EE378B Homework 1 Solution

Thanks to Yu Bai

Winter 2017

(a) **Implementation.** We implement the K-MEANS algorithm in python as the KMeans function (code attached).

An improvement is made upon the naive implementation. Notice that the step \( \sigma_i^{\text{new}} = \arg \min_{a \in [k]} \| x_i - c_a \|^2 \) is costly, as it requires \( nk \) computations on \( d \)-vectors. To improve its efficiency, we take advantage of the inner-product expansion. Note that

\[
\| x_i - c_a \|^2 = \| x_i \|^2 - 2\langle x_i, c_a \rangle + \| c_a \|^2.
\]

With the data matrix \( X \in \mathbb{R}^{n \times d} \) and centroid matrix \( C \in \mathbb{R}^{k \times d} \), it suffices to compute \( \| c_a \|^2 \) and \( XC^T \) to figure out the loss. In theory this saves a constant order of time, and in practice it is significantly faster.

We also implement two utility functions: ClusteringError computes the clustering error given true labels and clustered labels. It finds the minimum matching error over all permutations of the clustered labels, giving a permutation-invariant measure of error. SVDLowerbound computes the SVD lower bound of part (b) on a data matrix \( X \).

**Experiments.** We run experiments on the Gaussian mixture model as specified. Figure 1 shows the mean misclassification errors and their boxplots. Two facts are worth noticing:

- When \( s < 5 \), the mean misclassification error decreases as the separation \( s \) gets larger. This is as expected.
- There are some obvious wrongly-clustered instances when \( s > 5 \). Under some values of \( s \), 1 to 2 instances (out of 10) have misclassification error around 0.5 when the rest are all zero. A closer inspection on the output centroids reveal that the three centroids collapse to two in these instances. For example, with the population centroids being \((s, 0, 0), (0, s, 0), (0, 0, s)\), these outputs are around \((s, 0, 0), (0, s/2, s/2), (0, s/2, s/2)\). Note that this is a saddle point of the population risk!

We postulate that these outputs are also saddle points of the sample objective. Since we use random initializations, this can happen, and the algorithm is not quite able to escape such stationary points.

(b) **Deriving the lower bound.** For any partition \( \sigma \), the minimizing centroids \( c \) satisfy

\[
c_a = \frac{1}{n_a} \sum_{i \in \sigma^{-1}(a)} x_i = \frac{1}{n_a} X^T 1_{\sigma^{-1}(a)}.
\]
Figure 1: Performance of K-Means on Gaussian mixture data. $s \in \{0.5, 1, \ldots, 10\}$.

So we have

$$\mathcal{L}(\sigma) = \frac{1}{n} \sum_{i=1}^{n} \|x_i - c_{\sigma_i}\|^2_2$$

$$= \frac{1}{n} \sum_{a=1}^{k} \sum_{i \in \sigma^{-1}(a)} \|x_i - \frac{1}{n_a} X^T 1_{\sigma^{-1}(a)}\|^2_2 = \frac{1}{n} \sum_{a=1}^{k} \left[ \sum_{i \in \sigma^{-1}(a)} \|x_i\|^2_2 - \frac{1}{n_a} \|X^T 1_{\sigma^{-1}(a)}\|^2_2 \right]$$

$$= \frac{1}{n} \|X\|_{F}^2 - \frac{1}{n} \sum_{a=1}^{k} \frac{1}{n_a} \|X^T 1_{\sigma^{-1}(a)}\|^2_2.$$ 

To show the lower bound, observe that $\{\frac{1}{\sqrt{n_a}} 1_{\sigma^{-1}(a)} : a \in [k]\}$ forms an orthonormal set. So letting $P_k = [\frac{1}{\sqrt{n_1}} 1_{\sigma^{-1}(1)}, \ldots, \frac{1}{\sqrt{n_k}} 1_{\sigma^{-1}(k)}] \in \mathbb{R}^{n \times k}$, $P_k$ is a right projector. By the best projection property of the SVD, we get that

$$\sum_{a=1}^{k} \frac{1}{n_a} \|X^T 1_{\sigma^{-1}(a)}\|^2_2 = \sum_{a=1}^{k} \frac{1}{\sqrt{n_a}} \|X^T 1_{\sigma^{-1}(a)}\|^2_2 = \|X^T P_k\|_{F}^2 \leq \sum_{j=1}^{k} \sigma_j(X)^2.$$ 

