CSE 547: CSE 547 Homework 2

Due on April 26, 2017

Brian de Silva

Collaborators

I collaborated with Weston Barger and Emily Dinan on ***

Problem 1

Gaussian Random Projections and Inner Products [10 Points]

In this problem, you will show that inner products are approximately preserved using random projections. Let $\phi(x) = \frac{1}{\sqrt{m}}Ax$ represent our random projection of $x \in \mathbb{R}^d$, with A an $m \times d$ projection matrix with each entry sampled i.i.d from $N(0,1)$. (Note that each row of A is a random projection vector, $v^{(i)}$.)
The *norm preservation theorem* states that for all $x \in \mathbb{R}^d$, the norm of the random projection $\phi(x)$ approximately maintains the norm of the original x with high probability:

$$\Pr\left((1-\epsilon)\|x\|^2 \leq \|\phi(x)\|^2 \leq (1+\epsilon)\|x\|^2\right) \geq 1 - 2e^{-(\epsilon^2-\epsilon^3)m/4}, \tag{1}$$

where $\epsilon \in (0, 1/2)$.
Using the norm preservation theorem, prove that for any $u, v \in \mathbb{R}^d$ s.t. $\|u\| \leq 1$ and $\|v\| \leq 1$,

$$\Pr(|u \cdot v - \phi(u) \cdot \phi(v)| \geq \epsilon) \leq 4e^{-(\epsilon^2-\epsilon^3)m/4}. \tag{2}$$

Note that $u \cdot v$ is the original dot product, and $\phi(u) \cdot \phi(v)$ is the dot product for the random projections. This statement puts a probabilistic bound on the distance between the two dot products. (*Hint: Think about using Theorem (1) with $x = u + v$ and $x = u - v$*).

Solution:

First notice that

$$\begin{aligned}
&(1-\epsilon)\|x\|^2 \leq \|\phi(x)\|^2 \leq (1+\epsilon)\|x\|^2 \\
\iff &\|x\|^2 - \epsilon\|x\| \leq \|\phi(x)\|^2 \leq \|x\|^2 + \epsilon\|x\|^2 \\
\iff &-\epsilon\|x\|^2 \leq \|\phi(x)\|^2 - \|x\|^2 \leq \epsilon\|x\|^2 \\
\iff &\left|\|\phi(x)\|^2 - \|x\|^2\right| \leq \epsilon\|x\|^2.
\end{aligned}$$

Multiplying the above by -1 gives a similar statement:

$$-\epsilon\|x\|^2 \leq \|x\|^2 - \|\phi(x)\|^2 \leq \epsilon\|x\|^2.$$

Observe also that ϕ is a linear function, so for any $x, y \in \mathbb{R}^d$, $\phi(x+y) = \phi(x) + \phi(y)$. We will need this fact later. Following the hint and applying Theorem (1) to $x = u + v$ and $x = u - v$, we obtain

$$\Pr\left((1-\epsilon)\|u+v\|^2 \leq \|\phi(u+v)\|^2 \leq (1+\epsilon)\|u+v\|^2\right) \geq 1 - 2e^{-(\epsilon^2-\epsilon^3)m/4}$$

and

$$\Pr\left((1-\epsilon)\|u-v\|^2 \leq \|\phi(u-v)\|^2 \leq (1+\epsilon)\|u-v\|^2\right) \geq 1 - 2e^{-(\epsilon^2-\epsilon^3)m/4}.$$

Manipulating the inequalities as above, these two statements become

$$\Pr\left(\left|\|\phi(u+v)\|^2 - \|u+v\|^2\right| \leq \epsilon\|u+v\|^2\right) \geq 1 - 2e^{-(\epsilon^2-\epsilon^3)m/4}$$

and

$$\Pr\left(\left|\|\phi(u-v)\|^2 - \|u-v\|^2\right| \leq \epsilon \|u-v\|^2\right) \geq 1 - 2e^{-(\epsilon^2-\epsilon^3)m/4}.$$

These expressions are equivalent to the following

$$\Pr\left(\left|\|\phi(u+v)\|^2 - \|u+v\|^2\right| \geq \epsilon \|u+v\|^2\right) \leq 1 - \left(1 - 2e^{-(\epsilon^2-\epsilon^3)m/4}\right) = 2e^{-(\epsilon^2-\epsilon^3)m/4}$$

and

$$\Pr\left(\left|\|\phi(u-v)\|^2 - \|u-v\|^2\right| \geq \epsilon \|u-v\|^2\right) \leq 1 - \left(1 - 2e^{-(\epsilon^2-\epsilon^3)m/4}\right) = 2e^{-(\epsilon^2-\epsilon^3)m/4}.$$

