CSE426 Pattern Recognition – Final Project: Uppercase Classification

Yi Luo (yil712)

1. Summary Error Table

In this report, method 1 is *moment-space minimum-distance classifier*; method 2 is *moment-space classifier with identical covariances*; method 3 is *1NN in moment space*; method 4 is *5NN in moment space*. Their error rates on dataset A, B, C, and D are shown as Table 1.

Error Rate Table 1: Total error rates for each of the four classifiers

Test set: Method:	A	B	C	D
1	202	208	220	194
2	38	39	41	39
3	0	91	110	107
4	61	77	99	94

The best error rate E averaged over B, C, & D is achieved by method 2, which is E = (39+41+39)/3 = 39.67. However, these classifiers are all based on *moment-space* features.

We then consider to explore new features, in this experiment, we make use of 256 pixel features and figure out that *5NN in such pixel-space* can realize good performance. The error rate becomes E' = (8+22+14)/3 = 14.67 which drops the error by 63%. We display the summary on test dataset B, C, & D as follows:

Error Rate Table 2: Total error rates of 5NN in pixel-space

Test set: Method:	B	C	D
4 in pixel-space	8	22	14

In addition to try new features, we design and implement a new classifier which is the SVM in pixel-space. It achieves the best performance compared to all previous classifiers. The error rate turns to be E'' = (5+10+4)/3 = 6.33 which drops E by 84% and even cuts E' in half. The summary is as follows:

Error Rate Table 3: Total error rates of SVM in pixel-space

Test set: Method:	B	C	D
5	5	10	4

2. Confusion Tables

In this section, we show 10 confusion tables. According to section one, we have ***four*** classifiers on moment-space features, ***one*** 5NN classifier in pixel-space and ***one*** SVM classifier in pixel-space. For each of them, we select one test dataset from B, C, & D which achieves the best error rate and show its confusion table. But for *moment-space minimum-distance classifier* and *SVM in Pixel-space classifier*, we show all three confusion tables as required. Here, the mapping is 0-B, 1-C, 2-D, 4-E, 4-I, 5- J, 6-O, 7-R, 8-U, 9-V.

Confusion Table 1: Method 1 – 20 Moments, moment-space minimum-distance classifier (trained on A, tested on B)

	0	1	2	3	4	5	6	7	8	9	Error Type I
0	63	0	9	0	8	1	12	7	0	0	37
1	0	87	0	6	2	0	0	0	5	0	13
2	14	0	81	0	2	0	3	0	0	0	19
3	0	0	0	76	5	5	1	12	1	0	24
4	0	0	0	0	100	0	0	0	0	0	0
5	0	0	0	0	10	90	0	0	0	0	10
6	3	0	0	0	20	4	69	4	0	0	31
7	4	0	0	2	9	0	15	70	0	0	30
8	0	11	0	0	5	2	0	0	67	15	33
9	0	0	0	0	11	0	0	0	0	89	11
Error Type II	21	11	9	8	72	12	31	23	6	15	208

Confusion Table 2: Method 1 – 20 Moments, moment-space minimum-distance classifier (trained on A, tested on C)

	0	1	2	3	4	5	6	7	8	9	Error Type I
0	70	0	8	0	4	0	7	11	0	0	30
1	0	85	0	10	1	0	0	0	4	0	15
2	20	0	71	0	1	3	4	1	0	0	29
3	0	0	0	67	14	9	1	7	2	0	33
4	0	0	0	0	98	2	0	0	0	0	2
5	0	0	0	0	12	88	0	0	0	0	12
6	1	0	0	0	17	2	75	5	0	0	25
7	7	0	1	1	8	2	6	75	0	0	25
8	0	5	0	1	3	0	0	0	63	28	37
9	1	0	0	0	8	0	0	0	3	88	12
Error Type II	29	5	9	12	68	18	18	24	9	28	220

Confusion Table 3: Method 1 – 20 Moments, moment-space minimum-distance classifier (trained on A, tested on D)

