Foundations of Data Science and Machine Learning – *Homework 3*

Isaac Martin

Last compiled February 24, 2023

EXERCISE 1. (Completed this one last, too lazy to copy down the problem statement.)

Solution: There isn't a lot to say in this problem, it is mostly implementation and experimentation. We discuss our process and our results. The code is attached to the end of this document. Throughout this problem a kmeans++ initialization was used with 10 iterations.

(a) We first generate a matrix C (named "X" in the code) as the requested block matrix. We then create the random matrix A by drawing from a Bernoulli distribution at each entry, using the entry's value of 0.3 or 0.7 as the probability in the Bernoulli distribution. Finally, we generate a random permutation matrix P and conjugate A by P. We save the inverse of P for use later in "unshuffling" the labels generated by kmeans in each case.

We then use scikit-learn's implementation of kmeans to cluster the rows of A and store both the cost of the final clustering and the cluster labels for each row. By applying the inverse permutation to the list of labels, we can easily compare it to the natural permutation in the problem statement. Here is the unshuffled list of labels:

```
UNSHUFFLED LABELS:
 [2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1.]
```

Figure 1: List of labels generated by kmeans

It should resemble a list which looks like $[0, 0, ..., 0, 1, 1, ..., 1, 2, 2, ..., 2]$, and indeed it does, up to rearranging the 0's, 1's and 2's. Note that the index on the cluster is arbitrary, so this rearrangement is to be expected.

(b) We now apply the kmeans algorithm as done in part (a) for many different values of k and plot the costs in each case. People have noticed that the resulting graph often looks like an elbow – the cost drops tremendously at a certain choice of k, after which the addition of extra clusters only incrementally improves the cost of the obtained clustering. This is precisely what we observe here. The "elbow" point occurs at $k = 3$, meaning this is a good choice for the number of clusters.

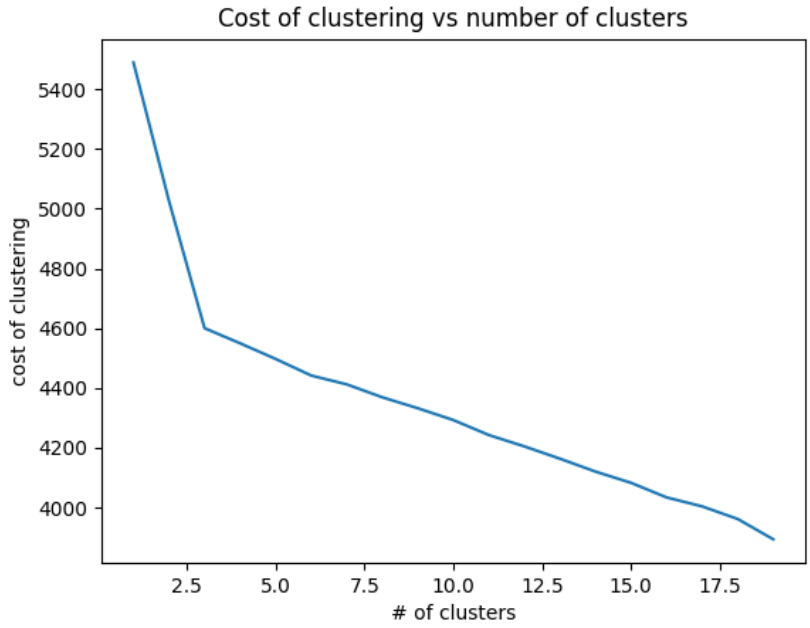

Figure 2: List of labels generated by kmeans

(c) We now apply the same procedure as in part (a), but this time we redefine $A = A\Phi$ where Φ is a 150×10 spherically sampled Gaussian matrix. We recover something resembling the original clustering, but it isn't great; we observe many rows have randomly jumped clusters.

```
UNSHUFFLED LABELS:
 [2. 0. 2. 0. 0. 1. 1. 0. 1. 2. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0.
 1. 0. 0. 0. 0. 2. 0. 2. 2. 0. 2. 1. 0. 2. 0. 0. 0. 2. 0. 2. 0. 1. 2. 0.
 0. 0. 1. 1. 1. 1. 1. 1. 1. 2. 0. 1. 1. 1. 0. 1. 1. 1. 1. 2. 2. 1. 2. 0.
 1. 1. 1. 2. 1. 0. 1. 1. 1. 2. 0. 1. 1. 1. 1. 1. 0. 2. 0. 1. 1. 1. 1. 1.
 0. 1. 1. 0. 0. 1. 1. 2. 0. 1. 2. 2. 1. 2. 2. 2. 2. 2. 2. 1. 2. 2. 2. 2.
 2. 1. 2. 0. 2. 2. 2. 2. 2. 1. 2. 0. 2. 2. 2. 2. 2. 2. 0. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 0. 2.]
```

Figure 3: List of labels generated by kmeans in problem 1 (c). Some of the natural clustering has been recovered, but it is far from a good match.