Now, recall that for any two events A and B, $\Pr(A \wedge B) \leq \Pr(A) + \Pr(B)$. Letting A be the event that

$$A: \quad \left|\|\phi(u+v)\|^2 - \|u+v\|^2\right| \geq \epsilon \|u+v\|^2$$

and B be the event

$$B: \quad \left|\|\phi(u-v)\|^2 - \|u-v\|^2\right| \geq \epsilon \|u-v\|^2,$$

it follows that the probability of both occuring is at most $\Pr(A) + \Pr(B) \leq 4e^{-(\epsilon^2-\epsilon^3)m/4}$. We saw previously that

$$\begin{aligned} &\left|\|\phi(u+v)\|^2 - \|u+v\|^2\right| \geq \epsilon \|u+v\|^2 \\ \iff &-\epsilon \|u+v\|^2 \leq \|u+v\|^2 - \|\phi(u)+\phi(v)\|^2 \leq \epsilon \|u+v\|^2 \end{aligned}$$

and

$$\begin{aligned} &\left|\|\phi(u-v)\|^2 - \|u-v\|^2\right| \geq \epsilon \|u-v\|^2 \\ \iff &-\epsilon \|u-v\|^2 \leq \|\phi(u)-\phi(v)\|^2 - \|u-v\|^2 \leq \epsilon \|u+v\|^2. \end{aligned}$$

If both inequalities hold simultaneously, we may add them together to obtain

$$-\epsilon\left(\|u+v\|^2 + \|u-v\|^2\right) \leq \|\phi(u)-\phi(v)\|^2 - \|\phi(u)+\phi(v)\|^2 + \|u+v\|^2 - \|u-v\|^2 \leq \epsilon\left(\|u+v\|^2 + \|u-v\|^2\right).$$

Using the Parallelogram law and Polarization identity, we can simplify this to

$$\begin{aligned} &-2\epsilon\left(\|u\|^2 + \|v\|^2\right) \leq 4\left(u \cdot v - \phi(u) \cdot \phi(v)\right) \leq 2\epsilon\left(\|u\|^2 + \|v\|^2\right) \\ \iff &\epsilon\left(\|u\|^2 + \|v\|^2\right) \leq 2\left(u \cdot v - \phi(u) \cdot \phi(v)\right) \leq \epsilon\left(\|u\|^2 + \|v\|^2\right) \\ \implies &2\left|u \cdot v - \phi(u) \cdot \phi(v)\right| \leq \epsilon\left(\|u\|^2 + \|v\|^2\right) \leq \epsilon(1+1) = 2\epsilon \\ \implies &\left|u \cdot v - \phi(u) \cdot \phi(v)\right| \leq \epsilon. \end{aligned}$$

Putting everything together, we have our result

$$\Pr\left(\left|u \cdot v - \phi(u) \cdot \phi(v)\right| \leq \epsilon\right) \leq 4e^{-(\epsilon^2-\epsilon^3)m/4}.$$

Problem 2

KD-Trees and LSH for Approximate Neighbor Finding [15 Points]

In this problem, you will use KD-trees and Locality Sensitive Hashing (LSH) to find exact and approximate nearest neighbors. To explore the performance of KD-trees, you can use the starter code in "NearestNeighbor.zip" on the course website. You may alter the class `KDTreeAnalysis` to generate a dataset and return the time it takes to query from the KD-tree on that dataset, although this is not a part of the assignment. For approximate nearest neighbor search, α can be set to a value greater than 1. The KD-tree implementation was downloaded from http://sourceforge.net/projects/java-ml.
For datasets with high dimensionality d, KD-trees will not scale well and other methods must be used. LSH offers a method for reducing the dimensionality of the data while still enabling approximate neighbor finding. For this portion of the problem, you will be using an artificially generated dataset consisting of term-document frequency for a vocabulary of size $d = 1000$ and $n = 100000$ documents. The data can be found in "sim_docdata.zip" on the course website. The zip file consists of two files, sim_docdata.mtx and test_docdata.mtx, which contain the term frequencies in a sparse matrix in Matrix Market format: each row is in the form "termid docid frequency". test_docdata.mtx contains a set of documents to use for querying nearest neighbors. The size of the test set is 500 documents. You will need to complete the classes `MethodComparison`, `GaussianRandomProjection`, and `LocalitySensitiveHash`.

(a) Implement a nearest neighbor search using LSH and m, the number of projections, in $\{5, 10, 20\}$. Although normally we would search 'until time runs out', for this problem, just search all bins that have a Hamming distance from the bin of the query data point <= 3. Record the average query time and the average distance to the nearest neighbor for the test set. Compare this with the average query time and average distance using a KD-tree with α in $\{1, 5, 10\}$. Note that the KD-tree implementation will be slow, and may take 5-10 minutes. Explain the trends found.

