Lecture 6
More tricks of the trade: sparsification and clustering

Bingqing Cheng

January 2020

Outline

"big data" is not easy to handle not just because the data often lives in high dimension, but also because there are usually a large number of sample points. If the high dimensionality of the data is the only problem, then the kernel trick will make things a lot easier. However, a large amount of data slows down the training, make it difficult to represent them in an unbiased way, and are more prone to be contaminated by outliers.

In this lecture, we will learn about the methods to sparsify the data, which are also the same techniques that can be used to sparsify the features. We will also talk about the clustering techniques, which can be used to find patterns inside the data set as well as be used to find representative data points.

1 Sparsification

Sparsify the sample points of the data set and dimensionality reduction are the two sides of the same coin. To see why this is the case, datasets are often represented by large real-valued design matrices $\mathbf{X}$ with the dimension $N \times D$, where N is the number of sample points, and each sample point is described by D features. To reduce the number of features so as to compress the design matrix into dimension $N \times d$ with $d \ll D$, one often just keep the first d most important feature vectors, which can be linear or nonlinear combinations of the raw features. For example, principal components analysis (PCA) is just this procedure applied to a suitably normalized data correlation matrix .

Another way to build this connection is to think in terms of the *kernel trick*: the kernel matrix $\mathbf{K}$ encodes all the information that is needed to train a model in the feature space. Sparsifying the data points can be think of as finding a low dimensional embedding of the kernel matrix, and the dimensionality reduction introduced in the previous lecture is similar to finding a low rank matrix that best approximate the design matrix $\mathbf{X}$.

1.1 Singular value decomposition (SVD)

We have encountered SVD when talking about PCA, as well as used it in the Python notebook. Here is a brief recap.

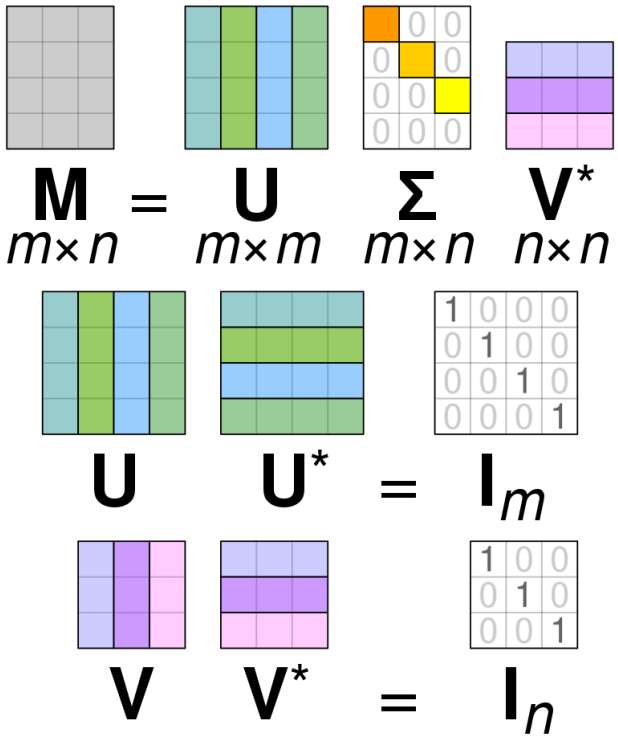

Figure 1: A schematic of singular value decomposition.

Recall the SVD of a general matrix $\mathbf{M} \in \mathcal{R}^{m \times n}$. Given $\mathbf{M}$, there exist orthogonal matrices $\mathbf{U} = [\mathbf{u}_2, \mathbf{u}_2, \ldots, \mathbf{u}_m]$ with $\mathbf{u}_i$ is a m-dimensional vector, and $\mathbf{V} = [\mathbf{v}_2, \mathbf{v}_2, \ldots, \mathbf{v}_n]$ with $\mathbf{v}_n$ is a n-dimensional vector such that

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \tag{1}$$

Here $\mathbf{\Sigma} = \mathbf{diag}(\sigma_1, \sigma_2, \ldots, \sigma_p), p = min(m, n)$ is a $m \times n$ diagonal matrix with the specified elements (σ_is) on the diagonal. The diagonal elements are all positive, and are ordered from the largest to smallest, that is $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$. The 3 matrices $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{\Sigma}$ constitute the SVD of $\mathbf{M}$: the σ_is are the singular values, the vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ are the left and right singular vectors, respectively.

