Lab #4
Manwen Li

Theory

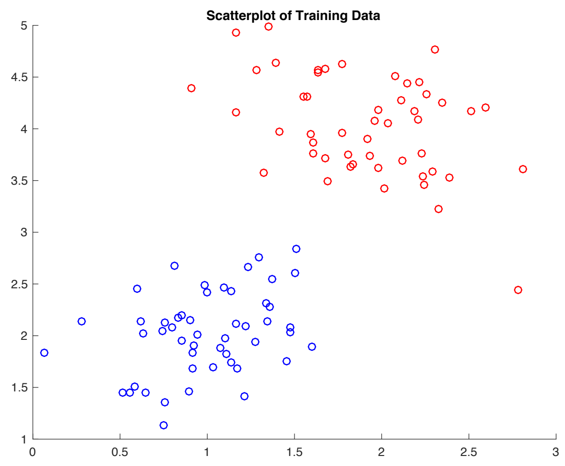
- The hyperplane is defined by
  $y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0$, where $\mathbf{x} \in \mathbb{R}^{\mathrm{D}}$**,** $\mathbf{w}$ the weight vector, w_0 the bias coefficient.
- By solving the optimization problem
  $\underset{\tilde{\mathbf{W}}}{\arg\min}\ \frac{1}{2}\,||\tilde{\mathbf{X}}\tilde{\mathbf{W}} - \mathbf{T}||_F^2$, we get $\widetilde{\mathbf{W}} = (\widetilde{\mathbf{X}}^{\mathrm{T}}\widetilde{\mathbf{X}})^{-1}\widetilde{\mathbf{X}}^{\mathrm{T}}\mathbf{T}$.
- What is the condition for a data point to be on the decision boundary between two classes k and j? What is the equation of that decision boundary?
  If a data point is on the decision boundary between class k and j, it satisfies the condition that $y_k(\mathbf{x}) = y_j(\mathbf{x})$, i.e. $\mathbf{w}_k{}^{\mathrm{T}}\mathbf{x} + w_{k0} = \mathbf{w}_j{}^{\mathrm{T}}\mathbf{x} + w_{j0}$.
  Also, we assign point $\mathbf{x}$ to class k if $y_k(\mathbf{x}) > y_j(\mathbf{x})$, for all $j \neq k$.
  The equation of the decision boundary is $y_k(\mathbf{x}) = y_j(\mathbf{x})$, and the corresponding hyperplane is
  $(\mathbf{w}_k - \mathbf{w}_j)^{\mathrm{T}}\mathbf{x} + (w_{k0} - w_{j0}) = 0.$

Experiment

1. 100 points are generated using the following code:

```
%1. Generate
mu_1 = [1;2];
mu_2 = [2;4];
sigma_1 = [[0.1,0.05];[0.05,0.2]];
sigma_2 = [[0.2,-0.1];[-0.1,0.3]];

rng(940730)
train_1 = mvnrnd(mu_1,sigma_1,50);
train_2 = mvnrnd(mu_2,sigma_2,50);
```

2.



Blue points are data with parameters $\mu_1, \Sigma_1,$ while red points are data wat parameters μ_2, Σ_2.

Matlab code:

```
%Scatterplot
figure
hold on
scatter(train_1(:,1),train_1(:,2),'b')
scatter(train_2(:,1),train_2(:,2),'r')
title('Scatterplot of Training Data')
hold off
```

3. Let X be the concatenation of data points train_1, train_2, then $\tilde{X}$ can be generated by:

```
X = [train_1;train_2];
X_tilde = [ones(100,1),X];
```

