Unsupervised Learning Review

Anthony R. Poggioli

In this notebook, I will review the unsupervised clustering and dimensionality reduction algorithms covered in the course Unsupervised Machine Learning, part of the IBM Machine Learning Professional Certificate. I will also include methods discussed in other resources, such as the `scikit-learn` page on clustering and Chapter 14 of *The Elements of Statistical Learning* by Hastie, Tibshirani, and Friedman. Note that the latter reference is an excellent introduction to the theoretical foundations of machine learning from a statistical perspective, and a PDF has been made available by Trevor Hastie at this page.

I am completing this review to solidify my understanding of the material covered in the Unsupervised Machine Learning course before completing a final project. These are my own reference notes. They are not comprehensive and may contain errors. I have not taken the time to generate figures to illustrate the concepts discussed in these notes.

Clustering Algorithms

K-Means

The k-means algorithm is the simplest clustering algorithm, and it is typically the first one introduced in a course on unsupervised learning. The steps of the algorithm are as follows:

1. *Parameter Selection*: The number of clusters k must be specified.
2. *Initialization*: Initial guesses for the k centroid locations must be specified. These are typically taken to be actual observations, but they can lie anywhere in the range of the data generating process.
3. *Cluster Mapping*: The Euclidean distance of each data point to every centroid is determined, and points are assigned to the cluster with the closest centroid.
4. *Centroid Shift*: The mean of all points in each cluster is determined, and the centroid of the cluster is shifted to this location.
5. *Convergence*: Steps 2 and 3 are repeated until convergence -- that is, until every cluster mean corresponds with the cluster centroid.

This algorithm seeks to minimize the *inertia* of the clustered data. Given k clusters indexed by $\alpha \in \{1, 2, \dots, k\}$ with centroids located at $\{\mu_\alpha\}$ and data points $\{\mathbf{x}_i\}$ indexed by $i \in \{1, 2, \dots, N\}$ and associated with the cluster α with the closest centroid, the inertia is defined as

$$I \equiv \sum_{\alpha} \sum_{i \in \alpha} ||\mathbf{x}_i - \mu_\alpha||_2^2,$$

where the sum over $i \in \alpha$ is understood as the sum over points $\mathbf{x}_i$ in the cluster indexed by α. Inertia as a measure of cluster coherence has several drawbacks:

- It assumes convex, isotropic clusters.
- It is not normalized in distance. This means that inertia will be sensitive to the inflated Euclidean distances associated with high-dimensional spaces
- It is not normalized by the number of points in each cluster. The algorithm therefore tends to favor clusters of roughly equal size.
- It is not normalized by the total number of points. In general, the inertia will become larger, suggesting worse performance, when the number of points is increased, even if the new points lie closer to the cluster centroids than the original points.

This metric can be compared to the *distortion*, which is the average squared distance of all points to their corresponding centrods. It is related to the inertia by

$$D \equiv \frac{1}{N} I$$

Unlke the inertia, this metric is normalized by the total number of points and does not in general increase when new points are introduced.

Given these definitions, some notes on the preceding algorithm are in order:

- *Parameter Selection*: Because this is an unsupervised method, there is no "ground truth" to compare our results to, and it is therefore difficult to tune hyperparameters like the number of clusters. One way to determine the optimal number of clusters is the *elbow method*, a heuristic method in which a measure of variance like inertia or distortion is plotted against the number of clusters. The measure of variance typically initially decreases rapidly in the number of clusters and then abruptly transitions to a slower rate of decrease. The "elbow" of this plot is identified as the slope break or the point where the curvature reaches its largest absolute value. The corresponding value of k is then taken as the optimal value. The elbow method sometimes fails. In this case, alternative methods like the silhouette method) can sometimes be applied.
- *Initialization*: K-means is only guaranteed to converge to a *locally optimal clustering*. Often, suboptimal clusterings are obtained because centroids are too close together resulting in overlapping clusters. One initialization scheme that seeks to mediate this by initializing with sufficiently distant points is as follows:
  1. Select the initial centroid for the first cluster from a uniform distribution across the data points. This point is designated μ_1^0.
  2. Select the initial centroid for the second cluster according to the distribution

$$p_i^{(2)} = \frac{\left|\left|\mathbf{x}_i - \mu_1^0\right|\right|_2^2}{\sum_i \left|\left|\mathbf{x}_i - \mu_1^0\right|\right|_2^2}$$

  3. Continue for each of the remaining $k-2$ centroids, choosing the initial location of the α-th centroid according to the distribution

$$p_i^{(\alpha)} = \frac{\min_{\beta \in \{1,2,\ldots,\alpha-1\}} \left|\left|\mathbf{x}_i - \mu_\beta^0\right|\right|_2^2}{\sum_i \min_{\beta \in \{1,2,\ldots,\alpha-1\}} \left|\left|\mathbf{x}_i - \mu_\beta^0\right|\right|_2^2}.$$

- *Cluster Mapping*: The use of Euclidean distance in mapping points to clusters and calculating centroid locations causes the algorithm to favor spherical, convex clusters, as discussed above. This also indicates that k-means is really only appropriate for continuous numerical data, though it is sometimes applied to numerically encoded categorical data.
- *Convergence*: The algorithm is guaranteed to converge in a finite number of steps; however, it is not guaranteed to converge to a *globally optimal clustering*, as noted above. This means that k-means algorithms should typically be run several times with different initializations, and the loss function (the inertia) should be compared to identify the most optimal clustering. This helps to avoid convergence to a highly suboptimal minimum.
- As a final note, mini-batch k-means uses random batching and parallelization to dramatically increase the computational efficiency of k-means at the expense of a(n often minimal) reduction in quality.

