Edward Zabrensky
861194954
Late Days used:0
Total =0

CS171 Assn2 Report

Part 0:

Downloaded the dataset and loaded it into the workspace by using the load function and the import data feature on MatLab. Use the load(breastcancerwisconsin) to load the mat file and then you can access breastcancerwinsconsin1 which is the matrix of data points.

Missing Values: I took care of the missing values by replacing them by 0 so that they would not affect the distance function.

I also took out the serial number of each data point because it was not useful.

Part 1:

```
function y_pred = knn_classifier(x_test, x_train, y_train, k, p)
    [sizeTest, ~] = size(x_test);%get size of x_test
    [sizeTrain, ~] = size(x_train);

    y_pred = zeros([sizeTest 1]);
    %calculate distance
    for i=1:sizeTest
       distanceBetweenAllPoints = zeros(sizeTrain, 2);
       distanceBetweenAllPoints(:,2) = y_train ;
       %calculate distance between x_test data point and all x_train
       for j=1:sizeTrain
           distanceBetweenAllPoints(j,1) = distance(x_test(i,:), x_train(j,:), p);
       end
       %sort distances
       distanceBetweenAllPoints = sortrows(distanceBetweenAllPoints);

       %get k nearest neighbors by grabbing first k data points, then use
       %mode on the vector for most frequent value
       distanceBetweenAllPoints = distanceBetweenAllPoints(1:k,2);
       y_pred(i,1) = mode(distanceBetweenAllPoints(1:k,1));
    end

end
```

```
Command Window
0.985714>>
```

I got a 98% accuracy when using my knn algorithm with an 80/20 split. I split the first 80% of the data as the training data and the last 20% as the test data.
The distance function used is the one from Assignment 1.
This function is implemented by calculating the distance between the test data point and every training data point. After calculating the distance I sorted the list and chose the first k datapoints or neighbors. Then I calculated the mode of those k neighbors to determine the label of the test datapoint.

```
%get xtrain, xtest and ytrain
%80percent train 20% is test
[n,~] = size(xdata);
eighteyCutOff = floor(n*.8);
eighteyCutOffPlus1 = eighteyCutOff+1;

xtrain = xdata(1:eighteyCutOff,:);
xtest = xdata(eighteyCutOffPlus1:n,:);
ytrain = ydata(1:eighteyCutOff,:);

%test with k = 1 and p = 2
k = 1;
p = 2;
ypred = knn_classifier(xtest, xtrain, ytrain, k, p);
sizeTest = n - eighteyCutOff;

%compute accuracy
count = 0;

for i=1:sizeTest
    if(ypred(i,1) == ydata(eighteyCutOff+i,1))
        count = count +1;
    end
    %fprintf('%i , %i\n', ypred(i,1), ydata(eighteyCutOff+i,1));
end
fprintf('%f',count/sizeTest);
```

//Test file used to get 98% accuracy. ‘Question1.m’

I take floor of 80% of the size of the xdata for the xtrain set

Then I take that 80% cut off plus 1 for xtest

Part2.1:

```
%shuffle rows
data = data(randperm(end),:);
xdata = data(:,(2:10));
ydata = data(:,11);

numFolds = 10;
[n,~] = size(xdata);
k = 1;
p = 2;
sizeTest = floor(n/numFolds);
ypred = zeros(sizeTest,1);
a=0;
accuracy = zeros(numFolds, 1);
sensitivity = zeros(numFolds, 1);
specificity = zeros(numFolds, 1);
```