The SVD expression can be equivalently wrote as,

$$\mathbf{U}^T\mathbf{M}\mathbf{V} = \mathbf{\Sigma}. \tag{2}$$

This form is more familiar due to its similarity with the eigenvalue equation in linear algebra.

Although the truncated SVD is widely used, the vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ may lack any meaning in terms of the field from which the data are drawn. For example, gene expression data, the eigenvector of images, and a linear combination of atomic environments are not particularly informative or meaningful. This fact should not be surprising. After all, the singular vectors are mathematical abstractions that can be calculated for any data matrix.

1.2 CUR Decomposition

CUR decomposition is a sparsification method that has been developed to deal with data where the information provided by the singular vectors cannot be properly interpreted. In analogy with the low-rank approximation obtained with a singular value decomposition, one writes

$$\mathbf{X} \approx= \mathbf{C}\,\mathbf{U}\,\mathbf{R} \tag{3}$$

such that $\mathbf{C}$ is made from actual columns of the original matrix $\mathbf{X}$, $\mathbf{R}$ is made from actual rows of $\mathbf{X}$. The objective is to find the best low-rank approximation to $\mathbf{X}$, but only using a small number of the actual data points.

Here we motivate the discussion for reducing the number of rows (i.e. data points) of the design matrix $\mathbf{X}$, but the CUR method can also be used to selecting a reduced number of columns (i.e. features).

Statistical Leverage Score The statistical leverage scores represent the column (or attribute) and row importance.

Each row r of the initial design matrix is given a normalized statistical leverage score calculated as

$$\pi_r = \frac{1}{k}\sum_{j=1}^{k}(\mathbf{u}_{j,r})^2, \tag{4}$$

where $\mathbf{u}_{j,r}$ is the r-th coordinate of the j-th left singular vector $\mathbf{u}_j$ of the of the left SVD matrix $\mathbf{U}$, and the sum is over $j = 1, 2, \ldots, k$ through the top k left singular vectors. k is called rank parameter here. Given that $\sum_{r=1}^{m}\pi_r = 1$, these scores form a probability distribution over the m rows.

Similarly, the normalized statistical leverage scores for all columns c are computed from the top k right singular vectors $\mathbf{v}_j$ as follows:

$$\pi'_c = \frac{1}{k}\sum_{j=1}^{k}(\mathbf{v}_{j,c})^2. \tag{5}$$

The importance score can also be weighted by a factor if one wants to prioritize the selection of a certain type of data points, e.g. if one wants to build a model that is most accurate for inputs in a certain range.

Row and column selection Most CUR schemes employ a probabilistic criterion for feature selection, to guarantee e.g. that if several nearly-identical features are present, any of them will have approximately the same probability of being selected. A typical workflow for selecting rows with high leverage scores (and reports their names, scores and ranks by importance) is described below:

Let's say $\hat{r}$ is the approximated (or expected) number of rows that users want to select. To realize row selections, first calculate the probability to select each row such as

$$\rho_r = \min(\pi_r, 1) \tag{6}$$

To make the selection deterministic, one can selected a column or row if the probability is greater than some threshold. Alternatively, for each row one can generate a random number from the uniform random distribution at $[0, 1]$ and select the row if the random number is smaller than the probability associated with that row.

1.3 Farthest Point Sampling

As a popular alternative, one can select the data points using a farthest-point sampling (FPS) approach. This is analogous to the strategy with which one can select uniformly-spaced reference points,so as to select fingerprints that are as diverse as possible for the data set being investigated. In a FPS scheme, successive points are chosen so as to maximize the Euclidean distance between them. After arbitrarily selecting the first fingerprint, each subsequent one is chosen as

$$k = \text{argmax}(\min_j |\mathbf{x}_k - \mathbf{x}_j|), \tag{7}$$

where j refers to all of the data points that have already been selected. The procedure is repeated until a pre-defined number of data points have been chosen.