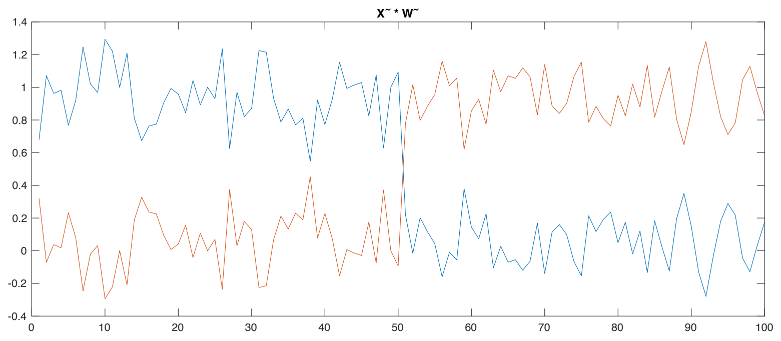
In this two-class case, matrix T should have two columns, generated by the following code:

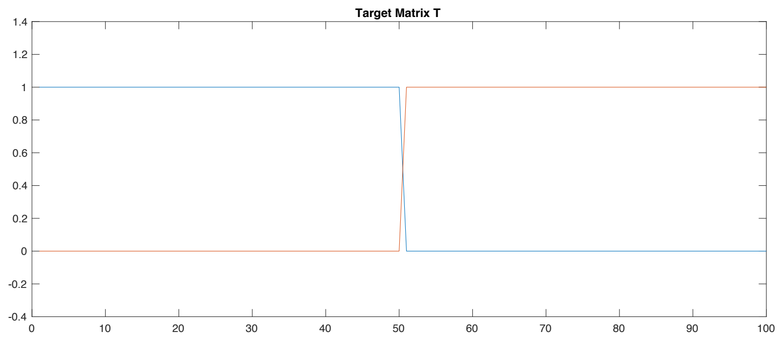
```
T_1 = [ones(50,1),zeros(50,1)];
T_2 = [zeros(50,1),ones(50,1)];
T = [T_1;T_2];
```

Then, $\widetilde{W} = (\tilde{X}^T\tilde{X})^{-1}\tilde{X}^T T$, and $\tilde{X}\widetilde{W}$ are computed by

```
W_tilde = (inv(X_tilde.' * X_tilde))*X_tilde.'*T;
X_W_tilde = X_tilde * W_tilde;
```

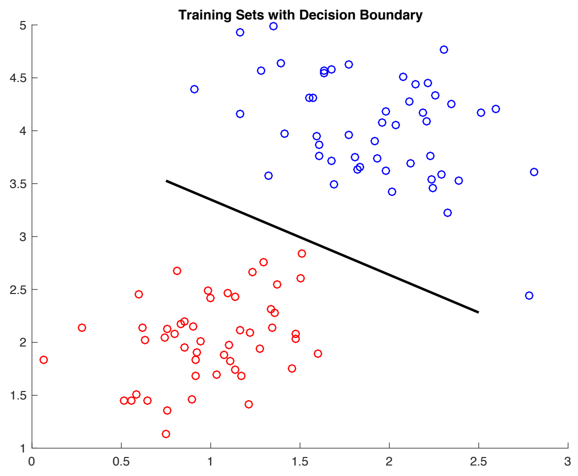
Plot of $\tilde{X}\widetilde{W}$:





The blue lines oscillate around 1 for the first 50 observations, while the red lines oscillate around 0. The red lines oscillate around 1 for the last 50 observations, while the blue lines oscillate around 0. This observation corresponds to the entries of T: the blue line denotes the first column of T and the red line denotes the second column of T. The first column of the first 50 rows are all 1, while the second column of the last 50 rows are all 1.

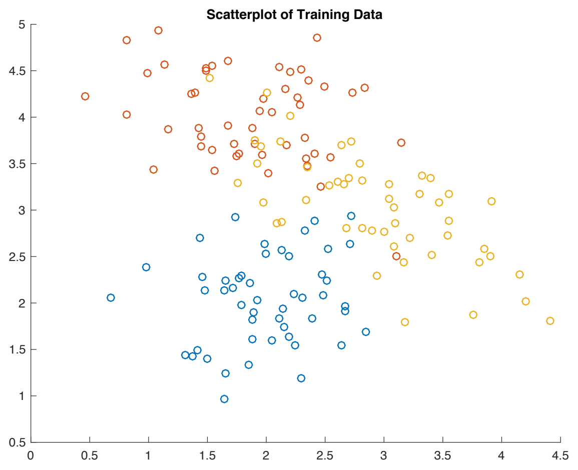
4. Decision boundary is in the form of a line that effectively separates the two training sets.



