Pragmatic Programming Techniques

Sunday, April 8, 2012

Machine Learning in R: Clustering

Clustering is a very common technique in unsupervised machine learning to discover groups of data that are "close-by" to each other. It is broadly used in customer segmentation and outlier detection.

It is based on some notion of "distance" (the inverse of similarity) between data points and use that to identify data points that are close-by to each other. In the following, we discuss some very basic algorithms to come up with clusters, and use R as examples.

K-Means

This is the most basic algorithm

1. Pick an initial set of K centroids (this can be random or any other means)
2. For each data point, assign it to the member of the closest centroid according to the given distance function
3. Adjust the centroid position as the mean of all its assigned member data points. Go back to (2) until the membership isn't change and centroid position is stable.
4. Output the centroids.

Notice that in K-Means, we not only require the distance function to be defined but also requiring the mean function to be specified as well. Of course, we also need K (the number of centroids) to be specified.

K-Means is highly scalable with O(n * k * r) where r is the number of rounds, which is a constant depends on the initial pick of centroids. Also notice that the result of each round is undeterministic. The usual practices is to run multiple rounds of K-Means and pick the result of the best round. The best round is one who minimize the average distance of each point to its assigned centroid.

Here is an example of doing K-Means in R

```
> km <- kmeans(iris[,1:4], 3)
> plot(iris[,1], iris[,2], col=km$cluster)
> points(km$centers[,c(1,2)], col=1:3, pch=8, cex=2)
> table(km$cluster, iris$Species)

setosa versicolor virginica
1      0         48        14
2     50          0         0
3      0          2        36
```

with the following visual output.

About Me

Ricky Ho

I am a software architect and consultant passionate in Distributed and parallel computing, Machine learning and Data mining, SaaS and Cloud computing.

View my complete profile

Popular Posts

MongoDb Architecture
NOSQL has become a very heated topic for large web-scale deployment where scalability and semi-structured data driven the DB requirement tow...

Designing algorithms for Map Reduce
Since the emerging of Hadoop implementation, I have been trying to morph existing algorithms from various areas into the map/reduce model. ...

NOSQL Patterns
Over the last couple years, we see an emerging data storage mechanism for storing large scale of data. These storage solution differs quite...

Couchbase Architecture
After receiving a lot of good feedback and comment on my last blog on MongoDb , I was encouraged to do another deep dive on another popular ...

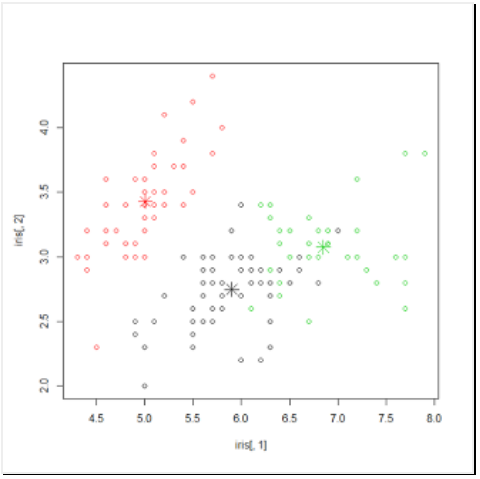
Predictive Analytics: Overview and Data visualization
I plan to start a series of blog post on predictive analytics as there is an increasing demand on applying machine learning technique to ana...

Predictive Analytics: Generalized Linear Regression
In the previous 2 posts, we have covered how to visualize input data to explore strong signals as well as how to prepare input data to a fo...

BigTable Model with Cassandra and HBase
Recently in a number of "scalability discussion meeting", I've seen the following pattern coming up repeatedly ... To make you...

Blog Archive

► 2013 (8)
▼ 2012 (18)
► November (1)
► October (1)



Hierarchical Clustering

In this approach, it compares all pairs of data points and merge the one with the closest distance.

