**Clustering**

**Clustering as a tool is not new.**

**It has been used in**

**-Biology**

**-Information retrieval**

**-Climate**

**-Psychology and Medicine**

**-Business**

**There can be various uses apart from ML**

**-Summarization**

**-Compression**

**-Finding nearest neighbours-Mahanoblis method**

**Different types of clustering:**

**-Hierarchical Vs Partitional**

**-Exclusive Vs Overlapping Vs Fuzzy**

**-Complete Vs Partial**

**Types of Clusters**

**-Well-Separated**

**-Prototype based**

**-Graph based-(Intelligence)**

**-Density based cluster-(traffic)**

**-Share Property-conceptual cluster-pattern recognition**

**Methods**

**-K-Means**

**-Agglomerative Hierarchical clustering**

**-DBScan**

* Biology. Biologists have spent many years creating a taxonomy (hi- erarchical classiﬁcation) of all living things: kingdom, phylum, class, order, family, genus, and species. Thus, it is perhaps not surprising that much of the early work in cluster analysis sought to create a discipline of mathematical taxonomy that could automatically ﬁnd such classiﬁ- cation structures. More recently, biologists have applied clustering to analyze the large amounts of genetic information that are now available. For example, clustering has been used to ﬁnd groups of genes that have similar functions.
* Information Retrieval. The World Wide Web consists of billions of Web pages, and the results of a query to a search engine can return thousands of pages. Clustering can be used to group these search re- sults into a small number of clusters, each of which captures a particular aspect of the query. For instance, a query of “movie” might return Web pages grouped into categories such as reviews, trailers, stars, and theaters. Each category (cluster) can be broken into subcategories (sub- clusters), producing a hierarchical structure that further assists a user’s exploration of the query results.
* Climate. Understanding the Earth’s climate requires ﬁnding patterns in the atmosphere and ocean. To that end, cluster analysis has been applied to ﬁnd patterns in the atmospheric pressure of polar regions and areas of the ocean that have a signiﬁcant impact on land climate.
* Psychology and Medicine. An illness or condition frequently has a number of variations, and cluster analysis can be used to identify these diﬀerent subcategories. For example, clustering has been used to identify diﬀerent types of depression. Cluster analysis can also be used to detect patterns in the spatial or temporal distribution of a disease.
* Business. Businesses collect large amounts of information on current and potential customers. Clustering can be used to segment customers into a small number of groups for additional analysis and marketing activities.

Clustering for Utility Cluster analysis provides an abstraction from in- dividual data objects to the clusters in which those data objects reside. Ad- ditionally, some clustering techniques characterize each cluster in terms of a cluster prototype; i.e., a data object that is representative of the other ob- jects in the cluster. These cluster prototypes can be used as the basis for a

489

number of data analysis or data processing techniques. Therefore, in the con- text of utility, cluster analysis is the study of techniques for ﬁnding the most representative cluster prototypes.

Summarization. Many data analysis techniques, such as regression or PCA, have a time or space complexity of *O*(*m*2) or higher (where *m* is the number of objects), and thus, are not practical for large data sets. However, instead of applying the algorithm to the entire data set, it can be applied to a reduced data set consisting only of cluster prototypes. Depending on the type of analysis, the number of prototypes, and the accuracy with which the prototypes represent the data, the results can be comparable to those that would have been obtained if all the data could have been used.

•

* Compression. Cluster prototypes can also be used for data compres- sion. In particular, a table is created that consists of the prototypes for each cluster; i.e., each prototype is assigned an integer value that is its position (index) in the table. Each object is represented by the index of the prototype associated with its cluster. This type of compression is known as vector quantization and is often applied to image, sound, and video data, where (1) many of the data objects are highly similar to one another, (2) some loss of information is acceptable, and (3) a substantial reduction in the data size is desired.
* Eﬃciently Finding Nearest Neighbors. Finding nearest neighbors can require computing the pairwise distance between all points. Often clusters and their cluster prototypes can be found much more eﬃciently. If objects are relatively close to the prototype of their cluster, then we can use the prototypes to reduce the number of distance computations that are necessary to ﬁnd the nearest neighbors of an object. Intuitively, if two cluster prototypes are far apart, then the objects in the corresponding clusters cannot be nearest neighbors of each other. Consequently, to ﬁnd an object’s nearest neighbors it is only necessary to compute the distance to objects in nearby clusters, where the nearness of two clusters is measured by the distance between their prototypes. This idea is made more precise in Exercise 25 on page 94.

This chapter provides an introduction to cluster analysis. We begin with a high-level overview of clustering, including a discussion of the various ap- proaches to dividing objects into sets of clusters and the diﬀerent types of clusters. We then describe three speciﬁc clustering techniques that represent

broad categories of algorithms and illustrate a variety of concepts: K-means, agglomerative hierarchical clustering, and DBSCAN. The ﬁnal section of this chapter is devoted to cluster validity—methods for evaluating the goodness of the clusters produced by a clustering algorithm. More advanced clustering concepts and algorithms will be discussed in Chapter 9. Whenever possible, we discuss the strengths and weaknesses of diﬀerent schemes. In addition, the bibliographic notes provide references to relevant books and papers that explore cluster analysis in greater depth.

# Overview

Before discussing speciﬁc clustering techniques, we provide some necessary background. First, we further deﬁne cluster analysis, illustrating why it is diﬃcult and explaining its relationship to other techniques that group data. Then we explore two important topics: (1) diﬀerent ways to group a set of objects into a set of clusters, and (2) types of clusters.

## What Is Cluster Analysis?

Cluster analysis groups data objects based only on information found in the data that describes the objects and their relationships. The goal is that the objects within a group be similar (or related) to one another and diﬀerent from (or unrelated to) the objects in other groups. The greater the similarity (or homogeneity) within a group and the greater the diﬀerence between groups, the better or more distinct the clustering.

In many applications, the notion of a cluster is not well deﬁned. To better understand the diﬃculty of deciding what constitutes a cluster, consider Figure 8.1, which shows twenty points and three diﬀerent ways of dividing them into clusters. The shapes of the markers indicate cluster membership. Figures 8.1(b) and 8.1(d) divide the data into two and six parts, respectively. However, the apparent division of each of the two larger clusters into three subclusters may simply be an artifact of the human visual system. Also, it may not be unreasonable to say that the points form four clusters, as shown in Figure 8.1(c). This ﬁgure illustrates that the deﬁnition of a cluster is imprecise and that the best deﬁnition depends on the nature of data and the desired results. Cluster analysis is related to other techniques that are used to divide data objects into groups. For instance, clustering can be regarded as a form of classiﬁcation in that it creates a labeling of objects with class (cluster) labels. However, it derives these labels only from the data. In contrast, classiﬁcation

* + - 1. Original points. (b) Two clusters.







