From BMI 214

Site: Project2

On this page... (hide)

Due electronically by **11:59PM, Thursday, October 31, 2013**.

Note that programming projects should be completed individually. You may discuss algorithms with others, but the coding should be done alone. You must explicitly name everyone with whom you discussed this project in the README.txt document and the project quiz you submit. Students must abide by the terms of the **Stanford Honor Code**.

Remember to consult **Piazza**, as many common questions will be asked and answered there. Students should actively participate in answering the questions of fellow students - remember this counts as part of your participation grade.

1. Introduction

In this project you will be implementing both K-nearest neighbor (a supervised machine learning algorithm) and K-means (an unsupervised machine learning algorithm) to analyze microarray data.

2. K nearest neighbors

Expression data was taken from 72 patients with acute leukemia, to see if expression alone can differentiate between acute lymphoblastic leukemia (ALL) and acute myeloid leukemia (AML). Although symptoms are similar for these two diseases, the response to treatment is quite variable between the two cancer types. For the first part of this project, you will use K-nearest neighbors to classify patients with acute leukemia.

The basic idea is the following: given an expression vector for an unclassified patient, find the K "closest" expression vectors from the classified patients and let them vote on the classification for the query patient. If some p percent or more of the K neighbors are ALL patients then classify the unknown patient as ALL, otherwise classify it as AML. Your function should take in the positive training set (ALL patient data), a negative training set (AML patient data), a positive integer for K, and a value between 0 and 1 for p. It should return a vector of predictions -- one prediction for each patient in the test set. Use Euclidean distance as your distance metric.

To assess the accuracy of a classifier, we typically use an approach known as n-fold cross-validation. In this strategy we:

1. Divide the training set into n groups randomly. Divide the positive set into n groups and the negative set into n groups and then combine them so that each of the final n groups have equal proportions of ALL and AML patients.
2. We use 1 of the n subsets as our test set and the other n-1 as our training set.
3. We repeat step 2, n times. We assess the test accuracy (sensitivity, specificity, or a combination of the two) by averaging over all n folds. You will be using 4-fold cross-validation for this project.

Some useful definitions:

- Sensitivity = TP/(TP+FN)
- Specificity = TN/(TN+FP)
- Accuracy = (TP+TN)/total

Where TP = number of true positives, FN = number of false negatives, FP = the number of false positives, and TN = the number of true negatives in the test set.

Data:

The dataset you will use for classification consists of microarray expression data for 72 leukemia patients. Experiments are described in detail in the paper:

T.R. Golub, D.K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J.P. Mesirov, H. Coller, M. Loh, J.R. Downing, M.A. Caligiuri, C.D. Bloomfield, and E.S. Lander, D. **Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression**. Science. 99:531-537.

The expression data for the 28 AML patients can be found in the tab delimited file **AML.dat**. Each column in the file represents a different AML patient. Each of the 7129 rows correspond to a different gene. The expression data for the 44 ALL patients can be found in **ALL.dat**. The genes in this data set are in the same order as those in the AML data set. A description of the 7129 genes can be found in **human_genes.txt**.

You can download all data files for KNN part through the individual links above or all together here: **knn.zip**

Implementation notes for KNN:

Your script should be run with the following command line arguments:

```
knn.py ALL.dat AML.dat k p
```

where *k* is the number of neighbors that have a "vote", and *p* is the minimum fraction of "votes" needed in order to classify a sample as positive.

Output should be to both <stdout> and a file called "knn.out" *exactly* as below, where 10 and 0.75 are used for k and p respectively (other #'s are made up):

```
k: 10
p: 0.75
accuracy: 0.68
sensitivity: 0.85
specificity: 0.90
```

Notes for n-fold cross validation, in the case when n=4:

- Divide the data into 4 equally (or as close to equally) sized sets. Note that you only need to randomize and split the data once.
- Make sure that the ratio of ALL to AML patients are the same in each set. You can do this by splitting the ALL and AML data into 4 sets separately and then combining them.
- Now take 3 of the 4 subsets and combine them to make the training set.
- Next run KNN on this training set using the remaining 1 (out of the 4) set to test the classification. We want to repeat this 4 times. Each time one of the 4 is left out as the test set, and the remaining 3 are used as the training set.
- NOTE: When partitioning data for the cross-validation step of KNN you cannot use a deterministic partition, but rather **must** divide the data randomly. Frequently you can get artifacts due to the ordering of the data. For example, imagine you are doing KNN on the yeast data to try to classify ribosomal genes. If you deterministically divided the data into 4 then all of the ribosomes would be in the same test set which would drastically affect your results.
- Additional information on n-fold cross-validation can be found **here**.

---

3. K-means

Analysis of microarray data can be on the experiment level (cluster/classify experiments), at the gene level (cluster/classify genes), or both (see **biclustering** if interested). In part I you performed k-nearest neighbors to classify microarray experiments based on gene expression. In this section, you will perform K-means clustering on the genes, clustering together genes who have similar expression profiles over a multitude of experimental conditions.

