Assignment2

Student Number: 45606134
Name: Weining Hu

Question 1

1.1

```
function [model] = knn(X,y,K)
% [model] = knn(X,y,k)
%
% Implementation of k-nearest neighbour classifer
model.X = X;
model.y = y;
model.K = K;
model.C = max(y);
model.predict = @predict;
end

function [yhat] = predict(model,Xtest)
%first create a matrix containing the squared Euclidean distances between
%all training and test points
X = model.X;
[N,D] = size(X);
[T,D] = size(Xtest);

% Dist will be the matrix (N*T)
Dist = X.^2*ones(D,T) + ones(N,D)*(Xtest').^2 - 2*X*Xtest';

%now iterate through the whole matrix to find the K smallest point with
%minimum distance, we just pick one column everytime and sort it

% create a matrix K_min(K*T) to store the index of traning data for k minimum
% distance
K = model.K;
K_min = zeros(K,T);
y = model.y;
C = model.C;
yhat = ones(T,1);
for t = 1:T
    test = Dist(:,t);
    [sortDist,sortIndex] = sort(test,'ascend');
    minIndex = sortIndex(1:K);
    % fill in the y values corrsponds to K smallest index a
    K_min(:,t) = y(minIndex);

    % Vote for different y values
    count = zeros(C,1);
    for c = 1:C
        count(c) = sum(y(minIndex)==c);
    end
    % The y with the maximum vote is our predicted value
```

```
    [M,I] = max(count);
    yhat(t) = I;
end
end
```

1.2 Choosing the number of neighbours
If the K gets smaller (approximate 1), the training error will get smaller (approximate 0). However, since there exist the trade-off , therefore the smaller the K the worse its training error approximate test error. When K is 1, there could be terrible overfit.
In practical, we could use K-fold cross-validation to find the model which minimize the generalization error.

1.3
Everytime with a new example, we have already calculated the distance between this example and all of the training data. If we know a selection algorithm(such as quick select), then we will know the k-th smallest in this unsorted list, we mark it as S. The running time will be O(N). Then, we run the second pass, we go through the unsorted list again and pick the elements that are smaller than the value of S and store the label corresponding to this element. At last, we run through the k labels we recorded and vote for different label and output the label with the maximum count as our predicted value.

The total running time would be O(N)+O(N)+O(k) => O(N)

Question2

2.1

The error: 0.19

```
function [model] = naiveBayes(X,y)
% [model] = naiveBayes(X,y,k)
%
% Implementation of navie Bayes classifier for binary features

% Compute number of training examples and number of features
[N,D] = size(X);

% Computer number of class lables
C = max(y);

counts = zeros(C,1);
for c = 1:C
    counts(c) = sum(y==c);
end
p_y = counts/N; % This is the probability of each class, p(y(i) = c)

% We will store:
%   p(x(i,j) = 1 | y(i) = c) as p_xy(j,1,c), which means that x(i,j) appear
%   p(x(i,j) = 0 | y(i) = c) as p_xy(j,2,c)
p_xy = ones(D,2,C);

for c = 1:C
    row = find(y==c);
    count_c = sum(y==c);
    %subtract the rows
    with_c = X(row,:);

    for j = 1:D
        count0 = sum(with_c(:,j)==0);
        count1 = count_c-count0;

        p_xy(j,2,c) = count0/count_c;
        p_xy(j,1,c) = count1/count_c;
    end
    %[sortf,sortIndex] = sort(c,'descend');
    %display(sortIndex(1:3));
end

model.C = C;
model.p_y = p_y;
model.p_xy = p_xy;
model.predict = @predict;
end

function [yhat] = predict(model,Xtest)
[T,D] = size(Xtest);
```

```
C = model.C;
p_y = model.p_y;
p_xy = model.p_xy;

yhat = zeros(T,1);
for i = 1:T
    probs = p_y; % This will be the probability for each class
    for j = 1:D
        if Xtest(i,j) == 1
            for c = 1:model.C
                probs(c) = probs(c)*p_xy(j,1,c);
            end
        else
            for c = 1:model.C
                probs(c) = probs(c)*p_xy(j,2,c);
            end
        end
    end
    [maxProb,yhat(i)] = max(probs);
end
end
```