Gaussian Mixture Models

Gaussian mixture models (GMMs) can be understood as the "soft" version of k-means clustering because they return the *probability* of membership in each of the k clusters, rather than just the cluster membership of each point. A Gaussian mixture is a weighted sum of Gaussians with density function

$$p\left(\mathbf{x}\right) = \sum_\alpha \pi_\alpha \mathcal{N}\left(\mathbf{x} \,|\mu_\alpha, \Sigma_\alpha\right),$$

with

$$\sum_\alpha \pi_\alpha = 1.$$

In the above, $\mathcal{N}\left(\mathbf{x} \,|\mu_\alpha, \Sigma_\alpha\right)$ is the normal distribution with mean μ_α and covariance matrix Σ_α. GMMs are typically fitted using the expectation-maximization (EM) algorithm, an iterative maximum liklihood parameter estimation procedure. Once fitted, the probability of membership of point $\mathbf{x}_i$ in Gaussian cluster α is given by

$$P\left(\alpha \,|\mathbf{x}_i\right) = \frac{\pi_\alpha \mathcal{N}\left(\mathbf{x}_i \,|\mu_\alpha, \Sigma_\alpha\right)}{\sum_\alpha \pi_\alpha \mathcal{N}\left(\mathbf{x}_i \,|\mu_\alpha, \Sigma_\alpha\right)},$$

i.e., by the total density associated with Gaussian cluster α at $\mathbf{x}_i$ normalized by the total density at $\mathbf{x}_i$.

Often, the number of free parameters is constrained by imposing conditions on the covariance matrices Σ_α. In particular, the class `GaussianMixture` in `sklearn.mixture` allows for the following constraints on the covariance matrices:

- `full` **[default]**: Each component has its own general covariance matrix.
- `tied` : All components share the same general covariance matrix.

- `diag` : Each component has its own diagonal covariance matrix.
- `spherical` : Each component has its own single variance.

Hierarchical Agglomerative Clustering

Rather than a single set of k clusters and the (probability of) membership of each point in a cluster, *hierarchical clustering* returns a hierarchy of clusters, ranging from N singleton clusters for each data point to one cluster containing every point. This hierarchy of clusters is visualized with a *dendrogram*, in which the root node is the all-encomposing cluster and leaves are the singleton clusters. See the Wikipedia page on dendrograms and examples from the page on hierarchical clustering for more details.

Hierarchical clustering algorithms may be either *divisive* or *agglomerative*. Divisive algorithms start with the unique cluster containing every point and iteratively break it down into smaller clusters until the N singleton clusters are reached. Agglomerative algorithms are the inverse of divisive algorithms, beginning with the N singleton clusters and merging clusters until the unique global cluster has been reached.

In hierarchical agglomerative clustering, the manner in which clusters are merged is specified by the *linkage criterion*. This criterion specifies *how we assign the dissimilarity between clusters that contain more than one point*. The *least dissimilar* pair of clusters is then linked. Each of the linkage criteria will depend on an underlying metric defining distances between individual points. As we will see, three of the four linkage criteria offered in the `sklearn.cluster.AgglomerativeClustering` class will allow us to pick the underlying metric. The four offered linkages are:

- `ward` **[default]**: The dissimilarity between clusters is the inertia of the composite cluster. Given clusters α and β, we denote their composite as $\alpha\beta$. The centroid of the composite cluster is denoted $\mu_{\alpha\beta}$. The Ward linkage is then

$$D\left(\alpha,\beta\right) = I\left(\alpha\beta\right) = \sum_{i\in\alpha\beta} \left|\left|\mathbf{x}_i - \mu_{\alpha\beta}\right|\right|_2^2.$$

- `complete` : The dissimilarity between clusters is the *maximum* pairwise distance between points within the clusters. Given clusters α and β and a pairwise distance function $d\left(\mathbf{x}_i, \mathbf{x}_j\right)$ defined according to a specified metric, the cluster distance $D\left(\alpha,\beta\right)$ is defined as

$$D\left(\alpha,\beta\right) = \max_{i\in\alpha, j\in\beta} d\left(\mathbf{x}_i, \mathbf{x}_j\right).$$

- `average` : The dissimilarity between clusters is the *average* pairwise distance between points with the clusters. Given clusters α and β containg N_α and N_β points, respectively, the distance $D\left(\alpha,\beta\right)$ between these clusters is given by

$$D\left(\alpha,\beta\right) = \frac{1}{N_\alpha N_\beta} \sum_{i\in\alpha, j\in\beta} d\left(\mathbf{x}_i, \mathbf{x}_j\right).$$

- `single`: The dissimilarity between clusters is the *minimum* pairwise distance between points within the clusters. This linkage is easily skewed by outliers lying close to other clusters. Given clusters α and β, the distance $D\left(\alpha, \beta\right)$ between these clusters is given by

$$D\left(\alpha, \beta\right) = \min_{i \in \alpha, j \in \beta} d\left(\mathbf{x}_i, \mathbf{x}_j\right).$$

Single linkage is sensitive to outliers that lie in between clusters. Complete linkage, on the other hand, handles such outliers well, but at the expense of sometimes failing to agglomerate large clusters. Both averge linkage and Ward linkage offer a compromise between these extremes.

Of these four linkages, only Ward linkage makes explicit use of the Euclidean distance. The other three are compatible with arbitrary metrics/similarities.

Roughly, the hierarchical agglomerative clustering algorithm goes as follows:

1. Begin with the N singleton clusters.
2. Calculate the dissimilarity of every pair of clusters based on the specified linkage (and metric if applicable).
3. Merge the pair of clusters with the minimum dissimilarity.
4. Repeat steps 2 and 3 until either the maximal cluster containing all points or the specified stopping criterion has been reached. Early stoppage with more than one cluster can be achieved by specifying a maximum distance or minimum similarity below or above which clusters are no longer merged.

The main advantages of hierarchical agglomerative clustering are the generation of the clustering hierarchy and the ability of the algorithm to identify clusters of distinct sizes.

DBSCAN

DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise. This method is based on the empirical density of data points. Roughly speaking, it identifies clusters with local density maxima and assumes that distinct clusters are separated by regions of low density. DBSCAN is somewhat similar to the GMMs discussed above. GMMs are also density-based in that they parameterize a GMM density function by maximizing the liklihood of the observed density (via the EM algorithm). However, there are crucial differences between DBSCAN and GMMs. Unlike GMMs, DBSCAN is *nonparametric* -- that is, it does not impose a functional form for the density function and then fit the parameters of this function. Unlike GMMs, DBSCAN is not a soft clustering algorithm because it does not return the probability of cluster membership. However, it is more general than k-means. In identifying cluster membership, DBSCAN differentiates between *core points* and *non-core* or *density-reachable points* based on the local density in the neighborhood of each point and their proximity to a local density maximum. (The definition of core and non-core points will be explained below.) Furthermore, DBSCAN classifies points that are in regions of low density far from any local density maximum as *outliers*. Because it allows for points to be classified as outliers, DBSCAN is regarded as a *true clustering algorithm* rather than a *partitioning algorithm*.