Plugging this into the above expression, we get

$$\mathcal{L}(\sigma) = \frac{1}{n} \|X\|_{F}^2 - \frac{1}{n} \sum_{a=1}^{k} \frac{1}{n_a} \|X^T 1_{\sigma^{-1}(a)}\|^2_2$$

$$\geq \frac{1}{n} \sum_{j=1}^{n \vee d} \sigma_j(X)^2 - \frac{1}{n} \sum_{j=1}^{k} \sigma_j(X)^2$$

$$= \frac{1}{n} \sum_{j=k+1}^{n \vee d} \sigma_j(X)^2.$$ 

This is the desired lower bound.

**Experiments.** We compute and plot the lower bounds in the Gaussian mixture experiment, compared with the actual losses achieved by K-Means. The lower bounds concentrate at
around 7 and the actual losses are about 10. Note that the value 10 makes sense: an oracle classifier achieves exactly this number at the population level since $\mathbb{E}[\|Z\|^2_2] = d = 10$ for $Z \sim \mathcal{N}(0, I_{10})$. For large values of $s$ we again see the degenerate instances. For small values of $s$, the actual losses are smaller, since some points are able to be clustered to a wrong but closer centroid.

![Figure 2: Loss functions and their SVD lower bounds (mean over 10 instances).](image)

Table 1: K-Means on the wheat seeds data.

<table>
<thead>
<tr>
<th>Loss</th>
<th>2.0530</th>
<th>2.0519</th>
<th>2.0530</th>
<th>2.0508</th>
<th>2.0508</th>
<th>2.0508</th>
<th>2.0515</th>
<th>2.0528</th>
<th>2.0508</th>
<th>2.0530</th>
</tr>
</thead>
<tbody>
<tr>
<td># Wrong clustering</td>
<td>17</td>
<td>15</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>14</td>
<td>16</td>
<td>17</td>
<td>17</td>
</tr>
</tbody>
</table>

We run the K-Means algorithm with 10 random initializations. Table 1 reports the losses and the number of misclassified observations. As is seen, the performance of K-Means with random initialization is quite stable on this real dataset and achieves a satisfactory misclassification error ($17/210 = 8.1\%$).

Further, the SVD lower bound equals 0.0932 on this dataset, far less than the realized values (around 2.05). This makes sense because the lower bound is tight only when the top $k$ left singular vectors of $X$ are the $k$ cluster indicators, which is clearly not the case here: the three spices also possess some similarity which can become the top signal.

(c) The wheat seeds dataset consists of 210 rows and 7 columns. As each column represents a feature that can be of varying units, we standardize all the columns to have mean 0 and standard deviation 1.
A Code for part (a),(b)

```python
import numpy as np

""
The Batch K-Means algorithm with squared Euclidean distance
Uses the inner product expansion to speed-up the re-labeling step
Inputs:
X: the n*d data matrix
k: number of clusters
sigma_0: the initial partition, labeled as {0,...,k-1}
eps: convergence criterion
Outputs:
sigma_new: the clustered labels
Loss_new: the loss
C: the centroids
""

def KMeans(X, k, sigma_0, eps=1e-6):
    n, d = X.shape
    sum_x_norms = np.power(np.linalg.norm(X), 2)
    sigma_new = sigma_0
    Loss_new = -1
    C = np.zeros((k,d))
    while 1:
        sigma = sigma_new
        Loss = Loss_new
        # Compute new centroids
        for j in range(k):
            ind_j = (sigma == j)
            if any(ind_j):
                C[j, :] = np.mean(X[ind_j, :], 0)
        # Assign new labels
        C_norms = np.reshape(np.power(np.linalg.norm(C, axis=1), 2), (1,k))
        xct_mat = np.matmul(X, np.transpose(C))
        dist_mat = -2 * xct_mat + np.matmul(np.ones((n,1)), C_norms)
        sigma_new = np.argmin(dist_mat, axis=1)
        Loss_new = 1.0/n * (sum_x_norms + sum([dist_mat[j,sigma_new[j]]
            for j in range(n)]))
        # If convergence criterion is satisfied, break
        if Loss != -1 and Loss - Loss_new < eps:
            break
    return sigma_new, Loss_new, C

""
Misclassification error for clustering
Computes the minimum misclassification error over all permutations
Inputs:
k: number of classes
""```
sigma_0, sigma_1: labels in the range \{0,...,k-1\} of the same length

Output:
min_err: misclassification error