(b) Implement a Gaussian random projection on the document data for m in $\{5, 10, 20\}$. Use these projections as the entries for a KD-tree. This results in an alternative approximate nearest neighbor search rather than using $\alpha > 1$. Again, record the average query time and average distance over the test set, and explain the trends found.

Solution:

Number of projections	Average query time (s)	Average distance to NN
5	0.482571	4.832050
10	0.105674	5.356869
20	0.054416	∞

Table 1: Statistics for approximate nearest neighbor search using a location sensitive hash table with different numbers of projections.

α	Average query time	Average distance to NN
1	63.911429	4.455368
5	22.306609	4.782051
10	19.210534	4.852166

Table 2: Statistics for approximate nearest neighbor search using a KD-tree different pruning parameters α.

Problem 2 continued on next page...

(a) Tables 1 and 2 show the average query times and average distances to the approximate nearest neighbors for the locality sensitive hashing and KD-tree implementations of the approximate nearest neighbor problem, respectively. As expected, the average query time using a locality sensitive hash table is *much* faster than that of the KD-tree. However, the KD-tree finds closer points to the input points on average. For $\alpha = 1$ this is to be expected since it finds the actual nearest neighbor. As α is increased the query time decreases as the tree can prune more aggressively, but the quality of the neighbor is diminished marginally.

For the implementation involving locality sensitive hashing, the average query time decreases as a function of the number of projections used. This is because if one uses m projections, then one has 2^m bins in which to place objects. If m is small then there are many fewer bins than there are objects in the training set, so bins can contain a large number of objects. For a fixed depth, when a query is made one has to search a fixed number of bins for a nearest neighbor. If each of these bins contains many objects then the time one must spend checking them for nearest neighbors grows (since one must compute the distance between the reference object and each of the objects in these bins). If we use more projections there are more bins over which the objects are distributed, so each bin contains fewer objects on average. This leads to faster query times on average.

The average distance to the returned nearest neighbor grows with the number of projections for the LSH implementation. This can be explained by similar reasoning to before. For small m lots of objects fall in the same bin. When we look for a nearest neighbor we are likely to search over a larger set of objects, often allowing us to find one that is closer to the reference object than we would have found for a larger value of m. Note that for $m = 20$ the average distance to the nearest neighbor is ∞. The ∞ just means that for at least one reference point, no nearest neighbor was located (using a depth of 3). This is because there are 2^20 bins, while only 100000 training objects (so there are about 10 times as many bins as objects). Many bins will be empty. There is some object in the test set which gets mapped to a bin with no nonempty bins within a distance of 3 of it (using Hamming-distance). If we were to increase the depth above 3 it is possible that a neighbor would be found.

Number of projections	**Average query time (s)**	**Average distance to NN**
5	0.003674	7.391327
10	0.074750	7.117428
20	1.420972	6.301496

Table 3: Statistics for approximate nearest neighbor search using Gaussian random projections with different numbers of projections fed into a KD-tree.

(b) Table 3 gives the average query time and average distance over the test set using Gaussian random projections. The first observation to make is that the average query time grows with the number of projections used. The number of projections is the dimension of the data points inserted into the KD-tree. Lookups become much slower for KD-trees containing higher dimensional objects, which explains this trend in the average query time. The average distance to the nearest neighbor that is returned decreases as the number of projections increases. This is because the more random projections that are used the more information about the original object that is captured. Given a reference object the KD-tree will return its true nearest neighbor *with respect to their representations in the projected space*. Consider the extreme case where only one projection is used. The KD-tree will compute distances between 1-dimensional scalars which are proxies for the true d-dimensional objects. If d is large then much information is lost when projecting these objects down to 1 dimension. As the dimension of the projected space grows we expect to capture more and more information about the original objects. Thus the KD-tree is better able to find nearest neighbors using these richer (higher-dimensional) representations than the lower-dimensional ones.

Problem 3

The Clustering Toolkit: K-Means and EM

In this problem, you will implement the k-means algorithm and EM to fit a Gaussian Mixture Model (GMM). This problem consists of 3 parts. In part 1, you will implement the two clustering algorithms and evaluate on a 2D synthetic dataset. In part 2, we will provide you with a small text dataset (from the BBC). Your task is to implement (or reuse) the same algorithms to cluster the documents. In the last part, you will implement the k-means algorithm in Hadoop MapReduce, and test your algorithm against a subset of a Wikipedia dataset.