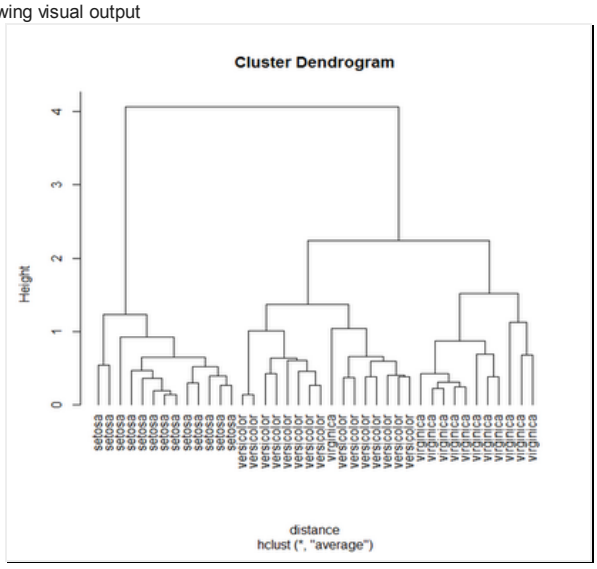
1. Compute distance between every pairs of point/cluster. Compute distance between pointA to pointB is just the distance function. Compute distance between pointA to clusterB may involve many choices (such as the min/max/avg distance between the pointA and points in the clusterB). Compute distance between clusterA to clusterB may first compute distance of all points pairs (one from clusterA and the other from clusterB) and then pick either min/max/avg of these pairs.
2. Combine the two closest point/cluster into a cluster. Go back to (1) until only one big cluster remains.

In hierarchical clustering, the complexity is O(n^2), the output will be a Tree of merge steps. It doesn't require us to specify K or a mean function. Since its high complexity, hierarchical clustering is typically used when the number of points are not too high.

Here is an example of doing hierarchical clustering in R

```
> sampleiris <- iris[sample(1:150, 40),]
> distance <- dist(sampleiris[,-5], method="euclidean")
> cluster <- hclust(distance, method="average")
> plot(cluster, hang=-1, label=sampleiris$Species)
```

with the following visual output



Fuzzy C-Means

Unlike K-Means where each data point belongs to only one cluster, in fuzzy cmeans, each data point has a fraction of membership to each cluster. The goal is to figure out the membership fraction that minimize the expected distance to each centroid.

$$J_m = \sum_{i=1}^{N}\sum_{j=1}^{C} u_{ij}^{m}\left\|x_i - c_j\right\|^2$$

The algorithm is very similar to K-Means, except that a matrix (row is each data point, column is

- ► September (1)
- ► August (2)
- ► July (1)
- ► June (3)
- ► May (3)
- ▼ April (3)
  - Basic graph analytics using igraph
  - Machine Learning in R: Clustering
  - MongoDb Architecture
- ► March (1)
- ► February (1)
- ► January (1)

► 2011 (6)
► 2010 (18)
► 2009 (31)
► 2008 (22)
► 2007 (11)

Search This Blog

Search

Labels

machine learning data mining map reduce Architecture Design Cloud computing algorithm NOSQL Hadoop scalability Distributed system parallel processing big data predictive analytics Design patterns SOA performance REST ensemble method recommendation engine

Pages

- Home

each centroid, and each cell is the degree of membership) is used.

1. Initialize the membership matrix U
2. Repeat step (3), (4) until converge
3. Compute location of each centroid based on the weighted fraction of its member data point's location.