The DBSCAN algorithm contains three essential hyperparameters:

1. *metric*: The metric by which distance between points is calculated.
2. N_{min}: The minimum number of neighbors within a distance ϵ of a point for that point to be considered a *core point*.
3. ϵ: The distance parameter introduced above, which specifies the neighborhood of each point.

Roughly speaking, DBSCAN approximates the density around each point by averaging the empirical density over a hypersphere of radius ϵ centered on that point. If the local density calculated in this manner exceeds $N_{\text{min}}/V_\epsilon^p \propto N_{\text{min}}/\epsilon^p$, where V_ϵ^p is the volume of a p-dimensional hypersphere of radius ϵ, then a region of high density has been identified and this point is labeled a core point. Of course, if a non-Euclidean metric is specified, the local volume will not be a hypersphere, but the volume will still scale as ϵ^p. More generally, ϵ defines the *ϵ-neighborhood* $\mathcal{N}_\epsilon\left(\mathbf{x}_i\right)$ of a point $\mathbf{x}_i$:

$$\mathcal{N}_\epsilon\left(\mathbf{x}_i\right) \equiv \left\{\mathbf{x} \left| d\left(\mathbf{x}, \mathbf{x}_i\right) < \epsilon\right.\right\}$$

where $d\left(\mathbf{x}, \mathbf{x}_i\right)$ is the distance between points $\mathbf{x}$ and $\mathbf{x}_i$ calculated according to some specified metric.

In addition to core points, non-core points are identified as those that:

1. Are neighbors of core points, and
2. Do not themselves have at least N_{min} neighbors.

Finally, outliers are defined as those points that:

1. Are not neighbors of core points, and
2. Are not themselves core points.

With these classifications, clusters are identified as *connected core and non-core points*. The DBSCAN algorithm goes roughly as follows:

1. Identify core points. Classification as a core point is based only on the local density, and points can therefore be classified as core points without knowing their cluster membership or which points are non-core points or outliers.
2. Select a random core point. This defines our first cluster.
3. Visit each neighbor of this core point, classifying them as core points, non-core points, or outliers. Every identified core and non-core point will be added to the same cluster.
4. Visit each neighbor of every subsequently identified core and non-core point, classifying them and adding them to the same cluster if they are core or non-core points.
5. After all points reachable within this cluster have been classified, randomly select another core point that has not yet been added to a cluster. This defines our next cluster.
6. Repeat steps 3 through 5 until all core points have been assigned to a cluster. Any remaining unclassified points are outliers.

DBSCAN is deterministic up to the ordering of clusters.

A key advantage of DBSCAN is that the number of clusters does not need to be specified *a priori*; however, this comes at the expense of introducing two new hyperparameters, N_{min} and ϵ. DBSCAN is particularly sensitive to ϵ, with small values tending to classify each point as an outlier and large values tending to place every point in the same cluster. The default value of ϵ will rarely be suited to a particular application. This can be understood intuitively by noting the following. First, a dataset represents an arbitrary number of samples drawn from some arbitrary generating distribution. The typical scale of the unnormalized empirical density of points can therefore vary by many orders of magnitude. Therefore, what constitutes a "high density" is highly specific to the particular problem under consideration.

Secondly, as discussed above, the "critical density" scales as $N_{\min}/\epsilon^p$, with p the number of features. Therefore, this critical density is anticipated to be highly sensitive to ϵ for small-to-intermediate values of this parameter. This will be especially true in high-dimensional feature spaces -- i.e., for large values of p.

Because it is based on local density rather than distance, DBSCAN does not implicitly assume convex clusters like k-means and GMMs. Additionally, because the metric can be specified, DBSCAN is applicable beyond continuous numerical data. DBSCAN is able to identify clusters of very different sizes, another advantage over k-means, but it cannot effectively handle clusters of different densities. Roughly speaking, it will only be able to identify local density maxima of similar magnitude.

Mean Shift

Like k-means, mean shift is a centroid-based algorithm, and it can be most easily understood by generalizing the k-means procedure. In k-means, we update a proposed centroid μ_α^t for cluster α at iteration t to the mean location of the points in the cluster at iteration t. The membership of a point in a given cluster is determined by the Euclidean distance from the proposed centroids: each point is placed in the cluster with the closest centroid. We define the cluster membership function

$$\mathcal{C}_t\left(\mathbf{x}\right) = \underset{\alpha}{\operatorname{argmin}} \left|\left|\mathbf{x} - \mu_\alpha^t\right|\right|,$$

which returns the cluster associated with a point $\mathbf{x}$. (We further specify that this function returns zero when a point is equidistant from more than one centroid.) With this function, we can define a cluster α at iteration t as

$$\mathcal{N}\left(\mu_\alpha^t\right) \equiv \left\{\mathbf{x}_i \in \mathcal{X} \left| \mathcal{C}_t\left(\mathbf{x}_i\right) = \alpha\right.\right\},$$

where $\mathcal{X} \equiv \{\mathbf{x}_i\}_{i=1}^N$ is the dataset. We adopt the notation $\mathcal{N}\left(\mu_\alpha^t\right)$ to emphasize that cluster membership defines a set of neighbors of the centroid μ_α^t. (If we considered all $\mathbf{x} \in \mathbb{R}^p$ and not just $\mathbf{x}_i \in \mathcal{X}$, $\mathcal{N}\left(\mu_\alpha^t\right)$ would define an *open neighborhood* about the point μ_α^t.) Averaging in k-means is done over this set of neighbors.

Furthermore, we weight each point in the cluster identically when calculating the cluster centroid in k-means. Generically, the weight of a point $\mathbf{x}$ relative to a second point $\mathbf{y}$ is given by the kernel function $K\left(\mathbf{x}, \mathbf{y}\right)$. In the k-means procedure, we take $K\left(\mathbf{x}_i, \mu_\alpha^t\right) = 1_{\mathcal{N}\left(\mu_\alpha^t\right)}\left(\mathbf{x}_i\right)$, where $1_{\mathcal{N}\left(\mu_\alpha^t\right)}\left(\mathbf{x}_i\right) \equiv \delta_{\alpha, \mathcal{C}_t\left(\mathbf{x}_i\right)}$ indicates membership of point $\mathbf{x}_i$ in the set $\mathcal{N}\left(\mu_\alpha^t\right)$. (Like the cluster membership function, this indicator function returns zero if the cluster membership is degenerate.) We see that we can equivalently define $\mathcal{N}\left(\mu_\alpha^t\right)$ as the support of the indicator function $1_{\mathcal{N}\left(\mu_\alpha^t\right)}\left(\mathbf{x}_i\right)$

.