1. 2D synthetic data

The k-means algorithm is an intuitive way to explore the structure of a dataset. Suppose we are given points $x^1, ..., x^n \in R^2$ and an integer $K > 1$, and our goal is to minimize the within-cluster sum of squares

$$J(\mu, \mathcal{Z}) = \sum_{i=1}^{n} \left\| x^i - \mu_{z^i} \right\|_2^2,$$

where $\mu = (\mu_1, ..., \mu_K)$ are the cluster centers with the same dimension of data points, and $\mathcal{Z} = (z^1, ..., z^n)$ are the cluster assignments, $z^i \in \{1, ..., K\}$. One common algorithm for finding an approximate solution is Lloyd's algorithm. To apply the Lloyd's k-means algorithm one takes a guess at the number of clusters (i.e., select a value for K) and initializes cluster centers $\mu_1, \ldots, \mu_K$ by picking K points (often randomly from $x^1, ..., x^n$). In practice, we often repeat multiple runs of Lloyd's algorithm with different initializations, and pick the best resulting clustering in terms of the k-means objective. The algorithm then proceeds by iterating through two steps:

(i) Keeping μ fixed, find the cluster classification $\mathcal{Z}$ to minimize $J(\mu, \mathcal{Z})$ by assigning each point to the cluster to which it is closest.

(ii) Keeping $\mathcal{Z}$ fixed, find new centers of the clusters μ for the $(m+1)^{th}$ step to minimize $J(\mu, \mathcal{Z})$ by averaging points within a cluster from the points in a cluster at the m^{th} step.

Terminate the iteration to settle on K final clusters if applicable.

(a) Assuming that the tie-breaking rule used in step (i) is consistent, does Lloyd's algorithm always converge in a finite number of steps? Briefly explain why.

(b) Implement Lloyd's algorithm on the two dimensional data points in data file "2DGaussianMixture.csv". Run it with some of your randomly chosen initialization points with different values of $K \in \{2, 3, 5, 10, 15, 20\}$, and show the clustering plots by color.

(c) Implement Lloyd's algorithm. Run it 20 times, each time with different initialization of K cluster centers picked at random from the set $\{x^1, ..., x^n\}$, with $K = 3$ clusters, on the two dimensional data points in data file 2DGaussianMixture.csv (Link is the "Synthetic Data" in the homework section). Plot in a single figure the original data (in gray), and all 20×3 cluster centers (in black) given by each run of Lloyd's algorithm. Also, compute the minimum, mean, and standard deviation of the within-cluster sums of squares for the clusterings given by each of the 20 runs.

(d) K-means++ is another initialization algorithm for K-means (by David Arthur and Sergei Vassilvitskii). The algorithm proceeds as follows:

(i) Pick the first cluster center μ_1 uniformly at random from the data $x^1, ..., x^n$.

(ii) For $j = 2, ..., K$:

- For each data point, compute its distance D_i to the nearest cluster center picked in a previous iteration:

$$D_i = \min_{j'=1,\ldots,j-1} \left\| x^i - \mu_{j'} \right\|.$$

- Pick the cluster center μ_j at random from $x^1, ..., x^n$ with probabilities proportional to $D_1^2, ..., D_n^2$.

(iii) Return μ as the initial cluster assignments for Lloyd's algorithm.

Replicate Part (c) using k-means++ as the initialization algorithm, instead of picking μ uniformly at random.

(e) Based on your observations in Part (c) and Part (d), is Lloyd's algorithm sensitive to initialization?

(f) One shortcoming of k-means is that one has to specify the value of K. Consider the following strategy for picking K automatically: try all possible values of K and choose K that minimizes $J(\mu, \mathcal{Z})$. Argue why this strategy is a good/bad idea. Suggest an alternative strategy.

(g) The two-dimensional real data in the file "2DGaussianMixture.csv" are generated from a mixture of Gaussians with three components. Implement EM for general mixtures of Gaussians (not just k-means). Initialize the means with the same procedure as in the k-means++ case. Initialize the covariances with the identity matrix. Run your implementation on the synthetic dataset, and:

(i) Plot the likelihood of the data over EM iterations.

(ii) After convergence, plot the data, with the most likely cluster assignment of each data point indicated by its color. Mark the mean of each Gaussian and draw the covariance ellipse for each Gaussian.

(iii) How do these results compare to those obtained by k-means? Why?