	0	1	2	3	4	5	6	7	8	9	Error Type I
0	71	0	5	0	5	1	8	10	0	0	29
1	0	79	0	6	1	0	0	0	14	0	21
2	18	0	75	0	1	4	2	0	0	0	25
3	0	0	0	72	13	3	1	11	0	0	28
4	0	0	0	0	99	1	0	0	0	0	1
5	0	0	0	0	6	94	0	0	0	0	6
6	1	0	0	1	13	2	79	4	0	0	21
7	7	0	0	2	3	2	9	77	0	0	23
8	0	3	0	1	9	2	0	0	74	11	26
9	0	0	0	0	9	0	0	0	5	86	14
Error Type II	26	3	5	10	60	15	20	25	19	11	194

Confusion Table 4: Method 2 – 20 Moments, moment-space classifier with identical covariances (trained on A, tested on B)

	0	1	2	3	4	5	6	7	8	9	Error Type I
0	93	0	3	0	1	2	0	1	0	0	7
1	0	99	0	0	1	0	0	0	0	0	1
2	7	0	87	1	0	0	5	0	0	0	13
3	0	0	0	97	2	0	0	0	1	0	3
4	0	0	0	0	100	0	0	0	0	0	0
5	0	0	0	0	5	95	0	0	0	0	5
6	0	0	1	0	0	0	99	0	0	0	1
7	2	0	0	0	0	0	2	96	0	0	4
8	0	1	0	0	2	0	0	0	97	0	3
9	0	0	0	0	2	0	0	0	0	98	2
Error Type II	9	1	4	1	13	2	7	1	1	0	39

Confusion Table 5: Method 3 – 20 Moments, 1NN in moment space (trained on A, tested on B)

	0	1	2	3	4	5	6	7	8	9	Error Type I
0	77	0	8	0	0	1	6	8	0	0	23
1	0	99	0	0	1	0	0	0	0	0	1
2	14	0	81	0	0	0	4	1	0	0	19
3	1	1	0	89	2	1	1	4	1	0	11
4	0	0	0	0	94	6	0	0	0	0	6
5	0	0	0	0	4	96	0	0	0	0	4
6	4	0	7	1	0	0	86	2	0	0	14
7	2	0	0	3	0	0	2	93	0	0	7
8	0	2	0	1	0	0	1	0	95	1	5
9	0	0	0	0	1	0	0	0	0	99	1
Error Type II	21	3	15	5	8	8	14	15	1	1	91

Confusion Table 6: Method 4 – 20 Moments, 5NN in moment space (trained on A, tested on B)

	0	1	2	3	4	5	6	7	8	9	Error Type I
0	81	0	9	0	0	1	1	8	0	0	19
1	0	99	0	0	1	0	0	0	0	0	1
2	16	0	81	0	0	0	3	0	0	0	19
3	1	0	0	91	2	1	0	4	1	0	9
4	0	0	0	0	97	3	0	0	0	0	3
5	0	0	0	0	1	99	0	0	0	0	1
6	2	0	2	0	0	1	93	2	0	0	7
7	2	0	0	2	0	0	4	92	0	0	8
8	0	3	0	0	0	0	3	0	92	2	8
9	0	0	0	0	2	0	0	0	0	98	2
Error Type II	21	3	11	2	6	6	11	14	1	2	77

Confusion Table 7: Method 4 – 256 Pixels, 5NN in Pixel-space (trained on A, tested on B)

	0	1	2	3	4	5	6	7	8	9	Error Type I
0	100	0	0	0	0	0	0	0	0	0	0
1	0	99	0	0	1	0	0	0	0	0	1
2	0	0	98	0	0	0	2	0	0	0	2
3	0	0	0	99	1	0	0	0	0	0	1
4	0	0	0	0	100	0	0	0	0	0	0
5	0	0	0	0	1	99	0	0	0	0	1
6	0	0	0	0	0	0	100	0	0	0	0
7	0	0	0	0	0	0	0	100	0	0	0
8	0	1	0	1	0	0	0	0	98	0	2
9	0	0	0	0	1	0	0	0	0	99	1
Error Type II	0	1	0	1	4	0	2	0	0	0	8