Matlab code:

```
%compute
W = [W_tilda(2,:); W_tilda(3,:)];
W0 = [W_tilda(1,:)];
x2_1 = (W0(1,2)-W0(1,1)-(W(1,1)-W(1,2))*0.75)/(W(2,1)-W(2,2));
x2_2 = (W0(1,2)-W0(1,1)-(W(1,1)-W(1,2))*2.5)/(W(2,1)-W(2,2));
%plot
figure
hold on
scatter(train_1(:,1),train_1(:,2),'r')
scatter(train_2(:,1),train_2(:,2),'b')
plot([0.75,2.5],[x2_1,x2_2],'black','LineWidth',2)
title('Training Sets with Decision Boundary')
hold off
```

5. We generate 3 groups of 50 points, which look like the following:



```
%Generate 3 groups of M = 50 points
%Code similar to #1

%Scatterplot
%code similar to #2
```

6. By calculating $\tilde{X}\widetilde{W}$ for the three classes, we get decision boundaries as the following, which classify the

points into three groups. Note that the three lines intersect at x ~= 2.3.

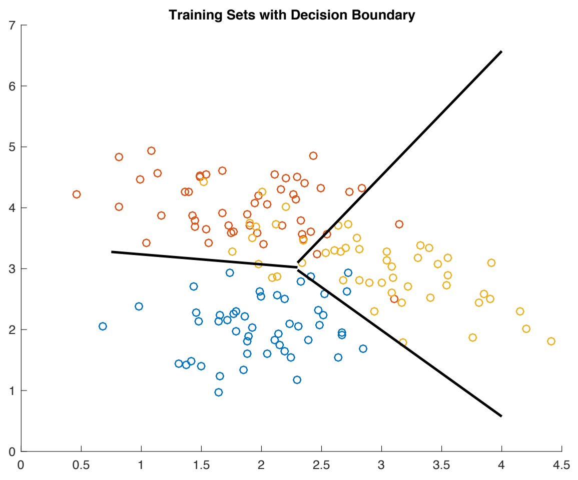


figure 0

Matlab code:

```
%Compute W
tic
X2 = [train_21;train_22;train_23];
X_tilde2 = [ones(150,1),X2];
%code similar to #3
…
toc

%plot decision boundary
%group 1 and 2
W2 = [W_tilde2(2,:); W_tilde2(3,:)];
W02 = [W_tilde2(1,:)];
p_1 = (W02(1,2)-W02(1,1)-(W2(1,1)-W2(1,2))*0.75)/(W2(2,1)-W2(2,2));
p_2 = (W02(1,2)-W02(1,1)-(W2(1,1)-W2(1,2))*2.3)/(W2(2,1)-W2(2,2));
… …
%plot
… …
```

7. After generating three groups of M = 100 test points and computing the predicted classification labels, we get accuracy of 84.33, using the formula c = 100 * (R/3M), where R is the total number of correctly corrective points.

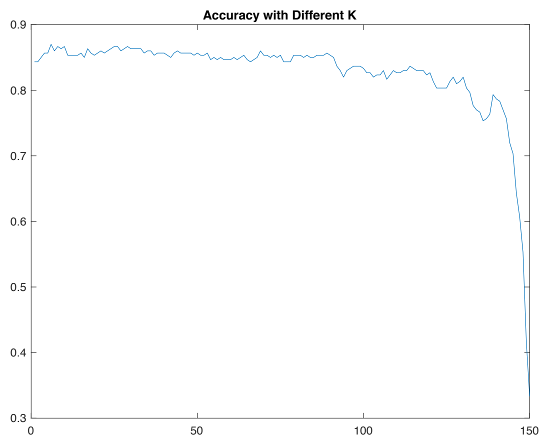
```
%first calculate X̃W̃, code similar to part 3.
%calculate R, group 1.
%code for group 2 and three is similar.
R = 0;
for i = 1:100
 [M,I]=max(X_W_tilde3(i,:));
    if I == 1
        R = R+1;
  end
end
```

