Cracking

the

Data Science Interview

101+ Data Science Questions & Solutions

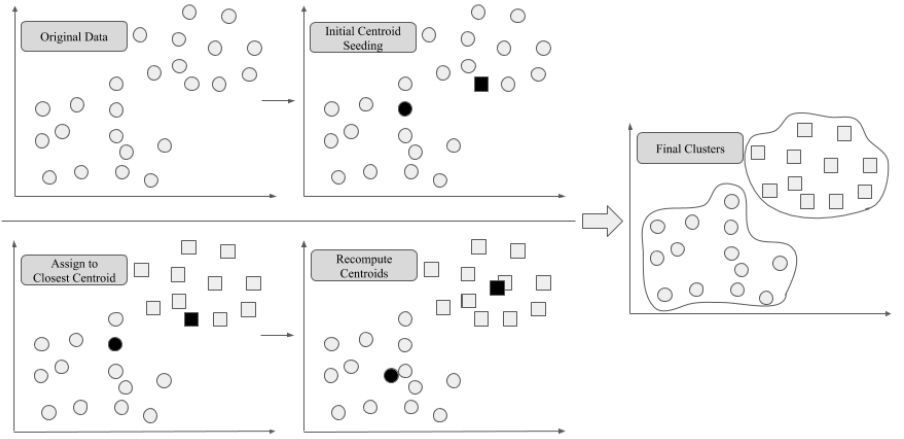
Maverick Lin

IX
Unsupervised Learning Algorithms

In unsupervised learning, the model receives the input X but no labeled output Y. What can we possibly do with just X? Well, we can actually learn representations of or detect patterns in the data. The findings can then be used in decision making, predicting future inputs, detect anomalies, etc...

► Clustering
Clustering is the problem of grouping data by similarity into *clusters*, which ideally reflect the similarities you are looking for. Of course, we must define what it means for two (or more) data points to be "similar" and "different"; this is usually done using a similarity/distance metric.

► k-Means Clustering
Intuition: We assume that there are k clusters or groups in your data and that each data points belongs to only one cluster. Therefore, a data point belongs to a cluster if that cluster's center (or centroid) is the closest cluster to that data point.



Algorithm
1. Choose a k. Select k distinct random points to serve as initial centroids.
2. Iterate until cluster assignments stop changing (or other stopping condition):
   (a) Assign each observation to the closest cluster centroid (closest defined by the distance metric).
   (b) For each of the k clusters, compute the new cluster centroid, which is the mean vector of all the observations assigned to cluster i.

Note: Since the results of the algorithm depend on the initial random centroid assignments, it is a good idea to repeat the algorithm from different random initializations to obtain the best overall results. We can use Mean-Squared Error (MSE) to determine which cluster assignment is better by taking the difference between all observations in a cluster and the cluster's centroid.

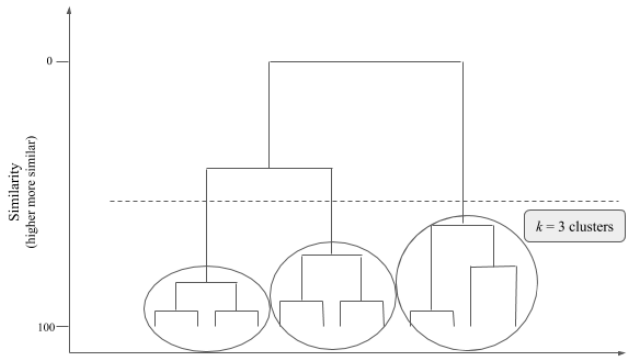
Advantages	Disadvantages
Simple	No optimal set of clusters
Interpretable	Sensitive to scale/outliers
Fast and efficient algorithm	Only handles numerical data
	Assumes there exists clusters to find
	Assumes spherical clusters (not elliptical, etc.)

Improving k-Means Clustering

- Scale or standardize data, remove outliers

▶ Hierarchical Clustering

Intuition: Suppose we don't want to commit to only having k clusters. An alternative is to build a hierarchy of clusters so you can decide the number of clusters later. An added benefit of this approach is that you also obtain a nice visualization (dendrogram) of how clusters are merged (or split) hierarchically; observations that fuse at the bottom are similar, where those at the top are quite different.