Solution:

(a) Yes, assuming a consistent deterministic tie-breaking rule is used, Lloyd's algorithm will always converge in a finite number of steps. Note that there are only a finite number of ways to group the data points into K clusters. These groupings completely determine the associated means. Further, observe that each step of Lloyd's algorithm which assigns at least one point to a new cluster decreases the value of $J(\mu, \mathcal{Z})$. Since there are only finitely many ways to assign the points to clusters, $J(\mu, \mathcal{Z})$ can only be decreased a finite number of times. Assuming a consistent deterministic tie-breaking rule is used to assign points equidistant from two cluster centers to clusters, once $J(\mu, \mathcal{Z})$ stops decreasing the algorithm will stop reassiging points to new clusters.

(b) Figure 1 shows the clusterings generated using various values of K.

(c) Figure 2 visualizes the centers chosen by the 20 runs of Lloyd's algorithm with random initializations. The minimum, mean, and standard deviation of the within-cluster sums of squares for the 20 runs were 6.184172, 8.340745, and 3.735294, respectively.

(d) Figure 3 shows the centers chosen by the 20 runs of Lloyd's algorithm with centers initialized with K-means++. The minimum, mean, and standard deviation of the within-cluster sums of squares for the 20 runs were 6.184172, 7.046801, and 2.587888, respectively.

(e) It seems that Lloyd's algorithm is sensitive to initialization. The K-means++ initialization procedure produced better results on average than the random initialization.

(f) This is a bad strategy because the larger K gets the smaller $J(\mathcal{Z}, \mu)$ can get. With lots of centers it becomes easy to place centers very close to all the points. In the extreme case where K is equal to the number of points,

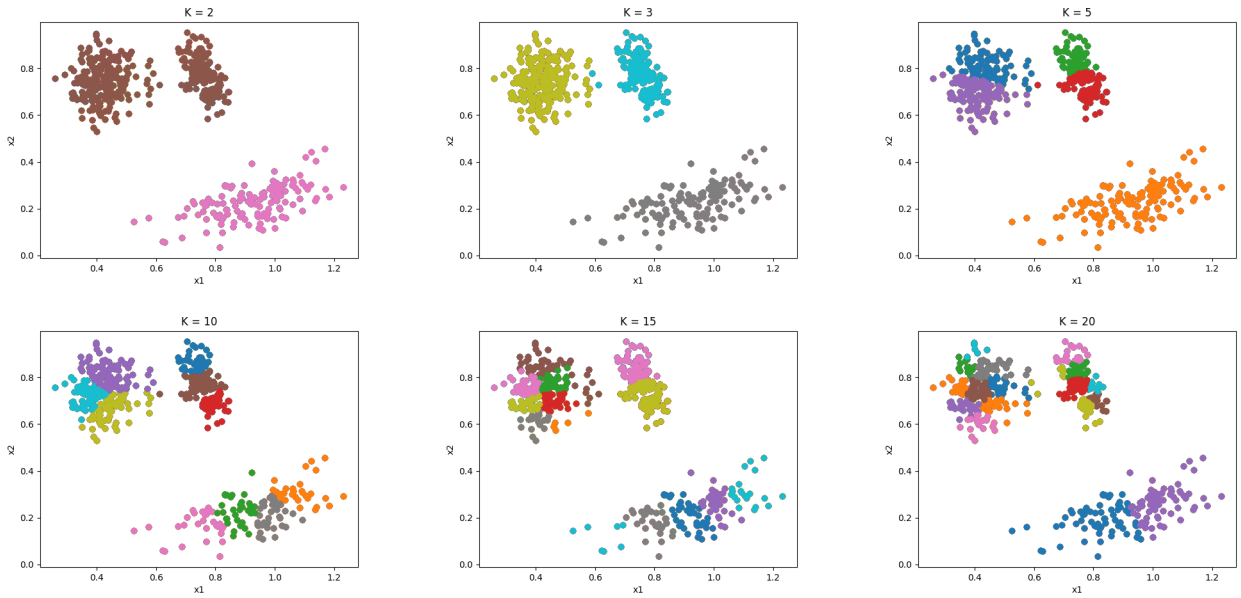


Figure 1: Clusters determined by K-means applied to the 2D data from `2DGaussianMixture.csv` for $K = 2, 3, 5, 10, 15, 20$.

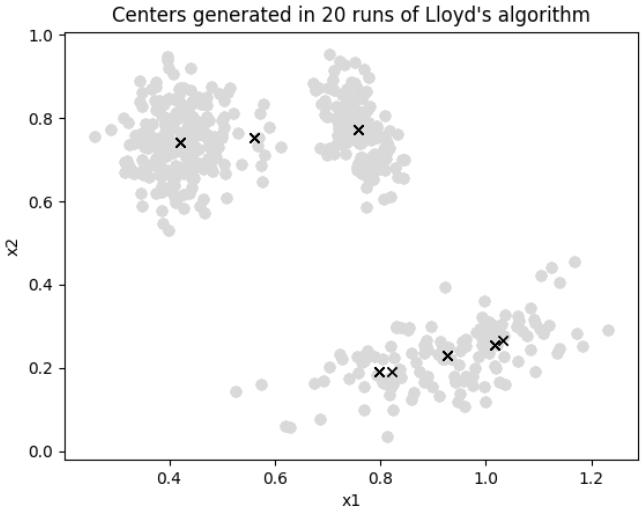


Figure 2: The centers (denoted by 'x') generated during 20 runs of Lloyd's algorithm with random initializations.