Confusion Table 8: Method 5 – 256 Pixels, SVM in Pixel-space (trained on A, tested on B)

	0	1	2	3	4	5	6	7	8	9	Error Type I
0	99	0	0	0	0	0	0	1	0	0	1
1	0	99	0	0	1	0	0	0	0	0	1
2	0	0	99	0	0	0	1	0	0	0	1
3	0	0	0	100	0	0	0	0	0	0	0
4	0	0	0	0	99	1	0	0	0	0	1
5	0	0	0	0	1	99	0	0	0	0	1
6	0	0	0	0	0	0	100	0	0	0	0
7	0	0	0	0	0	0	0	100	0	0	0
8	0	0	0	0	0	0	0	0	100	0	0
9	0	0	0	0	0	0	0	0	0	100	0
Error Type II	0	0	0	0	2	1	1	1	0	0	5

Confusion Table 9: Method 5 – 256 Pixels, SVM in Pixel-space (trained on A, tested on C)

	0	1	2	3	4	5	6	7	8	9	Error Type I
0	99	0	0	0	0	0	0	1	0	0	1
1	0	100	0	0	0	0	0	0	0	0	0
2	0	0	100	0	0	0	0	0	0	0	0
3	0	0	0	97	1	0	0	0	2	0	3
4	0	0	0	0	98	2	0	0	0	0	2
5	0	0	0	0	2	98	0	0	0	0	2
6	0	0	1	0	0	0	99	0	0	0	1
7	0	0	0	0	0	0	0	100	0	0	0
8	0	0	0	0	1	0	0	0	99	0	1
9	0	0	0	0	0	0	0	0	0	100	0
Error Type II	0	0	1	0	4	2	0	1	2	0	10

Confusion Table 10: Method 5 – 256 Pixels, SVM in Pixel-space (trained on A, tested on D)

	0	1	2	3	4	5	6	7	8	9	Error Type I
0	100	0	0	0	0	0	0	0	0	0	0
1	0	100	0	0	0	0	0	0	0	0	0
2	0	0	100	0	0	0	0	0	0	0	0
3	0	0	0	97	1	0	0	0	2	0	3
4	0	0	0	0	99	1	0	0	0	0	1
5	0	0	0	0	0	100	0	0	0	0	0
6	0	0	0	0	0	0	100	0	0	0	0
7	0	0	0	0	0	0	0	100	0	0	0
8	0	0	0	0	0	0	0	0	100	0	0
9	0	0	0	0	0	0	0	0	0	100	0
Error Type II	0	0	0	0	1	1	0	0	2	0	4

3. Comments and Discussions

We can see that method 1 has the poorest performance. It is because we give some improper assumptions such as the identity covariance matrix. Also, the feature has 20 dimensions and may not contain much enough useful information.

Then, by giving a more reasonable assumption – the shared covariance matrix is not an identity one, instead, it's the average covariance matrix –we get better results in method 2.

In method 3 we implement the 1NN classifier on 20-moment space and in method 4 we try the 5NN classifier. We can see that 5NN achieves better performance than 1NN. The thing is that in *k-NN* method, a small value of ***k*** makes the classifier too sensitive to neighbor data samples. Namely, noises can influence a lot if ***k*** is too small. On the contrary, a larger value of ***k*** alleviates this issue since the classification result is made based on more votes from ***k*** samples, making the classifier more robust to noises.

Then, we make use of 256 pixel features. Compared to method 4 which is also 5NN but in moment-space, we can see that 5NN in pixel-space achieves smaller error rate than 5NN in moment-space. It seems that a larger number of features are more likely to make better performance. But the problem is that more features require more multiplications and the computation would also consume more time.

Last, in method 5, we design and implement a new classifier which is SVM in pixel space. We use the popular ***libsvm*** package (http://www.csie.ntu.edu.tw/~cjlin/libsvm/) for this experiment. We select the simplest linear kernel function to train the model and realize good improvements of the error rate on all three test datasets B, C, & D. Compared to the original best error rate (averaged) E = 39.67, we get a much better error rate (averaged) E'' = 6.33 which drops the error by 84%.

