CS532
Pattern Recognition
Lab: Genomic Data Analysis and Classification

Group 3: Layla Barkal, Lisa Ossian, Louis Schultz

November 10, 2014

1 Intro

The objective of this lab is to build a classifier for a highly underdetermined system. An underdetermined system has more unknowns than it does equations and thus will either have no solution or infinitely many. In particular, we will be working with a gene expression dataset comprised of gene expression values of tissue samples from individuals with and without breast cancer. This is underdetermined data, as we have 8141 gene expression values for each of only 295 patients observed. To do this classification task, we will use singular value decomposition (SVD) and least squares. The SVD of a matrix decomposes it in such a way that we can identify the most important information in the matrix and remove noise. Using the SVD on our gene expression dataset, we can reduce the dimensionality of the data, so we only consider the most important genes. Least squares is a classification technique in which a vector of weights x is found to solve the equation $Ax = b$, where A represents the matrix of data and b represents the classifications. We will use least squares to solve for x and in turn use this to predict whether a new patient has cancer.

The SVD is considered a fundamental result of linear algebra. It can be used for a variety of applications, including image compression, search engine implementations, and recommendation systems. The winners of the Netflix Prize, a competition with a \$1 million grand prize in which the goal was to improve Netflix's recommendation accuracy by 10%, used the SVD in their algorithm.

2 Overview

The SVD takes a matrix $A \in \mathbb{R}^{m\times n}$ with rank $r \leq \min(m, n)$ and decomposes it into $A = U\Sigma V^T$. The matrix $U \in \mathbb{R}^{m\times r}$ has orthonormal columns which span the *columns* of A. The matrix $V \in \mathbb{R}^{n\times r}$ also has orthonormal columns, but these span the *rows* of A. The matrix $\Sigma \in \mathbb{R}^{r\times r}$ is a diagonal matrix containing the *singular values* of A. The singular values are in decreasing order — that is, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r$ — and in general can be thought to reflect the dominance (i.e., information contained in A) of the corresponding singular vectors (columns of U and V). This is one of the principle advantages of the SVD: U, Σ, and V are ordered such that earlier columns contain more information than later columns. It is in this way that we can use the SVD to reduce the dimensionality of A. Fundamentally, we seek to reduce the dimensionality of A because we believe (and plotting the singular values can confirm this belief) that most of the information in A can in fact be captured in a lower dimensional subspace.

We perform dimensionality reduction by simply setting all singular values past k to zero, $\sigma_{k+1} = \cdots = \sigma_r = 0$. Mechanically, this amounts to computing $A_2 = \sum_{i=1}^{k} \sigma_i u_i^T v_i$. To choose how many singular values to retain, we simply find the smallest k such that $\frac{\sum_{i=1}^{k}\sigma_i}{\sum_{i=1}^{r}\sigma_i}$. The value $\rho \in [0, 1]$ is chosen by the researcher and reflects the proportion of information in A we wish to retain. For our purposes, we'll set $\rho = 0.9$.

Once we have reduced the dimensionality of A by using the SVD, we can build our classifier. Given a vector of classifications $b \in \mathbb{R}^m$, the least squares problem is to find a vector of weights $\hat{x}$ such that

$\|b - A_2x\|^2$ is minimized. The standard method for solving least squares is to use the normal equations, $\hat{x} = (A_2^T A_2)^{-1} A_2^T b$. However, for underdetermined systems, we generally prefer the regularlized least squares solution, $\hat{x} = (A_2^T A_2 + \lambda I)^{-1} A_2 b$, where $\lambda > 0$. For sufficiently small λ, this will yield the minimum norm solution to the underdetermined system. In practice, we tend to set λ to be very small (e.g., $\lambda = 10^{-6}$).

