffers from MapReduce's inability to exploit the repetitive structure of the computation.
 - In fact, the MapReduce program executed on many machines can take much longer to finish than the sequential program.

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 the following:
 - Use `broadcast()` on the `Centers` file.
 - 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)`.
 - Aggregate this pair RDD or DataSet by `nearestCenterID`, emitting the average in each dimension.
 - Loop through the last 2 steps until convergence.
- Or we can just use the nice Spark machine learning library Spark MLlib.
 - Try to find the source code for K-means in `org.apache.spark.ml` (new version with DataSet) or `org.apache.spark.mllib` (old version with RDD).

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)
```

Alternative Parallel Version

- If computation on a single machine is faster than the corresponding MapReduce execution on many machines, consider the following alternative:
 - Usually the user does not know the best choice of parameter K. Hence in practice she would explore different values.
 - Since the initial choice of cluster centers has a significant impact on the clustering found, in practice many different initial configurations are explored.
- For each K and each initial configuration, the clustering can be computed independently. In MapReduce we can achieve this by creating a parameter file containing the different values of K in different lines. The same value of K may appear in several lines, indicating that multiple initial configurations are to be explored. Map reads such a line with parameters and computes the corresponding clustering locally.
 - Usually the parameter file will be smaller than the default file split size. To make sure that MapReduce creates multiple Map tasks, use [NLineInputFormat](#) to create small input splits based on lines in the file.
 - The file with the data objects would be made available to the Mappers through the distributed file cache. No further reading or transfer of the file is necessary.
- The pseudo-code is shown on the next page.

```
// The input file is copied to each worker using
// the distributed file cache
Class Mapper {
    data D

    setup() {
        D = read data from distributed file cache
    }

    map( ..., K ) {
        // Kmeans() is a sequential implementation
        // of the K-means algorithm
        clustering = Kmeans( D, K )