4. Source code

```
function [confusion1] = Proj_classifier1(FeatureA, FeatureB)

% to train the mu
u(1:10, 1:20) = 0; %10 classes * 20 features
for i = 1:10 % class
    for j = 1:100 % image
        for k = 1:20 % feature
            u(i, k) = u(i, k) +  FeatureA(i, j, k);
        end
    end
    for k = 1:20 % feature
        u(i, k) = u(i, k) / 100;
    end
end

% to test
confusion1(1:10, 1:10) = 0;
I = eye(20);
for real = 1:10 % for each class
    for j = 1:100 % to classify each image
        min = 32767;
        classified = real;
        for c = 1:10
            dist = 0;
            x(1:20) = 0;
            mu(1:20) = 0;
            for k = 1:20 % feature
                x(k) = FeatureB(real, j, k);
                mu(k) = u(c, k);
            end
            dist = (x - mu)*(I^-1)*(x - mu)';
            if (dist < min)
                min = dist;
                classified = c;
            end
        end
        confusion1(real, classified) = confusion1(real, classified) + 1;
    end
end

end
```

```
function [confusion2] = Proj_classifier2(FeatureA, FeatureB)

% to train the mu
u(1:10, 1:20) = 0; %10 classes * 20 features
for i = 1:10 % class
    for j = 1:100 % image
        for k = 1:20 % feature
            u(i, k) = u(i, k) +  FeatureA(i, j, k);
        end
    end
    for k = 1:20 % feature
        u(i, k) = u(i, k) / 100;
    end
end
% to train the covariance
avgCov(1:20, 1:20) = 0;
for i = 1:10 % class
    A(1:100, 1:20) = 0;
```

```
    for j = 1:100 % image
        for k = 1:20 % feature
            A(j, k) = FeatureA(i, j, k);
        end
    end
    covar = cov(A);
    avgCov = avgCov + covar;
end
avgCov = avgCov / 10;

% to test
confusion2(1:10, 1:10) = 0;

for real = 1:10 % for each class
    for j = 1:100 % to classify each image
        min = 32767;
        classified = real;
        for c = 1:10
            dist = 0;
            x(1:20) = 0;
            mu(1:20) = 0;
            for k = 1:20 % feature
                x(k) = FeatureB(real, j, k);
                mu(k) = u(c, k);
            end
            dist = (x - mu)*(avgCov^-1)*(x - mu)';
            if (dist < min)
                min = dist;
                classified = c;
            end
        end
        confusion2(real, classified) = confusion2(real, classified) + 1;
    end
end

end
```

```
function [confusion3] = Proj_classifier3(FeatureA, FeatureB)

confusion3(1:10, 1:10) = 0;

for real = 1:10 % for each class
    for i = 1:100 % to classify each image
        for k = 1:20 % feature
            crt(k) = FeatureB(real, i, k);
        end
        dist = 0;
        min = 32767;
        classified = real;
        for c = 1:10
            for j = 1:100
                for k = 1:20 % feature
                    target(k) = FeatureA(c, j, k);
                end
                dist = getMinkowski(crt, target, 20, 2);
                if (dist <= min)
                    min = dist;
                    classified = c;
                end
            end
        end
        confusion3(real, classified) = confusion3(real, classified) + 1;
    end
end
```

```
end
function [confusion4] = Proj_classifier4(FeatureA, FeatureB)

confusion4(1:10, 1:10) = 0;

for real = 1:10 % for each class
    for i = 1:100 % to classify each image
        classCount(1:10) = 0;

        for k = 1:20 % feature
            crt(k) = FeatureB(real, i, k);
        end
        dist = 0;
        min1 = 32767;
        min2 = 32767;
        min3 = 32767;
        min4 = 32767;
        min5 = 32767;
        c1 = real;
        c2 = real;
        c3 = real;