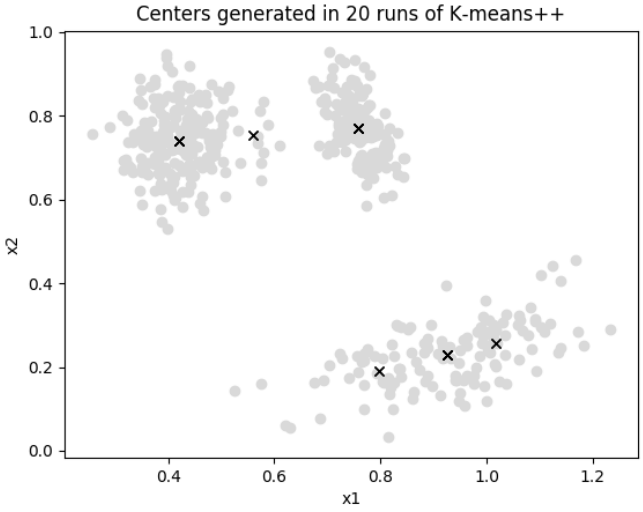


Figure 3: The centers (denoted by 'x') generated during 20 runs of Lloyd's algorithm with K-means++ initializations.

$J(\mathcal{Z}, \mu)$ can be made to be 0 by choosing each point as a center. On the other hand, $K = 1$, the center will be the mean of all the points and $J(\mathcal{Z}, \mu)$ will be large unless the points are all very close together to begin with.

A better strategy would be to experiment with a range of reasonable values of K and track how robust each is to differences in initialization. For each K would could run K-means multiple times on the same data set (using different initializations of the centers) and track how different the resulting clusters are between runs. One could track the similarity of clusters either by checking how similar the clusters are to one-another from run to run, or perhaps by observing fluctuations in $J(\mathcal{Z}, \mu)$. If one has selected a "good" value of K we might expect the algorithm to output somewhat similar clusters each time it is run. Thus we could select the value of K for which the clusters output by K-means showed the most robustness.

(g) My implementation of EM for GMMs terminated after the log-likelihood failed to change by more than 10^{-6} between successive iterations.

(i) Figure 4 shows the **log-likelihood** of the data over EM iterations.

(ii) Figure 5 plots the most likely cluster assignments of the data points along with the cluster centers and the covariance ellipses for each Gaussian.

(iii) This algorithm performed slightly better than K-means with the correct value of K (see the $K = 3$ plot of Figure 1). This is because the two points which K-means misclassifies are both technically closer to the wrong center. K-means does not take the distribution of same-cluster points into account. Since GMM does, it correctly classifies the points. Based on the covariance ellipses shown in Figure 5, it is much more clear that these two points belong in the cluster on the left rather than the one on the right. Since the GMM algorithm takes this into account, it puts the points in their correct cluster.

2. Clustering the BBC News [20 Points]

The dataset we will consider comes from the BBC (`http://mlg.ucd.ie/datasets/bbc.html`). The preprocessed dataset consists of the term-document frequency of 99 vocabulary and 1791 documents chosen from 5 categories: business, entertainment, politics, sport and tech.

From lecture, we learned that the term frequency vector is not a good metric because of its biases to frequent terms. Your first task is to convert the term frequency into tfidf using the following equations:

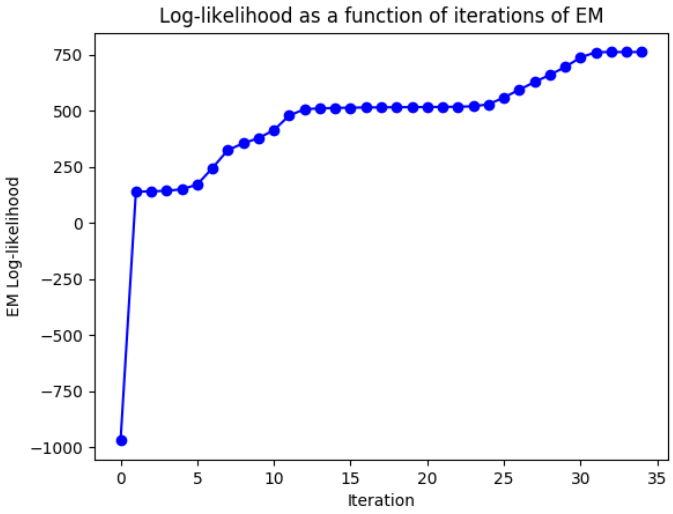


Figure 4: The log-likelihood of the data as a function of EM iterations.

