CS 181 Spring 2020 Section 7
Clustering
Solution

1 Part I

1.1 SVM Review

Support Vector Machines (SVMs) learn a decision boundary for binary classification problems using weight vector $\mathbf{w}$ and bias w_0. For some point $\mathbf{x}$, the boundary is defined by:

$$\mathbf{w}^\top \mathbf{x} + w_0 = 0$$

Given a setting of $\mathbf{w}, w_0$, we make a prediction on $\mathbf{x}$ by computing the discriminant function:

$$h(\mathbf{x}, \mathbf{w}, w_0) = \mathbf{w}^\top \mathbf{x} + w_0$$

and classify $\mathbf{x}$ as $y = 1$ if $h > 0$ and $y = -1$ otherwise. How good is our SVM? Of all the ways to cleanly separate the two classes when possible, we should pick a boundary that is furthest away from the closest points to the boundary! The *signed orthogonal distance* of $\mathbf{x}$ from boundary:

$$r(\mathbf{x}) = \frac{\mathbf{w}^\top \mathbf{x} + w_0}{||\mathbf{w}||}$$

When points are classified correctly, this is negative for points i with $y_i = -1$ and positive for those with $y_i = 1$. Then margin of the model is the smallest (over datapoints) *unsigned* distance. Multiply with y_i to get rid of signs:

$$\text{margin}(\mathbf{w}, w_0) = \min_i y_i r(\mathbf{x}_i) = \min_i \frac{y_i(\mathbf{w}^\top \mathbf{x} + w_0)}{||\mathbf{w}||} \quad \text{(want this to be large)}$$

The SVM objective is to maximize the margin (which itself is defined as a min over points!)! The $\frac{1}{||\mathbf{w}||}$ term can come out of the $\min_i$ since it doesn't depend on i. The objective to maximize is:

$$\max_{\mathbf{w}, w_0} \frac{1}{||\mathbf{w}||} \min_i y_i \left[\mathbf{w}^\top \mathbf{x}_i + w_0 \right]$$

In this hard-margin formulation (soft-margin out of scope for these notes), if the data is truly linearly separable than the margin satisfies:

$$\text{margin}(\mathbf{w}, w_0) = \min_i y_i r(\mathbf{x}_i) > 0.$$

Assuming data is separable, then the margin is some positive number. We can rescale $\mathbf{w}, w_0$ so that this positive margin is specifically 1 or greater without changing the orientation of the boundary. Then the boundary satisfies:

$$\text{margin}(\mathbf{w}, w_0) = \min_i y_i r(\mathbf{x}_i) > 1$$

Let's re-write the objective to make this constraint explicit (don't need to know how to justify to these re-writes, but see optimization theory if interested e.g. Stanford prof Boyd's videos on YouTube):

$$\max_{\mathbf{w}, w_0} \frac{1}{||\mathbf{w}||} \quad \text{s.t.} \quad \forall i, y_i(\mathbf{w}^\top \mathbf{x}_i + w_0) \geq 1$$

where s.t. stands for "such that" and $\forall_i$ means "for all i". A different-looking objective with the same solution is ($\mathbf{w}$ in numerator instead of denom):

$$\min_{\mathbf{w}, w_0} \frac{1}{2}||\mathbf{w}||^2 \quad \text{s.t.} \quad \forall i, y_i(\mathbf{w}^\top \mathbf{x}_i + w_0) \geq 1$$

We will shortly see why the min-rewrite is useful. With either of the last two objectives, we say they are "quadratic problems with linear constraints" which means they can be solved easily!