This suggests that the Gaussian projection isn't a great way to preserve the clustering structure of our original data.

(d) We now apply a projection to $\mathbb{R}^3$ using the first three right singular vectors of A. This proves to be a much better way to preserve the clustering structure of A; for while there is still some differences, it recovers the original clustering much better than the Gaussian projection. Problem 2 (b) explains this.

```
UNSHUFFLED LABELS:
 [2. 0. 2. 2. 2. 2. 2. 2. 2. 2. 2. 0. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 0. 1. 0. 2. 2. 2. 2. 2. 2. 1. 2. 2. 2. 2. 0. 2. 1. 2. 2. 2.
 2. 2. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 2. 1. 1. 1. 1.
 1. 1. 1. 2. 0. 0. 0. 1. 0. 1. 0. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.]
```

Figure 4: List of labels generated by kmeans in problem 1 (d). Most of the original clustering has been recovered, despite the fact that we have reduced from 150 dimensions to 3.

□

EXERCISE 2. Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be a matrix whose n rows are the data points $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$, and let $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$. Consider the k-means optimization problem: find a partition $C_1, ..., C_k$ which minimizes, among all partitions of $[n]$ into k subsets,

$$\text{cost}_{\mathcal{X}}(C_1, ..., C_k) := \sum_{j=1}^{k} \sum_{i \in C_j} \left\| \mathbf{x}_i - \frac{1}{|C_j|} \sum_{i \in C_j} \mathbf{x}_i \right\|_2^2 .$$

(a) Suppose that $\Phi : \mathbb{R}^d \longrightarrow \mathbb{R}^r$ with $r = \mathcal{O}(\log(n)/\varepsilon^2)$ is a random i.i.d. spherical Gaussian projection matrix and thus satisfies the JL lemma. Consider the projected points $\mathbf{y}_j = \Phi \mathbf{x}_j \in \mathbb{R}^r$ and suppose $\tilde{C}_1, ..., \tilde{C}_k$ are an optimal set of k-means clusters for the data points $\mathcal{Y} = \{\mathbf{y}_1, ..., \mathbf{y}_n\}$. That is,

$$\text{cost}_{\mathcal{Y}}(\tilde{C}_1, ..., \tilde{C}_n) = \min_{C_\bullet} \text{cost}_{\mathcal{Y}}(C_1, ..., C_k)$$

where the minimization is over all partitions of $\mathcal{Y}$ into k subsets. Show that the clusters $\tilde{C}_1, ..., \tilde{C}_k$ also represent a good clustering for the original dataset $\mathcal{X}$ in the sense that with high probability

$$\text{cost}_{\mathcal{X}}(\tilde{C}_1, ..., \tilde{C}_k) \leq (1 + \varepsilon) \min_{C_1, ..., C_k} \text{cost}_{\mathcal{X}}(C_1, ..., C_k).$$

(b) Suppose we now project the points $\mathbf{x}_j$ to k dimensions using the SVD of $\mathbf{X}$. Let $\mathbf{V}_k \in \mathbb{R}^{d \times k}$ be the matrix whose columns are the first right singular vectors of $\mathbf{X}$. Suppose the $\tilde{C}_1, ..., \tilde{C}_k$ are the optimal k-means clusters for the points $\mathbf{V}_k^\top \mathbf{x}_1, ..., \mathbf{V}_k^\top \mathbf{x}_n$.