        emit( K, clustering )
    }
}
```

Now we change gears and take a 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 the credit card transaction is fraudulent.
 - Predict if the customer will purchase the product.
 - Predict the probability of observing a certain species in a given environment.
- Classification and prediction are **supervised** learning methods: They rely on the availability of **labeled** training data, i.e., records where both input and correct output are known.
- 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: Data

- Given a data set of recent graduates, NEU wants to predict for each graduating senior if s/he will receive a job offer within a year of graduation.

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 a model based on the relationships between input attributes (name, age, GPA, major) and output (job offer).
- Usually many different models can be fit to the same training data. Clearly, some are more realistic (Model 1) than others (Model 2). However, in practice it usually is not this obvious. Note that the model picks up correlations in the data. These are not necessarily causal relationships, e.g., the name of a person is not causing the person to get a job offer. This is purely a coincidence because of the given 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

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.
- Notice that 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 have been 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.

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

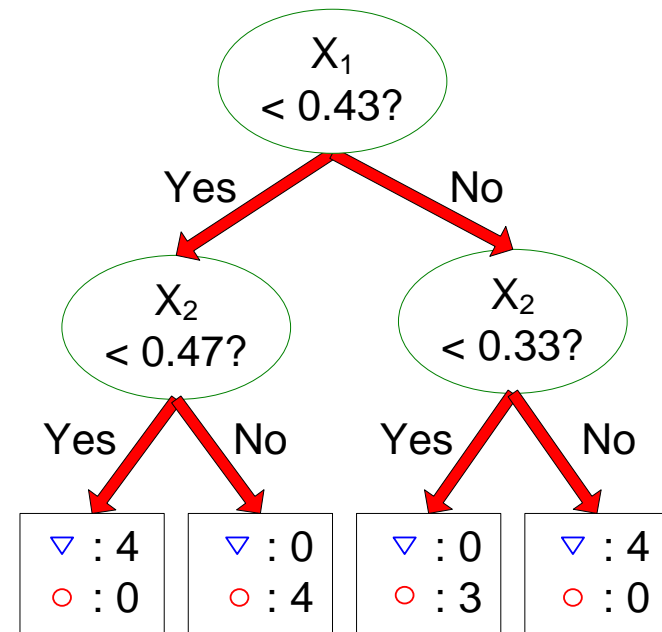
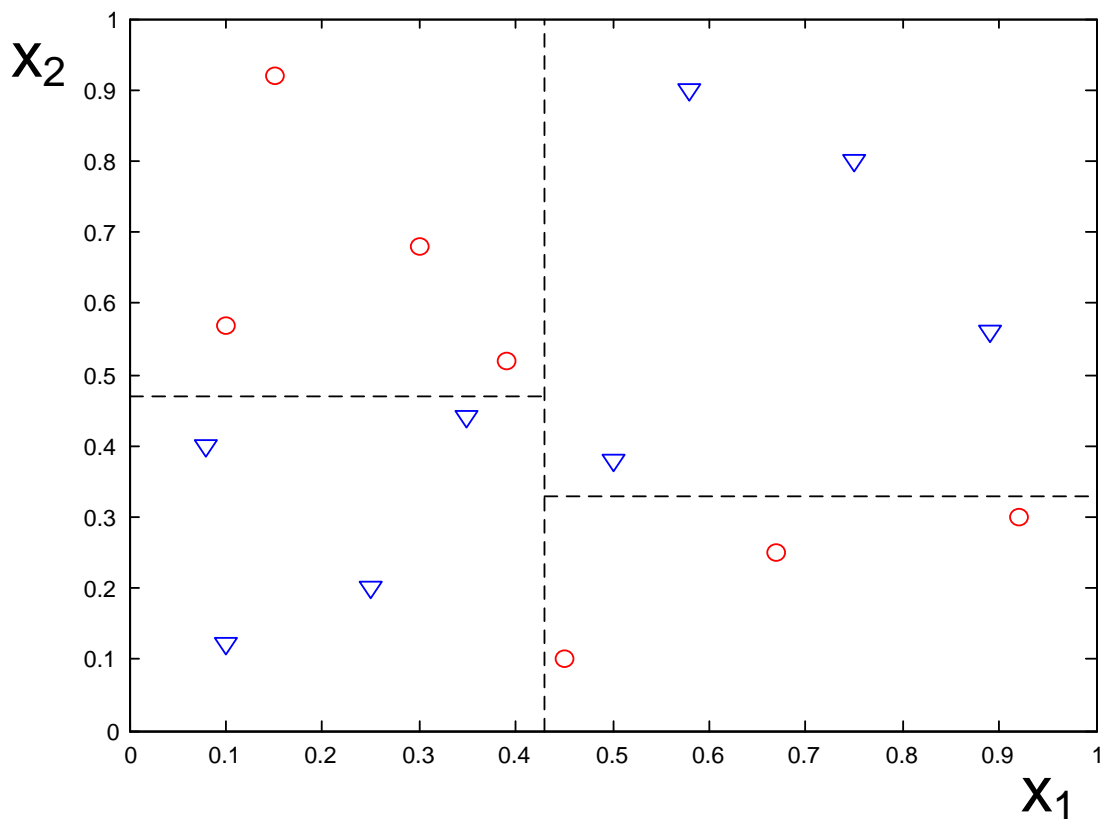
Model 2: IF (name = Joe OR name = Amy)
THEN job = Yes; ELSE job = No

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 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.

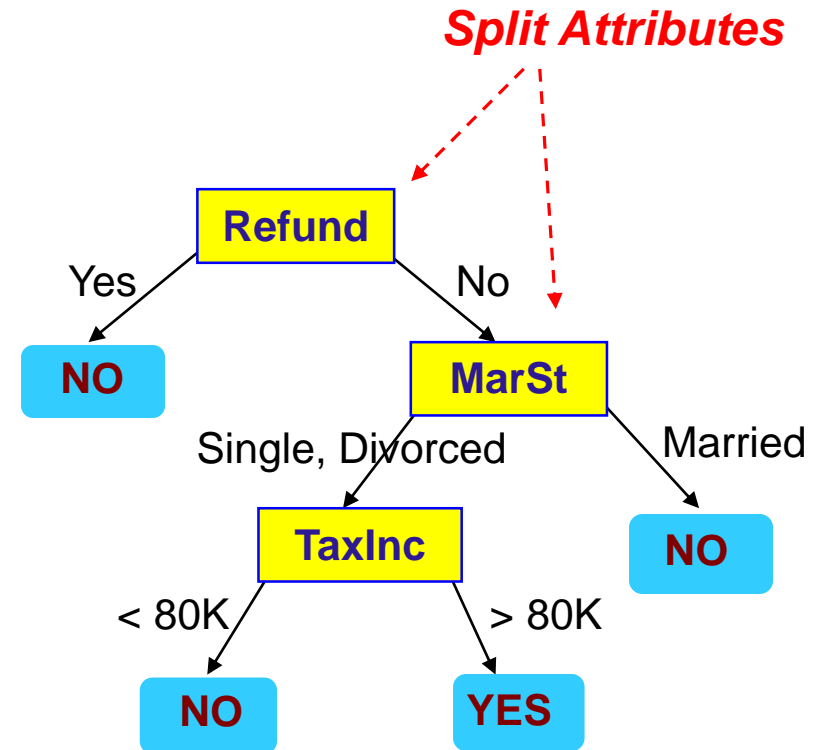
Tree Uniqueness

- For a given data set, there are usually many structurally different trees that achieve similar accuracy on the training data.