3 Warm-up

It is useful at this point to give a brief preview of the steps we will take in this warm-up. As a first step, we will use the SVD to reduce the dimensionality of our data matrix A; we'll call this matrix A_2. Next, we will use least squares to estimate $\hat{x}$ in the linear system $A_2x = b$. Because this is an underdetermined system, there exist either no solutions or infinitely many solutions. In this case, there will be infinitely many. Thus, we will need some criteria to choose a particular $\hat{x}$ from the infinitely many possible solutions. The criteria we will choose is to minimize the norm of the solution vector, which motivates the use of regularized least squares.

3.1 Setting up the classification problem

Later in the lab you will be given data on 295 patients, each with 8141 associated gene expression levels. You will also have information on whether the patient was diagnosed with breast cancer. Keeping in mind the methods presented above, think about how you would set up the problem in order to use gene expression information to predict whether a patient will be diagnosed with breast cancer.

(a) How will you set up the A matrix and b vector using the information described above? (Hint: In this case, since there are two possible outcomes, you may want to let $b_i = 1$ signify a patient has breast cancer and let $b_i = -1$ otherwise.)

(b) Conceptually, what information are you attempting to eliminate when performing dimensionality reduction on this data?

(c) When performing least squares on the reduced dataset, what will your $\hat{x}$ vector look like and what does it tell you?

3.2 Dimensionality reduction using SVD

The following Matlab code, which can be found in the file `dimensionality_reduction.m`, generates a matrix $A \in \mathbb{R}^{50\times200}$ and a vector of predictions $b \in \mathbb{R}^{50}$. Using A and b, complete the problems below.

```
q=2;
m=50;
n=200;
mu=1e-4;

rng(532);
O=randn(m,q);
P=O*randn(q,n-q);
b=[O,P]*randn(n,1)+randn(m,1);
x_true=inv([O,P]'*[O,P]+mu*eye(n))*[O,P]'*b;

Q=P;
for i=1:(n-q)
        Q(:,i)=Q(:,i)+(1e-2)*randn(m,1);
end

A=[O,Q];
x_hat=inv(A'*A+mu*eye(n))*A'*b;
```

(a) Use the Matlab function `svd` with the economy size option to compute the SVD of $A = U\Sigma V^T$. Plot the sprectrum $\sigma_1, \sigma_2, \ldots, \sigma_{10}$ of A. What is the rank of A? How many dimensions seem important?

(b) Calculate A_2, the approximation of A with minimum rank such that $\rho = 0.9$. What is the rank of this matrix?

(c) Using the code below, which can be found in the file `svd_plot.m`, visualize the dataset by projecting the columns and rows on to the first three principal component directions. Use the `rotate` tool in the Matlab plot to get different views of the three-dimensional projections.

```
xu = [];
yu = [];
zu = [];

for i = 1:n
    xu(:,i) = U(:,1)'*A(:,i);
    yu(:,i) = U(:,2)'*A(:,i);
    zu(:,i) = U(:,3)'*A(:,i);
end

figure(1)
scatter3(xu,yu,zu);

xv = [];
yv = [];
zv = [];

for i = 1:m
    xv(:,i) = A(i,:)*V(:,1);
    yv(:,i) = A(i,:)*V(:,2);
    zv(:,i) = A(i,:)*V(:,3);
end

figure(2)
scatter3(xv,yv,zv);
```

3.3 Estimating feature weights using least squares

Consider the code given in Warm-up 3.2 again. The vector `x_true` is the underlying linear relationship we seek to estimate.

(a) Compare `x_hat`, the estimate of `x_true` using regularized least squares, with `x_true`. How does this comparison depend on λ? (Hint: Compare the Euclidean distances between the estimates of x and the true x.)

(b) Compute `x_hat_2`, the estimate of `x_true` having first performed dimensionality reduction on A using the SVD (as in Warm-up 3.2(b)), before doing regularized least squares. Compare `x_hat_2` with `x_true`.

(c) Which of `x_hat` and `x_hat_2` appears to be a better estimate of `x_true`? Why do you think this is the case?