1.2 Dual Formulation of SVMs

We will now show an alternate view on the SVM problem. It will reveal an algorithm that look something analogous to KNN for regression. This new way of looking at SVMs will highlight some convenient ways of dealing with high-dimensional data! We left off with this loss function with constraints:

$$\min_{\mathbf{w}, w_0} \frac{1}{2}||\mathbf{w}||^2 \quad \text{s.t.} \quad \forall i, y_i(\mathbf{w}^\top \mathbf{x}_i + w_0) \geq 1$$

We use an idea called the Lagrangian (see optimization theory) to re-write this constrained objective so that some of the constraints end up in the main function to be optimized. We introduce *lagrange multipliers* a_i for each constraint (we have one for every datapoint) similarly to how you used λ in HW2 to write $\sum_k \pi_k = 1$ as $\lambda\Big(\sum_k \pi_k - 1\Big)$

$$\min_{\mathbf{w}, w_0} \left[\max_{a_i} \left(\frac{1}{2}\mathbf{w}^\top \mathbf{w} - \sum_i a_i \Big[y_i(\mathbf{w}^\top \mathbf{x}_i + w_0) - 1 \Big] \right) \right] \quad \text{s.t.} \quad a_i \geq 0$$

where we also used that $||\mathbf{w}||^2 = \mathbf{w}^\top \mathbf{w}$. Having changed the max to a min in the previous section and then applying the Lagrangian trick, we are left with a "min-max" problem that is quadratic with linear constraints. Beautifully/interestingly, some optimization theory tells us this can (finally) be re-written as:

$$\max_{a_i} \sum_i a_i - \frac{1}{2}\sum_i\sum_j a_i a_j y_i y_j (\mathbf{x}_i^\top \mathbf{x}_j) \quad \text{s.t.} \quad a_i \geq 0, \sum_i a_i y_i = 0$$

This has no **w** or w_0! Just a constrained optimization problem in terms of the a_i. We call the set of points $\{\mathbf{x}_i | a_i > 0\}$. the **support vectors** since they contribute to the objective value at its optimum.

How do we predict without $\mathbf{w}, w_0$? Hidden in the last step that removed $\mathbf{w}, w_0$ from the objective was the condition that $\mathbf{w} = \sum_i a_i y_i \mathbf{x}_i$. Hidden also was a condition that tells us to find any i with $a_i > 0$ and to set $w_0 = y_i \mathbf{w}^\top \mathbf{x}_i$. For a concise explanation see the cs181 SVM 2 lecture recap or David Sontag's MIT notes. Then our discriminant can be written as

$$\begin{aligned} h(\mathbf{x}, w, w_0) &= \mathbf{w}^\top \mathbf{x} + w_0 \\ \implies h(\mathbf{x}, a, w_0) &= \sum_i a_i y_i (\mathbf{x}_i^\top \mathbf{x}) + w_0 \end{aligned}$$

That is, to predict, we take dot products of the test point **x** with the dataset support vectors i.e. the set of $\mathbf{x}_i$ with $a_i > 0$. Qualitatively, this is like KNN: the complexity of prediction depends on the data rather than a fixed parameter vector. However, when the number of support vectors is small relative to the data dimension, this is cheap!

1.3 Basis Functions, Higher Dimensions, and Kernels

Suppose the data were not separable as-is but were separable using some basis ϕ. Lets just replace any **x** with $\phi(\mathbf{x})$ in the final objective:

$$\max_{a_i} \sum_i a_i - \frac{1}{2}\sum_i\sum_j a_i a_j y_i y_j \Big(\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)\Big) \quad \text{s.t.} \quad a_i \geq 0, \sum_i a_i y_i = 0$$

and in the discriminant function:

$$h(\mathbf{x}, a, w_0) = \sum_i a_i y_i \Big(\phi(\mathbf{x}_i)^\top \phi(\mathbf{x})\Big) + w_0$$

Suppose we needed to use a fairly high-dimensional basis function to achieve separability e.g. mapping to all powers up to 100. Well, notice that we don't explicitly need the values of each $\phi(\mathbf{x}_i)$ or $\phi(\mathbf{x})$ but we only need to know the result of the dot product of basis vectors on pairs. Then, we can directly define the **kernel function** for two vectors $\mathbf{x}, \mathbf{z}$