- In particular, for consistent data, there is more than one tree structure that perfectly represents the entire training data set.
- We look at two examples next.

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

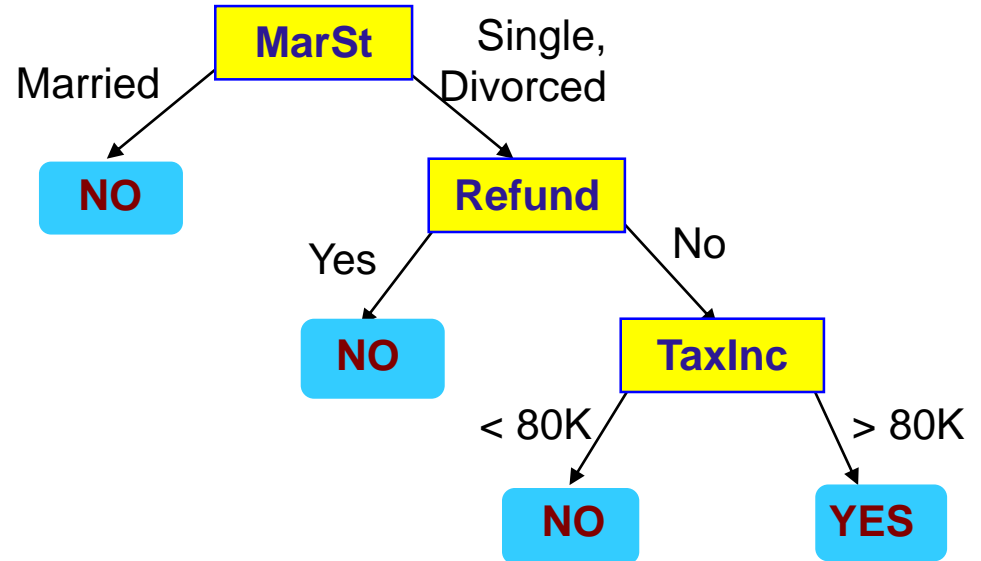
Training Data



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

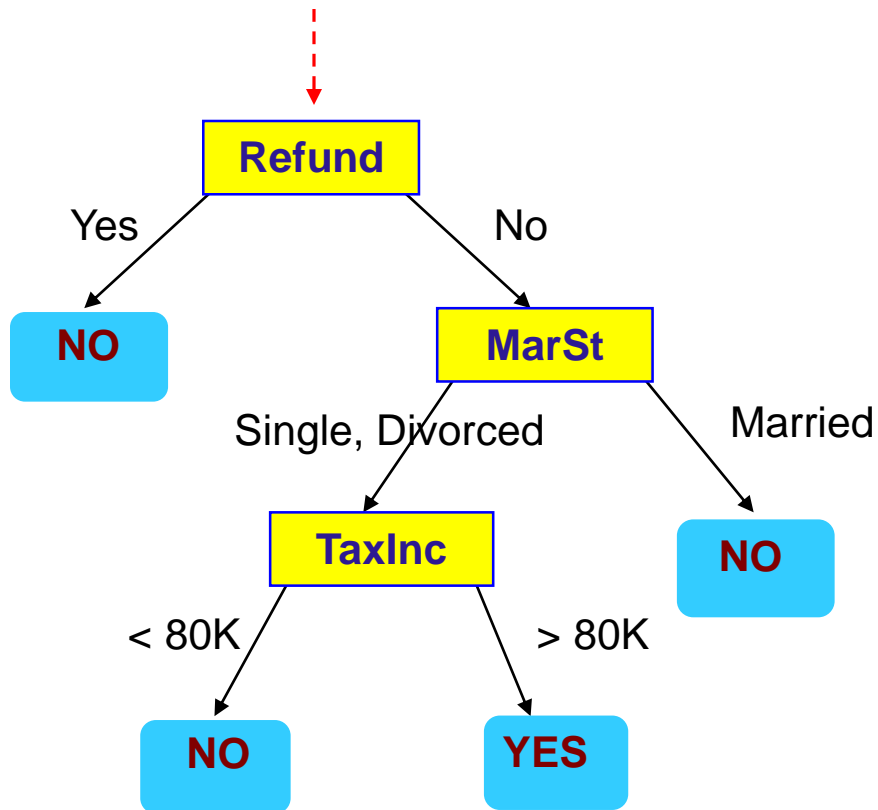


[Example source: "Introduction to Data Mining" by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]

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 this 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.
- Let us look at an example.

Start from the root of the tree



Test Data

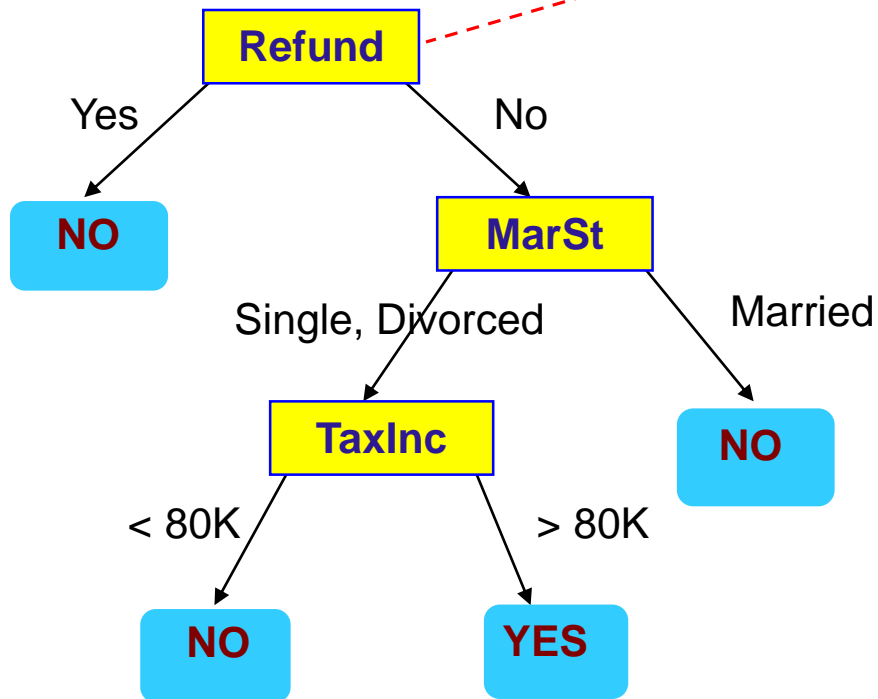
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

[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	?

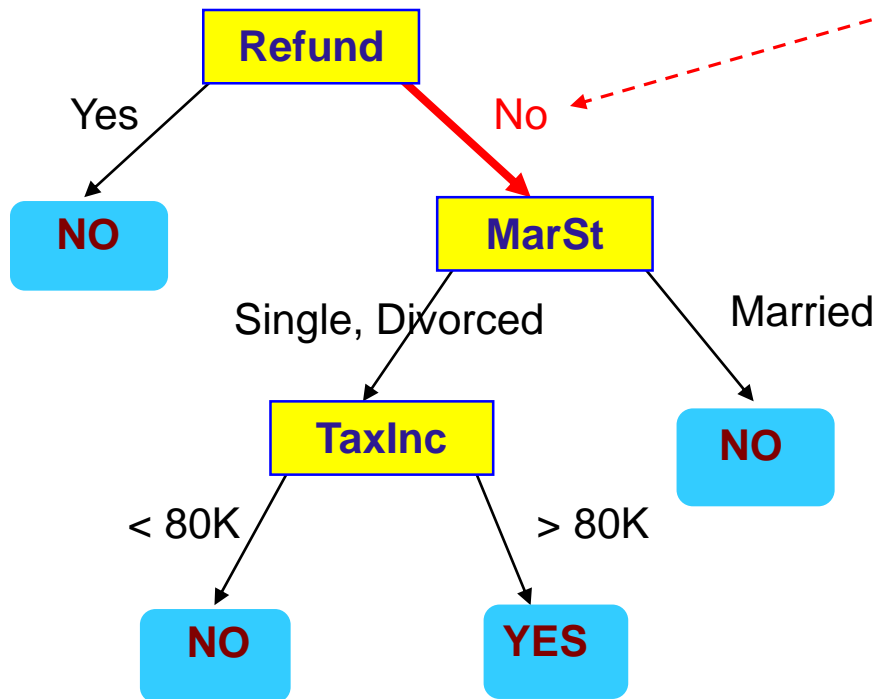


[Example source: "Introduction to Data Mining" by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]

Refund = No, therefore follow the right pointer.

Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