4 Classifying breast cancer data

For the main part of this lab, we will implement several classification techniques on an underdetermined dataset contained in `Breast_Cancer.mat`. Let us denote the matrix of 8141 gene expression values for 295 patients as our data matrix A, where $A \in \mathbb{R}^{295\times 8141}$, and let the vector of classifications for these 295 patients be called b, such that $b \in \mathbb{R}^{295}$. The techniques we will consider include regularized least squares without dimensionality reduction, a least squares approach using the truncated SVD of our data matrix A, and lastly, a technique in which we isolate the k most predictive genes and perform least squares for classification.

In designing the three classifiers mentioned, we will use cross validation. Each classifier will be run l times, and the final error of the classifier will be averaged over these l trials. The data will be divided randomly into training, tuning, and testing sets for regularized least squares (Section 4.1) and for least squares using the truncated SVD (Section 4.2). For these methods, we will use the training set to estimate $\hat{x}$ for each regularization parameter. Then, we will select the $\hat{x}$ associated with the lowest error rate on the tuning set. Using this best $\hat{x}$ to classify the testing set, we can compute the error for a trial. For least squares using the k most predictive genes (Section 4.3), there are no parameters to tune, so a tuning set will not be needed. The code for these three methods is contained in the files `regLs.m`, `truncSvdLs.m`, and `singleGeneLs.m`.

4.1 Regularized least squares

The regularized least squares problem is given by $\min \|b-Ax\|_2^2+\lambda\|x\|_2^2$. In the `regLs` function below, we calculate $\hat{x}$ on the training set with the closed-form solution $\hat{x} = (A^TA+\lambda I)^{-1}Ab$, for each regularization parameter $\lambda = 0.5, 1$, and 2.

```
unction finalError = regLs(fileName,trNum,tuNum,trials)
    % Load data and get A matrix of data and b vector of classifications.
    load(fileName,'-mat');
    A = X;
    b = Y(:,2);
    % The set of lambdas to consider.
    lambda = [0.5 1 2];
    % Initialize error sum, which will contain the sum of errors from all
    % trials.
    errorSum = 0;
    % For each trial, we want to train, tune, and test.
    for i = 1:trials
        % Get new random training, tuning, and testing sets.
        [A_tr,b_tr,A_tu,b_tu,A_te,b_te] = createSets(A,b,trNum,tuNum);
        % Initialize bestError at its max so we can find the lowest error.
        bestError = 1;
        bestLambda = 0;
        % Tune using each possible value of lambda.
        for k = 1:size(lambda,2)
            I = eye(size(A_tr,2));
            x_tr = inv(A_tr'*A_tr+lambda(k)*I)*A_tr'*b_tr;
            error = nnz(b_tu - sign(A_tu*x_tr))/size(b_tu,1);
            % Check if current error is better than bestError.
            if (error < bestError)
                bestError = error;
                bestLambda = lambda(k);
                x_tu = x_tr;
            end
        end
        testingError = nnz(b_te - sign(A_te*x_tu))/size(b_te,1);
        % Add testing error to error sum.
        errorSum = errorSum + testingError;
        fprintf('trial: %d lambda: %f error: %f\n',i,bestLambda,
            testingError);
    end
        % Get final error estimate by dividing the error sum by the total
        % number of trials.
        finalError = errorSum/trials;
        fprintf('final error: %f, accuracy: %f\n',finalError,1-finalError);
end
```

The function `createSets` for randomly generating the training, tuning, and testing sets for regularized least squares and for least squares using the truncated SVD (Section 4.2) is given below.

```matlab
function [A_tr,b_tr,A_tu,b_tu,A_te,b_te] = createSets(A,b,trNum,tuNum)
    % Generate a vector with numbers from 1 to size(A,1), representing the
    % indices of the rows of A.
    allRows = 1:size(A,1);
    % Shuffle around the row indices.
    allRows = allRows(randperm(length(allRows)));
    % Grab sets for training, tuning, and testing.
    tr = allRows(1:trNum);
    tu = allRows(trNum+1:trNum+tuNum);
    te = allRows(trNum+tuNum+1:size(allRows,2));
    % Generate training, tuning, and testing data matrices.
    A_tr = A(tr,:);
    A_tu = A(tu,:);
    A_te = A(te,:);
    % Generate training, tuning, and testing classifications.
    b_tr = b(tr);
    b_tu = b(tu);
    b_te = b(te);
end
```

The code above can be run with the command `regLs('Breast_Cancer',trNum,tuNum,trials)`.