The basic idea of K-means clustering is the following:

1. Choose an initial partition of the data into K clusters. The centers, called centroids, can be picked at random or specified by the user.
2. For each data point, assign it to a cluster such that the euclidean distance from the data point to the centroid is minimized.
3. For each cluster, recalculate the centroids based on all the data points that belong to that cluster.
4. Iterate steps 2 and 3 until convergence (no data points change cluster) or after a certain number of iterations.

Data:

The first set of data is made-up **test data** which can be found here: **testdata.dat**. Each row corresponds to one gene and each column corresponds to one experimental condition. Sample centroids for the test data, in which each row represents one of the initial centroids, can be found here: **testdata_centroids.dat**.

The second set of data is **experimental** and consists of microarrays spanning 79 experiments on **yeast**, including measurements of expression taken after environmental changes were imposed on the yeast. For example, some of these conditions were starvation (causing the yeast to form spores), changing the sugar supply (causing the yeast to ferment rather than respire), and synchronizing the cells to force them to pass through the stages of cell division at the same time. The experiments are described in detail in the paper:

> Eisen, M.B., Spellman, P.T., Brown, P.O., Botstein, D. **Cluster analysis and display of genome-wide expression patterns**. PNAS. 95:14863-14868.

The expression patterns for the 2467 genes can be found in the tab delimited data file called **yeast.dat**. Each column in the file represents one experimental condition. The file **yeast_experiments.txt** lists the experiments to which each column corresponds. The common gene names and a short functional annotation for the 2467 genes can be found in **yeast_gene_names.txt**. A sample centroid input file for the experimental data for K=3 can be found here: **experimental_centroids.txt**.

You can download all data files for the K-means part through the individual links above or all together here: **kmeans.zip**

Implementation notes for K-means:

Your script should be run with the following command line arguments:

```
kmeans.py k expression.dat max.it centroids.txt
```

where *k* is the number of centroids, *expression.dat* is the tab delimited file containing expression data, *max.it* is the maximum number of iterations allowed, and *centroids.txt* is an optional parameter specifying a file from which initial centroids should be read. If this filename is not present then centroids should be generated randomly.

Output should be written to a tab-delimited file called "kmeans.out". The output should contain the cluster assignment for each gene by index (i.e the first gene in the list is gene #1, etc.)

For example (K=3):

```
1    2
2    1
3    3
4    1
5    2
6    1
7    1
8    2
9    1
```

where the first column is the gene number and the second column designates a cluster assignment. The numbering of the clusters is arbitrary as long as two genes in the same cluster have the same cluster number.

Notes for stopping conditions:

- The max.it argument specifies the maximum number of iterations we allow before cutting off if convergence is not achieved. It is useful for testing (and grading) purposes. It allows you to get "pretty good" results in cases when it is taking a long time to converge. For this project we generally want it set pretty high (I will test it on 50 iterations).
- Do not rely on the maximum iteration parameter to stop the program. If k-means converges before the maximum number of iterations is reached then it should stop then.

---

4. Quiz

You will run your code to answer the questions in the quiz:

> **Project 2 Quiz**

Use this template to fill in your answers:

project2_quiz.txt

5. Submission contents

What you will need to submit:

1. Your scripts knn.py and kmeans.py. Remember, we expect well-commented code, especially if you want partial credit for "almost working" code.
2. A README file. Please call it README.txt and include:
   - Your name and SUNet ID
   - Instructions on how to run your programs
   - Pseudocode for KNN and KMEANS (concise)
   - Names of people with whom you discussed this project.
3. A file called project2_quiz.txt, containing your answers to the quiz.
4. Three plots, saved as JPG files: question1.jpg, question2.jpg, question3.jpg
5. A folder called "output" with the following output files:
   - KNN: knn.out.1, knn.out.5, knn.out.10, knn.out.15, knn.out.30, knn.out.50 for k = 1, 5, 10, 15, 30 and 50 respectively and p=0.5.
   - KMeans: kmeans_testdata.out using the test centroids and kmeans_experimental.out using K = 2, picking the first gene in yeast.dat and gene number 2467 as your starting centroids.

6. Submission instructions

Log in to a cardinal machine:

```
ssh sunetid@cardinal.stanford.edu
```

Place all of your relevant files into one directory. Do not include irrelevant files; please only include those listed above under submission contents.

```
~/biomedin214/p2/submit
```

cd into your submit folder.

```
cd ~/biomedin214/p2/submit
```

Run the class submission script:

```
/usr/class/biomedin214/bin/submit p2
```

Note the space before p2. Be sure to run the script from within your submission folder.

You may submit multiple times; simply re-run the script. Each new submission overwrites the previous submission. Your submission date will be the final submission received, and late days will be charged accordingly.