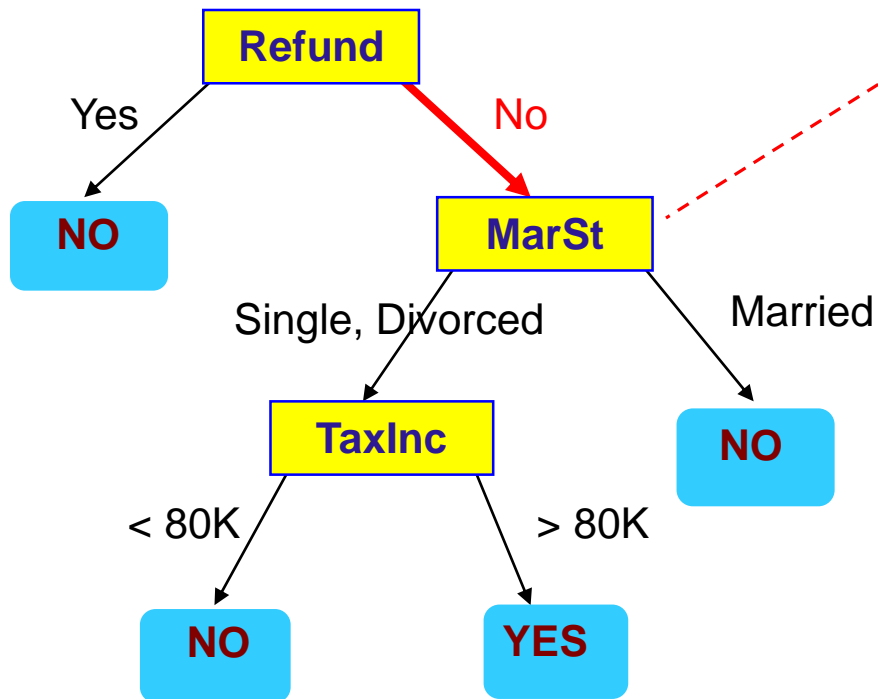


[Example source: "Introduction to Data Mining" by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Addison-Wesley, 2006]

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

Test Data

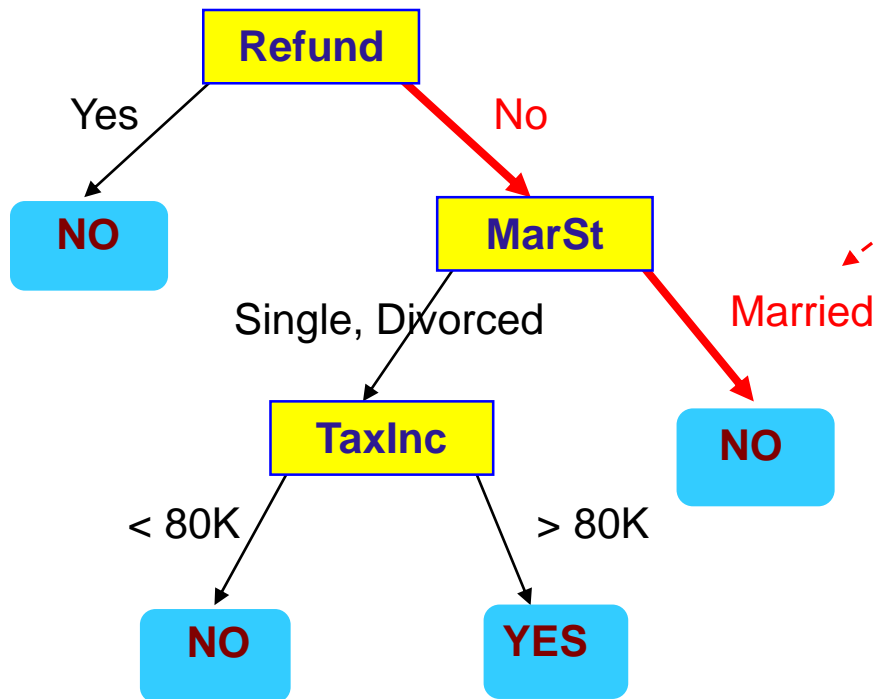
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



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

Test Data

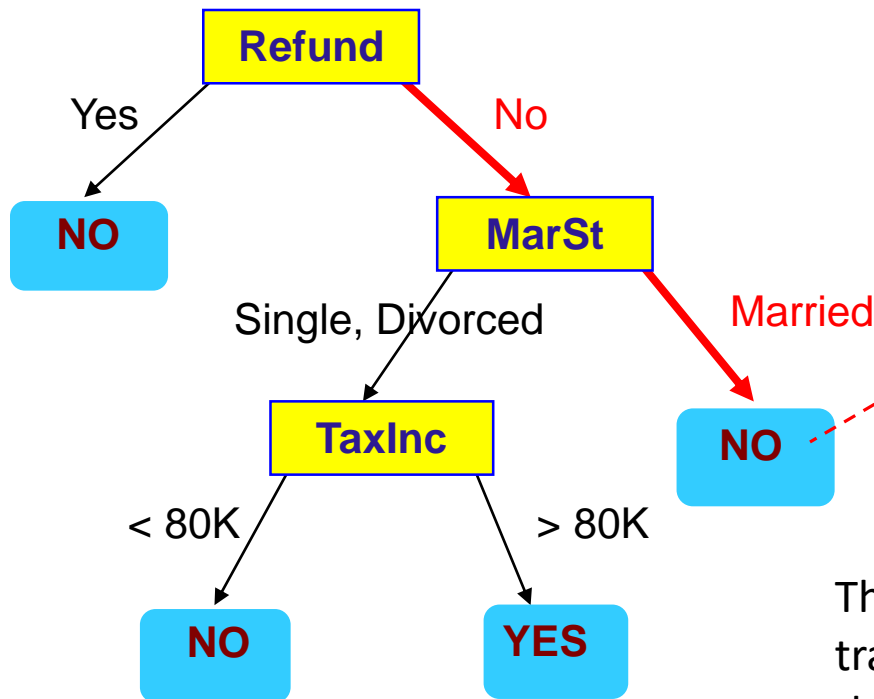
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Follow the corresponding pointer to the leaf node.

Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



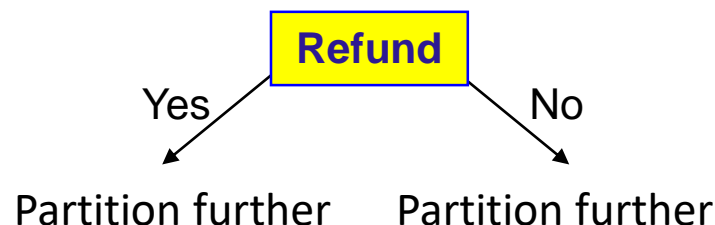
The leaf node reached only contains training records with Cheat = No. Hence the tree will return Cheat = No as the predicted class.

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 the training set is split, is selected based on a heuristic, e.g., information gain. Typical stopping conditions are (1) pure node (when all records in the node belong to the same class), (2) no split attributes left for further partitioning, and (3) too few records in the partition.
- To parallelize model training, one has to identify tasks that can be performed independently. For trees, there are two major opportunities:
 - A tree node partitions the training data set. In the example, records with Refund = Yes go one way, those with Refund = No go another. Each partition is then recursively split further. The split decisions for the partition with Refund = Yes are independent of the split decisions for the partition with Refund = No. Hence they can be performed in parallel without need for communication.
 - To find the best split predicate for a node, each attribute and its possible split values need to be explored. Given the training data and a candidate split predicate, the tree induction algorithm will compute a score. These scores can be computed independently for each attribute and possible split value. Afterward coordination is required to select the split predicate with the highest score.