I shuffle before cross validating so that there is diverse data in my partitions.

```
for w=1:numFolds
    if(w==1)
       xtest = xdata(1:sizeTest,:);
       xtrain = xdata(sizeTest+1:n,:);
       ytrain = ydata(sizeTest+1:n,:);
       ypred = knn_classifier(xtest, xtrain, ytrain, k, p);
    else
        a  =((w-1)*(sizeTest+1));
        xtest = xdata(a+1:a+sizeTest,:);
        xtrain = xdata(1:a,:);
        xtrain = [xtrain; xdata(a+sizeTest+1:n,:)];
        ytrain = ydata(1:a,:);
        ytrain = [ytrain; ydata(a+sizeTest+1:n,:)];
        %fprintf('test range: %i -%i, train trange: 1-%i, %i-n', a+1,a+sizeTest,a,a+sizeTest+1);
        ypred = knn_classifier(xtest, xtrain, ytrain, k, p);
    end
    %compute statistics here
    %compute accuracy
    count = 0;
    truePositive = 0;
    trueNegative = 0;
    falsePositive = 0;
    falseNegative = 0;
    for i=1:sizeTest
        if(ypred(i,1) == ydata(a+i,1))
            count = count +1;
        end
```

For Cross Validation I randomly shuffled all the rows of the data. Then I partitioned it into 10 equal pieces.

```
        if(ypred(i,1) == 4 && ydata(a+i,1) == 4)
            truePositive = truePositive + 1;
        end
        if(ypred(i,1) == 2 && ydata(a+i,1) == 2)
            trueNegative = trueNegative + 1;
        end
        if(ypred(i,1) == 4 && ydata(a+i,1) == 2)
            falsePositive = falsePositive + 1;
        end
        if(ypred(i,1) == 2 && ydata(a+i,1) == 4)
            falseNegative = falseNegative + 1;
        end

    end
    accuracy(w,1) = count/sizeTest;
    sensitivity(w,1) = truePositive/(truePositive+falseNegative);
    specificity(w,1) = trueNegative/(trueNegative+falsePositive);
    fprintf('fold %i: accuracy - %f, sensitivity - %f, specificity - %f\n',w, accuracy(w,1),sensitivity(w,1),specificity(w,1));
end
fprintf('Mean of accuracy: %f, Std: %f\n', mean(accuracy), std(accuracy));
fprintf('Mean of sensitivity: %f, Std: %f\n', mean(sensitivity), std(sensitivity));
fprintf('Mean of specificity: %f, Std: %f\n', mean(specificity), std(specificity));
```

