# MLSP homework 2

Mo Zhou <mzhou32@jhu.edu>

Accuracy Table

===

| | Validation Set (Dev) | Test Set |
|---|---|---|
| LDA | 0.66 | 0.51 |
| Lasso | ~0.06 | ~0.08 |
| LDA/Lasso | 0.11 | 0.08 |
| Kmeans/Lasso | 0.74 | 0.48 |
| LDA/Kmeans/Lasso | 0.13 | 0.04 |
| Gaussian | 0.80 | 0.62 |

Discussions

===

Question III:

According to the Gaussian classifier results, the I-vector representation is already well-clustered. However, each class are not

"orthogonal" in the I-vector representation space. As a result, it is very likely that an I-vector from any class can be approximately

reconstructed using the dictionary of another class. Hence the accuracy of the naive "Lasso" is very low.

Using Lasso on LDA representation vectors can slightly improve the performance on both validation (dev) dataset and test dataset,

however, this representation is still correlated between classes, and can still be reconstructed by LDA vectors from other classes

in most cases.

Question IV:

Kmeans with Lasso has a high performance than naive lasso. I speculate that kmeans suffers less from the outlier or noisy data points

than the original dictionary. Without ambiguous data points that may construct data from other classes, the Kmeans/lasso method

can perform well. Since the non-orthogonality of LDA vectors across different classes is not addressed by kmeans, the performance

of LDA/kmeans/lasso is not clearly improved.

Question V:

The gaussian classifier is intuitive and simple, but still performs well because the I-vector has sufficient semantic information

so that data points in each class are well-clustered. Namely, there is a relatively small discrepancy between the I-vector

representation of the dataset and the mixture of Gaussian assumption.

## Preliminary: load data from ../Data/**/Class_*.txt

```
% load training, validation, and testing set
% and do normalization.
clear all;
DataTrain = {};
DataVal = {};
DataTest = {};
ivecdim = 600;
numcls = 24;

for i = 1:numcls
    filename = sprintf('../Data/Train/Class_%d.txt', i);
    data = importdata(filename);
    DataTrain{i} = data ./ vecnorm(data, 2, 2);

    filename = sprintf('../Data/Dev/Class_%d.txt', i);
    data = importdata(filename);
    DataVal{i} = data ./ vecnorm(data, 2, 2);

    filename = sprintf('../Data/Eval/Class_%d.txt', i);
    data = importdata(filename);
    DataTest{i} = data ./ vecnorm(data, 2, 2);
end
clear filename;
clear i;
clear data;
```

## Question I: LDA Train, Val and Test

```
% LDA train
```

```
% global mean i-vector
gmean = zeros(1, ivecdim);
rows = 0;
for i = 1:numcls
    isum = sum(DataTrain{i}, 1);
    gmean = gmean + isum;
    rows = rows + size(DataTrain{i}, 1);
end
gmean = gmean / rows;
```

```
% class-wise centroids
centroids = zeros(numcls, ivecdim);
for i = 1:numcls
    imean = mean(DataTrain{i}, 1);
    centroids(i, :) = imean;
end
```

```
% between-class voc
SB = zeros(ivecdim, ivecdim);
for i = 1:numcls
    nl = size(DataTrain{i}, 1);
    diff = centroids(i, :) - gmean;
    tmp = nl * diff' * diff;
    SB = SB + tmp;
end
```

```
% within-class cov
SW = zeros(ivecdim, ivecdim);
for i = 1:numcls
    for j = 1:size(DataTrain{i}, 1)
        cursor = DataTrain{i};
        diff = cursor(j, :) - centroids(i, :);
        tmp = diff' * diff;
        SW = SW + tmp;
    end
end
```

```
% get LDA matrix
[V, D] = eigs(SB, SW, numcls-1);
```

```
% classifier training
LDATrain = {};
LDACentroids = zeros(numcls, numcls-1);
for i = 1:numcls
    lda = V' * DataTrain{i}';
    lda = lda ./ vecnorm(lda, 2, 2);
    LDATrain{i} = lda;
    LDACentroids(i, :) = mean(lda, 2);
end
```

```
LDACentroids = LDACentroids ./ vecnorm(LDACentroids, 2, 2);
```

```
% classifier testing
LDAVal = {};
LDATest = {};
for i = 1:numcls
    lda = V' * DataTest{i}';
    lda = lda ./ vecnorm(lda, 2, 2);
    LDATest{i} = lda;

    lda = V' * DataVal{i}';
    lda = lda ./ vecnorm(lda, 2, 2);
    LDAVal{i} = lda;
end
```

```
% accuracy on validation set
acc = 0;
count = 0;
for i = 1:numcls
    cossim = LDAVal{i}' * LDACentroids';
    [~,idx] = max(cossim, [], 2);
    acc = acc + sum(idx == i);
    count = count + length(idx);
end
acc = acc / count;
fprintf("Validation Accurcy: %f\n", acc);
```

```
Validation Accurcy: 0.660833
```

```
% accuracy on test set
acc = 0;
count = 0;
for i = 1:numcls
    cossim = LDATest{i}' * LDACentroids';
    [~,idx] = max(cossim, [], 2);
    acc = acc + sum(idx == i);
    count = count + length(idx);
end
acc = acc / count;
fprintf("Test Accuracy: %f\n", acc);
```

```
Test Accuracy: 0.512500
```