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^{m} \cdot x_i}{\sum_{i=1}^{N} u_{ij}^{m}}$$

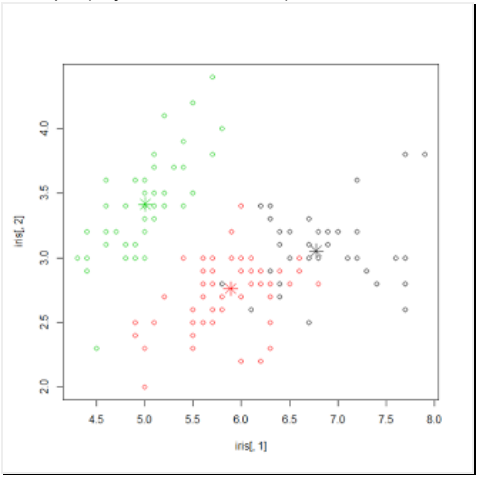
4. Update each cell as follows

$$u_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}}$$

Notice that the parameter m is the degree of fuzziness. The output is the matrix with each data point assigned a degree of membership to each centroids.
Here is an example of doing Fuzzy c-means in R

```
> library(e1071)
> result <- cmeans(iris[,-5], 3, 100, m=2, method="cmeans")
> plot(iris[,1], iris[,2], col=result$cluster)
> points(result$centers[,c(1,2)], col=1:3, pch=8, cex=2)
> result$membership[1:3,]
          1           2         3
[1,] 0.001072018 0.002304389 0.9966236
[2,] 0.007498458 0.016651044 0.9758505
[3,] 0.006414909 0.013760502 0.9798246
> table(iris$Species, result$cluster)
            1  2  3
setosa      0  0 50
versicolor  3 47  0
virginica  37 13  0
```

with the following visual output (very similar to K-Means)



Multi-Gaussian with Expectation-Maximization

Generally in machine learning, we will to learn a set of parameters that maximize the likelihood of observing our training data. However, what if there are some hidden variable in our data that we haven't observed. Expectation Maximization is a very common technique to use the parameter to estimate the probability distribution of those hidden variable, compute the expected likelihood and then figure out the parameters that will maximize this expected likelihood. It can be explained as follows ...

- Maximum Likelihood MLE
  - Pick Θ to maximize P(traindata | Θ)
- Handle partially observed data
  - What if traindata is not only partially observed with Z hidden and X observed ?
  - Now pick Θ to maximize P(X, Z | Θ) but we don't know what Z is.
  - We can only maximize $E_{over\ Z}(P(X, Z \mid \Theta))$. That is, pick Θ to maximize $E_{P(Z|X,\Theta)}(P(X, Z \mid \Theta))$
- Algorithm: Expectation Maximization
  - Iterate until converge
    - Compute P(Z|X,Θ)
    - Compute Θ' that maximize $E_{over\ Z}(P(X, Z \mid \Theta'))$
    - Set Θ to Θ'

Now, we assume the underlying data distribution is based on K centroids, each a multi-variate Gaussian distribution. To map Expectation / Maximization into this, we have the following.

- Imagine data point x_i is generated by the following process
  - Pick a centroid j with probability ϕ_j
  - From centroid$_j$, generate x_i from multi-variate normal distribution $N(\mu_j, \Sigma_j)$
- Recall: Pick Θ to maximize $E_{P(Z|X,\Theta)}(P(X, Z \mid \Theta))$
  - Parameter Θ is multi-normial ϕ_j , Normal distribution with mean μ_j and covariance matrix Σ_j
  - X = $\{x_i\}$ are observed data points
  - Z = $\{z_i\}$ are selected centroids (which are hidden)

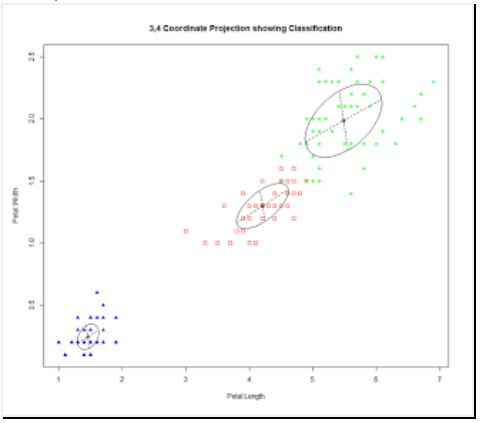
The order of complexity is similar to K-Means with a larger constant. It also requires K to be specified. Unlike K-Means whose cluster is always in circular shape. Multi-Gaussian can discover cluster with elliptical shape with different orientation and hence it is more general than K-Means.