2.2

The function frequency.m is included in the file as well

Code

```
function [word_count] = frequency(X,y)
% Compute number of training examples and number of features
[N,D] = size(X);

% Computer number of class lables
C = max(y);
word_count = zeros(C,3);


for c = 1:C
    row = find(y==c);
    count_c = sum(y==c);
    %subtract the rows
    with_c = X(row,:);
    wc = zeros(D,1);
    for j = 1:D
        count0 = sum(with_c(:,j)==0);
        count1 = count_c-count0;
        wc(j) = count1;


    end
    [sortf,sortIndex] = sort(wc,'descend');
    word_count(c,:) = sortIndex(1:3);
end
```

ans =

98	38	23
6	89	32
85	38	59
33	26	73

class1 (comp): 'windows' 'help' ‘email'
class2 (rec): ‘car’ ‘team’ ‘games’
class3 (sci): ‘space’ ‘help’ ‘nasa’
class4 (talk): ‘god’ ‘fact’ ‘question’

2.3
For T test data, for each data in the entry, we iterate through D features. For each of the feature, we iterate through C labels and find the label that gives the maximum probability. So that the total running time would be O(TDC).

2.4
There is a fundamental trade-off in learning theory. For simple model (like Naive Bayes), training error is a good approximation of testing error. Because of its simplicity, it does not fit the training data well.
Also, there is no free lunch theorem would apply. That there is no single method would goes for all dataset.

Question3

3.1

After the modification with bootstrap aggregation:

Validation error with decision tree: 0.05

Validation error with random forest: 0.04

```
function [model] = randomForest(X,y,depth,nBootstraps)
% w = randomForest(X,y,depth,nBootstraps)
%
% Computes Bootstrapped Random Tree Classifier

[N,D] = size(X);

for k = 1:nBootstraps
    sample = datasample(1:N,N,'Replace',true);
    model.subModel{k} = randomTree(X(sample,:),y(sample),depth);
end

model.predict = @predict;

end

function [y] = predict(model,X)

for k = 1:length(model.subModel)
    y(:,k) = model.subModel{k}.predict(model.subModel{k},X);
end
y = mode(y,2);
end
```

3.2
After the modification with random feature selection
Validation error with decision tree: 0.05
Validation error with random forest: 0.01

```
function [model] = randomStump(X,y)
% [model] = randomStump(X,y)
%
% Fits a decision stump that splits on a single variable,
%   assuming that X is binary {0,1}, and y is categorical {1,2,3,...,C}.

% Compute number of training examples and number of features
[N,D] = size(X);

% Computer number of class lables
C = max(y);

% Address the trivial case where we do not split
count = zeros(C,1);
for n = 1:N
    count(y(n)) = count(y(n)) + 1;
end
[maxCount,maxLabel] = max(count);

% Compute total entropy
p = count/sum(count); % Convert to probabilities
entropyTotal = -sum(p.*log0(p));

maxGain = 0;
splitVariable = [];
splitThreshold = [];
splitLabel0 = maxLabel;
splitLabel1 = [];

% Loop over features looking for the best split
if any(y ~= y(1))
    for d = randi([1,D],1)
        thresholds = sort(unique(X(:,d)));

        for t = thresholds'

            % Count number of class labels where the feature is greater than
threshold
            count1 = zeros(C,1);
            for n = find(X(:,d) > t)'
                count1(y(n)) = count1(y(n)) + 1;
            end
            count0 = count-count1;
```

```
            % Compute infogain
            p1 = count1/sum(count1);
            p0 = count0/sum(count0);
            H1 = -sum(p1.*log0(p1));
            H0 = -sum(p0.*log0(p0));
            prob1 = sum(X(:,d) > t)/N;
            prob0 = 1-prob1;
            infoGain = entropyTotal - prob1*H1 - prob0*H0;

            % Compare to minimum error so far
            if infoGain > maxGain
                % This is the lowest error, store this value
                maxGain = infoGain;
                splitVariable = d;
                splitThreshold = t;
                % Compute majority class
                [maxCount,splitLabel1] = max(count1);
                [maxCount,splitLabel0] = max(count0);
            end
        end
    end
end
model.splitVariable = splitVariable;
model.splitThreshold = splitThreshold;
model.label1 = splitLabel1;
model.label0 = splitLabel0;
model.predict = @predict;
end

function [y] = predict(model,X)
[T,D] = size(X);

if isempty(model.splitVariable)
    y = model.label0*ones(T,1);
else
    y = zeros(T,1);
    for n = 1:T
        if X(n,model.splitVariable) > model.splitThreshold
            y(n,1) = model.label1;
        else
            y(n,1) = model.label0;
        end
    end
end
end
```

