University of Washington
Department of Computer Science and Engineering / Department of Statistics

CSE 547 / Stat 548 Machine Learning (Statistics) for Big Data

Homework 2
Spring 2015

Issued: Thursday, April 16, 2015 **Due:** Thursday, April 30, 2015

Suggested Reading: Assigned Readings in Case Study II (see website).

Problem 2.1

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:

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

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 $\|v\| \leq 1$,

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

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 $x = u - v$*).

Problem 2.2

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) [8 Points] 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) [6 Points] 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.

NOTE: The Python version of the KD-tree is *very* slow. To run the Python code, set the `PYTHONPATH` to the root directory of the code base, as with the previous assignment. For example, `PYTHONPATH=. python analysis/MethodComparison.py`

Problem 2.3

The Clustering Toolkit: K-Means and EM [80 Points]

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. *Note: In this homework, you will run your Hadoop job on a single machine, rather than a cluster. However, since Hadoop is a distributed abstraction, the exact same code would work in the Cloud.*

The first 2 parts can be implemented in any language, such as MatLab, R or Python. However, you are not permitted to use built in functions beyond standard matrix operations and procedures to sample from Gaussians (e.g., please don't use standard clustering packages). The last part requires Java and Hadoop (see "Instructions for Hadoop Setup"). Starter code is provided for this part (see "Instructions for Starter Code").

1. 2D synthetic data [20 Points]

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:

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

- 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:

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

(b) 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.

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

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.

- Download "bbc_data.zip" from the course website.
- After unzipping the folder, there should be four files: bbc.mtx, bbc.terms, bbc.classes, and bbc.centers.
  - *.mtx: Original term frequencies stored in a sparse matrix in Matrix Market format: each row is in the form of "termid docid frequency".
  - *.terms: List of content-bearing terms in the corpus, with each line corresponding to a row of the sparse data matrix.
  - *.classes: Assignment of documents to natural classes, with each line corresponding to a document.
  - *.centers: Cluster centers for initializing the clusters, with each line corresponding to a center of the cluster.

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:

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

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

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

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

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

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.

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

The dataset for this part contains tfidf from a subset of Wikipedia documents.

- Download "smallwiki.zip" from the course website.
- After unzipping the folder, there should be three files:
  - tfidf.txt: Each row is in the form of "docid|termid1:tfidf1,termid2:tfidf2,...".
  - dictionary.txt: Map of term to term_id.
  - cluster0.txt: The initial clusters centers, with each line corresponding a cluster center in the form of "clusterid|termid1:tfidf1,termid2:tfidf2,...".

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:

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

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

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.)**

A Instructions for Hadoop Setup

If you already set up Hadoop, feel free to skip this section as we do not require using the specific version or configuration.
`http://hadoop.apache.org/docs/r1.2.1/single_node_setup.html` has detailed instructions for setting up a hadoop on a single machine.

For Windows users, we recommend you try on a Linux system if possible (you could install a Ubuntu on a virtual machine). Unfortunately if Windows is your only choice, you may have to go through many hackings.

Here we emphasize some key steps that you need to take care of as you walk through the instructions from the website.

1. The download section provides a list of Hadoop versions. We recommend you download Hadoop version 1.2.1 from
   `http://apache.mirrors.tds.net/hadoop/common/hadoop-1.2.1/hadoop-1.2.1.tar.gz`
2. After unzipping the folder, open `conf/hadoop-env.sh`, find line 9
   `#export JAVA_HOME=/usr/lib/xxxx`,
   and change it into
   `export JAVA_HOME=PATH_TO_YOUR_JAVA_HOME`.
   For Mac users, your Java path is probably "/Library/Java/Home".
   For Linux users, look for your java path under "/usr/lib/jvm/".
3. Hadoop provides three modes. We will only use the Pseudo-Distributed Mode. Pseudo-Distributed mode runs on a single machine but simulates a distributed computing environment.
4. Before you proceed to setup the Pseudo-Distributed mode, please **follow the instructions in the "StandAlone Operation" section and make sure you can repeat the "grep example".**.
5. To configure the Pseudo-Distributed Mode, please follow the "configuration" and "Setup passpharaseless ssh".
6. In the "Execution" step, notice there are extra steps:
   `$ bin/hadoop fs -put conf input`,
   `$ bin/hadoop fs -cat output/`
   This is because in Pseudo-Distributed mode, the hadoop program must read and write through HDFS (Hadoop Distributed File System). Here are some useful commands for hdfs.
   - List a directory in hdfs: `bin/hadoop fs -ls PATH_TO_DIR`
   - Create a directory in hdfs: `bin/hadoop fs -mkdir PATH_TO_DIR`

- Remove a directory in hdfs: `bin/hadoop fs -rmr PATH_TO_DIR`
- Copy files from local to hdfs: `bin/hadoop fs -put SOURCE_PATH TARGET_PATH`
- Copy files from hdfs to local: `bin/hadoop fs -get SOURCE_PATH TARGET_PATH`

You need to put the data files into HDFS. For example, the starter code assumes the data is in the location: `hdfs:///user/yourusername/kmeans/`. You can import your data into hdfs using the following commands:

```
bin/hadoop fs -mkdir kmeans bin/hadoop fs -put
/PATH/TO/DATA/smallwiki/tfidf.txt kmeans/tfidf.txt bin/hadoop fs -put
/PATH/TO/DATA/smallwiki/dictionary.txt kmeans/dictionary.txt
bin/hadoop fs -mkdir kmeans/cluster0 bin/hadoop fs -put
/PATH/TO/DATA/smallwiki/cluster0.txt
kmeans/cluster0/cluster0.txt
```

7. **Follow the instructions in the "Pseudo-Distributed Mode" section, and make sure you can repeat the "grep example" at the end.**
8. (Optional) If you would like to work on the word count example, a complete tutorial can be found at `http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html`.
9. If you have any trouble setting up Hadoop, please come to one of the office hours as soon as possible.

B Instructions for Starter Code

First, make sure you have hadoop running.
Java:

1. Download the Java version of "KmeansMR.zip" from the course website.
2. To import the starter code, go to the menu: "File" → "Import". Under "general", select "Existing Projects into Workspace", and click "Next". Choose "Select archive file", and find "stubs.zip" that you downloaded from the course website. Click Finish.
3. Right click on the imported project name "KmeansMR" → "Build Path" → "Add External Archives...". Select "hadoop-core-1.2.1.jar" from your hadoop directory and the project should compile.
4. To run your code with Hadoop, go to the menu: File → Export. Under "Java", select "JAR file", type in the destination file, for example: "kmeans.jar". Click Finish. In commandline, cd to your hadoop directory, type:

```
bin/hadoop jar YOUR_PATH/kmeans.jar
edu.uw.cs.biglearn.hadoop.kmeans.mapred.KmeansMRDriver
```

Python:

1. Download the Python version of "KmeansMR.zip" from the course website.
2. Update the path for hadoop and the path to the kmeans data within HDFS in the `KmeansMRDriver.py` file.
3. Run with `python KmeansMRDriver.py`

NOTE: In `KmeansMRDriver.java` or `KmeansMRDriver.py`, you may need to change the locations of the data in hdfs (replacing `haijie` with your username, assuming you followed the instructions above for putting the data into hdfs).
This will run Kmeans with unimplemented map and reduce functions for one iteration.
NOTE: Between runs of kmeans, you will need to remove the `kmeans/output` directory in hdfs: `bin/hadoop fs -rmr kmeans/output`.
If you use the starter code, the files you need to hand in are:

- KmeansMapper.java (or KmeansMapper.py)
- KmeansReducer.java (or KmeansReducer.py)