(c) Four clusters. (d) Six clusters.

**Figure 8.1.** Different ways of clustering the same set of points.

in the sense of Chapter 4 is supervised classiftcation; i.e., new, unlabeled objects are assigned a class label using a model developed from objects with known class labels. For this reason, cluster analysis is sometimes referred to as unsupervised classiftcation. When the term classiﬁcation is used without any qualiﬁcation within data mining, it typically refers to supervised classiﬁcation.

Also, while the terms segmentation and partitioning are sometimes used as synonyms for clustering, these terms are frequently used for approaches outside the traditional bounds of cluster analysis. For example, the term partitioning is often used in connection with techniques that divide graphs into subgraphs and that are not strongly connected to clustering. Segmentation often refers to the division of data into groups using simple techniques; e.g., an image can be split into segments based only on pixel intensity and color, or people can be divided into groups based on their income. Nonetheless, some work in graph partitioning and in image and market segmentation is related to cluster analysis.

## Diﬀerent Types of Clusterings

An entire collection of clusters is commonly referred to as a clustering, and in this section, we distinguish various types of clusterings: hierarchical (nested) versus partitional (unnested), exclusive versus overlapping versus fuzzy, and complete versus partial.

Hierarchical versus Partitional The most commonly discussed distinc- tion among diﬀerent types of clusterings is whether the set of clusters is nested

or unnested, or in more traditional terminology, hierarchical or partitional. A partitional clustering is simply a division of the set of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset. Taken individually, each collection of clusters in Figures 8.1 (b–d) is a partitional clustering.

If we permit clusters to have subclusters, then we obtain a hierarchical clustering, which is a set of nested clusters that are organized as a tree. Each node (cluster) in the tree (except for the leaf nodes) is the union of its children (subclusters), and the root of the tree is the cluster containing all the objects. Often, but not always, the leaves of the tree are singleton clusters of individual data objects. If we allow clusters to be nested, then one interpretation of Figure 8.1(a) is that it has two subclusters (Figure 8.1(b)), each of which, in turn, has three subclusters (Figure 8.1(d)). The clusters shown in Figures 8.1 (a–d), when taken in that order, also form a hierarchical (nested) clustering with, respectively, 1, 2, 4, and 6 clusters on each level. Finally, note that a hierarchical clustering can be viewed as a sequence of partitional clusterings and a partitional clustering can be obtained by taking any member of that sequence; i.e., by cutting the hierarchical tree at a particular level.

Exclusive versus Overlapping versus Fuzzy The clusterings shown in Figure 8.1 are all exclusive, as they assign each object to a single cluster. There are many situations in which a point could reasonably be placed in more than one cluster, and these situations are better addressed by non-exclusive clustering. In the most general sense, an overlapping or non-exclusive clustering is used to reﬂect the fact that an object can *simultaneously* belong to more than one group (class). For instance, a person at a university can be both an enrolled student and an employee of the university. A non-exclusive clustering is also often used when, for example, an object is “between” two or more clusters and could reasonably be assigned to any of these clusters. Imagine a point halfway between two of the clusters of Figure 8.1. Rather than make a somewhat arbitrary assignment of the object to a single cluster, it is placed in all of the “equally good” clusters.

In a fuzzy clustering, every object belongs to every cluster with a mem- bership weight that is between 0 (absolutely doesn’t belong) and 1 (absolutely belongs). In other words, clusters are treated as fuzzy sets. (Mathematically, a fuzzy set is one in which an object belongs to any set with a weight that is between 0 and 1. In fuzzy clustering, we often impose the additional con- straint that the sum of the weights for each object must equal 1.) Similarly, probabilistic clustering techniques compute the probability with which each

point belongs to each cluster, and these probabilities must also sum to 1. Be- cause the membership weights or probabilities for any object sum to 1, a fuzzy or probabilistic clustering does not address true multiclass situations, such as the case of a student employee, where an object belongs to multiple classes. Instead, these approaches are most appropriate for avoiding the arbitrariness of assigning an object to only one cluster when it may be close to several. In practice, a fuzzy or probabilistic clustering is often converted to an exclusive clustering by assigning each object to the cluster in which its membership weight or probability is highest.

Complete versus Partial A complete clustering assigns every object to a cluster, whereas a partial clustering does not. The motivation for a partial clustering is that some objects in a data set may not belong to well-deﬁned groups. Many times objects in the data set may represent noise, outliers, or “uninteresting background.” For example, some newspaper stories may share a common theme, such as global warming, while other stories are more generic or one-of-a-kind. Thus, to ﬁnd the important topics in last month’s stories, we may want to search only for clusters of documents that are tightly related by a common theme. In other cases, a complete clustering of the objects is desired. For example, an application that uses clustering to organize documents for browsing needs to guarantee that all documents can be browsed.

## Diﬀerent Types of Clusters

Clustering aims to ﬁnd useful groups of objects (clusters), where usefulness is deﬁned by the goals of the data analysis. Not surprisingly, there are several diﬀerent notions of a cluster that prove useful in practice. In order to visually illustrate the diﬀerences among these types of clusters, we use two-dimensional points, as shown in Figure 8.2, as our data objects. We stress, however, that the types of clusters described here are equally valid for other kinds of data.

Well-Separated A cluster is a set of objects in which each object is closer (or more similar) to every other object in the cluster than to any object not in the cluster. Sometimes a threshold is used to specify that all the objects in a cluster must be suﬃciently close (or similar) to one another. This idealistic deﬁnition of a cluster is satisﬁed only when the data contains natural clusters that are quite far from each other. Figure 8.2(a) gives an example of well- separated clusters that consists of two groups of points in a two-dimensional space. The distance between any two points in diﬀerent groups is larger than

the distance between any two points within a group. Well-separated clusters do not need to be globular, but can have any shape.

Prototype-Based A cluster is a set of objects in which each object is closer (more similar) to the prototype that deﬁnes the cluster than to the prototype of any other cluster. For data with continuous attributes, the prototype of a cluster is often a centroid, i.e., the average (mean) of all the points in the clus- ter. When a centroid is not meaningful, such as when the data has categorical attributes, the prototype is often a medoid, i.e., the most representative point of a cluster. For many types of data, the prototype can be regarded as the most central point, and in such instances, we commonly refer to prototype- based clusters as center-based clusters. Not surprisingly, such clusters tend to be globular. Figure 8.2(b) shows an example of center-based clusters.