Algorithm (Agglomerative or Bottom-Up)

- Treat each observation as its own cluster, so we begin with n clusters
- Calculate the distance between all pairs of observations/clusters using one of the following linkage methods:
  - Distance between the average distances in each cluster (Average)
  - Maximum distance between two points in each cluster (Max)
  - Minimum distance between two points in each cluster (Min)
  - Distance between the centroids of each cluster (Centroid)

- While number of clusters > 1:
  - Find the two clusters closest to each other and merge them into one
  - Recompute the distances between clusters
- Obtain a dendrogram and determine clusters by cutting the tree at the desired level; each connected component then forms a cluster

Advantages	Disadvantages
Simple	Once two clusters are merged, cannot unmerge
Interpretable	Breaks large clusters
No need to choose k	Computationally expensive
Easy to implement	Assumes there exists clusters to find

Improving Hierarchical Clustering

- Scale or standardize data, remove outliers
- Try the top-down approach (Divisive Method): assign all observations to a single cluster and then split the cluster into two least similar clusters and repeat until there are n clusters (one cluster for every observation)
- Try a different linkage criteria (max, min, average, weighted average, etc.)

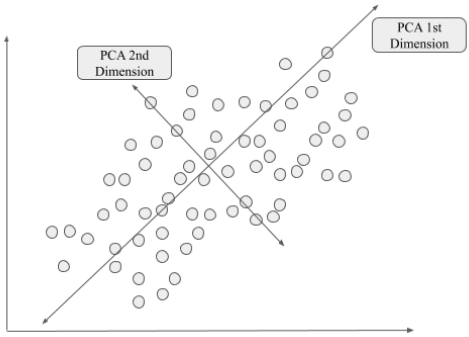
▶ Dimensionality Reduction

Dimensionality Reduction is the problem of reducing the number of features in your data to reduce storage space, lower computation cost, remove correlated features, and help better visualize the data.

▶ Principal Component Analysis (PCA)

Intuition: suppose we have a large dataset and we want to remove some features. PCA allows us to summarize a set of correlated features with a smaller set of independent features that collectively explain most of the data in the original set (or explains most of the variability); the idea of variability is important because higher variance along a dimension (or feature) means that more information is contained, which further implies that the dimension is more important.

Essentially, all we are doing is "dropping" the least important features by creating new, independent features in a lower dimensional space by taking the linear combination of the original features. The downside of such an approach is that the resulting independent variables are now less interpretable.

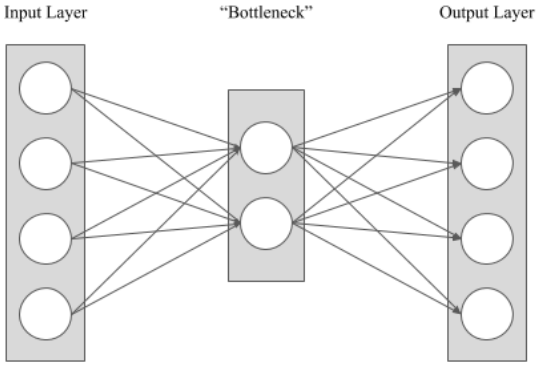


Algorithm
Select a dimension k where $k \leq D$ (D is the dimension of the original dataset)

- Compute the D-dimensional mean vector (or the mean of every feature)
- Compute the covariance matrix of the whole dataset: $\text{Cov}(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x}_i)(x_i - \bar{x}_i)^T$
- Compute eigenvectors $(e_1...e_D)$ and eigenvalues $(\lambda_1...\lambda_D)$
- Sort the eigenvectors by decreasing eigenvalues and select the largest k eigenvealues (and the corresponding eigenvectors) and create a D x k dimensional matrix W
- Use W to transform the original dataset into the new lower dimensional space: $y = W^T x$