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^\top \phi(\mathbf{z})$$

We can pick K such that we can compute it without ever computing an individual $\phi(\mathbf{x})$. For example, let's take

$$K(\mathbf{x},\mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^3$$

Example: Poly Kernels Lets consider $\mathbf{x} \in \mathbb{R}^2$ and see how we would represent this as a basis dot product. Write $K(\mathbf{x},\mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^3 = (x_1 z_1 + x_2 z_2)^3$ as a dot-product $\phi(\mathbf{x})^\top \phi(\mathbf{z})$. How is $\phi()$ defined?

$$\begin{aligned}
(\mathbf{x}^\top \mathbf{z})^3 &= (x_1 z_1 + x_2 z_2)^3 \\
&= (x_1 z_1 + x_2 z_2)(x_1 z_1 + x_2 z_2)(x_1 z_1 + x_2 z_2) \\
&= \left(x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2\right)(x_1 z_1 + x_2 z_2) \\
&= x_1^2 z_1^2 (x_1 z_1 + x_2 z_2) + 2 x_1 z_1 x_2 z_2 (x_1 z_1 + x_2 z_2) + x_2^2 z_2^2 (x_1 z_1 + x_2 z_2) \\
&= x_1^3 z_1^3 + x_2^3 z_2^3 3 x_1^2 z_1^2 x_2 z_2 + 3 x_2^2 z_2^2 x_1 z_1 \\
&= \left(x_1^3, x_2^3, 3x_1^2 x_2, 3x_2^2 x_1\right)^\top \left(z_1^3, z_2^3, 3z_1^2 z_2, 3z_2^2 z_1\right)
\end{aligned}$$

which implies the basis $\phi(\mathbf{x}) = [x_1^3, x_2^3, 3x_1^2 x_2, 3x_2^2 x_1]$. But we only picked $\mathbf{x} \in \mathbb{R}^2$ and a degree 3 basis. More generally if the data dimension is D and degree is q we have $O(D^q)$ terms! But if we just compute the function K we don't first need to map to these high-dimensional bases. Put another way, we can pick functions K that imply the use of very high-dim bases!

1.3.1 What's a valid kernel?

When training SVMs, we begin by computing the kernel matrix $\mathbf{K}$, over our training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. The kernel matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$, defined as $K_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$, expresses the kernel function applied between all pairs of training points.

Mercer's theorem tells us that any function K that yields a **positive semi-definite kernel matrix forms a valid kernel**, that is, corresponds to a matrix of dot-products under *some* basis ϕ. Recall that a positive semi-definite matrix $\mathbf{K}$ requires $\mathbf{z}^\top \mathbf{K}\mathbf{z} \geq 0$, $\forall\, \mathbf{z} \in \mathbb{R}^n$. Therefore instead of using an explicit basis, we can build kernel functions directly that fulfill this property.

Example: Scaling to make a new kernel Suppose K is a valid kernel. Show that $K^{\text{new}}(\mathbf{x}, \mathbf{x}') = cK(\mathbf{x}, \mathbf{x}')$ for $c > 0$ is also a valid kernel. You can either show the positive semi-definite property or explicitly construct the basis.

We have the kernel matrix $\mathbf{K}^{\text{new}} = c\mathbf{K}$. We need $\mathbf{v}^\top \mathbf{K}^{\text{new}} \mathbf{v} = c\mathbf{v}^\top \mathbf{K} \mathbf{v} \geq 0$, which we know to be true because $\mathbf{K}$ is positive semi-definite and $c > 0$.
Alternatively, take $\phi^{\text{new}}(\mathbf{x}) = \sqrt{c}\, \phi(\mathbf{x})$.