Graph-Based If the data is represented as a graph, where the nodes are objects and the links represent connections among objects (see Section 2.1.2), then a cluster can be deﬁned as a connected component; i.e., a group of objects that are connected to one another, but that have no connection to objects outside the group. An important example of graph-based clusters are contiguity-based clusters, where two objects are connected only if they are within a speciﬁed distance of each other. This implies that each object in a contiguity-based cluster is closer to some other object in the cluster than to any point in a diﬀerent cluster. Figure 8.2(c) shows an example of such clusters for two-dimensional points. This deﬁnition of a cluster is useful when clusters are irregular or intertwined, but can have trouble when noise is present since, as illustrated by the two spherical clusters of Figure 8.2(c), a small bridge of points can merge two distinct clusters.

Other types of graph-based clusters are also possible. One such approach (Section 8.3.2) deﬁnes a cluster as a clique; i.e., a set of nodes in a graph that are completely connected to each other. Speciﬁcally, if we add connections between objects in the order of their distance from one another, a cluster is formed when a set of objects forms a clique. Like prototype-based clusters, such clusters tend to be globular.

Density-Based A cluster is a dense region of objects that is surrounded by a region of low density. Figure 8.2(d) shows some density-based clusters for data created by adding noise to the data of Figure 8.2(c). The two circular clusters are not merged, as in Figure 8.2(c), because the bridge between them fades into the noise. Likewise, the curve that is present in Figure 8.2(c) also

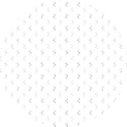
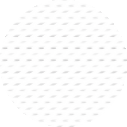
fades into the noise and does not form a cluster in Figure 8.2(d). A density- based deﬁnition of a cluster is often employed when the clusters are irregular or intertwined, and when noise and outliers are present. By contrast, a contiguity- based deﬁnition of a cluster would not work well for the data of Figure 8.2(d) since the noise would tend to form bridges between clusters.

Shared-Property (Conceptual Clusters) More generally, we can deﬁne a cluster as a set of objects that share some property. This deﬁnition encom- passes all the previous deﬁnitions of a cluster; e.g., objects in a center-based cluster share the property that they are all closest to the same centroid or medoid. However, the shared-property approach also includes new types of clusters. Consider the clusters shown in Figure 8.2(e). A triangular area (cluster) is adjacent to a rectangular one, and there are two intertwined circles (clusters). In both cases, a clustering algorithm would need a very speciﬁc concept of a cluster to successfully detect these clusters. The process of ﬁnd- ing such clusters is called conceptual clustering. However, too sophisticated a notion of a cluster would take us into the area of pattern recognition, and thus, we only consider simpler types of clusters in this book.

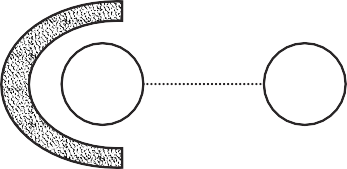
## Road Map

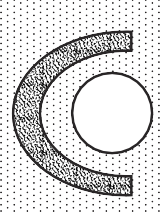
In this chapter, we use the following three simple, but important techniques to introduce many of the concepts involved in cluster analysis.

* K-means. This is a prototype-based, partitional clustering technique that attempts to ﬁnd a user-speciﬁed number of clusters (*K* ), which are represented by their centroids.
* Agglomerative Hierarchical Clustering. This clustering approach refers to a collection of closely related clustering techniques that produce a hierarchical clustering by starting with each point as a singleton cluster and then repeatedly merging the two closest clusters until a single, all- encompassing cluster remains. Some of these techniques have a natural interpretation in terms of graph-based clustering, while others have an interpretation in terms of a prototype-based approach.
* DBSCAN. This is a density-based clustering algorithm that produces a partitional clustering, in which the number of clusters is automatically determined by the algorithm. Points in low-density regions are classi- ﬁed as noise and omitted; thus, DBSCAN does not produce a complete clustering.

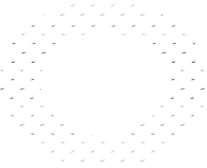
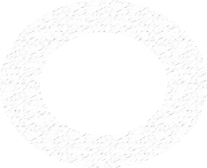
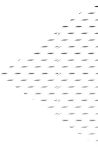
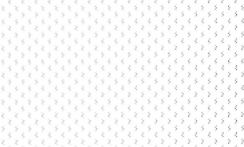


1. Well-separated clusters. Each point is closer to all of the points in its cluster than to any point in another cluster.
2. Center-based clusters. Each point is closer to the center of its cluster than to the center of any other cluster.





1. Contiguity-based clusters. Each point is closer to at least one point in its cluster than to any point in another cluster.
2. Density-based clusters. Clus- ters are regions of high density sep- arated by regions of low density.



1. Conceptual clusters. Points in a cluster share some general property that derives from the entire set of points. (Points in the intersection of the circles belong to both.)

**Figure 8.2.** Different types of clusters as illustrated by sets of two-dimensional points.

# K-means

Prototype-based clustering techniques create a one-level partitioning of the data objects. There are a number of such techniques, but two of the most prominent are K-means and K-medoid. K-means deﬁnes a prototype in terms of a centroid, which is usually the mean of a group of points, and is typically

applied to objects in a continuous *n*-dimensional space. K-medoid deﬁnes a prototype in terms of a medoid, which is the most representative point for a group of points, and can be applied to a wide range of data since it requires only a proximity measure for a pair of objects. While a centroid almost never corresponds to an actual data point, a medoid, by its deﬁnition, must be an actual data point. In this section, we will focus solely on K-means, which is one of the oldest and most widely used clustering algorithms.

## The Basic K-means Algorithm

The K-means clustering technique is simple, and we begin with a description of the basic algorithm. We ﬁrst choose *K* initial centroids, where *K* is a user- speciﬁed parameter, namely, the number of clusters desired. Each point is then assigned to the closest centroid, and each collection of points assigned to a centroid is a cluster. The centroid of each cluster is then updated based on the points assigned to the cluster. We repeat the assignment.

The K-means clustering technique is simple, and we begin with a description of the basic algorithm. We ﬁrst choose *K* initial centroids, where *K* is a user- speciﬁed parameter, namely, the number of clusters desired. Each point is then assigned to the closest centroid, and each collection of points assigned to a centroid is a cluster. The centroid of each cluster is then updated based on the points assigned to the cluster. We repeat the assignment and update steps until no point changes clusters, or equivalently, until the centroids remain the same.

K-means is formally described by Algorithm 8.1. The operation of K-means is illustrated in Figure 8.3, which shows how, starting from three centroids, the ﬁnal clusters are found in four assignment-update steps. In these and other ﬁgures displaying K-means clustering, each subﬁgure shows (1) the centroids at the start of the iteration and (2) the assignment of the points to those centroids. The centroids are indicated by the “+” symbol; all points belonging to the same cluster have the same marker shape.

Algorithm 8.1 Basic K-means algorithm.

1: Select *K* points as initial centroids.

2: repeat

3: Form *K* clusters by assigning each point to its closest centroid.