Based on these definitions, we can write the centroid updating procedure for k-means as

$$\mu_\alpha^{t+1} = \mu_\alpha^t + \Delta \mathbf{m}_\alpha^t,$$

with the *mean-shift vector*

$$\Delta \mathbf{m}_\alpha^t = \frac{\sum_{\mathbf{x}_i \in \mathcal{X}} 1_{\mathcal{N}(\mu_\alpha^t)}(\mathbf{x}_i)\,\mathbf{x}_i}{\sum_{\mathbf{x}_i \in \mathcal{X}} 1_{\mathcal{N}(\mu_\alpha^t)}(\mathbf{x}_i)} - \mu_\alpha^t \equiv \frac{\sum_{\mathbf{x}_i \in \mathcal{N}(\mu_\alpha^t)} K(\mathbf{x}_i, \mu_\alpha^t)\,\mathbf{x}_i}{\sum_{\mathbf{x}_i \in \mathcal{N}(\mu_\alpha^t)} K(\mathbf{x}_i, \mu_\alpha^t)} - \mu_\alpha^t$$

where $\mathcal{N}(\mu_\alpha^t) \equiv \operatorname*{supp}_{\mathbf{x}_i} (K(\mathbf{x}_i, \mu_\alpha^t))$ is understood as the support of the kernel function on $\mathbf{x}_i \in \mathcal{X}$. With this notation, the k-means algorithm terminates when $\Delta \mathbf{m}_\alpha^t \equiv 0$ for every α.

This prescription allows us to generalize the k-means procedure to arbitrary kernel functions. The implementation of mean shift using the `sklearn.cluster.MeanShift` class uses a *flat kernel*, defined by the indicator function $K_\epsilon(\mathbf{x}, \mathbf{y}) \equiv 1(\mathbf{x} \in \mathcal{B}_\epsilon(\mathbf{y}))$ indicating membership in the ball $\mathcal{B}_\epsilon(\mathbf{x}) \equiv \{\mathbf{x} \mid \ ||\mathbf{x}||_2 < \epsilon\}$.

Another common implementation is Gaussian mean shift, in which the kernel function is given by $K(\mathbf{x}, \mathbf{y}) = \exp\left(-||\mathbf{x} - \mathbf{y}||_2^2 / c\right)$. Gaussian mean shift is an expectation-maximization algorithm. In either the flat kernel or the Guassian kernel, the respective width parameters -- ϵ and c -- are referred to as the *bandwidth* of the kernel.

There are crucial differences between the interpretations and implementations of k-means and mean shift. In k-means, the number of clusters k is pre-defined, and the neighborhood of a candidate centroid is based only on cluster membership (assigned according to the cluster membership function defined above). Because the neighborhood of the centroid is not explicitly distance based, centroids in k-means will settle on the geometric center of each cluster, but not necessarily on points of high density.

On the other hand, the number of clusters in mean shift is not determined *a priori*. (Instead, the bandwidth is the only hyperparameter.) Furthermore, the neighborhood of a candidate centroid is based on the kernel support on the data and is inherently sensitive to the distance of points from the candidate centroid. The *hill climbing procedure* in mean shift will therefore move towards regions of higher density. *Mean shift is thus a mode-seeking algorithm, and the centroids identified by the algorithm will be the local density maxima.*

The outline of the mean shift procedure is as follows:

1. Select a data point randomly. This is the first candidate centroid.
2. Calculate the weighted mean in the neighborhood of this candidate centroid and shift the centroid to the location of the weighted mean.
3. Repeat step 2 until convergence -- i.e., until the mean-shift vector vanishes.
4. Repeat steps 1 through 3 for the remaining data points.
5. Those points leading to (nearly) the same centroid are placed in the same cluster. (In `scikit-learn`, a post-processing step eliminates near-duplicates to arrive at the final set of centoids.)

Two final notes are important:

- Mean shift does not scale well because the algorithm requires mutliple nearest neighbor calculations.
- If no bandwidth is specified in `sklearn.cluster.MeanShift`, it is approximated using the function `sklearn.cluster.estimate_bandwidth`. This function is much less scalable than `MeanShift` itself.

Spectral Clustering

Spectral clustering is particularly useful when clusters are non-convex. Non-convex clusters are difficult to capture using ordinary clustering methods like k-means that are based on the inherently spherically symmetric Euclidean distance. The spectral clustering method clusters not on the data directly, but on the eigenvectors of a *graph Laplacian*. The graph is obtained from the *symmetric* $N \times N$ *similarity matrix* $S \equiv \left(s_{ij}\right)$, where $s_{ij} \in [0, 1]$ is a measure of the *similarity* between points i and j. For example, given the Euclidean distance $d_{ij} \equiv \left|\left|\mathbf{x}_i - \mathbf{x}_j\right|\right|_2$ between points i and j, we might consider a measure of similarity obtained from the Gaussian radial basis function as $s_{ij} \equiv \exp\left(-d_{ij}^2/c\right)$, where c is a scale parameter. Note that this similarity metric introduces a bandwidth or scale hyperparameter c.

Given this similarity matrix S, we first construct the symmetric set $\mathcal{N}_K$ of nearest neighbors -- i.e., a pair $(i, j) \in \mathcal{N}_K$ if i is one of the k most similar points to j *or vice versa*. Now, we construct the *adjacency matrix* (or *weight matrix*) $W \equiv \left(w_{ij}\right)$ by setting

$$w_{ij} = s_{ij} 1_{\mathcal{N}_K}(i, j),$$

where $1_{\mathcal{N}_K}(i, j)$ is an indicator function on the set $\mathcal{N}_K$ -- i.e., it returns one if the pair $(i, j) \in \mathcal{N}_K$ and zero otherwise. We construct a weighted, undirected graph -- called the *mutual k-nearest neighbor graph* -- from this adjacency matrix by creating a node for every data point and an edge between nodes i and j only if $w_{ij} > 0$. The weight of each edge is given by the value of w_{ij}. Note that we have introduced another hyperparameter, the nearest-neighbor cutoff k. For more distant neighbors, we set the edge weights to zero. If we take $k = N$, we consider the case that all nodes in our graph are connected with edge weights given by the values of s_{ij}.