Show that the clusters $\mathbf{C}_1, ..., \mathbf{C}_k$ also represent a good clustering for the original dataset $\mathcal{X}$, in the sense that

$$\text{cost}_{\mathcal{X}}(\tilde{C}_1, ..., \tilde{C}_k)) \leq 2 \min_{C_1, ..., C_k} \text{cost}_{\mathcal{X}}(C_1, ..., C_k).$$

Proof:

(a) First consider a fixed $j \in \{1, ..., k\}$ and the following expression:

$$\sum_{i, \ell \in C_j} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 = \sum_{i \in C_j} \sum_{\ell \in C_j} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2.$$

Letting $\mu_j = \frac{1}{|C_j|}\sum_{i\in C_j}\mathbf{x}_i$ denote the centroid of $\{\mathbf{x}_i\}_{i\in C_j}$, we see that

$$\sum_{i\in C_j}\sum_{\ell\in C_j}\|\mathbf{x}_i-\mathbf{x}_j\|_2^2 = \sum_{i\in C_j}\sum_{\ell\in C_j}\|\mathbf{x}_i-\mu_j+\mu_j-\mathbf{x}_j\|_2^2$$

$$= \sum_{i\in C_j}\left(\sum_{\ell\in C_j}\|\mu_j-\mathbf{x}_\ell\|_2^2 \;+\; 2(\mathbf{x}_i-\mu_j)\cdot\sum_{\ell\in C_j}(\mu_j-\mathbf{x}_\ell) \;+\; |C_j|\cdot\|\mathbf{x}_i-\mu_j\|_2^2\right).$$

The second equality above follows from the fact that

$$\sum_{i=1}^n\|\mathbf{a}_i-\mathbf{c}+\mathbf{c}-\mathbf{x}\|^2 = \sum_{i=1}^n\|\mathbf{a}_i-\mathbf{c}\| \;+\; 2(\mathbf{c}-\mathbf{x})\cdot\sum_i(\mathbf{a}_i-\mathbf{c}) \;+\; n\cdot\|\mathbf{c}-\mathbf{x}\|^2,$$

which in turn can be derived by writing $\|\mathbf{a}_i-\mathbf{c}+\mathbf{c}-\mathbf{x}\|^2 = \langle\mathbf{a}_i-\mathbf{c}+\mathbf{c}-\mathbf{x}, \mathbf{a}_i-\mathbf{c}+\mathbf{c}-\mathbf{x}\rangle$, expanding by bilinearity and gathering up terms in a clever way. Using the fact that indexing over ℓ and i is equivalent together with the bilinearity properties of the inner product, we can continue our above chain of equalities to get that

$$\sum_{i,\ell\in C_j}\|\mathbf{x}_i-\mathbf{x}_j\|_2^2$$

$$= \sum_{i\in C_j}\left(\sum_{\ell\in C_j}\|\mu_j-\mathbf{x}_\ell\|_2^2 \;+\; 2(\mathbf{x}_i-\mu_j)\cdot\sum_{\ell\in C_j}(\mu_j-\mathbf{x}_\ell) \;+\; |C_j|\cdot\|\mathbf{x}_i-\mu_j\|_2^2\right)$$

$$= |C_j|\cdot\sum_{\ell\in C_j} \;+\; 2\left(\sum_{\ell\in C_j}(\mu_j-\mathbf{x}_\ell)\right)\cdot\sum_{i\in C_j}(\mathbf{x}_i-\mu_j) \;+\; |C_j|\cdot\sum_{i\in C_j}\|\mathbf{x}_i-\mu_j\|_2^2$$

$$= 2|C_j|\cdot\sum_{i\in C_j}\|\mathbf{x}_i-\mu_j\|_2^2 \;-\; 2\left\|\sum_{\ell\in C_j}(\mathbf{x}_i-\mu_j)\right\|_2^2.$$

The term $\sum_{\ell\in C_j}(\mathbf{x}_i-\mu_j)$ is 0 because μ_j is the centroid of $\{\mathbf{x}_i\}_{i\in C_j}$, hence

$$\sum_{i,\ell\in C_j}\|\mathbf{x}_i-\mathbf{x}_j\|_2^2 = 2|C_j|\cdot\sum_{i\in C_j}\|\mathbf{x}_i-\mu_j\|_2^2 = 2|C_j|\cdot\sum_{i\in C_j}\left\|\mathbf{x}_i-\frac{1}{|C_j|}\sum_{\ell\in C_j}\mathbf{x}_\ell\right\|_2^2.$$