4: Recompute the centroid of each cluster.

5: until Centroids do not change.

In the ﬁrst step, shown in Figure 8.3(a), points are assigned to the initial centroids, which are all in the larger group of points. For this example, we use the mean as the centroid. After points are assigned to a centroid, the centroid is then updated. Again, the ﬁgure for each step shows the centroid at the beginning of the step and the assignment of points to those centroids. In the second step, points are assigned to the updated centroids, and the centroids



* + - 1. Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Iteration 4.



**Figure 8.3.** Using the K-means algorithm to ﬁnd three clusters in sample data.

are updated again. In steps 2, 3, and 4, which are shown in Figures 8.3 (b), (c), and (d), respectively, two of the centroids move to the two small groups of points at the bottom of the ﬁgures. When the K-means algorithm terminates in Figure 8.3(d), because no more changes occur, the centroids have identiﬁed the natural groupings of points.

For some combinations of proximity functions and types of centroids, K- means always converges to a solution; i.e., K-means reaches a state in which no points are shifting from one cluster to another, and hence, the centroids don’t change. Because most of the convergence occurs in the early steps, however, the condition on line 5 of Algorithm 8.1 is often replaced by a weaker condition, e.g., repeat until only 1% of the points change clusters.

We consider each of the steps in the basic K-means algorithm in more detail and then provide an analysis of the algorithm’s space and time complexity.

Assigning Points to the Closest Centroid

To assign a point to the closest centroid, we need a proximity measure that quantiﬁes the notion of “closest” for the speciﬁc data under consideration. Euclidean (L2) distance is often used for data points in Euclidean space, while cosine similarity is more appropriate for documents. However, there may be several types of proximity measures that are appropriate for a given type of data. For example, Manhattan (L1) distance can be used for Euclidean data, while the Jaccard measure is often employed for documents.

Usually, the similarity measures used for K-means are relatively simple since the algorithm repeatedly calculates the similarity of each point to each centroid. In some cases, however, such as when the data is in low-dimensional

**Table 8.1.** Table of notation.

|  |  |
| --- | --- |
| Symbol | Description |
| x  *C*i ci c *m*i *m K* | An object.  The *i*th cluster.  The centroid of cluster *C*i. The centroid of all points.  The number of objects in the *i*th cluster. The number of objects in the data set.  The number of clusters. |

Euclidean space, it is possible to avoid computing many of the similarities, thus signiﬁcantly speeding up the K-means algorithm. Bisecting K-means (described in Section 8.2.3) is another approach that speeds up K-means by reducing the number of similarities computed.

Centroids and Objective Functions

Step 4 of the K-means algorithm was stated rather generally as “recompute the centroid of each cluster,” since the centroid can vary, depending on the proximity measure for the data and the goal of the clustering. The goal of the clustering is typically expressed by an objective function that depends on the proximities of the points to one another or to the cluster centroids; e.g., minimize the squared distance of each point to its closest centroid. We illus- trate this with two examples. However, the key point is this: once we have speciﬁed a proximity measure and an objective function, the centroid that we should choose can often be determined mathematically. We provide mathe- matical details in Section 8.2.6, and provide a non-mathematical discussion of this observation here.

Data in Euclidean Space Consider data whose proximity measure is Eu- clidean distance. For our objective function, which measures the quality of a clustering, we use the sum of the squared error (SSE), which is also known as scatter. In other words, we calculate the error of each data point, i.e., its Euclidean distance to the closest centroid, and then compute the total sum of the squared errors. Given two diﬀerent sets of clusters that are produced by two diﬀerent runs of K-means, we prefer the one with the smallest squared error since this means that the prototypes (centroids) of this clustering are a better representation of the points in their cluster. Using the notation in Table 8.1, the SSE is formally deﬁned as follows:

K

Σ Σ

SSE = *dist*(ci*,* x)2 (8.1)

i=1 x∈Ci

where *dist* is the standard Euclidean (L2) distance between two objects in Euclidean space.

Given these assumptions, it can be shown (see Section 8.2.6) that the centroid that minimizes the SSE of the cluster is the mean. Using the notation in Table 8.1, the centroid (mean) of the *i*th cluster is deﬁned by Equation 8.2.

1

ci =

*m*i

x (8.2)

x∈Ci

Σ

To illustrate, the centroid of a cluster containing the three two-dimensional points, (1,1), (2,3), and (6,2), is ((1 + 2 + 6)*/*3*,* ((1 +3+ 2)*/*3) = (3*,* 2).

Steps 3 and 4 of the K-means algorithm directly attempt to minimize the SSE (or more generally, the objective function). Step 3 forms clusters by assigning points to their nearest centroid, which minimizes the SSE for the given set of centroids. Step 4 recomputes the centroids so as to further minimize the SSE. However, the actions of K-means in Steps 3 and 4 are only guaranteed to ﬁnd a local minimum with respect to the SSE since they are based on optimizing the SSE for speciﬁc choices of the centroids and clusters, rather than for all possible choices. We will later see an example in which this leads to a suboptimal clustering.

Document Data To illustrate that K-means is not restricted to data in Euclidean space, we consider document data and the cosine similarity measure. Here we assume that the document data is represented as a document-term matrix as described on page 31. Our objective is to maximize the similarity of the documents in a cluster to the cluster centroid; this quantity is known as the cohesion of the cluster. For this objective it can be shown that the cluster centroid is, as for Euclidean data, the mean. The analogous quantity to the total SSE is the total cohesion, which is given by Equation 8.3.

K

Σ Σ

Total Cohesion = *cosine*(x*,* ci) (8.3)

i=1 x∈Ci

The General Case There are a number of choices for the proximity func- tion, centroid, and objective function that can be used in the basic K-means

**Table 8.2.** K-means: Common choices for proximity, centroids, and objective functions.

|  |  |  |
| --- | --- | --- |
| Proximity Function | Centroid | Objective Function |
| Manhattan (L1) | median | Minimize sum of the L1 distance of an ob- ject to its cluster centroid |
| Squared Euclidean (L2)  2 | mean | Minimize sum of the squared L2 distance of an object to its cluster centroid |
| cosine | mean | Maximize sum of the cosine similarity of  an object to its cluster centroid |
| Bregman divergence | mean | Minimize sum of the Bregman divergence  of an object to its cluster centroid |

algorithm and that are guaranteed to converge. Table 8.2 shows some possible choices, including the two that we have just discussed. Notice that for Man- hattan (L1) distance and the objective of minimizing the sum of the distances, the appropriate centroid is the median of the points in a cluster.