We define the *degree* of node i as $g_i \equiv \sum_j w_{ij}$, the total weight of all edges leading to it, and the matrix $G \equiv \mathrm{diag}\left(g_i\right)$. We can then construct an *unnormalized graph Laplacian* $\mathcal{L}$ as

$$\mathcal{L} \equiv G - W.$$

In spectral clustering, we find the q eigenvalues of $\mathcal{L}$ -- excluding the trivial constant eigenvector -- corresponding to the q *smallest* eigenvalues. We then construct the matrix $Z_{N \times q}$ whose columns are given by these q eigenvectors and perform a traditional clustering analysis like k-means on the rows of $Z_{N \times q}$. Note that this introduces a third hyperparameter, the number of clusters k'.

The Elements of Statistical Learning has a nice plausibility argument for why spectral clustering works. I will not reproduce it here, but in essence, zero eigenvalues correspond to connected subgraphs, and small, nonzero eigenvalues correspond to regions of the graph that are internally highly connected but have few connections with the rest of the graph. Put another way, if we consider the graph we have constructed as a Markov state model with (symmetric) transition probabilities proportional to the edge weights, then a random walker released on this graph would dwell for a long time in the clusters we have identified, punctuated intermittently by (rare) transitions between clusters.

The major advantage of spectral clustering is that it allows for the identification of non-convex clusters. The major disadvantage is that it typically introduces four hyperparameters: the type of similarity function used, the scale parameter of that similarity function, the number of nearest neighbors k used in constructing the mutual k-nearest-neighbors graph, and the number of clusters k' to be identified. What's more, the results of this algorithm can be highly sensitive to these hyperparameters, and, as in all unsupervised tehcniques, tuning can be extremeley difficult.

Dimensionality Reduction Techniques

PCA

In this section, I will review the simple but powerful mathematics behind principal component analysis (PCA). This is important both to understand the methodology behind PCA and to extend PCA to nonlinear feature combinations in kernel PCA.

The fundamental problem in dimensionality reduction is as follows: We have an $N \times p$ data matrix X whose rows are our N observations or data points, with each data point composed of p features. We assume that X is *centered* -- that is, we assume that the values have been shifted such that the mean of each feature (column) is zero. Often, $p < N$, but this is not always the case. We would like to construct $q < p$ (not necessarily linear) combinations of the p raw features such that the new $N \times q$ data matrix Z captures the "essential characteristics" of the original data while being embedded in a lower-dimensional space. In the context of machine learning, dimensionality reduction is often performed to combat the "curse of dimensionality" -- i.e., the inherent sparseness of high-dimensional data. Of course, in the preceding description, we will have to specify a metric of "essential characteristics" and a structure for the combination of features.

PCA looks for linear combinations of features that explain the highest degree of variance in the data. This is done by the extension of eigendecomposition techniques to non-square matrices called singular value decomposition (SVD). SVD searches for a decomposition of an $N \times p$ matrix into two rotations (orthogonal transformations) and a rectangular diagonal matrix:

$$X = UDV^T,$$

where U and V are orthogonal matrices of size $N \times N$ and $p \times p$, respectively, and D is a rectangular diagonal matrix whose diagonal entries can be ordered such that $d_1 \geq d_2 \geq \cdots \geq d_{\min(N,p)} \geq 0$. The diagonal values $\{d_i\}_{i=1}^{\min(N,p)}$ are always real and nonnegative, and they are referred to as the *singular values* of X. The number of nonzero singular values corresponds to the rank of X. The columns $\{\hat{\mathbf{u}}_i\}$ of U and $\{\hat{\mathbf{v}}_i\}$ of V form orthonormal bases and are referred to as *left-singular vectors* and *right-singular vectors*, respectively. This decomposition is unique up to the ordering of the singular values and columns of U and V.

Note that we have assumed that X is a real-valued matrix. Everything above still applies to a complex-valued matrix if we replace "orthogonal" with "unitary" and the transpose operator V^T with the complex transpose operator $V^\dagger$.

The *principal components* of X are obtained by expanding the rows of X (i.e., the data points) in the right-singular vectors:

$$Z \equiv XV \equiv UD,$$

where Z is the matrix of principal component rows, and we have used the orthogonality of V to obtain the second equivalence from the SVD of X. Note that the columns of V span $\mathbb{R}^p$. Given the ordering of singular values from largest to smallest, *dimensionality reduction is achieved by truncating the expansion in the right-singular vectors at some value $q < p$ -- that is we take as our basis $\{\hat{\mathbf{v}}_i\}_{i=1}^{q<p}$*. This dimensionality reduction scheme selects the q directions of largest variance in the original data cloud and projects onto these directions.

The connection of PCA to the data variance can be understood more concretely by considering the sample covariance matrix $S \equiv N^{-1}X^TX$. Inserting the above decomposition into the definition of S, we find

$$S \equiv \frac{1}{N}X^TX = \frac{1}{N}\left(UDV^T\right)^T\left(UDV^T\right) = V\left(\frac{1}{N}D^TD\right)V^T.$$

It is straightforward to show that the matrix D^TD is necessarily square-diagonal, and we therefore recognize our final result as the eigendecomposition of S. The matrix $N^{-1}D^TD$ is therefore the matrix of eigenvalues of the sample covariance matrix, and the eigenvectors of the covariance matrix are likewise given by the columns of V -- i.e., by the right-singular vectors. The eigenvalues of S are given by $s_i = d_i^2/N$, and we see that large values of d_i correspond to linear combinations of the raw features that make large contributions to the total covariance $\mathrm{Tr}[S] = \sum_i s_i$.