        c4 = real;
        c5 = real;
        for c = 1:10
            for j = 1:100
                for k = 1:20 % feature
                    target(k) = FeatureA(c, j, k);
                end
                classified = c;
                dist = getMinkowski(crt, target, 20, 2);
                if (dist < min1)
                    temp = min1;
                    min1 = dist;
                    dist = temp; % swap and pass on
                    tempc = c1;
                    c1 = classified;
                    classified = tempc; % swap and pass on
                end
                if (dist < min2)
                    temp = min2;
                    min2 = dist;
                    dist = temp;
                    tempc = c2;
                    c2 = classified;
                    classified = tempc;
                end
                if (dist < min3)
                    temp = min3;
                    min3 = dist;
                    dist = temp;
                    tempc = c3;
                    c3 = classified;
                    classified = tempc;
                end
                if (dist < min4)
                    temp = min4;
                    min4 = dist;
                    dist = temp;
                    tempc = c4;
                    c4 = classified;
                    classified = tempc;
                end
```

```
                if (dist < min5)
                    temp = min5;
                    min5 = dist;
                    dist = temp;
                    tempc = c5;
                    c5 = classified;
                    classified = tempc;
                end
            end
        end
        classCount(c1) = classCount(c1) + 1;
        classCount(c2) = classCount(c2) + 1;
        classCount(c3) = classCount(c3) + 1;
        classCount(c4) = classCount(c4) + 1;
        classCount(c5) = classCount(c5) + 1;

        classified = real;
        maxCount = 0;
        for d = 1:10
            if (classCount(d) > maxCount)
                maxCount = classCount(d);
                classified = d;
            end
        end

        confusion4(real, classified) = confusion4(real, classified) + 1;
    end
end

end
```

```
function [confusion4] = Proj_classifierP4(FeatureA, FeatureB)

confusion4(1:10, 1:10) = 0;
for real = 1:10 % for each class
    for i = 1:100 % to classify each image
        classCount(1:10) = 0;

        for k = 1:256 % feature
            crt(k) = FeatureB(real, i, k);
        end
        dist = 0;
        min1 = 32767;
        min2 = 32767;
        min3 = 32767;
        min4 = 32767;
        min5 = 32767;
        c1 = real;
        c2 = real;
        c3 = real;
        c4 = real;
        c5 = real;
        for c = 1:10
            for j = 1:100
                for k = 1:256 % feature
                    target(k) = FeatureA(c, j, k);
                end
                classified = c;
                dist = getMinkowski(crt, target, 256, 2);
                if (dist < min1)
                    temp = min1;
                    min1 = dist;
                    dist = temp; % swap and pass on
                    tempc = c1;
```

```
                    c1 = classified;
                    classified = tempc; % swap and pass on
                end
                if (dist < min2)
                    temp = min2;
                    min2 = dist;
                    dist = temp;
                    tempc = c2;
                    c2 = classified;
                    classified = tempc;
                end
                if (dist < min3)
                    temp = min3;