The last entry in the table, Bregman divergence (Section 2.4.5), is actually a class of proximity measures that includes the squared Euclidean distance, L2, the Mahalanobis distance, and cosine similarity. The importance of Bregman divergence functions is that any such function can be used as the basis of a K- means style clustering algorithm with the mean as the centroid. Speciﬁcally, if we use a Bregman divergence as our proximity function, then the result- ing clustering algorithm has the usual properties of K-means with respect to convergence, local minima, etc. Furthermore, the properties of such a cluster- ing algorithm can be developed for all possible Bregman divergences. Indeed, K-means algorithms that use cosine similarity or squared Euclidean distance are particular instances of a general clustering algorithm based on Bregman divergences.

2

For the rest our K-means discussion, we use two-dimensional data since it is easy to explain K-means and its properties for this type of data. But, as suggested by the last few paragraphs, K-means is a very general clustering algorithm and can be used with a wide variety of data types, such as documents and time series.

Choosing Initial Centroids

When random initialization of centroids is used, diﬀerent runs of K-means typically produce diﬀerent total SSEs. We illustrate this with the set of two- dimensional points shown in Figure 8.3, which has three natural clusters of points. Figure 8.4(a) shows a clustering solution that is the global minimum of



* + - * 1. Optimal clustering. (b) Suboptimal clustering.



**Figure 8.4.** Three optimal and non-optimal clusters.

the SSE for three clusters, while Figure 8.4(b) shows a suboptimal clustering that is only a local minimum.

Choosing the proper initial centroids is the key step of the basic K-means procedure. A common approach is to choose the initial centroids randomly, but the resulting clusters are often poor.

Example 8.1 (Poor Initial Centroids). Randomly selected initial cen- troids may be poor. We provide an example of this using the same data set used in Figures 8.3 and 8.4. Figures 8.3 and 8.5 show the clusters that re- sult from two particular choices of initial centroids. (For both ﬁgures, the positions of the cluster centroids in the various iterations are indicated by crosses.) In Figure 8.3, even though all the initial centroids are from one natu- ral cluster, the minimum SSE clustering is still found. In Figure 8.5, however, even though the initial centroids seem to be better distributed, we obtain a suboptimal clustering, with higher squared error.

Example 8.2 (Limits of Random Initialization). One technique that is commonly used to address the problem of choosing initial centroids is to perform multiple runs, each with a diﬀerent set of randomly chosen initial centroids, and then select the set of clusters with the minimum SSE. While simple, this strategy may not work very well, depending on the data set and the number of clusters sought. We demonstrate this using the sample data set shown in Figure 8.6(a). The data consists of two pairs of clusters, where the clusters in each (top-bottom) pair are closer to each other than to the clusters in the other pair. Figure 8.6 (b–d) shows that if we start with two initial centroids per pair of clusters, then even when both centroids are in a single

1. Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Iteration 4.



**Figure 8.5.** Poor starting centroids for K-means.

cluster, the centroids will redistribute themselves so that the “true” clusters are found. However, Figure 8.7 shows that if a pair of clusters has only one initial centroid and the other pair has three, then two of the true clusters will be combined and one true cluster will be split.

Note that an optimal clustering will be obtained as long as two initial centroids fall anywhere in a pair of clusters, since the centroids will redistribute themselves, one to each cluster. Unfortunately, as the number of clusters becomes larger, it is increasingly likely that at least one pair of clusters will have only one initial centroid. (See Exercise 4 on page 559.) In this case, because the pairs of clusters are farther apart than clusters within a pair, the K-means algorithm will not redistribute the centroids between pairs of clusters, and thus, only a local minimum will be achieved.

Because of the problems with using randomly selected initial centroids, which even repeated runs may not overcome, other techniques are often em- ployed for initialization. One eﬀective approach is to take a sample of points and cluster them using a hierarchical clustering technique. *K* clusters are ex- tracted from the hierarchical clustering, and the centroids of those clusters are used as the initial centroids. This approach often works well, but is practical only if (1) the sample is relatively small, e.g., a few hundred to a few thousand (hierarchical clustering is expensive), and (2) *K* is relatively small compared to the sample size.

The following procedure is another approach to selecting initial centroids. Select the ﬁrst point at random or take the centroid of all points. Then, for each successive initial centroid, select the point that is farthest from any of the initial centroids already selected. In this way, we obtain a set of initial

* 1. Initial points. (b) Iteration 1.

(c) Iteration 2. (d) Iteration 3.

**Figure 8.6.** Two pairs of clusters with a pair of initial centroids within each pair of clusters.

centroids that is guaranteed to be not only randomly selected but also well separated. Unfortunately, such an approach can select outliers, rather than points in dense regions (clusters). Also, it is expensive to compute the farthest point from the current set of initial centroids. To overcome these problems, this approach is often applied to a sample of the points. Since outliers are rare, they tend not to show up in a random sample. In contrast, points from every dense region are likely to be included unless the sample size is very small. Also, the computation involved in ﬁnding the initial centroids is greatly reduced because the sample size is typically much smaller than the number of points.

Later on, we will discuss two other approaches that are useful for produc- ing better-quality (lower SSE) clusterings: using a variant of K-means that



(a) Iteration 1. (b) Iteration 2.



(c) Iteration 3. (d) Iteration 4.



**Figure 8.7.** Two pairs of clusters with more or fewer than two initial centroids within a pair of clusters.

is less susceptible to initialization problems (bisecting K-means) and using postprocessing to “ﬁxup” the set of clusters produced.

Time and Space Complexity

The space requirements for K-means are modest because only the data points and centroids are stored. Speciﬁcally, the storage required is *O*((*m* + *K*)*n*), where *m* is the number of points and *n* is the number of attributes. The time requirements for K-means are also modest—basically linear in the number of data points. In particular, the time required is *O*(*I* ∗ *K* ∗ *m* ∗ *n*), where *I* is the number of iterations required for convergence. As mentioned, *I* is often small and can usually be safely bounded, as most changes typically occur in the

ﬁrst few iterations. Therefore, K-means is linear in *m*, the number of points, and is eﬃcient as well as simple provided that *K*, the number of clusters, is signiﬁcantly less than *m*.

## K-means: Additional Issues

Handling Empty Clusters

One of the problems with the basic K-means algorithm given earlier is that empty clusters can be obtained if no points are allocated to a cluster during the assignment step. If this happens, then a strategy is needed to choose a replacement centroid, since otherwise, the squared error will be larger than necessary. One approach is to choose the point that is farthest away from any current centroid. If nothing else, this eliminates the point that currently contributes most to the total squared error. Another approach is to choose the replacement centroid from the cluster that has the highest SSE. This will typically split the cluster and reduce the overall SSE of the clustering. If there are several empty clusters, then this process can be repeated several times.