1.4 Some more exercises

Exercise: Large Bases with Exp
Suppose $x \in \mathbb{R}$ and suppose we pick $K(x, x') = \exp(xx')$ where $\exp(z) = e^z$. If we rewrite $K(x, x') = \phi(x)^\top \phi(x')$ then how is the implied ϕ defined for this choice of K and what is the dimension of $\phi(x)$? *hint:* use

$$e^z = \lim_{i \to \infty} 1 + z + \ldots + \frac{z^i}{i!}$$

Solution:

Using the Taylor series expansion, we see that

$$K(x, x') = \exp(xx') = \lim_{i \to \infty} \left(1 + xx' + \cdots + \frac{(xx')^i}{i!}\right)$$

So by definition of dot product

$$K(x, x') = \left[1, x, \ldots, \frac{x^i}{i!}, \ldots\right]^\top \left[1, x', \ldots, \frac{(x')^i}{i!}, \ldots\right]$$

So therefore

$$\phi(x) = \left[1, x, \ldots, \frac{x^i}{\sqrt{i!}}, \ldots\right]^\top.$$

This choice of K implies infinite-dimensional bases! That is, there is no finite-dimensional vector ϕ such that $K(x, x') = \phi(x)^\top \phi(x')$!

Exercise: String Kernels
Let $\mathbf{s}$ and $\mathbf{s}'$ be strings. To measure how similar $\mathbf{s}$ and $\mathbf{s}'$ are, consider the "string kernel" $K(\mathbf{s}, \mathbf{s}')$, which returns the total number of distinct substrings (of any length) that $\mathbf{s}$ and $\mathbf{s}'$ have in common. For example, $K($`'aa'`$,$`'aab'`$) = 3$ because the substrings `''`, `'a'`, and `'aa'` are in common.

1. Compute $K($`'aza'`$,$`'zaz'`$)$.
2. What is the number of possible substrings of length 1, 2, and 3 in strings that are composed from a 26-letter alphabet?
3. Suppose we wanted to project a string into a higher-dimensional space such that we could represent via a 0 or 1 each of all possible substrings of length ≤ 3. How many dimensions would we need?
4. How does directly defining this string kernel help over computing the basis functions? Is it possible to compute the kernel itself efficiently?

Solution:

1. $K($`'aza'`$,$`'zaz'`$) = 5$ because substrings `''`,`'a'`,`'z'`,`'az'`,`'za'` are in common.
2. There are $26^1 = 26$ possible substrings of length 1, $26^2 = 676$ of length 2, and $26^3 = 17576$ of length 3.
3. Then $26 + 676 + 17576 = 18278$ features are required to represent all substrings of length ≤ 3.
4. In computing the kernel, we don't have to compute a feature representation for the data points (i.e. we don't have to find the presence/absence of each possible substring for $\mathbf{s}$ and $\mathbf{s}'$). Instead we can just write a program to find only the substrings that are in common. We avoid having to use costly representations to calculate the similarity between strings. This is the advantage of using kernel functions.

Exercise: Composing Kernels
A particularly nice corollary of Mercer's theorem is that it allows us to build more expressive kernels by composition. We already saw that positive scaling yields a new kernel. Now, use Mercer's theorem and the definition of a kernel matrix to prove that the following compositions are valid kernels, assuming $K^{(1)}$ and $K^{(2)}$ are valid kernels.

[Note: It suffices to show that a kernel is valid either by finding a particular $\boldsymbol{\phi}(\mathbf{x})$ that produces it, or by showing that the kernel matrix is positive semi-definite. Recall that a positive semi-definite matrix $\mathbf{K}$ requires $\mathbf{z}^\top \mathbf{K}\mathbf{z} \geq 0$, $\forall\ \mathbf{z} \in \mathbb{R}^n$.]

1. $K(\mathbf{x}, \mathbf{x}') = K^{(1)}(\mathbf{x}, \mathbf{x}') + K^{(2)}(\mathbf{x}, \mathbf{x}')$
2. $K(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})\, K^{(1)}(\mathbf{x}, \mathbf{x}')\, f(\mathbf{x}')$ where $f : \mathbb{R}^m$ to $\mathbb{R}$
3. $K(\mathbf{x}, \mathbf{x}') = K^{(1)}(\mathbf{x}, \mathbf{x}')\, K^{(2)}(\mathbf{x}, \mathbf{x}')$
   [Hint: Use the property that for any $\boldsymbol{\phi}(\mathbf{x})$, $K(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{x}')$ forms a positive semi-definite kernel matrix.]
4. $K(\mathbf{x}, \mathbf{x}') = \exp\left(K^{(1)}(\mathbf{x}, \mathbf{x}')\right)$
5. Finally use this analysis and previous identities to prove the validity of the Gaussian kernel:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{-||\mathbf{x} - \mathbf{x}'||_2^2}{2\sigma^2}\right)$$

Solution: In these solutions, let $\boldsymbol{\phi}^{(i)}(\mathbf{x})$ and $\mathbf{K}^{(i)}$ denote the underlying basis function and kernel matrix for kernel $K^{(i)}$, respectively.

