Homework 5 solution template
CMPSCI 370 Spring 2019, UMass Amherst
Name: DongWon Park

1 Decision trees

1. Empty decision tree.
   - accuracy on training set: 50%
   - accuracy on test set: 50%
   - explain your answer: Because it is the empty decision tree, the accuracy depends on the ratio of the data and its label (i.e., majority number of each label). The number of data set for '3' and '8' is equal which is 100 each. So whatever pixel you would take and predict its class, the accuracy will be 50%. Also it is 50% on the test set because test set also has same 100 elements of '3' and 100 of '8' data.

2. Decision tree of depth 1
   (a) The coordinate of pixel with the highest accuracy: $(x, y) =$ (19, 11) (row, column). Figure 1 visualizes the scores.

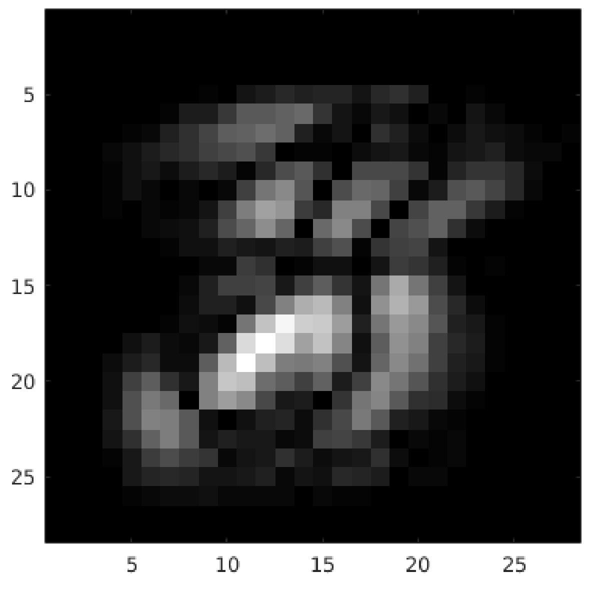


Figure 1: Visualizing scores

(b) Write down the decision tree that obtains the best classification

```
% in the form of (row, column)
if pixel(19,11) == 1 then predict 3, else predict 8
```

(c) Accuracy of decision tree with depth 1 on test set: 0.72 (144/200)

3. Decision tree of depth 2.

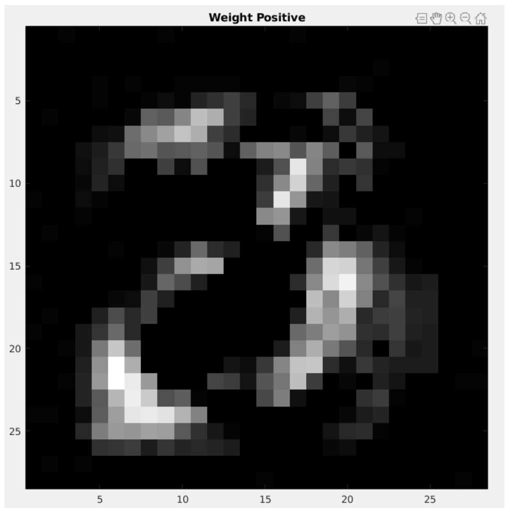
(a) Write down the decision tree in if-then-else statement

```
% in the form of (row, column)
if pixel(19,11) == 0
  if pixel(18,14) == 0
    predict 3;
  else
    predict 8;
  end
else
  if pixel(16,22) == 0
    predict 8;
  else
    predict 3;
  end
end
```

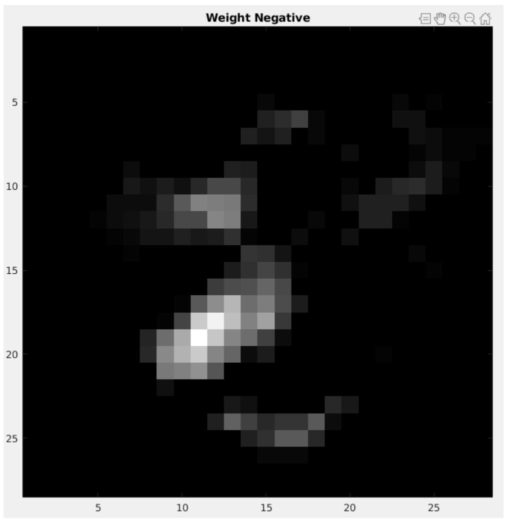
(b) Accuracy of decision tree with depth 2 on test set: 0.89 (178/200)

2 Linear classifier

- Accuracy of linear classifier on test set: 0.9450 (189/200)
- Figure 2 visualizes positive and negative parts of weights.