                    min3 = dist;
                    dist = temp;
                    tempc = c3;
                    c3 = classified;
                    classified = tempc;
                end
                if (dist < min4)
                    temp = min4;
                    min4 = dist;
                    dist = temp;
                    tempc = c4;
                    c4 = classified;
                    classified = tempc;
                end
                if (dist < min5)
                    temp = min5;
                    min5 = dist;
                    dist = temp;
                    tempc = c5;
                    c5 = classified;
                    classified = tempc;
                end
            end
        end
        classCount(c1) = classCount(c1) + 1;
        classCount(c2) = classCount(c2) + 1;
        classCount(c3) = classCount(c3) + 1;
        classCount(c4) = classCount(c4) + 1;
        classCount(c5) = classCount(c5) + 1;

        classified = real;
        maxCount = 0;
        for d = 1:10
            if (classCount(d) > maxCount)
                maxCount = classCount(d);
                classified = d;
            end
        end

        confusion4(real, classified) = confusion4(real, classified) + 1;
    end
end

end
```

```
function [confusion5] = Proj_classifierP5(FeatureA, FeatureB)

confusion5(1:10, 1:10) = 0;

train_data = ones(1000, 256);
row = 0;
```

```
for c = 1:10 % for each class
    for i = 1:100 % to classify each image
        row = row + 1;
        for k = 1:256 % feature
            train_data(row, k) = FeatureA(c, i, k);
        end
    end
end
train_label = [ones(100, 1); ones(100, 1)*2; ones(100, 1)*3; ...
            ones(100, 1)*4; ones(100, 1)*5; ones(100, 1)*6; ...
            ones(100, 1)*7; ones(100, 1)*8; ones(100, 1)*9; ones(100,
1)*10];
model = svmtrain(train_label, train_data, '-s 0 -c 1 -t 0');

test_data = ones(1000, 256);
row = 0;
for c = 1:10 % for each class
    for i = 1:100 % to classify each image
        row = row + 1;
        for k = 1:256 % feature
            test_data(row, k) = FeatureB(c, i, k);
        end
    end
end
test_label = [ones(100, 1); ones(100, 1)*2; ones(100, 1)*3; ...
            ones(100, 1)*4; ones(100, 1)*5; ones(100, 1)*6; ...
            ones(100, 1)*7; ones(100, 1)*8; ones(100, 1)*9; ones(100,
1)*10];
[predict_label, accuracy, dec_values] = svmpredict(test_label, test_data,
model);

index = 0;
for real = 1:10
    for i = 1:100 % images
        index = index + 1;
        classified = predict_label(index);
        confusion5(real, classified) = confusion5(real, classified) + 1;
    end
end
```

```
function [L, h, w, max, may, F] = readLetterImages(filename)

fid1 = fopen(filename);

size = 100;
numOfL = 1;
L(1:size,:,:) = 0; % all letter images
while ~feof(fid1)
    line = fgetl(fid1);
    if (line(1) == 'C')
        para = regexp(line,'\s','split');
        height = para{2};
        weight = para{3};
        h(numOfL) = str2num(height(2:end)); % to get the height of the
matrix
        w(numOfL) = str2num(weight(2:end)); % to get the weight of the
matrix
    end

    for i = 1:h(numOfL)
        line = fgetl(fid1);
        for j = 1:w(numOfL)
            if (line(j) == '.')
                L(numOfL, i, j) = 0;
```

```
            else
                L(numOfL, i, j) = 1;
            end
        end
    end

    numOfL = numOfL + 1;
end
fclose(fid1);

imgSize = 16;
F(1:size, 1:20) = 0;
for i = 1:size
    I(imgSize, imgSize) = 0;
    for x = 1:imgSize % w
        for y = 1:imgSize % h
            I(x, y) = L(i, y, x);