2 Clustering and pattern recognition

2.1 Introduction: why automatic clustering

Clustering algorithms aim to recognize groups of input points that are related to each other, and different from other groups of inputs. For instance, clusters could represent different classes of molecules, or configurations of a given system that are separated by a sparsely-populated, or seldom accessed region.

Clustering is used in a wide variety of applications, for example:

Segmentation You can cluster your customers based on their purchases and their activity on your website. This is useful to understand who your customers are and what they need, so you can adapt your products and marketing campaigns to each segment. For example, customer segmentation can be useful in recommender systems to suggest content that other users in the same cluster enjoyed.

Reduce the load of data analysis When you analyze a new dataset, it can be helpful to run a clustering algorithm, and then analyze each cluster separately.

Dimensionality reduction Once a dataset has been clustered, it is usually possible to measure each instance's affinity with each cluster (affinity is any measure of how well an instance fits into a cluster). Each instance's feature vector x can then be replaced with the vector of its cluster affinities. If there are k clusters, then this vector is k-dimensional. This vector is typically much lower-dimensional than the original feature vector, but it can preserve enough information for further processing.

Outlier detection Any instance that has a low affinity to all the clusters is likely to be an anomaly. For example, if you have clustered the users of your website based on their behavior, you can detect users with unusual behavior, such as an unusual number of requests per second. Anomaly detection is particularly useful in detecting defects in manufacturing, or for fraud detection.

Semi-supervised learning If you only have a few labels, you could perform clustering and propagate the labels to all the instances in the same cluster. This technique can greatly increase the number of labels available for a subsequent supervised learning algorithm, and thus improve its performance.

2.2 Gaussian mixture model

One hint that data might follow a mixture model is that the data looks multimodal, i.e. there is more than one peak in the distribution of data. Trying to fit a multimodal distribution with a unimodal (one peak) model will generally give a poor fit, as shown in the example below. Since many simple distributions are unimodal, an obvious way to model a multimodal distribution would be to assume that it is generated by multiple unimodal distributions. For several theoretical reasons, the most commonly used distribution in modeling real-world unimodal data is the Gaussian distribution. Thus, modeling multimodal data as a mixture of many unimodal Gaussian distributions makes intuitive sense. Furthermore, GMMs maintain many of the theoretical and computational benefits of Gaussian models, making them practical for efficiently modeling very large datasets.

A Gaussian mixture model is parameterized by two types of values, i) the mixture component weights and ii) the component means and variances (or covariances if more than one dimension). For a Gaussian mixture model with K components, the k^{th} component has a mean of μ_k and a variance σ_k^2 (or a covariance matrix of $\mathbf{C}_k$ for the multivariate case). The mixture component weights are defined as ϕ_k are the component k, and they are normalized as $\sum_{i=1}^{K} \phi_k = 1$ so that the total probability distribution normalizes to 1. There

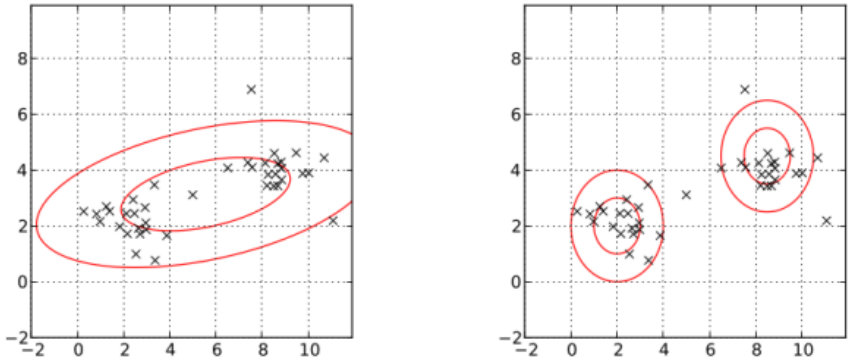

Figure 2: (Left) Fit with one Gaussian distribution (Right) Fit with Gaussian mixture model with two components.

are a number of algorithms that differ by how the weights and component means and variances are initialized and updated.