```

// For simplicity, 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 a file storing the tree structure
    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.
- 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 list of 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.
- For data sets with a huge number of attributes, e.g., text analysis applications in information retrieval, one should consider parallelizing the process of finding the best split attribute for a node. 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.

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 (e.g., information gain, Gini, gain ratio) for classification, or variance for prediction.

Training data

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

Split point 2.5 is the clear winner, because it perfectly separates the two 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.
 - In the example, assume task 0 only receives the records with A-values 1 and 3. Then the only possible middle point would be 2, not 1.5. Intuitively, task 0 cannot determine the right split candidates, because it is missing some of the data.
- Range-partitioning would help, can we find a cheaper solution?

Parallel Counting with Pre-Defined Split Points

- What is each task knew all possible split points from the beginning?
If all tasks count the left and right class distribution for the 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.
- Both round 1 and round 2 parallelize well.
 - If round 2 has multiple tasks, then finding the best split requires another round for finding the one with maximum score.
- The biggest challenge is [how to select the split point candidates](#) a priori.
 - The easiest solution is to pick evenly spaced values from the attribute 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.
- For more info, see [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]
 - This is the algorithm on which Spark's tree (ensemble) implementation in Mllib is based.

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 have to be made for a huge set of test records. The tree model has to be distributed to all worker machines; then each Map call can independently compute the prediction for a test record.

// The file with the decision tree is made available through the distributed file cache

```
Class Mapper {
```

```
    tree
```

```
    setup()
```

```
        tree = load tree from file cache
```

```
    map( ..., test record r ) {
```

```
        output = tree.makePrediction( r )
```

```
        emit( r, output)
```

```
    }
```

```
}
```

Decision Trees in Spark

- We can again rely on Mllib for a ready-to-use solution. Look at the source code to see how it is implemented.
 - The code below shows excerpts from the example in the Spark 2.3.2 documentation.

```
// spark.ml
val dt = new DecisionTreeClassifier()
  .setLabelCol("indexedLabel")
  .setFeaturesCol("indexedFeatures")

// Chain indexers and tree in a Pipeline.
val pipeline = new Pipeline()
  .setStages(Array(labelIndexer, featureIndexer,
    dt, labelConverter))

// Train model. This also runs the indexers.
val model = pipeline.fit(trainingData)

// Make predictions.
val predictions = model.transform(testData)
```

```
// spark.mllib
val model = DecisionTree.trainClassifier(trainingData,
  numClasses, categoricalFeaturesInfo,
  impurity, maxDepth, maxBins)

// Evaluate model on test instances and compute test error
val labelAndPreds = testData.map { point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
}
```

Other Classification and Prediction Methods

- In addition to decision trees, numerous other classification and prediction techniques have been proposed. Many are complex and hence not easy to parallelize. The following is a list of popular techniques:
 - Support Vector Machines (SVMs)
 - Artificial Neural Networks (ANNs)
 - Nearest Neighbor
 - Naïve Bayes
 - Bayesian networks
 - Regression.

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.

5. CYK: Q1

- For each of the following statements, select if it is true or false.
 - Consider the K-means MapReduce algorithm that performs each iteration in parallel.
1. While in early iterations, this algorithm transfers the entire input data set from Mappers to Reducer, in later iterations, as the clusters converge, less data will be transferred from Mappers to Reducer.
 2. The algorithm does not always converge, e.g., when K does not match the number of natural clusters in the data.
 3. K-means works well for *all* clustering problems, as long as we find a good initial configuration.
 4. K-means will always find the clustering with the minimal SSE for a given K.

Q2

- For each of the following statements, select if it is true or false.
- Consider the breadth-first MapReduce program for training a decision tree.
 1. The degree of parallelism in each iteration is limited by the number of different nodes at the corresponding level of the tree. Hence in the beginning it is one.
 2. Initially the number of Map calls increases from one iteration to the next, until it starts decreasing toward the end.
 3. The Map function is more likely to run into memory problems in later iterations, not in earlier ones.

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