## Problem II: Overcomplete Space Representation

```
% concatenate the matrices into a big one
A = [];
for i = 1:numcls
    A = [A; DataTrain{i}];
end
```

```
A = A';
A = single(A);
size(A)
```

```
ans = 1×2
         600       22491
```

```
% calculate indeces
cumlen = [];
for i = 1:numcls
    cumlen(i) = size(DataTrain{i}, 1);
end
cumlen = cumsum(cumlen);
clssta = [0, cumlen(1:end-1)] + 1;
clsend = cumlen;
```

```
% accuracy on the validation set
acc = 0;
count = 0;
for i = 1:numcls
    fprintf("%d", i);
    for j = 1:size(DataVal{i}, 1)
        cursor = DataVal{i};
        cursor = cursor(j, :)';
        errors = [];
        for k = 1:numcls
            Dc = A(:, clssta(k):clsend(k));
            B = lasso(Dc, cursor);
            ac = B(:, 1);
            reconst = Dc * ac;
            error = norm(cursor - reconst);
            errors = [errors, error];
        end
        [~, pred] = min(errors);
        if pred == i
            acc = acc + 1;
        end
        count = count + 1;
        fprintf(".");
        if j >= 10
            break
        end
    end
    fprintf("\n");
end
```

```
1
..........
2
..........
3
..........
4
..........
5
```

```
.........
6
.........
7
.........
8
.........
9
.........
10
.........
11
.........
12
.........
13
.........
14
.........
15
.........
16
.........
17
.........
18
.........
19
.........
20
.........
21
.........
22
.........
23
.........
24
.........
```

```
acc = acc / count;
fprintf("Lasso Validation Accuracy %f\n", acc);
```

```
Lasso Validation Accuracy 0.062500
```

```
% accuracy on the test dataset
acc = 0;
count = 0;
for i = 1:numcls
    for j = 1:size(DataTest{i}, 1)
        cursor = DataTest{i};
        cursor = cursor(j, :)';
        errors = [];
        for k = 1:numcls
            Dc = A(:, clssta(k):clsend(k));
            B = lasso(Dc, cursor);
            ac = B(:, 1);
            reconst = Dc * ac;
            error = norm(cursor - reconst);
```

```
            errors = [errors, error];
        end
        [~, pred] = min(errors);
        if pred == i
            acc = acc + 1;
        end
        count = count + 1;
        fprintf(".");
        if j >= 10
            break
        end
    end
    fprintf("\n");
end
```

```
acc = acc / count;
fprintf("Lasso Test Accuracy %f\n", acc);
```

```
Lasso Test Accuracy 0.079167
```