2.3 Kmeans

One of the simplest clustering methods, k-means clustering, can be seen as a special case of a GMM, in which the Gaussians are all taken to have the same diagonal covariance matrix, and a spread that is small compared to the distance between cluster centers.

How does the algorithm work? Suppose you are given the centroids, then You can easily label all the instances in the dataset by assigning each of them to the cluster whose centroid is closest. Conversely, if you are given all the instance labels, you can easily locate all the centroids by computing the mean of the instances for each cluster. In reality, you are given neither the labels nor the centroids, so one way to proceed is to work iteratively:

1) first placing the centroids randomly (e.g., by picking k instances at random and using their locations as centroids).

2) Then label the instances, update the centroids, label the instances, update the centroids, and so on until the centroids stop moving. The algorithm is guaranteed to converge in a finite number of steps (usually quite small).

You can see the algorithm in action in Figure 3: the centroids are initialized randomly, then the instances are labeled, then the centroids are updated, the instances are relabeled, and so on. As you can see, in just three iterations, the algorithm has reached a clustering that seems close to optimal.

2.4 DBSCAN

One of the shortfalls of the KMeans algorithm is that one has to pre-assign the number of clusters, but usually it is not obvious how many clusters there

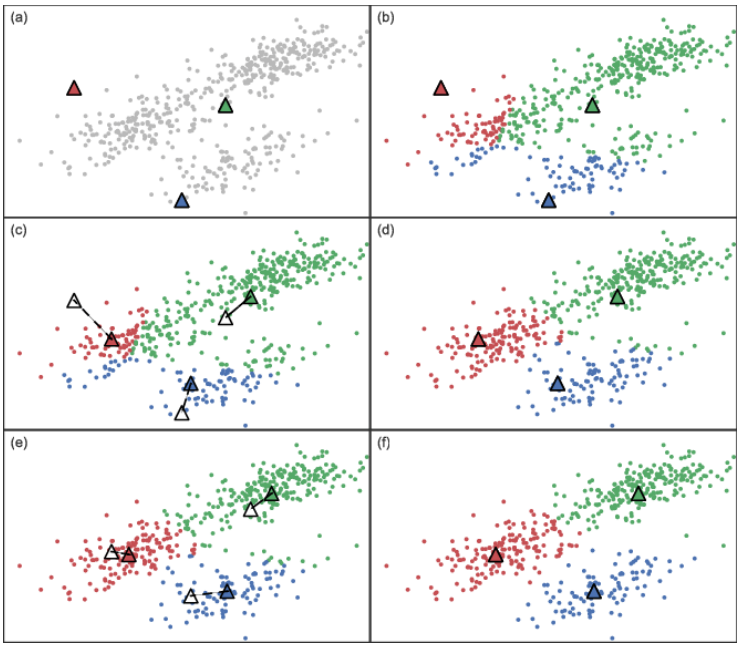

Figure 3: How the k-means clustering algorithm updates.

should be. Although there are some algorithms that allow the measurement of the quality of clustering (e.g. Silhouette analysis), so one can try out different number of clusters and then decide which one is the best, it is far from convenient. To circumvent this, one can use density-based clustering techniques, which does not require the number of clusters in the input parameters.

One commonly-used density-based clustering technique is DBSCAN, in which regions with high density are identified by checking whether many points fall within a prescribed distance of a given point, and contiguous regions with high density define separate clusters. Inputs that do not belong to any of these high-density regions are discarded as noise.

This algorithm defines clusters as continuous regions of high density. Here is how it works:

1) For each instance, the algorithm counts how many instances are located within a small distance (epsilon) from it. This region is called the instance's -neighborhood.

2) If an instance has at least minsamples instances in its ϵ-neighborhood (including itself), then it is considered a core instance. In other words, core instances are those that are located in dense regions.

3) All instances in the neighborhood of a core instance belong to the same cluster. This neighborhood may include other core instances; therefore, a long sequence of neighboring core instances forms a single cluster.

4) Any instance that is not a core instance and does not have one in its

neighborhood is considered an anomaly.

This algorithm works well if all the clusters are dense enough and if they are well separated by low-density regions.