Taking sums over all $j\in\{1,...,k\}$, we then get that

$$\begin{aligned}\text{cost}_{\mathcal{X}}(C_1,...,C_k) &= \sum_{j=1}^k\sum_{i\in C_j}\left\|\mathbf{x}_i-\frac{1}{|C_j|}\sum_{i\in C_j}\mathbf{x}_i\right\|_2^2\\ &= \sum_{j=1}^k\frac{1}{2|C_j|}\sum_{i,\ell\in C_j}\|\mathbf{x}_i-\mathbf{x}_\ell\|_2^2,\end{aligned}$$

as suggested by the hint. This form of the cost function integrates more favorably with the properties of the Johnson-Lindenstrauss theorem, since for $r > C\cdot\frac{\log(n/\delta)}{\varepsilon^2}$, we get that

$$\left\|\mathbf{y}_i-\mathbf{y}_j\right\|_2^2 = \|\Phi(\mathbf{x}_i-\mathbf{x}_\ell)\|_2^2 \le (1+\varepsilon)\|\mathbf{x}_i-\mathbf{x}_\ell\|_2^2$$

occurs with probability at least $1-\delta$ and therefore

$$\begin{aligned}\text{cost}_{\mathcal{Y}}(C_1,...,C_k) &= \sum_{j=1}^{k}\frac{1}{2|C_j|}\sum_{i,\ell\in C_j}\|\mathbf{y}_i-\mathbf{y}_\ell\|_2^2 \\ &\leq (1+\varepsilon)\sum_{j=1}^{k}\frac{1}{2|C_j|}\sum_{i,\ell\in C_j}\|\mathbf{x}_i-\mathbf{x}_\ell\|_2^2 = (1+\varepsilon)\,\text{cost}_{\mathcal{X}}(C_1,...,C_k)\end{aligned}$$

also occurs with probability at least $1-\delta$ for any partition $C_1 \sqcup ... \sqcup C_k = \{1..n\}$. Combining this with the other bound from the JL theorem we have that

$$(1-\varepsilon)\,\text{cost}_{\mathcal{X}}(C_1,...,C)k \leq \text{cost}_{\mathcal{Y}}(C_1,...,C_k) \leq (1+\varepsilon)\,\text{cost}_{\mathcal{X}}(C_1,...,C_k)$$

for all partitions $C_\bullet$ of $\mathcal{X}$. Since this holds for all partitions of $\mathcal{X}$ it also holds for the partition $\tilde{C}_\bullet$ which minimizes $\text{cost}_{\mathcal{Y}}$, hence

$$(1-\varepsilon)\,\text{cost}_{\mathcal{X}}(\tilde{C}_1,...,\tilde{C}_k) \leq \text{cost}_{\mathcal{Y}}(\tilde{C}_1,...,\tilde{C}_k) \leq \text{cost}_{\mathcal{Y}}(C_1,...,C_k) \leq (1+\varepsilon)\,\text{cost}_{\mathcal{X}}(C_1,...,C_k).$$

We then have that

$$\text{cost}_{\mathcal{X}}(\tilde{C}_1,...,\tilde{C}_k) \leq \frac{1+\varepsilon}{1-\varepsilon}\,\text{cost}_{\mathcal{Y}}(C_1,...,C_k).$$

This is not quite what we want. However, we chose $r = \mathcal{O}(\log(n)/\varepsilon^2)$, which only means that $r = C\log(n)/\varepsilon^2$ for some C. Rescale C and choose a new $\varepsilon' \in (0,1)$ so that

$$r = 9C\cdot\frac{\log(n)}{\varepsilon'^2} \implies \varepsilon = 3\sqrt{\frac{C\log(n)}{r}} = 3\varepsilon.$$

If our original ε was less than $1/3$, then

$$\begin{aligned}1-3\varepsilon > 0 &\iff 0 < \varepsilon(1-3\varepsilon) \\ &\iff 1+\varepsilon < 1+3\varepsilon-\varepsilon-3\varepsilon^2 \\ &\iff \frac{1+\varepsilon}{1-\varepsilon} < 1+3\varepsilon = 1+\varepsilon'.\end{aligned}$$

Thus, for ε', we have

$$\text{cost}_{\mathcal{X}}(\tilde{C}_1,...,\tilde{C}_k) \leq \frac{1+\varepsilon}{1-\varepsilon}\,\text{cost}_{\mathcal{Y}}(C_1,...,C_k) \leq (1+\varepsilon')\,\text{cost}_{\mathcal{Y}}(C_1,...,C_k).$$

This is perfectly fine, since the change $\varepsilon \longrightarrow \varepsilon'$ corresponds to scaling r by r, and hence we still have that $r = \mathcal{O}(\log(n)/\varepsilon^2)$ for our original choice of ε. This gives us the desired result.