1. We have the kernel matrix $\mathbf{K} = \mathbf{K}^{(1)} + \mathbf{K}^{(2)}$. We need

   $$\mathbf{v}^\top \mathbf{K}\mathbf{v} = \mathbf{v}^\top (\mathbf{K}^{(1)} + \mathbf{K}^{(2)})\mathbf{v} = \mathbf{v}^\top \mathbf{K}^{(1)}\mathbf{v} + \mathbf{v}^\top \mathbf{K}^{(2)}\mathbf{v} \geq 0,$$

   which we know to be true because $\mathbf{K}^{(1)}, \mathbf{K}^{(2)}$ are positive semi-definite.

   Alternatively, take $\boldsymbol{\phi}(\mathbf{x}) = \left[\phi_1^{(1)}(\mathbf{x}), \ldots, \phi_d^{(1)}(\mathbf{x}), \phi_1^{(2)}(\mathbf{x}), \ldots, \phi_d^{(2)}(\mathbf{x})\right]^\top$, the concatenation of $\boldsymbol{\phi}^{(1)}(\mathbf{x}), \boldsymbol{\phi}^{(2)}(\mathbf{x})$

2. Take $\boldsymbol{\phi}(\mathbf{x}) = f(\mathbf{x})\boldsymbol{\phi}^{(1)}(\mathbf{x})$.

3. Take $\boldsymbol{\phi}(\mathbf{x}) = \left[\phi_1^{(1)}(\mathbf{x})\phi_1^{(2)}(\mathbf{x}), \ldots, \phi_1^{(1)}(\mathbf{x})\phi_d^{(2)}(\mathbf{x}), \phi_2^{(1)}(\mathbf{x})\phi_1^{(2)}(\mathbf{x}), \ldots, \phi_d^{(1)}(\mathbf{x})\phi_d^{(2)}(\mathbf{x})\right]^\top$, the flattened vector for outer product $\boldsymbol{\phi}^{(1)}(\mathbf{x}) \otimes \boldsymbol{\phi}^{(2)}(\mathbf{x})$. The order of terms does not matter.

4. (a) We have

   $$K(\mathbf{x}, \mathbf{x}') = \exp\left(K^{(1)}(\mathbf{x}, \mathbf{x}')\right) = \exp\left(\boldsymbol{\phi}^{(1)}(\mathbf{x})^\top \boldsymbol{\phi}^{(1)}(\mathbf{x}')\right) = \prod_{i=1}^{d} \exp\left(\phi_i^{(1)}(\mathbf{x})\phi_i^{(1)}(\mathbf{x}')\right)$$

   We recognize the multiplicand to be a valid kernelThen, recognize that a product of valid kernels is a valid kernel. Alternatively, write

   $$K(\mathbf{x}, \mathbf{x}') = \exp(K^{(1)}(\mathbf{x}, \mathbf{x}')) = \lim_{i\to\infty} \left(1 + \boldsymbol{\phi}^{(1)}(\mathbf{x})^\top \boldsymbol{\phi}^{(1)}(\mathbf{x}') + \cdots + \frac{(\boldsymbol{\phi}^{(1)}(\mathbf{x})^\top \boldsymbol{\phi}^{(1)}(\mathbf{x}'))^i}{i!}\right)$$

   which can also be recognized to be valid given our previous conclusions.