## Problem III: LASSO with LDA

```
% assemble dataset
% concatenate the matrices into a big one
LA = [];
for i = 1:numcls
    LA = [LA; LDATrain{i}'];
end
LA = LA';
LA = single(LA);
size(LA)
```

```
ans = 1×2
          23       22491
```

```
% accuracy on the validation set
acc = 0;
count = 0;
for i = 1:numcls
    assert(size(LDAVal{1, i}, 2) == 100);
    fprintf(".");
    for j = 1:size(LDAVal{i}, 2)
        cursor = LDAVal{1, i};
        cursor = cursor(:, j)';
        errors = [];
        for k = 1:numcls
            Dc = LA(:, clssta(k):clsend(k));
            B = lasso(Dc, cursor);
            ac = B(:, 1);
            reconst = Dc * ac;
            error = norm(cursor - reconst);
            errors = [errors, error];
```

```
        end
        [~, pred] = min(errors);
        if pred == i
            acc = acc + 1;
        end
        count = count + 1;
        %if j > 10
        %    break
        %end
    end
end
```

```
.....................
```

```
acc = acc / count;
fprintf("Lasso/LDA Validation Accuracy %f\n", acc);
```

```
Lasso/LDA Validation Accuracy 0.110833
```

```
% accuracy on the test dataset
acc = 0;
count = 0;
for i = 1:numcls
    assert(size(LDATest{1, i}, 2) == 100);
    fprintf(".");
    for j = 1:size(LDATest{i}, 2)
        cursor = LDATest{1, i};
        cursor = cursor(:, j)';
        errors = [];
        for k = 1:numcls
            Dc = LA(:, clssta(k):clsend(k));
            B = lasso(Dc, cursor);
            ac = B(:, 1);
            reconst = Dc * ac;
            error = norm(cursor - reconst);
            errors = [errors, error];
        end
        [~, pred] = min(errors);
        if pred == i
            acc = acc + 1;
        end
        count = count + 1;
        %if j > 5
        %    break
        %end
    end
end
```

```
.....................
```

```
acc = acc / count;
fprintf("Lasso/LDA Validation Accuracy %f\n", acc);
```

```
Lasso/LDA Validation Accuracy 0.081250
```

## Kmeans + Sparse / Kmeans&LDA + Sparse

```
% learn code book
book = {};
KA = [];
for i = 1:numcls
    samples = DataTrain{i};
    [~, centroids] = kmeans(samples, 55);
    book{i} = centroids;
    KA = [KA; centroids];
end
KA = KA';
```

```
% Kmeans + Sparse : validation
acc = 0;
count = 0;
for i = 1:numcls
    for j = 1:size(DataVal{i}, 1)
        cursor = DataVal{i};
        cursor = cursor(j, :)';
        errors = [];
        for k = 1:numcls
            Dc = KA(:, (k-1)*55+1:k*55);
            B = lasso(Dc, cursor);
            ac = B(:, 1);
            reconst = Dc * ac;
            error = norm(cursor - reconst);
            errors = [errors, error];
        end
        [~, pred] = min(errors);
        if pred == i
            acc = acc + 1;
        end
        count = count + 1;
        fprintf(".");
        %if j > 5
        %    break
        %end
    end
    fprintf("\n");
end
```

.....

```
acc = acc / count;
fprintf("KMeans/Sparse: Validation Accuracy %f\n", acc);
```

```
KMeans/Sparse: Validation Accuracy 0.740833
```

```
% Kmeans + Sparse : Test
acc = 0;
count = 0;
```

```
for i = 1:numcls
    for j = 1:size(DataTest{i}, 1)
        cursor = DataTest{i};
        cursor = cursor(j, :)';
        errors = [];
        for k = 1:numcls
            Dc = KA(:, (k-1)*55+1:k*55);
            B = lasso(Dc, cursor);
            ac = B(:, 1);
            reconst = Dc * ac;
            error = norm(cursor - reconst);
            errors = [errors, error];
        end
        [~, pred] = min(errors);
        if pred == i
            acc = acc + 1;
        end
        count = count + 1;
        fprintf(".");
        %if j > 5
        %    break
        %end
    end
    fprintf("\n");
end
```

.....

```
acc = acc / count;
fprintf("KMeans/Sparse: Test Accuracy %f\n", acc);
```

```
KMeans/Sparse: Test Accuracy 0.482083
```

```
% learn code book from LDA vectors
Lbook = {};
LKA = [];
for i = 1:numcls
    samples = LDATrain{i}';
    [~, centroids] = kmeans(samples, 55);
    book{i} = centroids;
    LKA = [LKA; centroids];
end
LKA = LKA';
```

```
% Kmeans/LDA + Sparse : Validation
acc = 0;
count = 0;
for i = 1:numcls
    for j = 1:size(LDAVal{i}, 1)
        cursor = LDAVal{i}';
        cursor = cursor(j, :);
        errors = [];
        for k = 1:numcls
```

```
            Dc = LKA(:, (k-1)*55+1:k*55);
            B = lasso(Dc, cursor);
            ac = B(:, 1);
            reconst = Dc * ac;
            error = norm(cursor - reconst);
            errors = [errors, error];
        end
        [~, pred] = min(errors);
        if pred == i
            acc = acc + 1;
        end
        count = count + 1;
        fprintf(".");
        if j > 10
            break
        end
    end
    fprintf("\n");
end
```