c is not 100 because the fact that the three groups of training data are not completely separable, which means that we cannot find lines that perfectly classify the points into three classes which perfectly coincide with their original labels. From the graph in #6, we see that there are red and blue points in class “yellow”, and yellow points in class “blue” and “red”. Since there are overlapping regions between group “red”, group “yellow”, and group “blue”, we cannot separate the data points with lines. Thus, when performing this method into test data, the accuracy cannot be 100.

8. Knn function is the following code:

```
function [Class] = Knn(Test, Train, Label, K)

Test = Test.';
Train = Train.';
[D, N] = size(Test);
[D, M] = size(Train);

% distance between testing and training data
for i = 1:N
    for j = 1:M
     dist(i,j) = norm(Test(:,i)-Train(:,j));
    end
end
%d is N*M

%case 1: K = 1, find the closest point and take its class label
if K == 1
    [minimum, IndexTrain] = min(dist,[],2);
    Class = Label(IndexTrain);
%calse 2: K>1, sort distance and find the nearest k neighbors
else
    %sort distance
    [sorted, IndexTrain] = sort(dist.');
    %find location
    IndexTrain = IndexTrain(1:K,:);
    KnnClass = Label(IndexTrain);
    %get how many class labels
    uniqueLabel = unique(Label);
    class_number = length(uniqueLabel);
    %count how many points are have the same label, for each class label
    for i = 1:class_number
        count_class(i,:) = sum(KnnClass == uniqueLabel(i));
    end
    %find the label with the most points
    [maximum, label_majority] = max(count_class, [], 1);
    Class = uniqueLabel(label_majority);
End
```

9. By computing the values of the classification accuracy for the K-NN algorithm for various values of K and the corresponding accuracy (c = 100R/(3M)), we get the following graph (and K from 1 to 120 for a better visual representation).



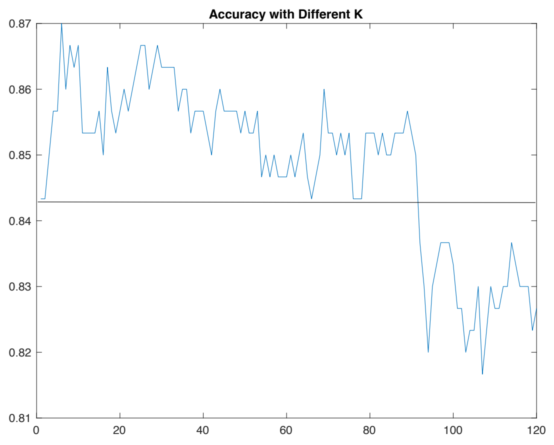


figure 1 figure 2

From the graph we can see that the maximum value of accuracy is c = 87. Through the function
`[maxprecision, best_k] = max(accuracy);`
we see that the value that yields the highest accuracy of 87 is K = 6.

Trend: From K = 1 to K = 6, accuracy increases as K increases. After K = 6, the accuracy oscillates around c = 3.855 (and very slowly decreases). However, at around K = 130, the accuracy drops sharply, as the result of overfitting. At K = 150, the accuracy is close to 33.33.

Compare with LS classification:

The accuracy using LS classification is 84.33, which is less than the maximum accuracy of 87 using K-NN. The black horizontal line in figure 2 is the accuracy using LS classification; thus we observe that when using K-NN, most values of K will give accuracy higher than 84.33. But the difference between accuracy is small.