(a) Setting $l = 1$, the size of the training set to 40, and the size of the tuning set to 40, what is the error rate on the testing set? Considering that our data is underdetermined, does this seem like a reasonable estimate? Why or why not?

(b) Keeping $l = 1$, experiment with the size of the training and tuning sets. What sizes of these sets seem to give the lowest error rate?

(c) Keeping $l = 1$ and using the best training and tuning set sizes found in (b), experiment with λ by modifying `lambda` in the `regLs` function. Identify a value of λ that seems to give you the lowest error rate.

(d) Setting $l = 10$ and using the best training and tuning set sizes found in (b), what is the average error rate over all trials? Which value of λ seems to give the lowest error rate?

(e) What is a problem with using this method on our data?

4.2 Least squares using truncated SVD

Gene expression data is prone to being noisy because there are so many influencing factors. These factors can include what the patient ate that morning, how the biological specimen was collected, pipetting error during sample preparation, and heterogeneity of hybridization during the microarray assay, among other sources of variation.

Luckily, we can reduce the dimensionality of our data to get rid of most of this noise. One method for doing this is the truncated SVD, where we consider only the k dimensions that contain the most information. How much information is contained in a given direction can be determined by the size of the singular value. Our $\hat{x}$ for this technique can be computed with $\hat{x} = \sum_{i=1}^{k} \frac{u_i^T b}{\sigma_i} v_i$, where u_i and v_i represent the ith columns of U and V, respectively.

Least squares using the truncated SVD is implemented in the `truncSvdLs` function below. The best value of k is selected during the tuning phase from lines 19 to 31. This value of k represents the k dimensions we should reduce our data to in order to give us the lowest error rate on the tuning set.

Run this function with the command `truncSvdLs('Breast_Cancer',trNum,tuNum,trials)`.

```matlab
1  function finalError = truncSvdLs(fileName,trNum,tuNum,trials)
2      % Load data and get A matrix of data and b vector of classifications.
3      load(fileName,'-mat');
4      A = X;
```

```
 5    b = Y(:,2);
 6    % Initialize error sum, which will contain the sum of errors from all
 7    % trials.
 8    errorSum = 0;
 9    % For each trial, we want to train, tune, and test.
10    for i = 1:trials
11        % Get new random training, tuning, and testing sets.
12        [A_tr,b_tr,A_tu,b_tu,A_te,b_te] = createSets(A,b,trNum,tuNum);
13        % Compute SVD.
14        [U,D,V] = svd(A_tr,0);
15        % Initialize bestError at its max so we can find the lowest error.
16        bestError = 1;
17        bestK = 0;
18        % Tune using each possible value of k.
19        for k = 1:trNum
20            x_tr = zeros(size(A_tr,2),1);
21            for j = 1:k
22                x_tr = x_tr + ((U(:,j)'*b_tr)/D(j,j))*V(:,j);
23            end
24            error = nnz(b_tu - sign(A_tu*x_tr))/size(b_tu,1);
25            % Check if current error is better than bestError.
26            if (error < bestError)
27                bestError = error;
28                bestK = k;
29                x_tu = x_tr;
30            end
31        end
32        testingError = nnz(b_te - sign(A_te*x_tu))/size(b_te,1);
33        % Add testing error to error sum.
34        errorSum = errorSum + testingError;
35        fprintf('trial: %d k: %d error: %f\n',i,bestK,testingError);
36    end
37        % Get final error estimate by dividing the error sum by the total
38        % number of trials.
39        finalError = errorSum/trials;
40        fprintf('final error: %f, accuracy: %f\n',finalError,1-finalError);
41 end
```

(a) Use the Matlab function `svd` with the economy size option to compute the SVD of our data matrix $A = U\Sigma V^T$. Plot the sprectrum $\sigma_1, \sigma_2, \ldots, \sigma_{295}$ of A. What is the rank of A? How many dimensions seem important?