.....

```
acc = acc / count;
fprintf("KMeans/LDA/Sparse: Val Accuracy %f\n", acc);
```

```
KMeans/LDA/Sparse: Val Accuracy 0.125000
```

```
acc = 0;
count = 0;
for i = 1:numcls
    for j = 1:size(LDAVal{i}, 1)
        cursor = LDAVal{i}';
        cursor = cursor(j, :);
        errors = [];
        for k = 1:numcls
            Dc = LKA(:, (k-1)*55+1:k*55);
            B = lasso(Dc, cursor, 'Lambda', 0.5);
            ac = B(:, 1);
            reconst = Dc * ac;
            error = norm(cursor - reconst);
            errors = [errors, error];
        end
        [~, pred] = min(errors);
        if pred == i
            acc = acc + 1;
        end
        count = count + 1;
        fprintf(".");
    end
    fprintf("\n");
end
```

.....

```
acc = acc / count;
fprintf("KMeans/LDA/Sparse: Val Accuracy %f\n", acc);
```

```
KMeans/LDA/Sparse: Val Accuracy 0.041667
```

```
% Kmeans/LDA + Sparse : Test
acc = 0;
count = 0;
for i = 1:numcls
    for j = 1:size(LDATest{i}, 1)
        cursor = LDATest{i}';
        cursor = cursor(j, :)';
        errors = [];
        for k = 1:numcls
            Dc = LKA(:, (k-1)*55+1:k*55);
            B = lasso(Dc, cursor);
            ac = B(:, 1);
            ac = alpha((k-1)*55+1:k*55);
            reconst = Dc * ac;
            error = norm(cursor - reconst);
            errors = [errors, error];
        end
        [~, pred] = min(errors);
        if pred == i
            acc = acc + 1;
        end
        count = count + 1;
        fprintf(".");
    end
    fprintf("\n");
end
```

```
.....................................................................................................
```

```
acc = acc / count;
fprintf("KMeans/Sparse: Test Accuracy %f\n", acc);
```

```
KMeans/Sparse: Test Accuracy 0.041667
```

## Problem V: Gaussian Classifier

```
% reload original data
OA = [];
for i = 1:numcls
    filename = sprintf('../Data/Train/Class_%d.txt', i);
    data = importdata(filename);
    DataTrain{i} = data;
    OA = [OA; data];

    filename = sprintf('../Data/Dev/Class_%d.txt', i);
    data = importdata(filename);
    DataVal{i} = data;
```

```
    filename = sprintf('../Data/Eval/Class_%d.txt', i);
    data = importdata(filename);
    DataTest{i} = data;
end
```

```
% train gaussian classifier
gcov = cov(OA); % 600x600
% class-wise centroids
gc = zeros(numcls, ivecdim);
for i = 1:numcls
    imean = mean(DataTrain{i}, 1);
    gc(i, :) = imean;
end
igcov = inv(gcov);
```

```
% Gaussian validation
acc = 0;
count = 0;
for i = 1:numcls
    for j = 1:size(DataVal{i}, 1)
        cursor = DataVal{i};
        cursor = cursor(j, :)';
        likelihood = zeros(1, numcls);
        for k = 1:numcls
            % the coefficient parts are discarded. does not affect argmax
            g = exp(-0.5 * (cursor - gc(k, :)')' * igcov * (cursor - gc(k, :)'));
            likelihood(k) = g;
        end
        [~, pred] = max(likelihood);
        if pred == i
            acc = acc + 1;
        end
        count = count + 1;
        fprintf(".");
    end
    fprintf("\n");
end
```

.....

```
acc = acc / count;
fprintf('Gaussian Validation Set Accuracy %f\n', acc);
```

```
Gaussian Validation Set Accuracy 0.797500
```

```
% Gaussian test
acc = 0;
count = 0;
for i = 1:numcls
    for j = 1:size(DataTest{i}, 1)
        cursor = DataTest{i};
        cursor = cursor(j, :)';
```

```
        likelihood = zeros(1, numcls);
        for k = 1:numcls
            % the coefficient parts are discarded. does not affect argmax
            g = exp(-0.5 * (cursor - gc(k, :)')' * igcov * (cursor - gc(k, :)'));
            likelihood(k) = g;
        end
        [~, pred] = max(likelihood);
        if pred == i
            acc = acc + 1;
        end
        count = count + 1;
        fprintf(".");
    end
    fprintf("\n");
end
```

.....

```
acc = acc / count;
fprintf('Gaussian Test Set Accuracy %f\n', acc);
```

```
Gaussian Test Set Accuracy 0.617083
```