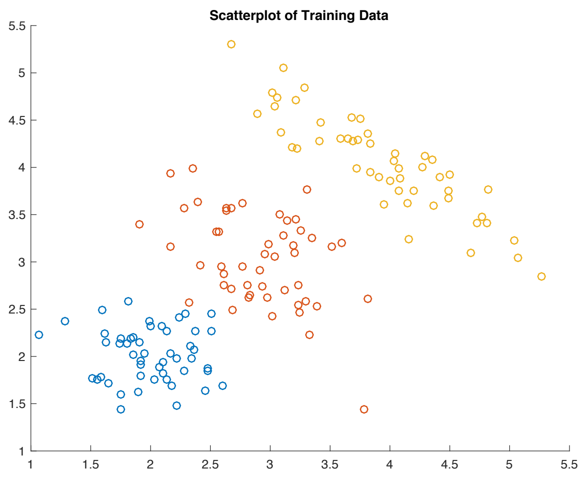
However, it takes the K-NN method longer to do the classification work than LS classification. This happens because I used both for loop and nested for loop in the KNN function; the computation cost is larger than matrix computation in LS classification.
LS classification: Elapsed time is 0.001241(training) + 0.000490(testing) = 0.0017 seconds.
K-NN: Elapsed time is 0.046954 seconds.

Which classification method is better? If we have a very large training or testing data set, the time cost of K-NN method becomes huge, and thus LS classification method is much better since: a) it is much faster, and b) the accuracy it gives is not much smaller than that given by K-NN. If we are dealing with data sets similar to the ones we did in question #5-9, then K-NN is slightly better due to its slightly higher accuracy. I would recommend LS classification.

Matlab code:

```
%Test for maximum accuracy and corresponding K
labels = [ones(1,50),ones(1,50)+1,ones(1,50)+2];
test_labels = [ones(1,100),ones(1,100)+1,ones(1,100)+2];

for k = 1:120
    [Class_k] = Knn(X3,X2,labels, k);
    counter = 0;
    for i = 1:300
        if Class_k(i) == test_labels(i)
            counter = counter+1;
        end
    end
    accuracy(k) = counter/300;
end
[maxprecision, best_k] = max(accuracy);

%Plot accuracy with respect to values of K
figure
plot(accuracy)
title('Accuracy with Different K')

tic
[Class] = Knn(X3,X2,labels,6);
toc

temp2 = 0;
for i = 1:300
    if Class(i) == test_labels(i)
        temp2 = temp2+1;
    end
end
```

10. Here's our new training data:



After performing LS classification, here's what I got with $\tilde{X}\tilde{W}$ and decision boundaries:

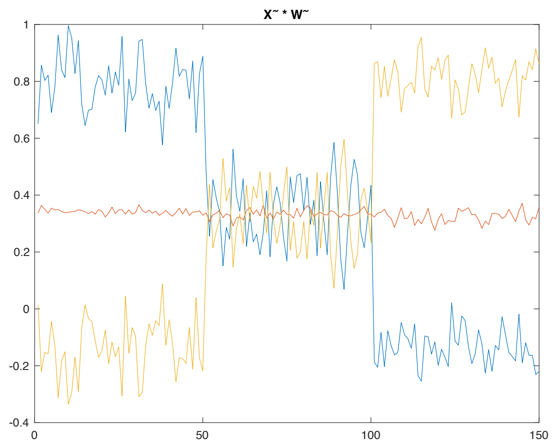


figure 3

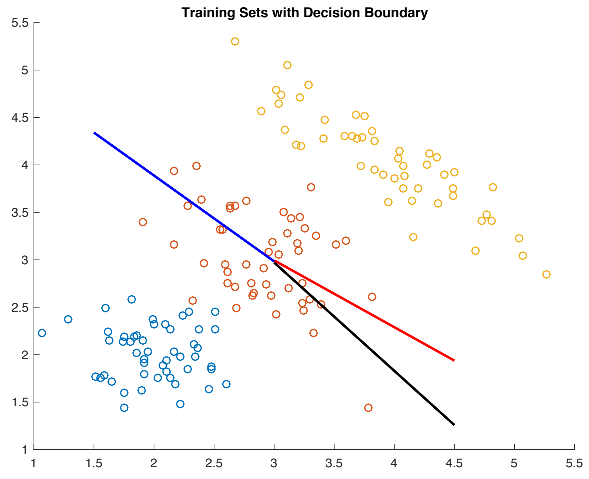


figure 4

From figure 3, we observe that $\tilde{X}\tilde{W}$ here only attempts to minimize