(a) positive weights



(b) negative weights

Figure 2: Visualizing weights

3 Nearest neighbor classifier

Figure 3 plots the test accuracy against the number of k for k nearest-neighbor classifier

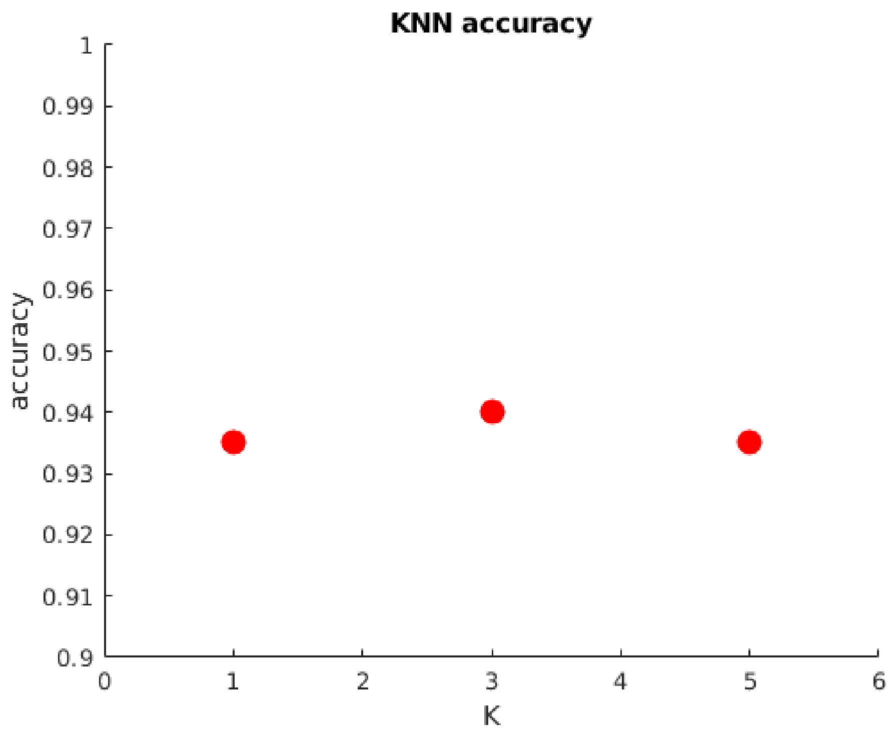
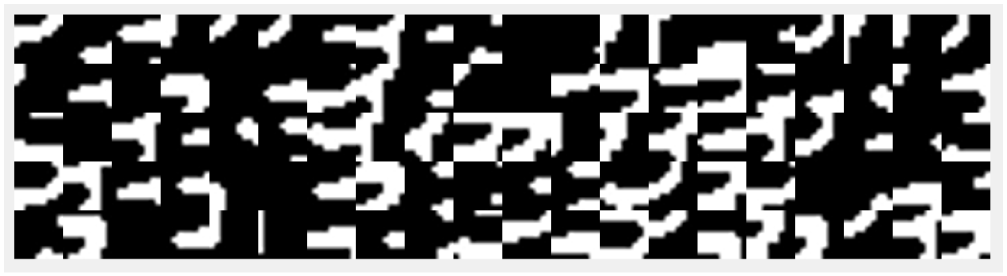


Figure 3: Accuracy of k-nearest-neighbor classifier

4 Bag-of-visual-words representations

1. Visualize the dictionary



2. **Extra credit**: test accuracy: 96.5%

5 Solution code

Include the source code for your solutions as seen below (only the files you implemented are necessary). In latex the command verbatiminput{alignChannels.m} allows you to include the code verbatim as seen below. Regardless of how you do this the main requirement is that the included code is readable (use proper formatting, variable names, etc.) A screenshot of your code works to provided you include a link to source files.