ommand Window

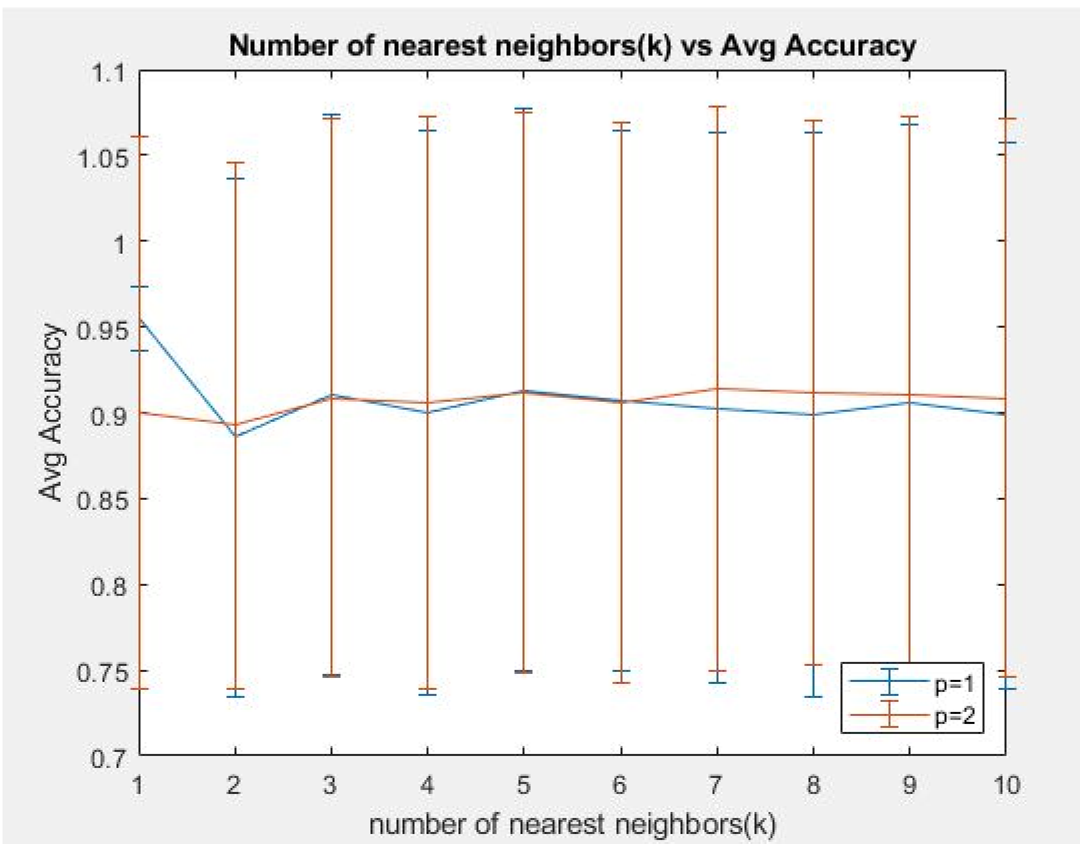
```
fold 1: accuracy - 0.985507, sensitivity - 1.000000, specificity - 0.977778
fold 2: accuracy - 0.942029, sensitivity - 0.789474, specificity - 1.000000
fold 3: accuracy - 0.942029, sensitivity - 0.937500, specificity - 0.945946
fold 4: accuracy - 0.971014, sensitivity - 1.000000, specificity - 0.955556
fold 5: accuracy - 0.956522, sensitivity - 0.962963, specificity - 0.952381
fold 6: accuracy - 0.971014, sensitivity - 0.913043, specificity - 1.000000
fold 7: accuracy - 0.956522, sensitivity - 0.900000, specificity - 0.979592
fold 8: accuracy - 0.956522, sensitivity - 0.913043, specificity - 0.978261
fold 9: accuracy - 0.985507, sensitivity - 0.952381, specificity - 1.000000
fold 10: accuracy - 0.927536, sensitivity - 0.888889, specificity - 0.952381
Mean of accuracy: 0.959420, Std: 0.019081
Mean of sensitivity: 0.925729, Std: 0.061621
Mean of specificity: 0.974189, Std: 0.021482
```

I got around 96% accuracy for cross validation with a std of .019.
The function will iterate between all partitions as the test set. So there is 1 test set and 9 training at each iteration.

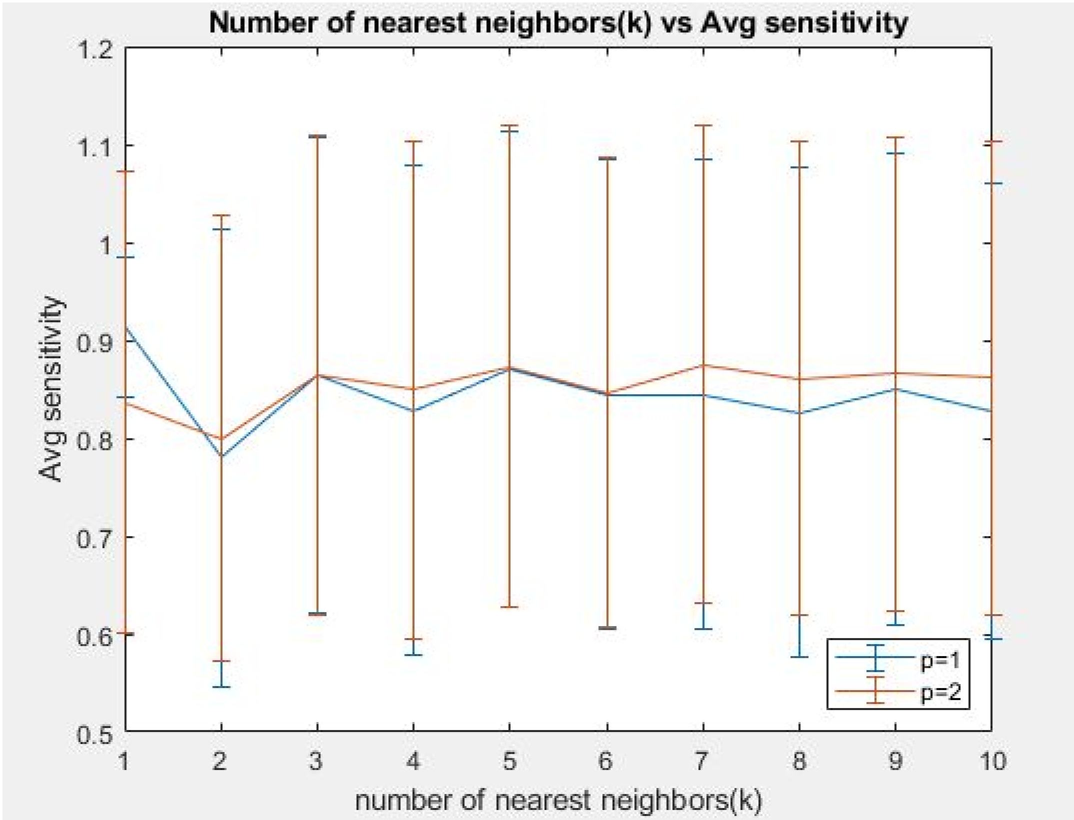
Part2.2:

For k=1:10 and p=1 and p=2 cross validated perfomance I got these metrics(with graphs below).
k:1, p:1
Mean of accuracy: 0.955072, Std: 0.018648
Mean of sensitivity: 0.913699, Std: 0.071672
Mean of specificity: 0.978242, Std: 0.020230
k:1, p:2
Mean of accuracy: 0.900000, Std: 0.160434
Mean of sensitivity: 0.836362, Std: 0.236064
Mean of specificity: 0.932327, Std: 0.132157
k:2, p:1
Mean of accuracy: 0.885507, Std: 0.150992
Mean of sensitivity: 0.780124, Std: 0.233288
Mean of specificity: 0.940881, Std: 0.127053
k:2, p:2
Mean of accuracy: 0.892754, Std: 0.153255
Mean of sensitivity: 0.799852, Std: 0.227857
Mean of specificity: 0.940706, Std: 0.127073
k:3, p:1
Mean of accuracy: 0.910145, Std: 0.163082
Mean of sensitivity: 0.864192, Std: 0.242342
Mean of specificity: 0.934214, Std: 0.132467
k:3, p:2
Mean of accuracy: 0.908696, Std: 0.162688
Mean of sensitivity: 0.864192, Std: 0.244836
Mean of specificity: 0.932131, Std: 0.131477
k:4, p:1
Mean of accuracy: 0.900000, Std: 0.164882
Mean of sensitivity: 0.828537, Std: 0.250379
Mean of specificity: 0.936101, Std: 0.133044
k:4, p:2
Mean of accuracy: 0.905797, Std: 0.166544
Mean of sensitivity: 0.849180, Std: 0.254851
Mean of specificity: 0.934214, Std: 0.132467
k:5, p:1
Mean of accuracy: 0.913043, Std: 0.163824
Mean of sensitivity: 0.869963, Std: 0.242439
Mean of specificity: 0.934214, Std: 0.132467
k:5, p:2
Mean of accuracy: 0.911594, Std: 0.163746
Mean of sensitivity: 0.872954, Std: 0.245974
Mean of specificity: 0.931889, Std: 0.132295
k:6, p:1
Mean of accuracy: 0.907246, Std: 0.157313
Mean of sensitivity: 0.844747, Std: 0.239601
Mean of specificity: 0.938184, Std: 0.126548
k:6, p:2
Mean of accuracy: 0.905797, Std: 0.162717
Mean of sensitivity: 0.846612, Std: 0.240121
Mean of specificity: 0.936101, Std: 0.133044
k:7, p:1
Mean of accuracy: 0.902899, Std: 0.160522
Mean of sensitivity: 0.844747, Std: 0.239601
Mean of specificity: 0.931889, Std: 0.132295
k:7, p:2
Mean of accuracy: 0.914493, Std: 0.164457
Mean of sensitivity: 0.875487, Std: 0.244369
Mean of specificity: 0.933775, Std: 0.132909
k:8, p:1
Mean of accuracy: 0.898551, Std: 0.164535
Mean of sensitivity: 0.826440, Std: 0.249754
Mean of specificity: 0.933775, Std: 0.132909
k:8, p:2
Mean of accuracy: 0.911594, Std: 0.158679