$$\frac{1}{2}||\tilde{\mathbf{X}}\tilde{\mathbf{W}} - \mathbf{T}||_F^2$$

instead of approximating each data point's class label. Therefore, my LS classification in figure 4 is a failure.

Here are some observations of figure 4:

a. **Decision boundaries**: Most of the points in the "red" label are misclassified as "blue" class and "yellow" class. The region of "red" class is too small that most of the points are misclassified. In fact, *the accuracy is only 70 under LS classification* method. However, *KNN method gives a maximum accuracy of 98* with K = 6, which is much higher than that in LS classification.
b. **Time cost**: it takes us 0.002417 seconds to perform LS classification (training + testing) and 0.051855 seconds to perform K-NN. K-NN is still slower than LS classification.
c. The centers of three data groups in figure 4 is on a straight line, while those in figure 0 is not on a line.
d. **Separable**: Classes in figure 4 look more separable than in the previous case, since there are less class overlaps in figure 4.
e. **Class by class accuracy**: under LS classification, accuracies for class 1 (blue) and class 3 (yellow) are 100, while the accuracy for class 2 (red) is only 42. This indicates that while points with original label "blue" and "yellow" are correctly classified, most of the points with original label "red" are misclassified.

Possible reasons why decision boundaries in figure 4 look “incorrect”:

a. Decision boundaries given by the LS classification are lines that intersect in one point. They partition the data points into three regions. But the centers of the three groups of points in figure 4 locate on a line, the decision boundaries in the form of intersecting lines cannot separate the points correctly. Thus, although classes in figure 4 look more separable, they cannot be separated by LS classification.
b. LS classification corresponds to maximum likelihood under the assumption of a Gaussian conditional distribution, whereas binary target vectors clearly have a distribution that is far from Gaussian. (Textbook)

Conclusion: in this case, K-NN is better with better assumption and much higher accuracy compared to LS classification, even though it takes K-NN longer to do the classification work.