(b) Setting $l = 10$, the size of the training set to 65, and the size of the tuning set to 110, compute the average value of k over all trials. Plot a horizontal line at this average on top of the plot you created in (a). Does this look like a reasonable guess of how many dimensions seem important?

(c) Setting $l = 50$ and keeping the sizes of the training and testing sets the same as in (b), how does your error rate compare to the error rate you obtained in (b)?

(d) Keeping $l = 50$ and keeping the sizes of the training and testing sets the same as in (b), set k at line 19 to be equal to the average value of k you found in (b) rounded to the nearest integer. For instance, if the value of k you found in (b) was 31.19, then line 19 should be replaced with `for k = 31:31`. What error rate do you obtain?

4.3 Least squares using k most predictive genes

Though the technique considered in Section 4.2 reduces the dimensionality of the data, it would be nice to know if certain genes are more important in predicting the presence of the disease. It seems unlikely that all 8141 genes are important to the predictive process. The technique considered in this section allows us to identify the k most predictive genes. Note that as we mentioned above, this method does not involve tuning. Therefore, we have a new function `createSets` without the tuning input argument.

```matlab
function [A_tr, b_tr, A_te, b_te] = createSets(A,b,trNum)
    % Generate a vector with numbers from 1 to size(A,1), representing the
    % indices of the rows of A.
    allRows = 1:size(A,1);
    % Shuffle around the row indices.
    allRows = allRows(randperm(length(allRows)));
    % Grab sets for training, tuning, and testing.
    tr = allRows(1:trNum);
    te = allRows(trNum+1:size(allRows,2));
    % Generate training, tuning, and testing data matrices.
    A_tr = A(tr,:);
    A_te = A(te,:);
    % Generate training, tuning, and testing classifications.
    b_tr = b(tr);
    b_te = b(te);
```

After splitting the data into a random training and testing set, we perform least squares on each individual gene. This can be seen in the function `kTopGenesMinError` below from lines 13 to 16. Notice that we add a column of ones to our training and testing sets. This is so we can fit the data to a line with any intercept. After l trials, this function returns the indices of the k genes that produce the lowest average error rates (`minErrorsGenes`) along with these error rates (`minErrors`).

```matlab
1  function [minErrors, minErrorsGenes] = kTopGenesMinError(A,b,kTopGenes,trNum
       ,trials)
2      % Initialize a vector that will contain sums of testingError by gene
3      % for all trials.
4      sumErrorVec = zeros(size(A,2),1);
5      % For each random trial, calculate testingError using each gene
6      % individually for classification.
7      for j = 1:trials
8          % Create a new random training and testing set.
9          [A_tr, b_tr, A_te, b_te] = createSets(A,b,trNum);
10         % Get testingError for each gene and add it to sumErrorVec at the
11         % appropriate place.
12         for i = 1:size(A,2)
13             geneA_tr = [A_tr(:,i) ones(size(A_tr,1),1)];
14             geneA_te = [A_te(:,i) ones(size(A_te,1),1)];
15             % Calculate x_tr using LS.
16             x_tr = inv(geneA_tr'*geneA_tr)*geneA_tr'*b_tr;
17             % Calculate testingError and add it to sumErrorVec.
18             testingError = nnz(b_te - sign(geneA_te*x_tr))/size(b_te,1);
19             sumErrorVec(i) = sumErrorVec(i) + testingError;
20         end
21     end
22     % Average the sumErrorVec by dividing each element by the number of
23     % trials.
24     avgErrorVec = sumErrorVec./trials;
25     % Find k genes that produce the smallest errors.
26     [sorted idx] = sort(avgErrorVec);
27     minErrors = sorted(1:kTopGenes);
28     minErrorsGenes = idx(1:kTopGenes);
29 end
```