Here is an example of doing multi-Gaussian with EM in R

```
> library(mclust)
> mc <- Mclust(iris[,1:4], 3)
> plot(mc, data=iris[,1:4], what=c('classification'),
       dimens=c(3,4))
> table(iris$Species, mc$classification)

            1  2  3
setosa     50  0  0
versicolor  0 45  5
virginica   0  0 50
```

with the following visual output



Density-based Cluster

In density based cluster, a cluster is extend along the density distribution. Two parameters is important: "eps" defines the radius of neighborhood of each point, and "minpts" is the number of neighbors within my "eps" radius. The basic algorithm called DBscan proceeds as follows

1. First scan: For each point, compute the distance with all other points. Increment a neighbor count if it is smaller than "eps".
2. Second scan: For each point, mark it as a core point if its neighbor count is greater than "minpts"
3. Third scan: For each core point, if it is not already assigned a cluster, create a new cluster and assign that to this core point as well as all of its neighbors within "eps" radius.

Unlike other cluster, density based cluster can have some outliers (data points that doesn't belong to any clusters). On the other hand, it can detect cluster of arbitrary shapes (doesn't have to be

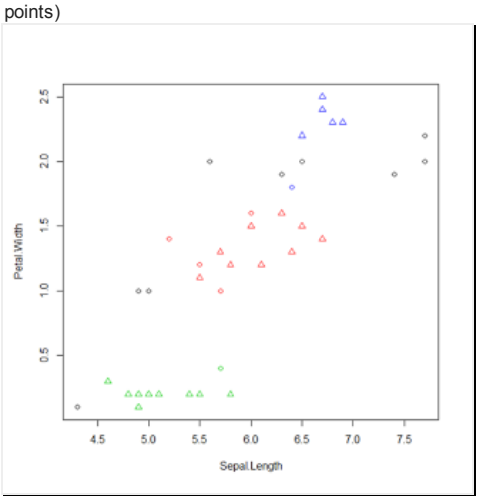
circular at all)

Here is an example of doing DBscan in R

```
> library(fpc)
# eps is radius of neighborhood, MinPts is no of neighbors
# within eps
> cluster <- dbscan(sampleiris[,-5], eps=0.6, MinPts=4)
> plot(cluster, sampleiris)
> plot(cluster, sampleiris[,c(1,4)])
# Notice points in cluster 0 are unassigned outliers
> table(cluster$cluster, sampleiris$Species)

setosa versicolor virginica
0      1          2         6
1      0         13         0
2     12          0         0
3      0          0         6
```

with the following visual output ... (notice the black points are outliers, triangles are core points and circles are boundary points)



Although this has covered a couple ways of finding cluster, it is not an exhaustive list. Also here I tried to illustrate the basic idea and use R as an example. For really large data set, we may need to run the clustering algorithm in parallel. Here is my earlier blog about how to do K-Means using Map/Reduce as well as Canopy clustering as well.

Posted by Ricky Ho at 10:10 PM

+7 Recommend this on Google

Labels: clustering, data mining, machine learning, R, unsupervised learning

2 comments:

Satpreet said...

Excellent article!

December 4, 2012 at 11:01 AM

Matthew Orlinski said...

Hey,

Thanks for this. I've followed it and found it really interesting. Just a quick comment.

In the **Multi-Gaussian with Expectation-Maximization** example the plot causes an error **formal argument "data" matched by multiple actual arguments**.

To solve it you can change the plot command to **plot(mc, what=c('classification'), dimens=c(3,4))**

April 2, 2013 at 10:26 AM

Post a Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)