Outliers

When the squared error criterion is used, outliers can unduly inﬂuence the clusters that are found. In particular, when outliers are present, the resulting cluster centroids (prototypes) may not be as representative as they otherwise would be and thus, the SSE will be higher as well. Because of this, it is often useful to discover outliers and eliminate them beforehand. It is important, however, to appreciate that there are certain clustering applications for which outliers should not be eliminated. When clustering is used for data com- pression, every point must be clustered, and in some cases, such as ﬁnancial analysis, apparent outliers, e.g., unusually proﬁtable customers, can be the most interesting points.

An obvious issue is how to identify outliers. A number of techniques for identifying outliers will be discussed in Chapter 10. If we use approaches that remove outliers before clustering, we avoid clustering points that will not clus- ter well. Alternatively, outliers can also be identiﬁed in a postprocessing step. For instance, we can keep track of the SSE contributed by each point, and eliminate those points with unusually high contributions, especially over mul- tiple runs. Also, we may want to eliminate small clusters since they frequently represent groups of outliers.

Here, we delve into the mathematics behind K-means. This section, which can be skipped without loss of continuity, requires knowledge of calculus through partial derivatives. Familiarity with optimization techniques, especially those based on gradient descent, may also be helpful.

As mentioned earlier, given an objective function such as “minimize SSE,” clustering can be treated as an optimization problem. One way to solve this problem—to ﬁnd a global optimum—is to enumerate all possible ways of di- viding the points into clusters and then choose the set of clusters that best satisﬁes the objective function, e.g., that minimizes the total SSE. Of course, this exhaustive strategy is computationally infeasible and as a result, a more practical approach is needed, even if such an approach ﬁnds solutions that are not guaranteed to be optimal. One technique, which is known as gradient descent, is based on picking an initial solution and then repeating the fol- lowing two steps: compute the change to the solution that best optimizes the objective function and then update the solution.

We assume that the data is one-dimensional, i.e., *dist*(*x, y*) = (*x* − *y*)2. This does not change anything essential, but greatly simpliﬁes the notation.

Derivation of K-means as an Algorithm to Minimize the SSE

In this section, we show how the centroid for the K-means algorithm can be mathematically derived when the proximity function is Euclidean distance and the objective is to minimize the SSE. Speciﬁcally, we investigate how we can best update a cluster centroid so that the cluster SSE is minimized. In mathematical terms, we seek to minimize Equation 8.1, which we repeat here, specialized for one-dimensional data.

K

Σ Σ

SSE = (*c*i − *x*)2 (8.4)

i=1 x∈C*i*

Here, *C*i is the *i*th cluster, *x* is a point in *C*i, and *c*i is the mean of the *i*th

cluster. See Table 8.1 for a complete list of notation.

We can solve for the *k*th centroid *c*k, which minimizes Equation 8.4, by diﬀerentiating the SSE, setting it equal to 0, and solving, as indicated below.

*∂* SSE = *∂* Σ Σ (*c*

K

− *x*)

*∂c*k

2

*∂c*k

K

Σ Σ

i

i=1 x∈C*i*

= *∂* (*c*

2

*∂c*k i

− *x*)

i=1 x∈C*i*

Σ

= 2 ∗ (*c*k − *x*k) = 0

x∈C*k*

Σ 2 ∗ (*c*

x∈C*k*

− *x* ) = 0 ⇒ *m c*

= Σ *x* ⇒ *c*

x∈C*k*

= 1 Σ *x*

x∈C*k*

x∈C*k*

x∈C*k*

x∈C*k*

Thus, as previously indicated, the best centroid for minimizing the SSE of a cluster is the mean of the points in the cluster.

k

k

k

k

k

k

*m*k

k

Derivation of K-means for SAE

To demonstrate that the K-means algorithm can be applied to a variety of diﬀerent objective functions, we consider how to partition the data into *K* clusters such that the sum of the Manhattan (L1) distances of points from the center of their clusters is minimized. We are seeking to minimize the sum of the L1 absolute errors (SAE) as given by the following equation, where *dist*L1

is the L1 distance. Again, for notational simplicity, we use one-dimensional

data, i.e., *dist*L1 = |*c*i − *x*|.

K

Σ Σ

SAE = *dist*L1 (*c*i*, x*) (8.5)

i=1 x∈C*i*

We can solve for the *k*th centroid *c*k, which minimizes Equation 8.5, by diﬀerentiating the SAE, setting it equal to 0, and solving.

*∂* SAE = *∂* Σ Σ |*c*

K

− *x*|

*∂c*k

*∂c*k

K

Σ Σ |

i

i=1 x∈C*i*

= *∂ c*

*∂c*k i

− *x*|

i=1 x∈C*i*

=

Σ

x∈C*k*

*∂*

*∂c* |*c*k

k

− *x*| = 0

xΣ∈C*k*

*∂*

*∂c* |*c*k

k

− *x*| = 0 ⇒ *sign*(*x* − *c*k

x∈C*k*

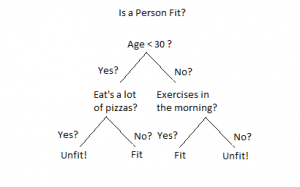
Σ

) = 0

If we solve for *c*k, we ﬁnd that *c*k = *median*{*x* ∈ *C*k}, the median of the points in the cluster. The median of a group of points is straightforward to compute and less susceptible to distortion by outliers.

**Introduction**

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.

[](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-1.png)

An example of a decision tree can be explained using above binary tree. Let’s say you want to predict whether a person is fit given their information like age, eating habit, and physical activity, etc. The decision nodes here are questions like ‘What’s the age?’, ‘Does he exercise?’, ‘Does he eat a lot of pizzas’? And the leaves, which are outcomes like either ‘fit’, or ‘unfit’. In this case this was a binary classification problem (a yes no type problem).

There are two main types of Decision Trees:

1. **Classification trees** (Yes/No types)

What we’ve seen above is an example of classification tree, where the outcome was a variable like ‘fit’ or ‘unfit’. Here the decision variable is **Categorical**.

1. **Regression trees** (Continuous data types)

Here the decision or the outcome variable is **Continuous**, e.g. a number like 123.

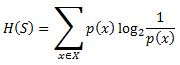
**Working**

Now that we know what a Decision Tree is, we’ll see how it works internally. There are many algorithms out there which construct Decision Trees, but one of the best is called as **ID3 Algorithm**. ID3 Stands for **Iterative Dichotomiser 3**.

Before discussing the ID3 algorithm, we’ll go through few definitions.

**Entropy**

Entropy, also called as Shannon Entropy is denoted by H(S) for a finite set S, is the measure of the amount of uncertainty or randomness in data.

[](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-2.jpg)

Intuitively, it tells us about the predictability of a certain event. Example, consider a coin toss whose probability of heads is 0.5 and probability of tails is 0.5. Here the entropy is the highest possible, since there’s no way of determining what the outcome might be. Alternatively, consider a coin which has heads on both the sides, the entropy of such an event can be predicted perfectly since we know beforehand that it’ll always be heads. In other words, this event has **no randomness** hence it’s entropy is zero.