Once we have identified the k genes that produce the lowest error rates, we can create a new matrix $\tilde{A}$ composed only of these genes, which we denote `newA` in the `singleGeneLs` function below. Next, we perform least squares using $\tilde{A}$ over l more random trials. By averaging the error rates from these l trials, we obtain the final error rate.

```
function finalError = singleGeneLs(fileName,kTopGenes,trNum,trials)
    % Load data and get A matrix of data and b vector of classifications.
    load(fileName,'-mat');
    A = X;
    b = Y(:,2);
    % Find the minimum errors over all trials and the genes that produce
    % these minimum errors.
    [minErrors,minErrorGenes] = kTopGenesMinError(A,b,kTopGenes,trNum,
        trials);
    % Make a new matrix composed only of the minimum-error genes.
    newA = A(:,minErrorGenes);
    % Initialize sum of testingError.
    sumError = 0;
    % For each random trial, calculate testingError.
    for j = 1:trials
        % Create a new random training and testing set.
        [A_tr,b_tr,A_te,b_te] = createSets(newA,b,trNum);
        % Calculate x_tr using LS.
        x_tr = inv(A_tr'*A_tr)*A_tr'*b_tr;
        % Calculate testingError and add it to sumError.
        testingError = nnz(b_te - sign(A_te*x_tr))/size(b_te,1);
        sumError = sumError + testingError;
    end
    % Average sumError by dividing by the number of trials.
    finalError = sumError/trials;
    fprintf('final error: %f accuracy: %f\n',finalError,1-finalError);
end
```

The command `singleGeneLs('Breast_Cancer',kTopGenes,trNum,trials)` runs the code above.

(a) Setting $l = 10$, the size of the training set to 70, and the number of genes to consider to 10, what error rate do you obtain?

(b) Setting $l = 50$ and the size of the training set to 70, compute error rates for $k = 1, 2, \ldots, 50$. Plot these results. How does the error rate change as k increases?

(c) Keeping $l = 50$, the size of the training set at 70, and using the best value of k that you found in (b), run the code. Display the vector of k gene IDs corresponding to the lowest error rate.

(d) Now modify the `createSets` function so that the training set contains the same amount of negative and positive training instances. Keeping the input arguments the same as in (c), how does your error rate compare to the one you found in (c)?

4.4 Gene Interpretation

The breast cancer data used here was gathered from a clinical trial performed in 2002.[1] The purpose of the trial was to validate an algorithm that used gene expression data to diagnose patients. This algorithm was proposed a year before the trial in a separate paper.[2] In this paper, they took samples from patients already diagnosed as having or not having breast cancer and measured their gene expression levels. From this data, they identified 70 genes that together had the best ability to separate the groups. They averaged the expression levels for these 70 genes across the patients they knew were disease-free to create an average disease-free pattern.

In the subsequent clinical trial, newly enrolled patients had their gene expression values for the same 70 genes compared to the averaged disease-free pattern of expression. If the correlation was stronger than 0.3, the patients were diagnosed as being disease-free. Otherwise, they were diagnosed with breat cancer.

What are some reasons the k gene IDs you found in 4.3(c) may not match the 70 genes the researchers used in their algorithm?

4.5 Conclusion

What are some drawbacks of the methods we considered above? Which method would you recommend? Consider the nature of genomic datasets: is the lowest error rate nessassarily associated with the best method? Why or why not?

4.6 References

1. van de Vijver, M. J., He, Y. D., Van't Veer, L. J., Dai, H., Hart, A. A., Voskuil, D. W., ... Bernards, R. (2002). A Gene-Expression Signature as a Predictor of Survival in Breast Cancer. New England Journal of Medicine, 347. doi:10.1056/NEJMoa021967

2. van't Veer, L. J., Dai, H., Van de Vijver, M. J., He, Y. D., Hart, A. A., Mao, M., ... Friend, S. H. (2002). Gene expression profiling predicts clinical outcome of breast cancer. Nature, 415, 530-536. doi:10.1038/415530a