(b) We first prove that $\text{cost}_{\mathcal{X}}(C_1,...,C_k) = \|\mathbf{X}-MM^\top\mathbf{X}\|_F^2$ where $M \in \mathbb{R}^{n\times k}$is defined by $M_{ij} = \frac{1}{\sqrt{|C_j|}}$ if $i \in C_j$ and 0 otherwise. Notice that each row of M has only one nonzero element at index (i,j) where $i \in C_j$, and hence

$$[MM^\top]_{ij} = [M]_{i,\bullet}\cdot[M]_{j,\bullet} = \begin{cases}\frac{1}{|C_{\ell_i}|} & i,j \in C_{\ell_i} \\ 0 & \text{else}\end{cases}$$

for some $\ell_i = 1, ..., k$. In particular, each diagonal element $[MM^\top]_{ii}$ is nonzero and is equal to $1/|C_{\ell_i}|$ where $i \in C_{\ell_i}$. Thus, when we multiply a row $[MM^\top]_{i,\bullet}$ by a column $[\mathbf{X}]_{\bullet,j}$ of $\mathbf{X}$, the result is a sum

$$[MM^\top \mathbf{X}]_{ij} = \frac{1}{|C_{\ell_i}} \sum_{a \in C_{\ell_i}} \mathbf{x}_a^j$$

where C_{ℓ_i} is the partition containing i and $\mathbf{x}_a^j$ is the jth term in the data point $\mathbf{x}_a$. That is, $[MM^\top \mathbf{X}]_{ij}$ is the sum of the jth components of all data points belonging to C_{ℓ_i} scaled by $1/|C_{\ell_i}|$. The ith row of $MM^T\mathbf{X}$ is therefore the centroid of the ℓ_ith cluster $\{\mathbf{x}_j\}_{j \in C_{\ell_i}}$. Denoting by μ_{ℓ_i} the ℓ_ith centroid, we see that

$$\begin{aligned}
\|\mathbf{X} - MM^\top \mathbf{X}\|_F^2 &= \sum_{i=1}^{n} \|[\mathbf{X} - MM^\top \mathbf{X}]_{i,\bullet}\|^2 \\
&= \sum_{i=1}^{n} \|\mathbf{x}_i - \mu_{\ell_i}\|^2 \\
&= \sum_{j=1}^{k} \sum_{i \in C_j} \|\mathbf{x}_i - \mu_i\|^2 = \mathrm{cost}_{\mathcal{X}}(C_1, ..., C_k).
\end{aligned}$$

This gives us yet another expression for the k-means cost function.

We now turn to the problem in earnest. Let $\{v_1, ..., v_k, v_{k+1}, ..., v_d\}$ be the complete list of right singular vectors for X, and note that they form a complete orthonormal basis for $\mathbb{R}^d$. Here we use the formulation of SVD in which V and U are both orthogonal matrices (not $d \times r$ matrices) but whose smallest singular vectors might correspond to singular values of 0. Let W be the matrix whose columns are $v_{k+1}, ..., v_d$. Then $V = V_k \oplus W$ is a $d \times d$ orthogonal matrix, preserves the Frobenius norm, and hence

$$\begin{aligned}
\mathrm{cost}_{\mathcal{X}}(\tilde{C}_1, ..., \tilde{C}_k) &= \|\mathbf{X} - \tilde{M}\tilde{M}^\top \mathbf{X}\|_2^2 = \|(\mathbf{X} - \tilde{M}\tilde{M}^\top \mathbf{X})(V_k \oplus W)\|_2^2 \\
&= \|(\mathbf{X}V_k - \tilde{M}\tilde{M}^\top \mathbf{X}V_k) \oplus (\mathbf{X}W - \tilde{M}\tilde{M}^\top \mathbf{X}W)\|_2^2 \\
&= \|(\mathbf{X}V_k - \tilde{M}\tilde{M}^\top \mathbf{X}V_k)\|_2^2 + \|(\mathbf{X}W - \tilde{M}\tilde{M}^\top \mathbf{X}W)\|_2^2
\end{aligned}$$

where $\tilde{M}$ is the matrix defined earlier corresponding to the clustering $\tilde{C}_\bullet$. If we can bound both of these summands by $\mathrm{cost}_{\mathcal{X}}(C_1, ..., C_k)$ for an arbitrary clustering $C_\bullet$, then we will be done. Let M denote the matrix corresponding to this $C_\bullet$.