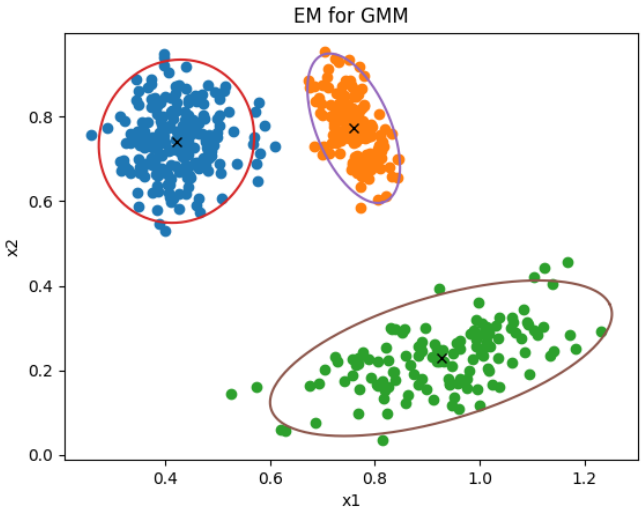


Figure 5: The most likely cluster assignments after termination of the EM algorithm. The means of the clusters are denoted by 'x.' The 95% confidence intervals are shown as solid curves.

$$\text{tf}(t,d) = \frac{f(t,d)}{max\{f(w,d) : (w \in d)\}} \quad (3)$$

$$\text{idf}(t,D) = \log \frac{|D|}{|\{d \in D : t \in d\}|} \quad (4)$$

$$\text{tfidf}(t,d,D) = tf(t,d) \times idf(t,D) \quad (5)$$

(a) Convert the term-doc-frequency matrix into a term-doc-tfidf matrix. For each term t, take the average tfidf over each class $C_i = \{\text{documents in class } i\}$:

$$\text{avg\_tfidf}(t, C_i, D) = \frac{1}{|C_i|} \sum_{d \in C_i} \text{tfidf}(t,d,D)$$

For each class C_i, report the 5 terms with the highest $AvgTfidf$ for the class (e.g Tech: spywar:0.69, aol:0.58, ... Business: ...).

(b) Run k-means with $K = 5$ for 5 iterations, using the centers in "*.centers" for initialization.
Plot the classification error (0/1 loss) versus the number of iterations. *Note: Don't use the optimal mapping procedure from the previous question; you should keep the class ids as they were in the initialization file.*

(c) Run EM with $K = 5$ for 5 iterations. Using "*.centers" as the mean and identity as the covariance of the initial clusters. Initialize $\pi_{1,\ldots,K}$ uniformly.
You need to be careful when updating the covariance matrix Σ_k during the M-step. In particular, the MLE can be ill-conditioned because the data is sparse. To handle this case, we can perform a shrinkage on the MLE: $\hat{\Sigma} = (1-\lambda)\hat{\Sigma}_{MLE} + \lambda I$, which is equivalent to a MAP estimate of the posterior distribution with some prior. In this problem, please use $\lambda = 0.2$.
Plot the classification error (0/1 loss) versus number of iterations.
Plot the log-likelihood versus the number of iterations.

Solution:

Class 0		**Class 1**		**Class 2**		**Class 3**		**Class 4**	
wider	0.7317	publish	0.5767	misdirect	0.8668	venic	0.7518	spywar	0.5942
observ	0.6131	paramount	0.5519	float	0.5538	seychel	0.4871	aol	0.5685
decid	0.4212	kelli	0.5211	gerhard	0.4623	shoe	0.4553	175	0.4589
stabilis	0.2782	hodgson	0.5054	beith	0.4221	vicechairman	0.3791	xbox	0.3765
unit	0.2688	coldplai	0.4762	reimburs	0.4221	coral	0.3094	restat	0.3467

Table 4: The five terms with the highest average tf-idf for each of the five classes in the BBC dataset.

(a) The terms with the highest average tf-idf scores for the five classes are given in Table 4.

(b) Figure 6 plots the classification error for this dataset using K-means with $K = 5$ and initial means given in the `bbc.centers` file.

(c) Figure 7a plots the 0/1 loss and Figure 7b plots the log-likelihood over five iterations of EM applied to the BBC data.