```
import itertools
def ClusteringError(sigma_0, sigma_1, k):
    min_err = 1
    for p in itertools.permutations(range(k)):
        sigma_1_permuted = [p[j] for j in sigma_1]
        err = np.mean([sigma_0[j] != sigma_1_permuted[j]
                        for j in range(len(sigma_0))])
        if err < min_err:
            min_err = err
    return min_err
```

SVD lower bound for KMeans

Inputs:
X: the n*d data matrix
k: number of clusters

Output:
L: the SVD lower bound

```
def SVDLowerbound(X, k):
    n, d = X.shape
    _, s, _ = np.linalg.svd(X)
    L = np.sum(np.power(s[k:], 2)) / n
    return L
```

# Part (a): experiments with KMeans
s_vec = np.linspace(0.5, 10, 20)
s_num = len(s_vec)
n_rep = 10
Err_mat = np.zeros((s_num, n_rep))
Loss_mat = np.zeros((s_num, n_rep))
Loss_lb = np.zeros((s_num, n_rep))
n, d = 10000, 10
k = 3
for i in range(s_num):
    s = s_vec[i]
    for j in range(n_rep):
        # Create the dataset
        X = np.random.randn(n, d)
        y = np.zeros(n, dtype=np.int)
        for l in range(n):
            y[l] = np.random.randint(k)
X[1, y[1]] += s

# Perform KMeans clustering
sigma, Loss, C = KMeans(X, k, np.random.randint(k, size=n))
Loss_mat[i,j] = Loss
Err_mat[i,j] = ClusteringError(y, sigma, k)

# Compute the SVD lower bound
Loss_lb[i,j] = SVDLowerbound(X, k)
print("s=%.1f, error=%.4f, Loss=%.6f, SVDLowerbound=%.6f"
      % (s, Err_mat[i,j], Loss, Loss_lb[i,j]))
print("s=%.1f finished with clustering error %f"
      % (s, np.mean(Err_mat[i,:])))

# Plot the results
import matplotlib.pyplot as plt
plt.figure(1, figsize=(11,5))
plt.subplot('121')
plt.plot(s_vec, np.mean(Err_mat, 1), 'o-')
plt.xlabel('s')
plt.ylabel('Error')
plt.title('Mean misclassification error')
plt.xticks(range(2,22,2), range(1,11,1))

plt.subplot('122')
plt.boxplot(np.transpose(Err_mat))
plt.ylim((-0.1,0.7))
plt.xticks(range(2,22,2), range(1,11,1))
plt.xlabel('s')
plt.ylabel('Error')
plt.title('Boxplot of misclassification errors')
plt.savefig('kmeans_err.jpg')
plt.close()
B Code for part (c)

```python
# Part (c): the seeds dataset
seeds = np.genfromtxt('seeds_dataset.txt', delimiter='	')
n, d = seeds.shape
d -= 1
seeds_X = seeds[:,:d]
seeds_y = seeds[:,d] - 1

# Standardize X
for i in range(d):
    seeds_X[:,i] = (seeds_X[:,i] - np.mean(seeds_X[:,i])) / np.std(seeds_X[:,i])

n_rep = 10
k = 3
Loss_seeds = np.zeros(n_rep)
NErr_seeds = np.zeros(n_rep, dtype=np.int)
for j in range(n_rep):
    sigma_0 = np.random.randint(k, size=n)
    sigma, Loss, C = KMeans(seeds_X, k, sigma_0)
    err = ClusteringError(seeds_y, sigma, k)
    Loss_seeds[j] = Loss
    NErr_seeds[j] = err * n

Loss_lb_seeds = SVDLowerbound(seeds_X, k)
```

7