5.a scoreFeatures.m

```
function score = scoreFeatures(x, y)

THREE = 3;
EIGHT = 8;
score = zeros(size(x, 1), size(x, 2));

% loop over all feature pixels of a digit image
for i=1:size(x,1)       % size(x) = 28x28x1x200
    for j=1:size(x,2)   % size(y) = 1:200
        % get indexes of data x splitting by its value of 0, 1
        ind_NO = x(i,j,:,:)==0;
        ind_YES = x(i,j,:,:)==1;

        % get subset of data x
        NO = x(ind_NO);  % subset of data x on which x(i,j) == 0
        YES = x(ind_YES); % subset of data x on which x(i,j) == 1

        % count each subset
        NO_THREE = sum(y(ind_NO)==THREE);
        NO_EIGHT = sum(y(ind_NO)==EIGHT);
        YES_THREE = sum(y(ind_YES)==THREE);
        YES_EIGHT = sum(y(ind_YES)==EIGHT);

        % score = #of majority answer(3 or 8) in NO + #of majority answer in YES
        score(i,j) = max(NO_THREE, NO_EIGHT) + max(YES_THREE, YES_EIGHT);
    end
end
```

```
% get index of the highest score and print the highest score value
[max_x, max_y] = find(score==max(max(score)))
```

5.b code training a decision tree

```
function best_f = DT(x, y)

% get score of 28*28 matrix
score = scoreFeatures(x, y);

% visualize the score
subplot(2,2,1:2)
imagesc(score); colormap gray; axis image

% get best predict feature
[max_x, max_y] = find(score==max(max(score)));

% print the coordiante of the best feature and its score
best_f = [max_x, max_y];
best_f = [best_f(1,1), best_f(1,2)] % If there is a tie, pick one arbitrarily.
score(best_f(1,1), best_f(1,2)) % printf the highest score

end
```

5.c code evaluate a decision tree on test set

```
THREE = 3;
EIGHT = 8;

% load data
d = load('data.mat');
x = d.data.train.x;
y = d.data.train.y;
test_x = d.data.test.x;
test_y = d.data.test.y;

% ********** Depth 1 **********
% train the Decision Tree and get the best feature
best_f = DT(x, y);

% apply this rule to the test set
NO = [];
YES = [];
NO_label = [];
YES_label = [];
for i=1:size(test_x,4)
    % if test pixel is 0, then place it into NO and save its location
    if test_x(best_f(1,1),best_f(1,2),1,i)==0
        NO = cat(4,NO,test_x(:,:,:,i));
        NO_label = cat(2,NO_label,test_y(:,i));
    else
    % if test pixel is 1, then place it into YES and save its location
        YES = cat(4,YES,test_x(:,:,:,i));
        YES_label = cat(2,YES_label,test_y(:,i));
    end
end

% Get the accuracy of the test set in Depth 1
acc_test = sum(NO_label==THREE) + sum(YES_label==EIGHT)
num_test = size(test_x,4)
acc_test_pcnt = acc_test / num_test



% ********** Depth 2 **********
% *** subdataset: NO ***
% train the Decision Tree and get the best feature
best_f = DT(NO, NO_label);

% apply this rule to the test set
NO_D2 = [];
YES_D2 = [];
NO_D2_label = [];
YES_D2_label = [];
for i=1:size(NO,4)
    % if test pixel is 0, then place it into NO and save its location
    if NO(best_f(1,1),best_f(1,2),1,i)==0
        NO_D2 = cat(4,NO_D2,NO(:,:,:,i));
        NO_D2_label = cat(2,NO_D2_label,NO_label(:,i));
    else
    % if test pixel is 0, then place it into YES and save its location
        YES_D2 = cat(4,YES_D2,NO(:,:,:,i));
        YES_D2_label = cat(2,YES_D2_label,NO_label(:,i));
    end
end

% Get the accuracy of the test set in Depth 2 -subdata: NO
acc_test_D2_NO = sum(NO_D2_label==THREE) + sum(YES_D2_label==EIGHT)
num_test_D2_NO = size(NO_D2,4) + size(YES_D2,4)
acc_test_D2_NO_pcnt = acc_test_D2_NO / num_test_D2_NO


% ********** Depth 2 **********
% *** subdataset: YES ***
% train the Decision Tree and get the best feature
best_f = DT(YES, YES_label);

% apply this rule to the test set
NO_D2 = [];
YES_D2 = [];
NO_D2_label = [];
YES_D2_label = [];
for i=1:size(YES,4)
    % if test pixel is 0, then place it into NO and save its location
    if YES(best_f(1,1),best_f(1,2),1,i)==0
        NO_D2 = cat(4,NO_D2,YES(:,:,:,i));
        NO_D2_label = cat(2,NO_D2_label,YES_label(:,i));
    else
    % if test pixel is 0, then place it into YES and save its location
        YES_D2 = cat(4,YES_D2,YES(:,:,:,i));
        YES_D2_label = cat(2,YES_D2_label,YES_label(:,i));
    end
end

% Get the accuracy of the test set in Depth 2 -subdata: YES
acc_test_D2_YES = sum(NO_D2_label==EIGHT) + sum(YES_D2_label==THREE)
num_test_D2_YES = size(NO_D2,4) + size(YES_D2,4)
acc_test_D2_YES_pcnt = acc_test_D2_YES / num_test_D2_YES


% ************** Final accuracy in Depth 2 *********************
acc_test_D2_total = acc_test_D2_NO + acc_test_D2_YES
num_test_D2_total = num_test_D2_NO + num_test_D2_YES
acc_test_D2_total_pctn = acc_test_D2_total / num_test_D2_total
```