The first term is easy. Note first that $\tilde{C}_\bullet$ is an optimal choice of clustering for $\mathbf{X}V_k$, and hence

$$\|(\mathbf{X}V_k - \tilde{M}\tilde{M}^\top \mathbf{X}V_k)\|_2^2 = \mathrm{cost}_{\mathcal{Y}}(\tilde{C}_\bullet) \leq \mathrm{cost}_{\mathcal{Y}}(C_\bullet) = \|(\mathbf{X}V_k - MM^\top \mathbf{X}V_k)\|_2^2.$$

However, by what we have above,

$$\|(\mathbf{X}V_k - MM^\top \mathbf{X}V_k)\|_2^2 \leq \|(\mathbf{X}V_k - MM^\top \mathbf{X}V_k)\|_2^2 + \|(\mathbf{X}W - MM^\top \mathbf{X}W)\|_2^2 = \mathrm{cost}_{\mathcal{X}}(C_\bullet),$$

so we have that $\mathrm{cost}_{\mathcal{Y}}(\tilde{C}_\bullet) \leq \mathrm{cost}_{\mathcal{X}}(C_\bullet)$.

The next term is much harder. The first thing we must note is that the matrix $I_n - MM^\top$ has operator norm 1. This can be seen via direct computation by looking at its action on an arbitrary norm 1 vector. There

truly ought to be a better way to see this – I worked this out on paper but do not entirely trust my result, and hence don't include it here.

However, if we take $\|I_n - MM^\top\|_2 \leq 1$ as a given, then we see that

$$\|\mathbf{X}W - MM^\top \mathbf{X}W\|_F^2 = \|(I_n - MM^\top)\mathbf{X}W\|_F^2 \leq \|\mathbf{X}W\|_F^2.$$

This last equality follows from the definition of the operator norm – for each column c_i in $\mathbf{X}W$, $\|(I_n - MM^\top)c_i\| \leq \|c_i\|$, and hence

$$\|(I_n - MM^\top)\mathbf{X}W\| \leq \|c_1\|^2 + ... + \|c_d\|^2 = \|\mathbf{X}W\|_F^2.$$

The matrix $\mathbf{X}W$ is interesting. Writing $X = U\Sigma V^\top$, we see that by the normalicy of the right singular vectors

$$XW = U\Sigma V^\top W = U\Sigma \cdot \left(\begin{array}{c|c} 0 & 0 \\ \hline 0 & I_{d-k} \end{array}\right) = U \cdot \left(\begin{array}{c|c} 0 & 0 \\ \hline 0 & \Sigma - \Sigma_k \end{array}\right),$$

meaning that XW is the result of taking the product of U together with the diagonal matrix consisting of only the final $d - k$ singular values. By the previous homework, we then get that

$$\|\mathbf{X}W\|_F^2 = \sigma_{k+1}^2 + ... + \sigma_d^2 = \|\mathbf{X} - \mathbf{X}_k\|_F^2.$$

This is something we can work with. By the hint, $MM^\top \mathbf{X}$ has rank k, but we have seen that X_k minimizes $\|X - C\|_F^2$ among rank k matrices, hence

$$\|(\mathbf{X}W - \tilde{M}\tilde{M}^\top \mathbf{X}W)\|_2^2 \leq \|\mathbf{X}W\|_F^2 = \|X - X_k\|_F^2 \leq \|\mathbf{X} - MM^\top \mathbf{X}\|_F^2.$$

This is the desired bound. Putting this all together, we get that

$$\begin{aligned} \text{cost}_{\mathcal{X}}(\tilde{C}_\bullet) &= \|(\mathbf{X}V_k - \tilde{M}\tilde{M}^\top \mathbf{X}V_k)\|_2^2 + \|(\mathbf{X}W - \tilde{M}\tilde{M}^\top \mathbf{X}W)\|_2^2 \\ &\leq \text{cost}_{\mathcal{X}}(C_\bullet) + \text{cost}_{\mathcal{X}}(C_\bullet) = 2\,\text{cost}_{\mathcal{X}}(C_\bullet). \end{aligned}$$