In particular, lower values imply less uncertainty while higher values imply high uncertainty.

**Information Gain**

Information gain is also called as Kullback-Leibler divergence denoted by IG(S,A) for a set S is the effective change in entropy after deciding on a particular attribute A. It measures the relative change in entropy with respect to the independent variables.

[Decision Trees modified](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-3.jpg)

Alternatively,

[Decision Trees modified](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-4.jpg)

where IG(S, A) is the information gain by applying feature A. H(S) is the Entropy of the entire set, while the second term calculates the Entropy after applying the feature A, where P(x) is the probability of event x.

Let’s understand this with the help of an example

Consider a piece of data collected over the course of 14 days where the features are Outlook, Temperature, Humidity, Wind and the outcome variable is whether Golf was played on the day. Now, our job is to build a predictive model which takes in above 4 parameters and predicts whether Golf will be played on the day. We’ll build a decision tree to do that using **ID3 algorithm.**

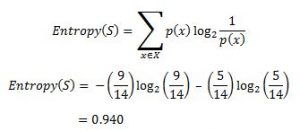
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Day | Outlook | Temperature | Humidity | Wind | Play Golf |
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | Hot | High | Weak | Yes |
| D4 | Rain | Mild | High | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | Yes |
| D6 | Rain | Cool | Normal | Strong | No |
| D7 | Overcast | Cool | Normal | Strong | Yes |
| D8 | Sunny | Mild | High | Weak | No |
| D9 | Sunny | Cool | Normal | Weak | Yes |
| D10 | Rain | Mild | Normal | Weak | Yes |
| D11 | Sunny | Mild | Normal | Strong | Yes |
| D12 | Overcast | Mild | High | Strong | Yes |
| D13 | Overcast | Hot | Normal | Weak | Yes |
| D14 | Rain | Mild | High | Strong | No |

ID3 Algorithm will perform following tasks recursively

1. Create root node for the tree
2. If all examples are positive, return leaf node ‘positive’
3. Else if all examples are negative, return leaf node ‘negative’
4. Calculate the entropy of current state H(S)
5. For each attribute, calculate the entropy with respect to the attribute ‘x’ denoted by H(S, x)
6. Select the attribute which has maximum value of IG(S, x)
7. Remove the attribute that offers highest IG from the set of attributes
8. Repeat until we run out of all attributes, or the decision tree has all leaf nodes.

Now we’ll go ahead and grow the decision tree. The initial step is to calculate H(S), the Entropy of the current state. In the above example, we can see in total there are 5 No’s and 9 Yes’s.

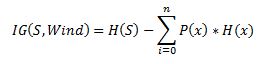
|  |  |  |
| --- | --- | --- |
| Yes | No | Total |
| 9 | 5 | 14 |

[](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-5.jpg)

Remember that the Entropy is 0 if all members belong to the same class, and 1 when half of them belong to one class and other half belong to other class that is perfect randomness. Here it’s 0.94 which means the distribution is fairly random.

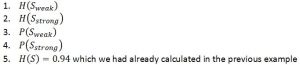
**Now the next step is to choose the attribute that gives us highest possible Information Gain** which we’ll choose as the root node.

Let’s start with ‘Wind’

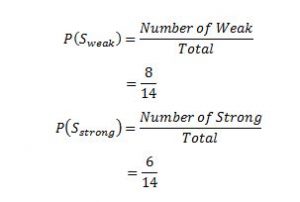
[](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-6.jpg)

where ‘x’ are the possible values for an attribute. Here,  attribute ‘Wind’ takes two possible values in the sample data, hence x = {Weak, Strong}

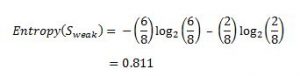
We’ll have to calculate:

[[](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-77.jpg)](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-77.jpg)Amongst all the 14 examples we have **8 places where the wind is weak and 6 where the wind is Strong**.

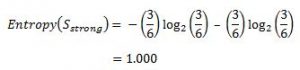
|  |  |  |
| --- | --- | --- |
| Wind = Weak | Wind = Strong | Total |
| 8 | 6 | 14 |

[](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-8.jpg)

Now out of the 8 Weak examples, 6 of them were ‘Yes’ for Play Golf and 2 of them were ‘No’ for ‘Play Golf’. So, we have,

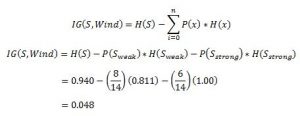
[](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-9.jpg)

Similarly, out of 6 Strong examples, we have **3 examples where the outcome was ‘Yes’ for Play Golf and 3 where we had ‘No’ for Play Golf**.

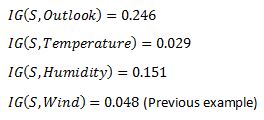
[](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-10.jpg)

Remember, here half items belong to one class while other half belong to other. Hence we have perfect randomness.

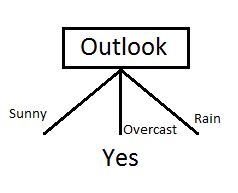
Now we have all the pieces required to calculate the Information Gain,

[](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-11.jpg)

Which tells us the Information Gain by considering ‘Wind’ as the feature and give us information gain of **0.048**. Now we must similarly calculate the Information Gain for all the features.

[](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-12.jpg)

We can clearly see that IG(S, Outlook) has the highest information gain of 0.246,**hence we chose Outlook attribute  as the root node**. At this point, the decision tree looks like.

[](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-13.jpg)

Here we observe that whenever the outlook is Overcast, Play Golf is always ‘Yes’, it’s no coincidence by any chance, the simple tree resulted because of **the highest information gain is given by the attribute Outlook**.

Now how do we proceed from this point? We can simply apply **recursion**, you might want to look at the algorithm steps described earlier.

Now that we’ve used Outlook, we’ve got three of them remaining Humidity, Temperature, and Wind. And, we had three possible values of Outlook: Sunny, Overcast, Rain. Where the Overcast node already ended up having leaf node ‘Yes’, so we’re left with two subtrees to compute: Sunny and Rain.

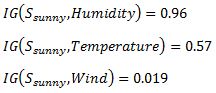
[Decision Trees modified](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-33.jpg)

Table where the value of Outlook is Sunny looks like:

|  |  |  |  |
| --- | --- | --- | --- |
| Temperature | Humidity | Wind | Play Golf |
| Hot | High | Weak | No |
| Hot | High | Strong | No |
| Mild | High | Weak | No |
| Cool | Normal | Weak | Yes |
| Mild | Normal | Strong | Yes |

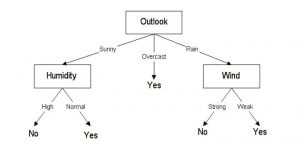
[Decision Trees modified](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-14.jpg)

In the similar fashion, we compute the following values

[](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-15.jpg)

As we can see the **highest Information Gain is given by Humidity**. Proceeding in the same way with [Decision Trees modified](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-55.jpg) will give us Wind as the one with highest information gain. The final Decision Tree looks something like this.