3.3

The smaller the number of features(k) that each decision stump considers, the simpler the models and therefore training error is a good approximation of the testing error. But it does not fit the training data well.

Question4 K means cluster

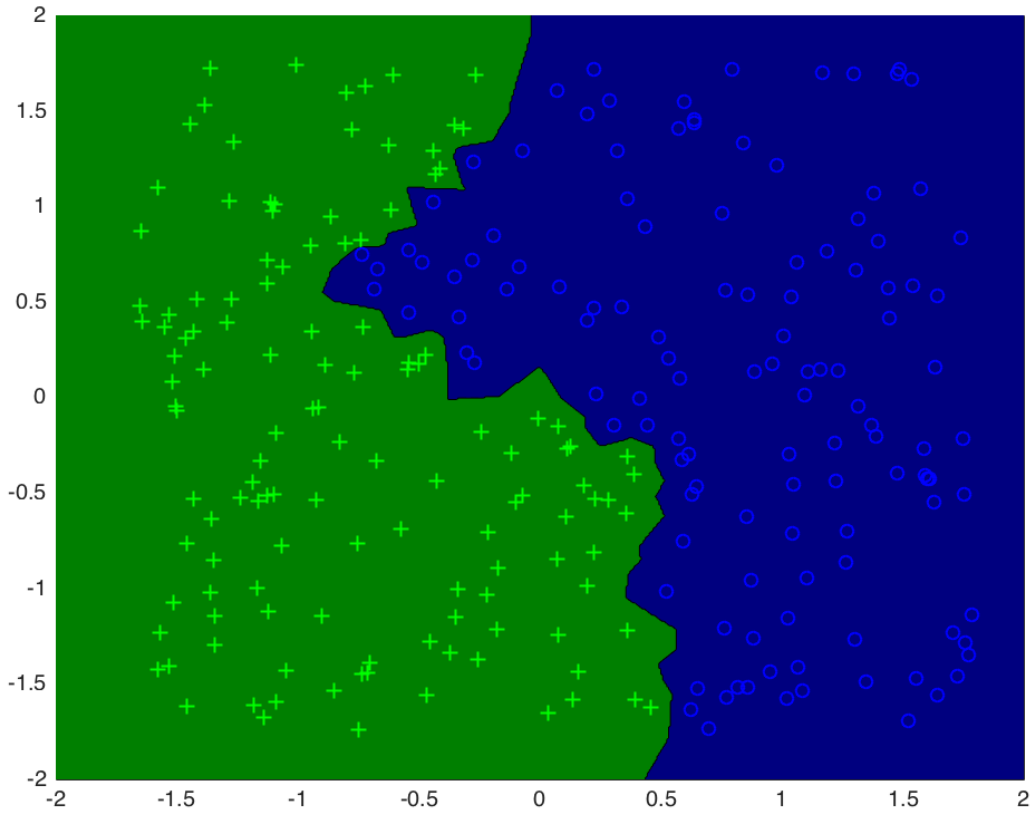
4.1

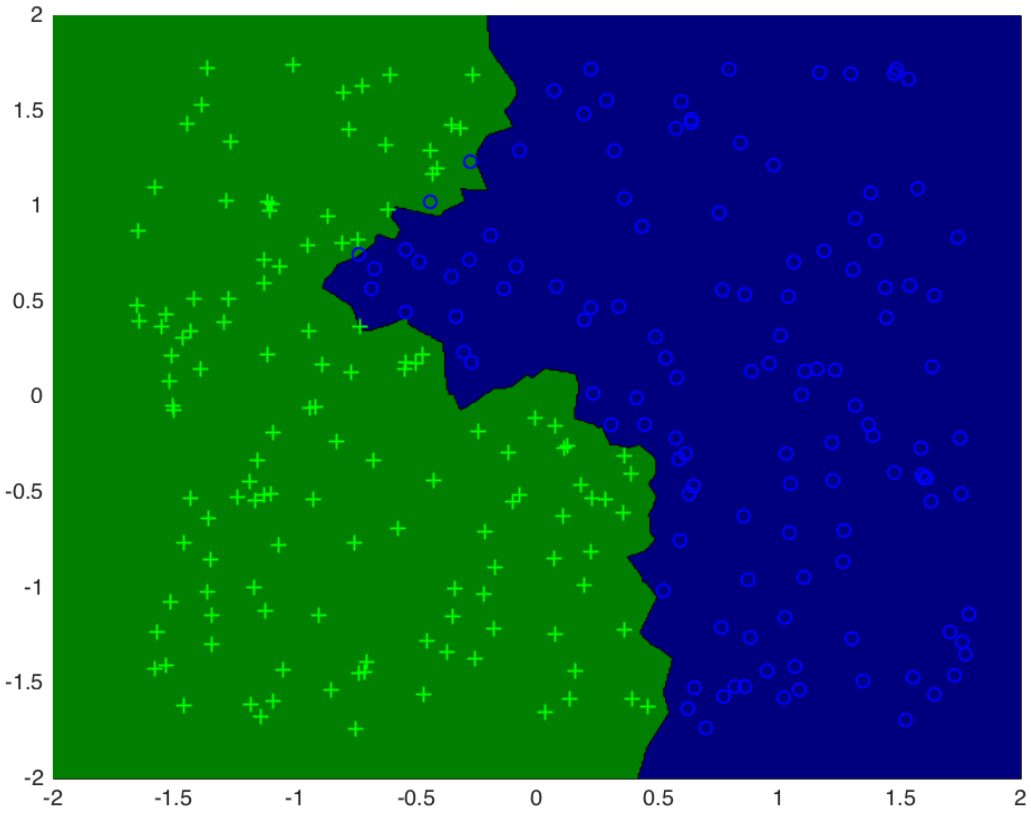
After using the k means plus plus initialization, it improves the chance of finding the four true clusters.