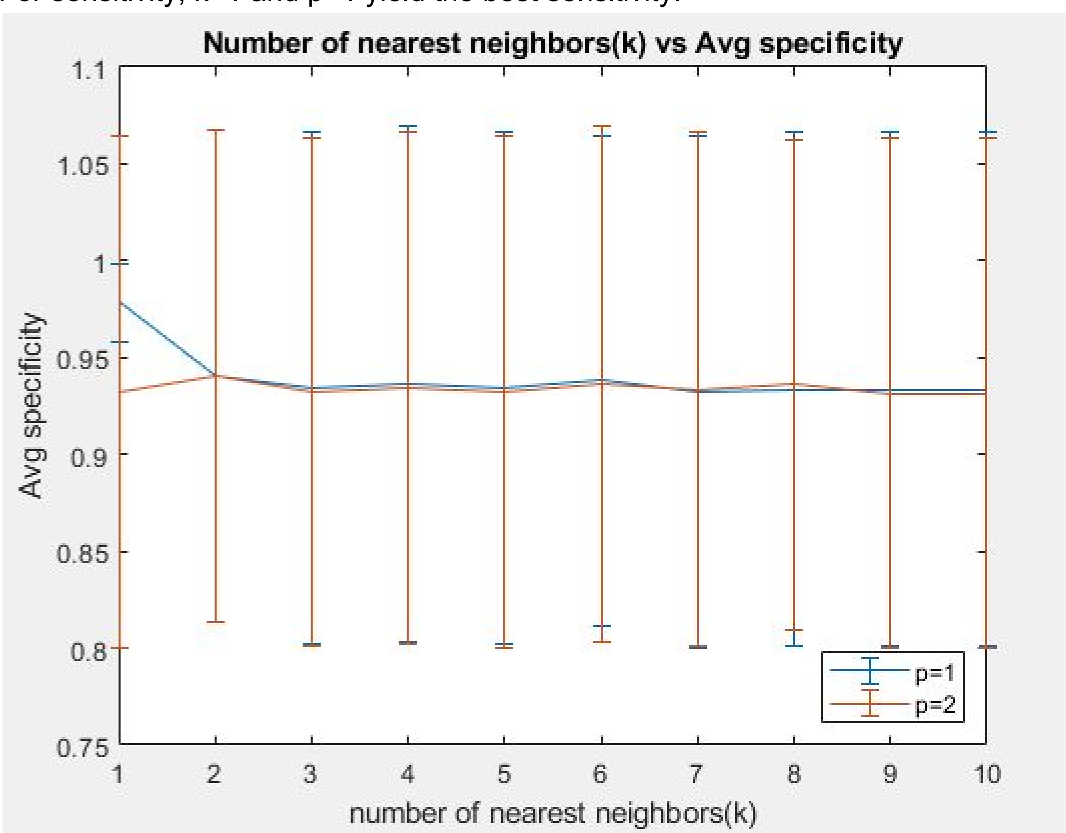
Mean of sensitivity: 0.861201, Std: 0.241471
Mean of specificity: 0.935859, Std: 0.126449
k:9, p:1
Mean of accuracy: 0.905797, Std: 0.162142
Mean of sensitivity: 0.850060, Std: 0.240065
Mean of specificity: 0.933775, Std: 0.132909
k:9, p:2
Mean of accuracy: 0.910145, Std: 0.162939
Mean of sensitivity: 0.865963, Std: 0.241851
Mean of specificity: 0.931692, Std: 0.131916
k:10, p:1
Mean of accuracy: 0.898551, Std: 0.159201
Mean of sensitivity: 0.827202, Std: 0.233207
Mean of specificity: 0.933775, Std: 0.132909
k:10, p:2
Mean of accuracy: 0.908696, Std: 0.162544
Mean of sensitivity: 0.861963, Std: 0.241924
Mean of specificity: 0.931692, Std: 0.131916
.