Finally, we can also understand PCA through the eigendecomposition of the *Gram* or *inner-product matrix* $K \equiv XX^T$. The ij-th component of this matrix gives the inner product of data points i and j. Inserting the SVD of X into the definition of K, we obtain

$$K \equiv XX^T = \left(UDV^T\right)\left(UDV^T\right)^T = U\left(DD^T\right)U^T.$$

DD^T is an $N \times N$ square-diagonal matrix, and we therefore recognize our result as the eigendecomposition of K, with the eigenvalues given by d_i^2 and the eigenvectors by the columns of U -- that is, by the left-singular vectors. We can therefore obtain the principal component matrix $Z \equiv XV \equiv UD$ from this eigendecomposition. Note that we have assumed in our previous development that X is centered. If we have an uncentered Gram matrix K, we can calculate the *double-centered Gram matrix* $\tilde{K}$ as

$$\tilde{K} = \left(\mathbb{I}_N - M_N\right)K\left(\mathbb{I}_N - M_N\right),$$

where $\mathbb{I}_N$ is the $N \times N$ identity matrix, $M_N \equiv N^{-1}\mathbf{1}_N\mathbf{1}_N^T$ is an $N \times N$ matrix whose entries are all $1/N$, and $\mathbf{1}_N \equiv (1, 1, \ldots, 1)^T$ is a column vector of N ones. After double-centering K, we may apply the analysis outlined above to $\tilde{K}$.

To summarize, we have shown that there are three equivalent ways to obtain the principal component matrix Z from the data matrix X:

1. Directly from the SVD of X.
2. From the eigendecomposition of the sample covariance matrix S.
3. From the eigendecomposition of the Gram/inner-product matrix K.

The third of these methods will be the foundation for extending PCA to nonlinear feature combinations, outlined in the following section on kernel PCA.

Kernel PCA

The *kernel function* $K(\mathbf{x}_1, \mathbf{x}_2)$ is foundational to support vector machines and other kernel-based techniques. The function $K(\mathbf{x}_1, \mathbf{x}_2)$ is a measure of *similarity* between data points $\mathbf{x}_1$ and $\mathbf{x}_2$. It can also be considered as a Gram matrix in a high-dimensional space obtained by the nonlinear transformation of the data matrix X. In this context, we introduce the *nonlinear feature map* $\phi : \mathbb{R}^p \to \mathcal{V}$, where $\mathcal{V}$ is a high-dimensional inner-product space. In this case, the kernel function $K\left(\mathbf{x}_i, \mathbf{x}_2\right) \equiv \left\langle \phi\left(\mathbf{x}_1\right), \phi\left(\mathbf{x}_2\right)\right\rangle_{\mathcal{V}}$ is equivalent to the inner-product of $\mathbf{x}_1$ and $\mathbf{x}_2$ after they have been mapped to the high-dimensional space $\mathcal{V}$. Note that we do not need to know ϕ explicitly. Very roughly speaking, ϕ is guaranteed to exist if K is a symmetric positive-definite kernel. (See, e.g., the Wikipedia page on Mercer's Theorem for more details.)

Interpreting the kernel function as a Gram matrix in this implicitly defined high-dimensional inner-product space allows us to obtain the principal components in $\mathcal{V}$ according to the third procedure outlined in the preceding section. That is, we first double-center the kernel function:

$$\tilde{K} = \left(\mathbb{I}_N - M_N\right) K \left(\mathbb{I}_N - M_N\right),$$

where $K \equiv \left(K\left(\mathbf{x}_i, \mathbf{x}_j\right)\right)$ is the symmetric $N \times N$ matrix of values of the kernel function evaluated on all pairs of data points. We then obtain the eigendecomposition $\tilde{K} = U\left(D^2\right)U^T$ of the centered kernel matrix and construct the principal components as $Z = UD$. This technique is known as *kernel PCA*. There are a few things to note about this technique:

1. Though ϕ represents an implicit mapping to a high-dimensional space, we only need to consider the $N \times N$ matrix of inner products in this space -- not the high-dimensional vectors of $\phi\left(X\right)$ themselves. This is the key advantage of the "kernel trick" and kernel-based techniques generally.
2. Kernel PCA can still be used to find lower ($q < p$) dimensional representations of the data matrix X by truncating the columns of D. If we retain only the first q columns of D, then $Z = UD$ is the product of the $N \times N$ matrix U and the $N \times q$ matrix D, giving an $N \times q$ nonlinear principal component matrix.
3. The principal components obtained from the kernel are *nonlinear combinations* of the raw features. Potentially, this can allow for greater variance to be capture in the same number of principal components and hence for a better compressed representation of the data than that obtained from ordinary (linear) PCA.
4. Kernel PCA is also often the starting point for cluster analysis. A low-dimensional nonlinear representation of the data is constructed from kernel PCA and an ordinary clustering algorithm

like k-means is then performed on this representation. This is useful for data that is not linearly separable and/or for clusters that are non-convex, which are difficult to capture with ordinary clustering methods.

Multidimensional Scaling

Multidimensional scaling (MDS) seeks to map points in $\mathbb{R}^p$ to a lower dimensional *optimal manifold*, where optimality here is defined in terms of the preservation of *distances* or *dissimilarities* -- or sometimes *similarities* -- between points. Various loss functions are considered in identifying this optimal manifold.

Least Squares/Kruskal-Shephard Scaling

We seek values $\mathbf{z}_1, \ldots, \mathbf{z}_N$ on the lower dimensional manifold that minimize the *stress function*

$$S_M\left(\mathbf{z}_1, \ldots, \mathbf{z}_N\right) \equiv \sum_{i \neq j}\left(d_{ij} - \left|\left|\mathbf{z}_i - \mathbf{z}_j\right|\right|_2\right)^2,$$

where d_{ij} is the Euclidean distance between $\mathbf{x}_i$ and $\mathbf{x}_j$.

Sammon Mapping

This is a variation on least squares scaling in which we minimize

$$S_{\mathrm{Sm}}\left(\mathbf{z}_1, \ldots, \mathbf{z}_N\right) \equiv \frac{\sum_{i \neq j}\left(d_{ij} - \left|\left|\mathbf{z}_i - \mathbf{z}_j\right|\right|_2\right)^2}{d_{ij}}.$$

Here, the emphasis is on preserving small distances.