5.
   - $K_0(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$ is a valid kernel by definition of the kernel (it is the inner product of $\mathbf{x}$ and $\mathbf{x}'$).
   - Thus $K_1(\mathbf{x}, \mathbf{x}') = \exp(2\mathbf{x}^\top \mathbf{x}')$ is also a valid kernel
   - Note that $K(\mathbf{x}, \mathbf{x}') = \exp(-\mathbf{x}^\top \mathbf{x})\exp(2\mathbf{x}^\top \mathbf{x}')\exp(-\mathbf{x}'^\top \mathbf{x}') = f(\mathbf{x})K_1 f(\mathbf{x}')$, where $f(\mathbf{x}) = \exp(-\mathbf{x}^\top \mathbf{x})$.
   - We proved $K(\mathbf{x}, \mathbf{x}')$ is a kernel.

   Note: a common mistake is saying $\exp(-\mathbf{x}^\top \mathbf{x})$ is a kernel. It is not.

2 Part II

2.1 Motivation

We now move onto **unsupervised learning**, where the objective is to learn the structure of unlabeled data. In other words, we are looking for groups, or **clusters** among the data. Clustering algorithms are useful not only for finding groups in data, but also to **extract features** of the data that summarize the most important information about the data in a compressed way.

2.2 Setup

For most clustering algorithms, we need some kind of a **metric** to specify the notion of "distance" between the data points.

Now that the metric is well-defined, the next thing we need to do is to decide **how many groups we want**. Sometimes you know the ideal number of groups in advance (*e.g.* clustering the 26 letters in the alphabet). Other times, you need to decide if you'd like a more compressed representation with more information loss by having the number of groups small, or a less compressed representation with less information loss by having the number of groups large.

Suppose our data set is $\{\mathbf{x}_i\}_{i=1}^{n}$, then our objective is to find the ideal assignment of the data set to the clusters, by assigning to each of the n data points, a binary **responsibility vector** $\mathbf{r}_i$, which is all zeros except one component, which corresponds to the assigned cluster.

2.3 K-Means Algorithm

The idea is to represent each cluster by the point in data space that is the average of the data assigned to it. For some choice of K and random initialization of clusters, the K-Means Algorithm (also called Lloyd's algorithm) is:

Repeat until convergence (none of the responsibility vectors change):

1. For each data point, update its responsibility vector by assigning it to the cluster with the closest mean.
2. For each cluster, $\{\boldsymbol{\mu}_k\}_{k=1}^{K}$, update its mean to be the mean of the data points currently assigned to that cluster.

2.3.1 Derivation

We begin by defining a loss function that the K-Means Algorithm minimizes via coordinate descent:

$$\mathcal{L}(\{\mathbf{r}_i\}_{i=1}^n, \{\boldsymbol{\mu}_k\}_{k=1}^K) = \sum_{i=1}^n \sum_{k=1}^K r_{ik}||\mathbf{x}_i - \boldsymbol{\mu}_k||^2$$

First, we want to choose r_i that minimizes the loss, holding all else constant. This is when we assign data points to the clusters with means closest to them:

$$r_{ik} = \begin{cases} 1 & \text{if } k = argmin_{k'}||\mathbf{x}_i - \boldsymbol{\mu}_{k'}|| \\ 0 & \text{otherwise} \end{cases}$$

This is the first step of each iteration of the K-means algorithm!

Second, we want to choose μ_k that minimizes the loss, holding all else constant. For a given k, the squared loss is:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\mu}_k) &= \sum_{i=1}^n r_{ik}||\mathbf{x}_i - \boldsymbol{\mu}_k||^2 \\ &= \sum_{i=1}^n r_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k)^T(\mathbf{x}_i - \boldsymbol{\mu}_k) \end{aligned}$$

Taking the derivative and setting it to zero,

$$\begin{aligned} \frac{\partial \mathcal{L}(\boldsymbol{\mu}_k)}{\partial \boldsymbol{\mu}_k} &= -2\sum_{i=1}^n r_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k) = 0 \\ \boldsymbol{\mu}_k &= \frac{\sum_{i=1}^n r_{ik}\mathbf{x}_i}{\sum_{i=1}^n r_{ik}} \end{aligned}$$

This is the second step of each iteration of the K-means algorithm!