Advantages	Disadvantages
Removes correlated features	Output is not easily interpretable
May improve algorithm performance	Must standardize data prior PCA
Helps visualize data in lower dimensions	Loses some information
May help to reduce overfitting	
May help reduce noise	

► Autoencoders
Intuition: mentioned briefly in the section for Neural Networks, but we'll go a little more in-depth here. While PCAs are by far the most popular method for dimensionality reduction, autoencoders can still be useful. Autoencoders work by compressing the dataset into a lower dimensional space and then recreating the original dataset from the lower dimensional representation. The training is done by comparing the reconstructed output to the original input and minimising a loss function (e.g. MSE). It is comprised of two parts: the encoder (transforms from high dimension to low dimension) and the decoder (transforms from low dimension back into high dimension).



Algorithm

- Initialize neural network and weights
- For 0...X training epochs:
  - Select input X and feed through the network to generate a reconstructed version Z
  - Calculate the loss between X and Z and update weights using gradient descent

Advantages	Disadvantages
Helps reduce dimensions	Uninterpretable
May learn non-linear feature representations	Prone to overfitting
May help reduce noise	Hard to train

▶ Self-Organizing Maps

Intuition: recall how neural networks are comprised of interconnected neurons (nodes). In this case, we also have a network of nodes (arranged in a bi-dimensional lattice) and each node contains a weight vector. We then iterate through the training data and after each observation, we select the node with the most similar weight vector to the observation and update that neuron's weight vector (and its neighbors) to be more like the observation. Over time, certain neurons' weight vectors will recognize certain patterns. The goal is to have the lower dimensional "map" approximate the original, higher-dimensional data.

Why bi-dimensional? The idea is to map higher-dimensional observations into a lower dimension (usually two-dimensions or three-dimensions) so we can visualize and interpret the result.

Algorithm

- Initialize the weights of each neuron randomly
- For 0...X training epochs:
  - Select an observation from the training data

- Find the most similar (or winning) neuron i for the observation using a distance metric
- Adjust the weights w_n of nearby neurons: $w_n = w_n + \alpha h(i)(x - w_n)$, where α is the learning rate and $h(i)$ is the neighborhood function that returns high values for neuron i and its neighbors

► Additional

The algorithms mentioned in this chapter are some of the more popular algorithms used. There are other more advanced algorithms such as: Gaussian Mixture Models, Hidden Markov Models, Expectation-Maximization Algorithm, t-SNE, General Bayesian Networks, etc... These are outside the scope of this book... for now.

► Interview Questions

9.1 What is the purpose of dimensionality reduction? What is it used for? What methods do you know?

9.2 What is the cluster analysis problem? What clustering methods do you know?

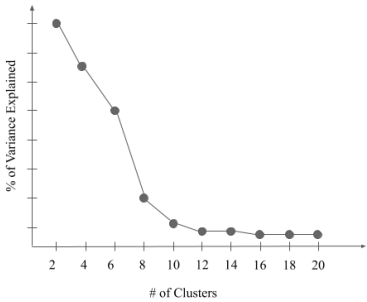
9.3 How do you take millions of Apple users (with hundreds of transactions each and among thousands of products) and group them into meaningful segments?

9.4 Explain what a local optimum is and why it is important in a specific context, such as k-means clustering. What are specific ways of determining if you have a local optimum problem? What can be done to avoid them?

9.5 Is feature scaling important prior to applying k-means clustering? If so, why?

9.6 What is a method for finding an optimal number of cluster in k-means? How do you determine the quality of a clustering?

9.7 What should be the best choice for number of clusters using the elbow method based on the following k-means results:



9.8 What are some stopping conditions for k-means?

9.9 Assume you want to cluster 7 observations into 3 clusters using k-means. After the first iteration, clusters C1, C2, C3 have the following observations:

- C1: (2, 1), (3, 2), (7, 3)
- C2: (2, 4), (5, 2)
- C3: (3, 3), (7, 7)

What are the new cluster centroids?

9.10 Explain how you would go about building a recommender system.

9.11 How is k-NN different from k-means clustering?