5.d code for training and evaluating the linear model

```
% load data
d = load('data.mat');
x = d.data.train.x;
y = d.data.train.y;
test_x = d.data.test.x;
test_y = d.data.test.y;

% ***** Train the model *****
% Reshape the training data
xrs = reshape(x, 28*28, 200);

% Get a model by training with the train data set
model = linearTrain(xrs, y);


% ***** Evaluate the model *****
% Reshape the test data
xrs_test = reshape(test_x, 28*28, 200);

% Get a prediction with test data
ypred = linearPredict(model, xrs_test);

% Calculate the accuracy
predict_correct = sum(ypred == test_y)
predict_wrong = sum(ypred ~= test_y)
accuracy = predict_correct / (predict_correct + predict_wrong)



% ***** Visualization *****
w = model.w(1:end-1);
wp = w.*(w>0);
wn = -w.*(w<0);
subplot(1,2,1);
```

```
imagesc(reshape(wp, [28 28])); colormap gray; axis image
title('Weight Positive')
subplot(1,2,2);
imagesc(reshape(wn, [28 28])); colormap gray; axis image
title('Weight Negative')
```

5.e code for k-nearest neighbor classification

```
function accuracy = KNN(test_x, train_x, y, K)

test_label = [];
for i=1:size(test_x,4)      % for each TEST images
    distances = [];
    for j=1:size(train_x,4)       % for each TRAIN images

        % reshape images into 1D vector to make it easier to calculate the distance
        vec_test = reshape(test_x(:,:,:,i),[1 size(test_x,1)*size(test_x,2)]);
        vec_train = reshape(train_x(:,:,:,j),[1 size(train_x,1)*size(train_x,2)]);

        % get the distance
        d = pdist2(double(vec_test), double(vec_train));

        % record all distances from this TEST image to all TRAIN images
        distances = cat(2,distances,d);
    end

    % get the K nearest neighbors by finding k minimum distances
    min_d = mink(distances, K);

    % get its indexes
    [tf, loc] = ismember(min_d, distances);

    % get its labels from the TRAIN data
    labels = y(loc);

    % pick its label by seeing the majority of K nearest neighbors
    test_label(i) = mode(labels);
end


% Evaluate the accuracy
correct = sum(y==test_label)
wrong = sum(y~=test_label)
accuracy = correct / (correct + wrong)

end
```