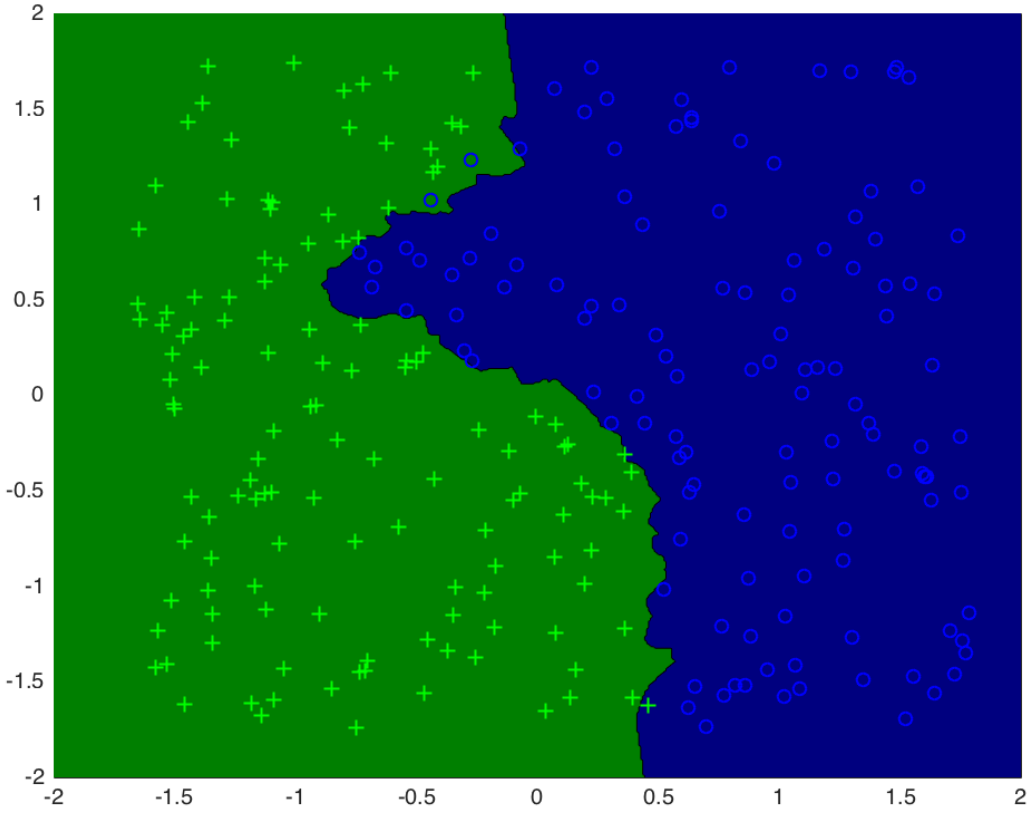
```
function [model] = clusterKmeans(X,K)
% [model] = clusterKmeans(X,K)
%
% K-means clustering

[N,D] = size(X);

% Choose random points to initialize means
%means = zeros(K,D);
%for k = 1:K
 %   i = ceil(rand*N);
  %  means(k,:) = X(i,:);
%end

% apply kmeans plus plus to initial the k means
X_square = X.^2*ones(D,K);
means = zeros(K,D);

for k = 1:K
    if (k == 1)
        i = ceil(rand*N);
        means(k,:) = X(i,:);
    else
        distance = zeros(N,1);
        d = (X_square + ones(N,D)*(means').^2 - 2*X*means');
        for n = 1:N
            distance(n) = min(d(n,:));
        end
        p = distance/sum(distance);
        index = sampleDiscrete(p);
        means(k,:) = X(index,:);
    end
end


X2 = X.^2*ones(D,K);
while 1
    means_old = means;

    % Compute Euclidean distance between each data point and each mean
    distances = sqrt(X2 + ones(N,D)*(means').^2 - 2*X*means');
```

```
    % Assign each data point to closest mean
    [~,clusters] = min(distances,[],2);

    % Compute mean of each cluster
    means = zeros(K,D);
    for k = 1:K
        means(k,:) = mean(X(clusters==k,:),1);
    end

    % If we only have two features, make a colored scatterplot
    if D == 2
        clf;hold on;
        colors = getColors;
        for k = 1:K
            h = plot(X(clusters==k,1),X(clusters==k,2),'.');
            set(h,'Color',colors{k});
        end
        pause(.25);
    end

    fprintf('Running K-means, difference = %f\n',max(max(abs(means-
means_old))));

    if max(max(abs(means-means_old))) < 1e-5
        break;
    end
end

model.means = means;
model.clusters = clusters;
```

Comment on the timing: The k means plus plus cut the time in half because of the reduction in iteration. And it is more probable to find the real clusters.


2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2


2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2


2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2