Classical Scaling

In classical scaling, we instead consider a measure of similarity s_{ij}, often taken to be the centered inner product $s_{ij} = \left\langle \mathbf{x}_i - \bar{\mathbf{x}}, \mathbf{x}_j - \bar{\mathbf{x}} \right\rangle$. We then minimize

$$S_C\left(\mathbf{z}_1, \ldots, \mathbf{z}_N\right) \equiv \sum_{ij}\left(s_{ij} - \left\langle \mathbf{z}_i - \bar{\mathbf{z}}, \mathbf{z}_j - \bar{\mathbf{z}} \right\rangle\right)^2.$$

Defining the matrix of squared Euclidean distances $D \equiv \left(d_{ij}^2\right)$ and the matrix of centered inner products $\tilde{K} \equiv \left(s_{ij}\right)$, one can show (*The Elements of Statistical Learning*, Section 18.5.2) that

$$\tilde{K} = -\frac{1}{2}\left(\mathbb{I}_N - M_N\right) D \left(\mathbb{I}_N - M_N\right).$$

If the similarities are centered inner-products, classical scaling is exactly equivalent to principal components.

Shephard-Kruskal Non-metric Scaling

The non-metric stress function is given by

$$S_{NM}\left(\mathbf{z}_1,\ldots,\mathbf{z}_N\right)\equiv\frac{\sum_{i\neq j}\left[\left|\left|\mathbf{z}_i-\mathbf{z}_j\right|\right|_2-\theta\left(d_{ij}\right)\right]^2}{\sum_{i\neq j}\left|\left|\mathbf{z}_i-\mathbf{z}_j\right|\right|_2^2},$$

where $\theta\left(d_{ij}\right)$ is an arbitrary increasing function. This stress function is sensitive to preserving the *rank* of similarities.

Non-negative Matrix Factorization

We can motivate non-negative matrix factorization (NMF) by reconsidering centroid-based clustering algorithms and PCA as means of developing a *compressed* representation of our $N\times p$ data matrix X. In the context of clustering, this compressed representation is obtained by replacing data points by the centroid associated with its cluster. The data matrix can then be written as

$$X\approx WH,$$

where W is an $N\times k$ matrix of *unitary* rows -- that is, rows that each contain exactly one non-zero element equal to one -- and H is a $k\times N$ matrix whose rows are composed of the k centroids, each of length N.

We can develop a similar formulation for PCA. Typically, we consider the lower-dimensional representation of our data matrix directly in PCA -- that is, we consider the principal components $Z_q=XV_q\in\mathbb{R}^{N\times q}$, where V_q is the matrix whose columns are composed of the first q eigenvectors of the sample covariance matrix. However, we can reexpress this lower-dimensional representation in the original feature space by multiplying the projections of the data points on the reduced dimensional matrix of covariance eigenvectors V_q by the first q eigenvectors of the sample covariance expressed in the coordinates of the original space. That is, we can write our compressed version of the original data matrix as $X=XV_qV_q^T$. To see this more concretely, consider

$$X=\begin{pmatrix}\mathbf{x}_1\\ \vdots\\ \mathbf{x}_N\end{pmatrix};\quad V_q=\begin{pmatrix}\mathbf{v}_1 & \ldots & \mathbf{v}_q\end{pmatrix}\Longrightarrow XV^T=\begin{pmatrix}\mathbf{x}_1\cdot\mathbf{v}_1 & \ldots & \mathbf{x}_1\cdot\mathbf{v}_q\\ \vdots & \ddots & \vdots\\ \mathbf{x}_N\cdot\mathbf{v}_1 & \ldots & \mathbf{x}_N\cdot\mathbf{v}_q\end{pmatrix},$$

where the $\mathbf{x}_i$'s are understood to be row vectors, the $\mathbf{v}_i$'s are understood to be column vectors, and the notation $\mathbf{a}\cdot\mathbf{b}\equiv\mathbf{ab}\equiv\mathbf{b}^T\mathbf{a}^T$ indicates the inner-product between a row vector $\mathbf{a}$ and column vector $\mathbf{b}$. Then multiplying V_q^T by this matrix gives

$$XV_qV_q^T=\begin{pmatrix}\sum_{i=1}^q\left(\mathbf{x}_1\cdot\mathbf{v}_i\right)\mathbf{v}_i^T\\ \vdots\\ \sum_{i=1}^q\left(\mathbf{x}_N\cdot\mathbf{v}_i\right)\mathbf{v}_i^T\end{pmatrix}$$

,

where the vectors $\mathbf{v}_i$ are of course expressed in the coordinates of the original feature space. This therefore gives a reduced representation of the original data cloud in the original embedding feature space. Defining $W\equiv XV_q$ to be our $N\times q$ matrix of weights and $H\equiv V_q^T$ to be our $q\times p$ matrix of basis vectors, we see that we again have the form

$$X \approx WH.$$

We note that the first of these procedures, centroid-based clustering, represents data points as a positive linear combination of representative points (the centroids) but is limited in that it identifies each point with one centroid (that corresponding to the point's cluster). On the other hand, PCA allows for a more general linear combination, but it allows for positive and negative contributions to this linear combination. This hinders the interpretability of the representative vectors (eigenvectors of the covariance matrix) when dealing with strictly positive data like text and images.

NMF decomposes a non-negative data matrix X into a weighted sum of non-negative components. It is again represented by the approximation

$$X \approx WH,$$

but with $x_{ij}, W_{ij}, H_{ij} \geq 0 \, \forall i, j$. It is more general than centroid-based clustering because it does not require the rows of W to be unitary, and it is more interpretable than PCA for non-negative data because the basis vectors (rows of H) are combined additively. This proceudre only makes sense if we require that the number of free components in WH be less than that in X itself. That is, we require $Nq + qp = q\left(N+p\right) < Np \Longleftrightarrow q < Np/\left(N+p\right)$. We also require that $q < \min\left(N, p\right)$. Otherwise, we could choose our q basis vectors to span $\mathbb{R}^p$ in the case that $p < N$ and recreate the data set exactly, or we could take our q basis vectors to be the N data points in the case $N < p$ and again recreate our data exactly. However, $Np/\left(N+p\right)$ is necessarily smaller than both N and p, and therefore the requirement

$$q < \frac{Np}{N+p}$$

is more stringent than the requirement that $q < \min\left(N, p\right)$.