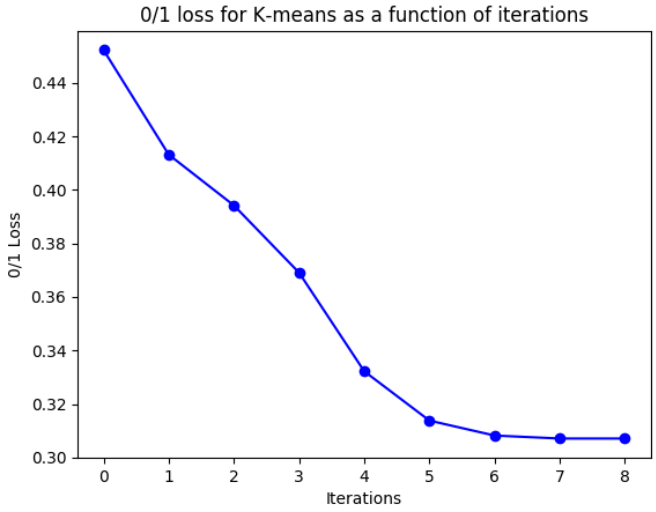
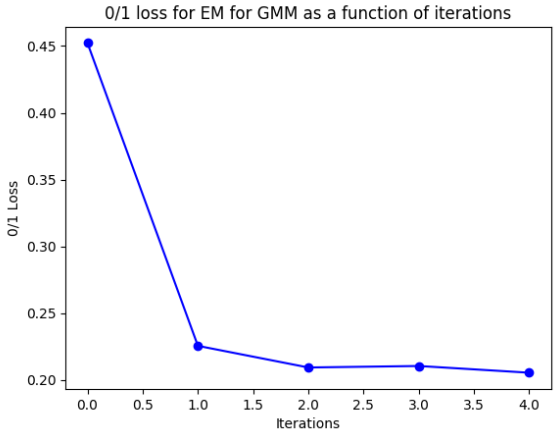
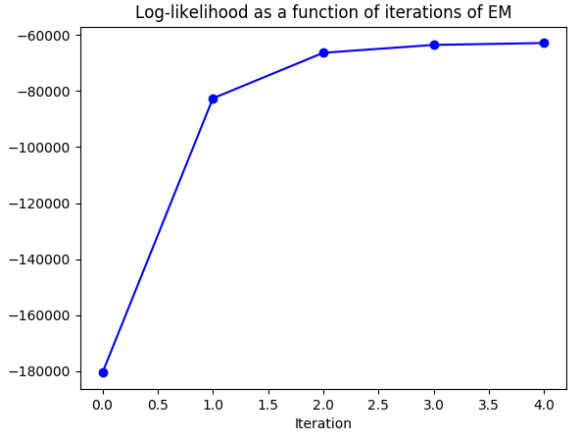


Figure 6: The 0/1 loss as a function of iterations of K-means with $K = 5$ (BBC data)



(a) The 0/1 loss as a function of iterations of EM for GMM



(b) The log-likelihood as a function of iterations of EM for GMM

Figure 7: Some results from applying five iterations of EM for GMM to the BBC data.

3. Scaling up K-Means with MapReduce [35 Points]

The k-means algorithm fits naturally in the MapReduce Framework. Specifically, each iteration of k-means corresponds to one MapReduce cycle:

- The map function maps each document with key being the cluster id.
- During the shuffling phase, each documents with the same cluster id will be sent to the same reducer.
- The reduce function reduces on the clusterid and updates the cluster center.

Because we do not have the ground truth labels for this dataset, the evaluation will be done on the k-means objective function:

$$J(\mathcal{Z}, \mu) = \sum_{i=1}^{N} \|x^i - \mu_{z^i}\|_2^2 \tag{6}$$

$$(\mathcal{Z}^*, \mu^*) = \arg\min_{\mathcal{Z}, \mu} J(\mathcal{Z}, \mu) \tag{7}$$

where z^i is the cluster assignment of example i, μ_j is the center of cluster j.
To get the value of $J(\mathcal{Z}, \mu)$ for each iteration, the reducer with key j will also compute and output the sum of the squares of L_2 distance in cluster j. At the end of each iteration, you can examine the mean and standard deviation of the clusters.
Run k-means with $K = 20$ for 5 iterations. Use the initial cluster centers from "center0.txt".

1. **Plot the objective function value $J(\mathcal{Z}, \mu)$ versus the number of iterations.**
2. **For each iteration, report mean (top 10 words in tfidf) of the largest 3 clusters (by total squared L_2 distance). (Use the printClusters function in the starter code.)**