Since this holds for all partitions $C_\bullet$, it holds for the one which minimizes $\text{cost}_{\mathcal{X}}(C_\bullet)$ as well, and so we have our result. □

EXERCISE 3. Find the mapping $\varphi(\mathbf{x})$ that gives rise to the polynomial kernel

$$K(\mathbf{x}, \mathbf{y}) = (x_1 x_2 + y_1 y_2)^2.$$

Proof: Consider the map $\varphi : \mathbb{R}^2 \longrightarrow \mathbb{R}^3$ defined $\varphi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1 x_2)$. Interestingly, this is similar to the map one considers from a polynomial ring $R[x_1, x_2]$ to its 2nd Veronese subring $R[x_1^2, x_1 x_2, x_2^2]$. We then have that

$$\begin{aligned} \varphi(\mathbf{x}) \cdot \varphi(\mathbf{y})^T &= (x_1^2, x_2^2, \sqrt{2}x_1 x_2) \cdot (y_1^2, y_2^2, \sqrt{2}y_1 y_2) \\ &= x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 y_1 x_2 y_2 \\ &= (x_1 y_1 + x_2 y_2)^2, \end{aligned}$$

hence φ gives rise to the desired kernel. □

EXERCISE 4. Consider a Support Vector Machine with "soft margin" constrains which allows for misclassification: Given training data $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)$ where $\mathbf{x}_j \in \mathbb{R}^d$ and $y_j \in \{-1, +1\}$, the SVM is

$$\min_{\mathbf{w},b,\xi} \lambda\|\mathbf{w}\|_2^2 + \frac{1}{n}\sum_{j=1}^{n}\xi \quad \text{s.t. } y_j \cdot (\langle \mathbf{w}, \mathbf{x}_j\rangle - b) \geq 1 - \xi_j, \quad \xi_j \geq 0.$$

(a) Discuss the relationship between the parameter λ and the allowable misclassification error.

(b) Where does a data point lie relative to where the margin is when $\xi_j = 0$? Is this data point classified correctly?

(c) Where does a data point lie relative to where the margin is when $0 < \xi_j \leq 1$? Is this data point classified correctly?

(d) Where does a data point lie relative to where the margin is when $\xi_j > 1$? Is this data point classified correctly?

Proof:

(a) The understanding of this case seems to follow from (b) through (d). If we take λ to be large, then $\mathbf{w}$ will shrink to compensate and $\langle \mathbf{w}, \mathbf{x}_i\rangle$ will be relatively small. This will mean that the penalty ξ_i for misclassifying data point $\mathbf{x}_i$ will be relatively small. Hence large λ emphasizes a large margin but allows for *more* misclassifications. Conversely, small λ allows us to choose larger magnitudes of $\mathbf{w}$, which penalizes misclassifications more drastically.

(b) When $\xi_j = 0$ we have that $y_j \cdot (\langle \mathbf{w}, \mathbf{x}_j\rangle - b) \geq 1$. This means that the data point is classified correctly *and* is outside the margin.

(c) When $0 < \xi_j \leq 1$, then ξ_j contributes to penalizing the cost function. Because this is still an optimal solution, this means that we cannot make ξ_j smaller, and thus

$$1 - \xi_j \leq y_j \cdot (\langle \mathbf{w}, \mathbf{x}_j\rangle - b\rangle) < 1.$$

(d) If the optimal solution to the cost function includes a value $\xi_j > 1$, then as before, it means we cannot make ξ_j any smaller without adding error. This means that $0 > y_j \cdot (\langle \mathbf{w}, \mathbf{x}_j\rangle - b)$, and hence $\mathbf{x}_j$ is misclassified.