2.3.2 Number of Clusters

There is not an especially well justified method to choose the number of clusters when using K-means. One approach is to plot K vs the objective criterion, and look for a “knee” or “kink” where progress slows down.

2.3.3 Notes

Lloyd’s algorithm finds a locally optimal solution.

It is generally a good idea to **standardize** the data to account for unsatisfying result due to dimension mismatch.

Lastly, when for the metric we are using for the given data set, a "mean" does not make sense, we might instead use a **K-Medoids Algorithm**. This algorithm requires the cluster centers to be a data point in the data set.

2.4 Hierarchical Agglomerative Clustering

Hierarchical clustering constructs a tree over the data, where the leaves are individual data items, while the root is a single cluster that contains all of the data. When drawing the dendrogram, for the clustering to be valid, the distances between the two groups being merged should be monotonically increasing. The algorithm is as follows:

1. Start with n clusters, one for each data point.
2. Measure the distance between clusters. This will require an inter-cluster distance measurement that we will define shortly.
3. Merge the two 'closest' clusters together, reducing the number of clusters by 1. Record the distance between these two merged clusters.
4. Repeat step 2 until we're left with only a single cluster.

The main decision in using HAC is what the distance criterion should be between groups.

2.4.1 The Min-Linkage Criterion

For two groups indexed by i and i', the idea is to merge groups based on the shortest distance over all possible pairs:

$$DIST_{\min}(\{\mathbf{x}_i\}_{i=1}^{n}, \{\mathbf{x}_{i'}\}_{i'=1}^{n'}) = \min_{i,i'} ||\mathbf{x}_i - \mathbf{x}_{i'}||.$$

2.4.2 The Max-Linkage Criterion

For two groups indexed by i and i', the idea is to merge groups based on the largest distance over all possible pairs:

$$DIST_{\max}(\{\mathbf{x}_i\}_{i=1}^{n}, \{\mathbf{x}_{i'}\}_{i'=1}^{n'}) = \max_{i,i'} ||\mathbf{x}_i - \mathbf{x}_{i'}||$$

2.4.3 The Average-Linkage Criterion

For two groups indexed by i and i', the idea is to average over all possible pairs between the groups:

$$DIST_{\text{avg}}(\{\mathbf{x}_i\}_{i=1}^{n}, \{\mathbf{x}_{i'}\}_{i'=1}^{n'}) = \frac{1}{nn'} \sum_{i=1}^{n} \sum_{i'=1}^{n'} ||\mathbf{x}_i - \mathbf{x}_{i'}||$$

2.4.4 The Centroid-Linkage Criterion

For two groups indexed by i and i', the idea is to look at the difference between the groups' centroids:

$$DIST_{\text{cent}}(\{\mathbf{x}_i\}_{i=1}^{n}, \{\mathbf{x}_{i'}\}_{i'=1}^{n'}) = || \left(\frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \right) - \left(\frac{1}{n'} \sum_{i'=1}^{n'} \mathbf{x}_{i'} \right) ||$$

2.4.5 Exercise on your own: K-means and HAC

What are three important differences between K-means and HAC?

K-Means has exactly K clusters whereas HAC can be used in a way where the number of clusters are determined after the fact, via inspecting the dendrogram. K-Means is randomized: the final clusters depend on the initial random centers whereas HAC is deterministic. HAC forms a hierarchy, which can provide additional understanding relative to a flat clustering.

2.4.6 Exercise on your own: Scaling to Large Dimensions

$\boxed{\text{Explain the ‘curse of dimensionality’ and how it is related to HAC.}}$

The curse of dimensionality refers to the problem where distances become meaningless in very large dimensional spaces. The problem is that the distance between two examples with some informative features but lots and lots of random features will be approximately the same (try this out in simulation if you don’t see why!).

This means that non-parametric (‘instance based’) methods such as HAC that use pairwise distances between examples (vs between examples and prototypes in K-means) become less useful in higher dimensions.