        end
    end
    max(i) = getMA(1, 0, I, imgSize, imgSize) / getMA(0, 0, I, imgSize,
imgSize);
    may(i) = getMA(0, 1, I, imgSize, imgSize) / getMA(0, 0, I, imgSize,
imgSize);
    F(i, 1) = getCM(2, 1, max(i), may(i), I, imgSize, imgSize);
    F(i, 2) = getCM(1, 2, max(i), may(i), I, imgSize, imgSize);
    F(i, 3) = getCM(3, 1, max(i), may(i), I, imgSize, imgSize);
    F(i, 4) = getCM(2, 2, max(i), may(i), I, imgSize, imgSize);
    F(i, 5) = getCM(1, 3, max(i), may(i), I, imgSize, imgSize);
    F(i, 6) = getCM(4, 1, max(i), may(i), I, imgSize, imgSize);
    F(i, 7) = getCM(3, 2, max(i), may(i), I, imgSize, imgSize);
    F(i, 8) = getCM(2, 3, max(i), may(i), I, imgSize, imgSize);
    F(i, 9) = getCM(1, 4, max(i), may(i), I, imgSize, imgSize);
    F(i, 10) = getCM(5, 1, max(i), may(i), I, imgSize, imgSize);
    F(i, 11) = getCM(4, 2, max(i), may(i), I, imgSize, imgSize);
    F(i, 12) = getCM(3, 3, max(i), may(i), I, imgSize, imgSize);
    F(i, 13) = getCM(2, 4, max(i), may(i), I, imgSize, imgSize);
    F(i, 14) = getCM(1, 5, max(i), may(i), I, imgSize, imgSize);
    F(i, 15) = getCM(6, 1, max(i), may(i), I, imgSize, imgSize);
    F(i, 16) = getCM(5, 2, max(i), may(i), I, imgSize, imgSize);
    F(i, 17) = getCM(4, 3, max(i), may(i), I, imgSize, imgSize);
    F(i, 18) = getCM(3, 4, max(i), may(i), I, imgSize, imgSize);
    F(i, 19) = getCM(2, 5, max(i), may(i), I, imgSize, imgSize);
    F(i, 20) = getCM(1, 6, max(i), may(i), I, imgSize, imgSize);
end

end
```

```
function [L, h, w, F] = readLetterImagesPixel(filename)

fid1 = fopen(filename);

size = 100;
numOfL = 1;
L(1:size,:,:) = 0; % all letter images
while ~feof(fid1)
    line = fgetl(fid1);
    if (line(1) == 'C')
        para = regexp(line,'\s','split');
        height = para{2};
        weight = para{3};
        h(numOfL) = str2num(height(2:end)); % to get the height of the
matrix
        w(numOfL) = str2num(weight(2:end)); % to get the weight of the
matrix
```

```
    end

    for i = 1:h(numOfL)
        line = fgetl(fid1);
        for j = 1:w(numOfL)
            if (line(j) == '.')
                L(numOfL, i, j) = 0;
            else
                L(numOfL, i, j) = 1;
            end
        end
    end

    numOfL = numOfL + 1;
end
fclose(fid1);

imgSize = 16;
F(1:size, 1:256) = 0;
for i = 1:size
    count = 0;
    for x = 1:imgSize % w
        for y = 1:imgSize % h
            count = count + 1;
            F(i, count) = L(i, y, x);
        end
    end
end

end
```

```
function [] = preprocessImg(input, output)

fid1 = fopen(input);

size = 100;
numOfL = 1;
L(1:size,:,:) = 0; % all letter images
while ~feof(fid1)
    line = fgetl(fid1);
    if (line(1) == 'C')
        para = regexp(line,'\s','split');
        height = para{2};
        weight = para{3};
        index = para{4};
        h(numOfL) = str2num(height(2:end)); % to get the height of the
matrix
        w(numOfL) = str2num(weight(2:end)); % to get the weight of the
matrix
        b(numOfL) = str2num(index(2:end));
    end

    for i = 1:h(numOfL)
        line = fgetl(fid1);
        for j = 1:w(numOfL)
            if (line(j) == '.')
                L(numOfL, i, j) = 0;
            else
                L(numOfL, i, j) = 1;
            end
        end
    end

    numOfL = numOfL + 1;
```

```
end
fclose(fid1);

fid2 = fopen(output, 'W');
dataSize = 16;
for i = 1:size
    fprintf(fid2, 'C h16 w16 b%d\n', b(i));
    up = fix((dataSize - h(i)) / 2);
    down = dataSize - h(i) - up;
    left = fix((dataSize - w(i)) / 2);

    for m = 1:up
        for n = 1:dataSize