In their 1999 *Nature* paper, Lee and Seung develop a loss function for fitting the above approximation by identifying the element $\left(WH\right)_{ij}$ associated with the j-th feature of data point i, x_{ij}, as the mean of a Poisson generating process for the non-negative data component x_{ij}. The Poisson distribution is a discrete distribution. We can get around this difficulty by introducing a discretization Δx, chosen to be small enough that $x_{ij}/\Delta x$ is guaranteed to be a whole number for all x_{ij}. The log of the Poisson distribution given the parameter $\left(WH\right)_{ij}$ is then given by

$$\log p\left(\frac{x_{ij}}{\Delta x}\middle| \left(WH\right)_{ij}\right) = \frac{x_{ij}\log\left(WH\right)_{ij} - \left(WH\right)_{ij}}{\Delta x} + f\left(x_{ij}, \Delta x\right),$$

with $f\left(x_{ij}, \Delta x\right)$ a function of x_{ij} and Δx only. Since we want to choose the $\left(WH\right)_{ij}$ to maximize the above liklihood, this function and the fixed denominator in the first term are irrelevant, and we can take our loss function as

$$L\left(W, H\right) = -\sum_{ij}\left[x_{ij}\log\left(WH\right)_{ij} - \left(WH\right)_{ij}\right].$$

Local minima of this loss function can be found iteratively by the algorithm of Lee and Seung (2001). This is discussed in Chapter 14 of *The Elements of Statistical Learning*.

The loss function implemented in `sklearn.decomposition.NMF` is distinct from the one given above (which is discussed in *The Elements of Statistical Learning*). `scikit-learn` allows for the objective function to be given by any of the *beta-divergences*. Divergences are measures of the dissimilarity between probability distributions, and the beta-divergences generalize classic divergences like the Kullback-Leibler (KL) divergence, Frobenius norm (or Euclidean distance), and the Itakura-Saito (IS) divergence. The beta-divergences are defined by

$$D_\beta\left(WH\,|X\right) \equiv \frac{1}{\beta\left(\beta-1\right)}\sum_{ij}\left[(WH)_{ij}^{\beta} + \left(\beta-1\right)x_{ij}^{\beta} - \beta(WH)_{ij}x_{ij}^{\beta-1}\right], \quad \beta\in\mathbb{R}\setminus\{0,1\}$$

In fact, the limits of this expression for $\beta\to 0,1$ are well-defined, and the beta-divergence can be made continuous by defining

$$\begin{aligned} D_0\left(WH\,|X\right) \equiv \lim_{\beta\to 0} D_\beta\left(WH\,|X\right) &= \sum_{ij}\left[\frac{(WH)_{ij}}{x_{ij}} - \log\frac{(WH)_{ij}}{x_{ij}} - 1\right] \\ &\equiv \sum_{ij}\frac{1}{x_{ij}}\left[(WH)_{ij} - x_{ij}\log\left(WH\right)_{ij}\right] + \sum_{ij}\left(\log x_{ij} - 1\right) \end{aligned}$$

and

$$D_1\left(WH\,|X\right) \equiv \lim_{\beta\to 1} D_\beta\left(WH\,|X\right) = \sum_{ij}\left[(WH)_{ij}\log\frac{(WH)_{ij}}{x_{ij}} - (WH)_{ij} + x_{ij}\right]$$

Both of these limits are obtained using the result $\lim_{\gamma\to 0}\left(x^\gamma - y^\gamma\right)/\gamma = \log(x/y)$. Though $D_0\left(WH\,|X\right)$ looks similar to the negative Poisson log-liklihood defined above, they are not equivalent due to the factor of $1/x_{ij}$ appearing in the first term. This can be seen by taking the gradient of $D_0\left(WH\,|X\right)$ with respect to W_{ij} and H_{ij}:

$$\partial_{W_{ij}} D_0 = \sum_k \frac{H_{jk}}{x_{ik}}\left(1 - \frac{x_{ik}}{\sum_l W_{il}H_{lk}}\right) = 0,$$

and

$$\partial_{H_{ij}} D_0 = \sum_k \frac{W_{ki}}{x_{kj}}\left(1 - \frac{x_{kj}}{\sum_l W_{kl}H_{lj}}\right) = 0.$$

We see that the $1/x_{ij}$ terms are still under a sum and cannot be cancelled.

$D_0\left(WH\,|X\right)$ is the IS divergence; $D_1\left(WH\,|X\right)$ reduces to the KL divergence when $\sum_{ij}(WH)_{ij} = \sum_{ij} x_{ij} = 1$ -- i.e., when we are considering normalized probability distributions.

Finally, we note that the beta-divergence for $\beta = 2$ gives the Frobenius norm of the quantity $WH - X$:

$$D_2\left(WH\,|X\right) = \frac{1}{2}\sum_{ij}\left[(WH)_{ij}^2 + x_{ij}^2 - 2(WH)_{ij}x_{ij}\right] \equiv \frac{1}{2}\sum_{ij}\left[(WH)_{ij} - x_{ij}\right]^2$$
$$\equiv \frac{1}{2}||WH - X||_{\text{Frob}}^2.$$

In addition to the specification of β in the loss function, the loss function in `sklearn.decomposition.NMF` allow for regularity conditions to be imposed on the components of W and H.

NMF is particularly suited to inherently non-negative data like text and images. In this case, the restriction of NMF to positive weights and components can be advantageous for interpretability. *The Elements of Statistical Learning* also discusses a now standard example from Lee and Seung (1999), in which they demonstrate that, unlike vector quantization -- which is equivalent to k-means -- and PCA, NMF decomposes images into individually interpretable features that can be reassembled into compressed versions of the original image.

The interpretability of NMF is hindered by the non-uniqueness of the factorization even in the case where $X = WH$ holds exactly. The features determined by NMF will in general be sensitive to the initialization, and this ambiguity makes the interpretation of the features so identified less clear.

Archetypal Analysis

This method represents data points as a *convex combination* of *prototypes* (the rows of H) that are themselves convex combinations of the original data points. The latter constraint forces the proptypes to lie in the *convex hull* of the data cloud. In this sense, the prototypes are "pure" or "archetypal" and referred to as *archetypes*.

Mathematically, we again have the approximation $X \approx WH$, where, in addition to the non-negativity constraints on the elements of W and H, we also require $\sum_j w_{ij} = 1\,\forall i$. Furthermore, as noted above, we require the prototypes to be convex combinations of the original data points:

$$H = BX$$

with $b_{ij} \geq 0\,\forall i, j$ and $\sum_j b_{ij} = 1\,\forall i$.

As with NMF, the optimization problem is typically non-convex and sensitive to initialization.

Archetypes identified by this procedure tend to lie on the relative boundary of the convex hull of the data cloud.