5.f code for evaluating k-nearest neighbor classificaiton

```
THREE = 3;
EIGHT = 8;

% load data
d = load('data.mat');
```

```
x = d.data.train.x;
y = d.data.train.y;
test_x = d.data.test.x;
test_y = d.data.test.y;

% Run KNN by different Ks
accuray_K1 = KNN(test_x, x, y, 1);
accuray_K3 = KNN(test_x, x, y, 3);
accuray_K5 = KNN(test_x, x, y, 5);
accuracy = [accuray_K1; accuray_K3; accuray_K5];

% Plot the accuracy
figure()
scatter([1;3;5], accuracy, 1300, 'r', '.'); ylim([0.9 1]); xlim([0 6])
xlabel('K')
ylabel('accuracy')
title('KNN accuracy')
```

5.g constructDictionary.m

```
function C = constructDictionary(x, K, ps)
% x: [h, w, 1, n]

C = zeros(ps, ps, K);

% Decide the boundary: number of pixels that can take in the patch
% from each direction of LEFT/UPPER or RIGHT/BOTTOM
% LU : LEFTT/UPPER  RB: RIGHT/BOTTOM
LU = floor((ps-1)/2);
if mod(ps,2) == 0  % if patch size ps is even
    RB = floor((ps+1)/2);
else               % if patch size ps is odd
    RB = floor((ps-1)/2);
end

% generate t patches from each image
% loop over each image from TRAIN dataset x
features = [];
for i=1:size(x,4)
    % randomly draw t patches
    t = 20;
    for j=1:t
        % generate random center pixel(a,b)
        % randomized patch's center will be located within boundary of image

        % random center pixel
        pixel = randi([1+LU size(x,1)-RB],1,2);

        % get the patch of size ps*ps within the patch boundary
        patch = x(pixel(1)-LU:pixel(1)+RB, pixel(2)-LU:pixel(2)+RB);

        % flatten the patch
        patch_flat = reshape(patch, [1 ps*ps]);
        % record the extracted patch
        features = cat(1,features,patch_flat);
    end
end

% classify the features(patch) by running kmeans
% loc = kmeans(features, K);
% uloc = unique(loc);

% reshape back the features into original shape and record into C
% for i=1:size(uloc,1)
%     a = reshape(features(uloc(i),:),[ps ps]);
%     C(:,:,i) = a;
% end

[IDX, B] = kmeans(features, K);
B = transpose(double(B));
C = reshape(B, [ps ps K]);
```

5.h encodeImage.m

```
function y = encodeImage(im, C)

y = ones(size(C, 3), 1);

% get patch size
ps = size(C,1);

% Decide the boundary: number of pixels that can take in the patch
% from each direction of LEFT/UPPER or RIGHT/BOTTOM
% LU : LEFTT/UPPER  RB: RIGHT/BOTTOM
LU = floor((ps-1)/2);
if mod(ps,2) == 0  % if patch size ps is even
    RB = floor((ps+1)/2);
else               % if patch size ps is odd
    RB = floor((ps-1)/2);
end

% loop over all pixels in im
for i=LU+1:size(im,1)-RB
    for j=LU+1:size(im,2)-RB
        % get a patch within the boundary and convert it to double
        patch = im(i-LU:i+RB, j-LU:j+RB);
        patch = double(patch);

        % variable to save least SSD(Sumof-Squared-Distance) and its location
        least_dist = inf;
        least_loc = 0;

        % loop over all feature in C and find the least SSD
        for k=1:size(C,3)
%                dist = pdist2(patch, C(:,:,k));   % This is too slow
            dist = distance(patch, C(:,:,k));
            if dist < least_dist
                least_dist = dist;
                least_loc = k;
            end
        end

        % increase the histogram
        y(least_loc) = y(least_loc) + 1;
    end
end

end

function dist = distance(p1, p2)
    dist = 0.0;
    for i=1:size(p1,2)
        for j=1:size(p2,2)
            dist = dist + (p1(i,j) - p2(i,j))^2;
        end
    end
end
```