            fprintf(fid2, '.');
        end
        fprintf(fid2, '\n');
    end

    for m = 1:h(i)
        for n = 1:dataSize
            if (n <= left || n > left + w(i))
                fprintf(fid2, '.');
            else
                if (L(i, m, n - left) == 0)
                    fprintf(fid2, '.');
                else
                    fprintf(fid2, 'x');
                end
            end
        end
        fprintf(fid2, '\n');
    end

    for m = 1:down
        for n = 1:dataSize
            fprintf(fid2, '.');
        end
        fprintf(fid2, '\n');
    end
end

fclose(fid2);
end
```

```
function [FeatureA, FeatureB, FeatureC, FeatureD] =
getNormalizedCMFeature()

FeatureA = zeros(10, 100, 20);
for i = 1:10
    input = strcat('data/A-', num2str(i), '-pre.txt');
    [L, h, w, max, may, F] = readLetterImages(input);
    FeatureA(i, :, :) = F;
end


for k = 1:20
    sum = 0;
    for i = 1:10
        for j = 1:100
            sum = sum + FeatureA(i, j, k)^2;
        end
    end
    rms = sqrt(sum/(10*100));
    FeatureA(:,:,k) = FeatureA(:,:,k) / rms;
```

```
end

FeatureB = zeros(10, 100, 20);
for i = 1:10
    input = strcat('data/B-', num2str(i), '-pre.txt');
    [L, h, w, max, may, F] = readLetterImages(input);
    FeatureB(i, :, :) = F;
end

for k = 1:20
    sum = 0;
    for i = 1:10
        for j = 1:100
            sum = sum + FeatureB(i, j, k)^2;
        end
    end
    rms = sqrt(sum/(10*100));
    FeatureB(:,:,k) = FeatureB(:,:,k) / rms;
end

FeatureC = zeros(10, 100, 20);
for i = 1:10
    input = strcat('data/C-', num2str(i), '-pre.txt');
    [L, h, w, max, may, F] = readLetterImages(input);
    FeatureC(i, :, :) = F;
end

for k = 1:20
    sum = 0;
    for i = 1:10
        for j = 1:100
            sum = sum + FeatureC(i, j, k)^2;
        end
    end
    rms = sqrt(sum/(10*100));
    FeatureC(:,:,k) = FeatureC(:,:,k) / rms;
end

FeatureD = zeros(10, 100, 20);
for i = 1:10
    input = strcat('data/D-', num2str(i), '-pre.txt');
    [L, h, w, max, may, F] = readLetterImages(input);
    FeatureD(i, :, :) = F;
end

for k = 1:20
    sum = 0;
    for i = 1:10
        for j = 1:100
            sum = sum + FeatureD(i, j, k)^2;
        end
    end
    rms = sqrt(sum/(10*100));
    FeatureD(:,:,k) = FeatureD(:,:,k) / rms;
end
end
```

```
function [cm] = getCM5(p, q, max, may, I, w, h)

cm = 0;

for x = 1:w
    for y = 1:h
       cm = cm + (x - max)^p * (y - may)^q * I(x, y);
```

```
    end
end

end
```

```
function [ma] = getMA(p, q, I, w, h)

ma = 0;

for x = 1:w
    for y = 1:h
       ma = ma + x^p * y^q * I(x, y);
    end
end

end
```

```
function [FeaturePA, FeaturePB, FeaturePC, FeaturePD] = getPixelFeature()

FeaturePA = zeros(10, 100, 256);
for i = 1:10
    input = strcat('data/A-', num2str(i), '-pre.txt');
    [L, h, w, F] = readLetterImagesPixel(input);
    FeaturePA(i, :, :) = F;
end

FeaturePB = zeros(10, 100, 256);
for i = 1:10
    input = strcat('data/B-', num2str(i), '-pre.txt');
    [L, h, w, F] = readLetterImagesPixel(input);
    FeaturePB(i, :, :) = F;
end

FeaturePC = zeros(10, 100, 256);
for i = 1:10
    input = strcat('data/C-', num2str(i), '-pre.txt');
    [L, h, w, F] = readLetterImagesPixel(input);
    FeaturePC(i, :, :) = F;
end

FeaturePD = zeros(10, 100, 256);
for i = 1:10
    input = strcat('data/D-', num2str(i), '-pre.txt');
    [L, h, w, F] = readLetterImagesPixel(input);
    FeaturePD(i, :, :) = F;
end

end
```

```
function [distance] = getMinkowski(a, b, d, k)

distance = 0;

for i = 1:d
    distance = distance + (abs(a(i) - b(i)))^k;
end

distance = distance ^ (1/k);

end
```