Matlab code for #10:

```
%10. repeat
mu_41 = [2;2];
sig_41 = [[0.1,0];[0,0.1]];
mu_42 = [3;3];
sig_42 = [[0.2,-0.1];[-0.1,0.3]];
mu_43 = [4;4];
sig_43 = [[0.4,-0.3];[-0.3,0.3]];
rng(940730)
train_41 = mvnrnd(mu_41,sig_41,50);
train_42 = mvnrnd(mu_42,sig_42,50);
train_43 = mvnrnd(mu_43,sig_43,50);

%Compute W
X4 = [train_41;train_42;train_43];
X_tilde4 = [ones(150,1),X4];
T_41 = [ones(50,1),zeros(50,1),zeros(50,1)];
T_42 = [zeros(50,1),ones(50,1),zeros(50,1)];
T_43 = [zeros(50,1),zeros(50,1),ones(50,1)];
T4 = [T_41;T_42;T_43];
W_tilde4 = (inv(X_tilde4.' * X_tilde4))*X_tilde4.'*T4;
X_W_tilde4 = X_tilde4 * W_tilde4;
figure
plot(X_W_tilde4)
title('X^~ * W^~')
%Scatterplot
figure
hold on
scatter(train_41(:,1),train_41(:,2))
scatter(train_42(:,1),train_42(:,2))
scatter(train_43(:,1),train_43(:,2))
title('Scatterplot of Training Data')
hold off

%%
%plot decision boundary
%group 1 and 2
W4 = [W_tilde4(2,:); W_tilde4(3,:)];
W04 = [W_tilde4(1,:)];
p_41 = (W04(1,2)-W04(1,1)-(W4(1,1)-W4(1,2))*3)/(W4(2,1)-W4(2,2));
p_42 = (W04(1,2)-W04(1,1)-(W4(1,1)-W4(1,2))*4.5)/(W4(2,1)-W4(2,2));

%group 1 and 3
p_43 = (W04(1,3)-W04(1,1)-(W4(1,1)-W4(1,3))*1.5)/(W4(2,1)-W4(2,3));
p_44 = (W04(1,3)-W04(1,1)-(W4(1,1)-W4(1,3))*3)/(W4(2,1)-W4(2,3));
```

```
%group 2 and 3
p_45 = (W04(1,3)-W04(1,2)-(W4(1,2)-W4(1,3))*3)/(W4(2,2)-W4(2,3));
p_46 = (W04(1,3)-W04(1,2)-(W4(1,2)-W4(1,3))*4.5)/(W4(2,2)-W4(2,3));

figure
hold on
scatter(train_41(:,1),train_41(:,2))
scatter(train_42(:,1),train_42(:,2))
scatter(train_43(:,1),train_43(:,2));
plot([3,4.5],[p_41,p_42],'r','LineWidth',2)
plot([1.5,3],[p_43,p_44],'b','LineWidth',2)
plot([3,4.5],[p_45,p_46],'black','LineWidth',2)
title('Training Sets with Decision Boundary')
hold off

%%
%LS result Testing data
rng(940730);
test_51 = mvnrnd(mu_41,sig_41,100);
test_52 = mvnrnd(mu_42,sig_42,100);
test_53 = mvnrnd(mu_43,sig_43,100);

tic
X5 = [test_51;test_52;test_53];
X_tilde5 = [ones(300,1),X5];
T_51 = [ones(100,1),zeros(100,1),zeros(100,1)];
T_52 = [zeros(100,1),ones(100,1),zeros(100,1)];
T_53 = [zeros(100,1),zeros(100,1),ones(100,1)];
T5 = [T_51;T_52;T_53];
X_W_tilde5 = X_tilde5 * W_tilde4;
toc

%calculate R
R2 = 0;
for i = 1:100
    [M2,I2]=max(X_W_tilde5(i,:));
    if I2 == 1
        R2 = R2+1;
    end
end

for i = 101:200
    [M2,I2]=max(X_W_tilde5(i,:));
    if I2 == 2
        R2 = R2+1;
    end
end

for i = 201:300
    [M2,I2]=max(X_W_tilde5(i,:));
    if I2 == 3
        R2 = R2+1;
    end
end

c2 = 100*R2/(3*100);

%%
% Accuracy class by class

%class 1
R_cbc1 = 0;
for i = 1:100
```

```
    [M2,I2]=max(X_W_tilde5(i,:));
    if I2 == 1
        R_cbc1 = R_cbc1+1;
    end
end

c_cbc1 = 100*R_cbc1/100;

%class 2
R_cbc2 = 0;
for i = 101:200
    [M2,I2]=max(X_W_tilde5(i,:));
    if I2 == 1
        R_cbc2 = R_cbc2+1;
    end
end

c_cbc2 = 100*R_cbc2/100;


%class 3
R_cbc3 = 0;
for i = 1:100
    [M2,I2]=max(X_W_tilde5(i,:));
    if I2 == 1
        R_cbc3 = R_cbc3+1;
    end
end

c_cbc3 = 100*R_cbc3/100;

%%
%KNN Testing
labels2 = [ones(1,50),ones(1,50)+1,ones(1,50)+2];
test_labels2 = [ones(1,100),ones(1,100)+1,ones(1,100)+2];

tic
[Class2] = Knn(X5,X4,labels2, 6);
toc

temp3 = 0;
for i = 1:300
    if Class2(i) == test_labels2(i)
        temp3 = temp3+1;
    end
end

c3 = 100*temp3/300;
```