The final Decision Tree looks something like this.

[](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-16.jpg)

**Code:**

Let’s see an example in Python

**import** pydotplus

**from** sklearn.datasets **import** load\_iris

**from** sklearn **import** tree

**from** IPython.display **import** Image, display

\_\_author\_\_ = "Mayur Kulkarni <mayur.kulkarni@xoriant.com>"

**def load\_data\_set**():

*"""*

*Loads the iris data set*

***:return****:        data set instance*

*"""*

iris = load\_iris()

**return** iris

**def train\_model**(iris):

*"""*

*Train decision tree classifier*

***:param*** *iris:    iris data set instance*

***:return****:        classifier instance*

*"""*

clf = tree.DecisionTreeClassifier()

    clf = clf.fit(iris.data, iris.target)

**return** clf

**def display\_image**(clf, iris):

*"""*

*Displays the decision tree image*

***:param*** *clf:     classifier instance*

***:param*** *iris:    iris data set instance*

*"""*

dot\_data = tree.export\_graphviz(clf, out\_file=None,

                                    feature\_names=iris.feature\_names,

                                    class\_names=iris.target\_names,

                                    filled=True, rounded=True)

    graph = pydotplus.graph\_from\_dot\_data(dot\_data)

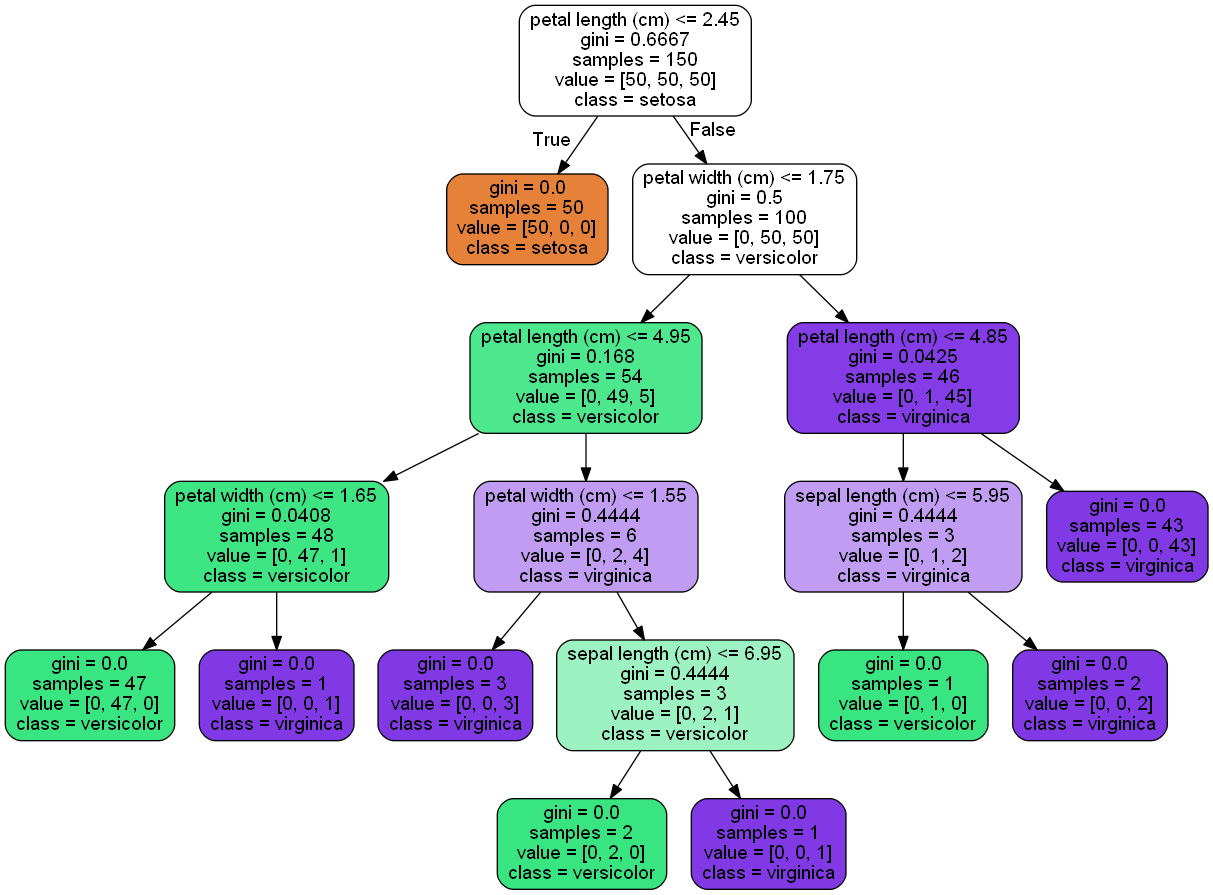
    display(Image(data=graph.create\_png()))

**if** \_\_name\_\_ == '\_\_main\_\_':

    iris\_data = load\_iris()

    decision\_tree\_classifier = train\_model(iris\_data)

    display\_image(clf=decision\_tree\_classifier, iris=iris\_data)

[](http://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-88.png)

**Conclusion:**

Below is the summary of what we’ve studied in this blog:

1. Entropy to measure discriminatory power of an attribute for classification task. It defines the amount of randomness in attribute for classification task. Entropy is minimal means the attribute appears close to one class and have a good discriminatory power for classification
2. Information Gain to rank attribute for filtering at given node in the tree. The ranking is based on high information gain entropy in decreasing order.
3. The recursive ID3 algorithm that creates a decision tree.

**References**:

1. Information Gain: <https://en.wikipedia.org/wiki/Information_gain_in_decision_trees>
2. Entropy: <https://en.wikipedia.org/wiki/Entropy_(information_theory)>
3. ID3: <https://en.wikipedia.org/wiki/ID3_algorithm>
4. Entropy YouTube video: <https://www.youtube.com/watch?v=O__7lAqni7A>
5. ID3 Example: <https://www.cise.ufl.edu/~ddd/cap6635/Fall-97/Short-papers/2.htm>

[Data Science](https://www.xoriant.com/blog/tag/data-science)[Decision Tree](https://www.xoriant.com/blog/tag/decision-tree)[Deep Learning](https://www.xoriant.com/blog/tag/deep-learning)[machine learning](https://www.xoriant.com/blog/tag/machine-learning)