□

```
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt


# generates the desired block matrix
def gen_block_matrix(blocksize=50, permute=True):
    diag = 0.7
    offdiag = 0.3
    repeat = 3

    # should be 150
    d = repeat * blocksize

    # create the blocks
    A = 0.7 * np.ones((blocksize, blocksize))
    B = 0.3 * np.ones((blocksize, blocksize))

    # create the block matrix
    C = np.block([[A, B, B], [B, A, B], [B, B, A]])

    # convert to the matrix of ones using Bernoulli distribution
    X = np.random.binomial(n=1, p=C)

    # permute
    rng = np.random.default_rng()
    P = rng.permutation(np.identity(d))
    if permute:
        X = np.matmul(P, np.matmul(X, np.transpose(P)))

    # labels for the "natural" cluster of the rows prior to permutation, permuted to match.
    # ASSUMES CLUSTERING WITH k = 3
    nat_labels = [0] * blocksize + [1] * blocksize + [2] * blocksize
    return X, P, nat_labels


# do the clustering
# X: data
# k: number of clusters
# n_init: number of iterations to run
def train_kmeans(X, k=3, n_init=10):
    kmeans = KMeans(n_clusters=k, init="k-means++", n_init=10)  # <-- init=1, verbose=2
    kmeans.fit(X)
```

```
    return kmeans


# code for problem 1a
def prob1a():
    # generate data
    X, P, nat_labels = gen_block_matrix(blocksize=50, permute=True)

    # perform k-means
    kmeans = train_kmeans(X, n_init=10)
    unshuffled_labels = np.matmul(np.array(kmeans.labels_), P)

    # Read off the labels of the cluster and the cost.
    # unshuffled_labels ought to look like the natural clustering,
    # i.e. [0,0,0,0,0,1,1,1,1,1,2,2,2,2,2].
    print("\n\n-----------------------------------\nProblem 1 (a)\n--------------")
    print("\nUNSHUFFLED LABELS:\n", unshuffled_labels)
    print("\nCost achieved:", kmeans.inertia_)
    print("\n---end---\n-----------------------------------")


# def prob1b():
def prob1b():
    # generate data
    X, P, nat_labels = gen_block_matrix(blocksize=50, permute=True)

    # cluster for various values of k and get costs
    ks = list(range(1, 20))
    kmeans = [train_kmeans(X, k=i, n_init=10) for i in ks]
    costs = [km.inertia_ for km in kmeans]

    print("\n\n-----------------------------------\nProblem 1 (b)\n--------------")
    print("\n     <displaying graph>\n")
    print("\n---end---\n-----------------------------------")

    # should look like an elbow. The "point" of the elbow is qualitatively a good choice of k
    # see the "elbow method"
    plt.plot(ks, costs)
    plt.xlabel("# of clusters")
    plt.ylabel("cost of clustering")
    plt.title("Cost of clustering vs number of clusters")
    plt.show()
```

```
def prob1c():
    mu, sigma = 0, 1
    col = 150
    row = 10

    # generate spherical projection matrix of size row x col
    # the gaussian vectors are of length 150, not of length 10. This may be wrong.
    Phi = np.array([np.random.normal(mu, sigma, col) for i in range(row)])
    for i in range(row):
        Phi[i, :] = Phi[i, :] / np.linalg.norm(Phi[i, :])

    # generate the data to cluster
    X, P, nat_labels = gen_block_matrix(blocksize=50, permute=True)
    Xn = np.matmul(X, np.transpose(Phi))

    # do clustering
    kmeans = train_kmeans(Xn, n_init=10)
    shuffled_labels = np.array(kmeans.labels_)
    unshuffled_labels = np.matmul(shuffled_labels, P)

    # Read off the labels of the cluster and the cost.
    print("\n\n-----------------------------------\nProblem 1 (c)\n--------------")
    print("\nSHUFFLED LABELS:\n", shuffled_labels)
    print("\nUNSHUFFLED LABELS:\n", unshuffled_labels)
    print("\nCost achieved:", kmeans.inertia_)
    print("\n---end---\n-----------------------------------")


def prob1d():
    # generate the data to cluster
    X, P, nat_labels = gen_block_matrix(blocksize=50, permute=True)

    # get the svd stuff
    U, S, Vh = np.linalg.svd(X)

    # truncate Vh
    # try adding more columns -- you'll quickly recover the original clustering.
    V = Vh[:, :3]
    Xn = np.matmul(X, V)

    # do clustering
    kmeans = train_kmeans(Xn, n_init=10)
```

```
    shuffled_labels = np.array(kmeans.labels_)
    unshuffled_labels = np.matmul(shuffled_labels, P)

    # Read off the labels of the cluster and the cost.
    print("\n\n-----------------------------------\nProblem 1 (d)\n--------------")
    print("\nSHUFFLED LABELS:\n", shuffled_labels)
    print("\nUNSHUFFLED LABELS:\n", unshuffled_labels)
    print("\nCost achieved:", kmeans.inertia_)
    print("\n---end---\n-----------------------------------")


if __name__ == "__main__":
    prob1a()
    prob1b()
    prob1c()
    prob1d()
```