For Accuracy p=1 and k=1 yield the best accuacy.



For sensitivity, k=1 and p=1 yield the best sensitivity.



For specificity, k=1 and p=1 yield the best specificity

```
for i=1:sizeTest
    if(ypred(i,1) == ydata(a+i,1))
        count = count +1;
    end
    if(ypred(i,1) == 4 && ydata(a+i,1) == 4)
        truePositive = truePositive + 1;
    end
    if(ypred(i,1) == 2 && ydata(a+i,1) == 2)
        trueNegative = trueNegative + 1;
    end
    if(ypred(i,1) == 4 && ydata(a+i,1) == 2)
        falsePositive = falsePositive + 1;
    end
    if(ypred(i,1) == 2 && ydata(a+i,1) == 4)
        falseNegative = falseNegative + 1;
    end

end
accuracy(w,1) = count/sizeTest;
sensitivity(w,1) = truePositive/(truePositive+falseNegative);
specificity(w,1) = trueNegative/(trueNegative+falsePositive);
```

I measured sensitivity by using the formula truePositive/(truePositive+falseNegative)
And I measured specificity by trueNegative/(trueNegative+falsePositive)

I used these rules to figure out which is positive or negative for specificity and senstivity:
positive = 4
negative = 2
true positive = got 4 and actual is 4
false positive = got 4 and actual is 2
true negative = got 2 and actual is 2
false negative = got 2 and actual is 4

Part3.1

.
I used
Train_perceptron function

```
function w = train_perceptron(input_x,input_y, w_init)
    w = w_init;
    numData = size(input_x(:,1),1);
    attributeSize = size(input_x(1,:),2);
    %set learning rate alpha
    alpha = .8;

    %for each data point
    for i=1:numData
            net =  0;
            %compute net
            for j=1:attributeSize
                net = net + input_x(i,j)*w(j,1);
            end
            %compute sigmoid of function
            net = 1.0/(1+exp(-net));
            %calculate error
            change = input_y(i, 1) - net;
            %adjust weight
           for j=1:attributeSize
                w(j,1) = w(j,1) + alpha*change*input_x(i,j);
           end
    end

end
```

For train_perceptron I went through each data point and implemented this formula from the slides:

For each training data point **x**

1. Compute $y = sigmoid(\sum_{i=1}^{n} w_i x_i)$
2. Compute error = $(d_j - y)$
3. “Correct” the weights: $w_i \leftarrow w_i + error^* x_i$

For the error I used gradient descent with an alpha of .8 .

```
function y_pred = classify_perceptron(input_x,w)
%CLASSIFY_PRECEPTRON Summary of this function goes here
%   Detailed explanation goes here
    sizeX = size(input_x(:,1),1);
    y_pred = zeros(sizeX,1);
    for m=1:sizeX
        sum = 0;
        for i=1:size(input_x(1,:),2)
            sum = sum + w(i,1)*input_x(m,i);
        end
        if(sign(sum) <0)
            y_pred(m,1) = 4;
        else
            y_pred(m,1) = 2;
        end
    end
end
```

For classify_perceptron function for each data point I multiplied each weight by the attribute. Then I used sign as the activation function. I defined -1 as ‘4’ class and +1 as ‘2’ class.

Evaluating Perceptron:
First I shuffled the rows to create sufficient diversity between partitions of cross validation.

```
%shuffle rows
data = data(randperm(end),:);
xdata = data(:,(2:10));
ydata = data(:,11);
```

```
    for d=1:numFolds
        %when q=1 initialize w to 0 else to random
        for q=1:2
        %q=1;
        w = zeros(size(xdata(1,:), 2),1);

            %get xtrain and ytrain
            a  =((d-1)*(sizeTest+1));
            xtest = xdata(a+1:a+sizeTest,:);
            xtrain = xdata(1:a,:);
            xtrain = [xtrain; xdata(a+sizeTest+1:n,:)];
            ytrain = ydata(1:a,:);
            ytrain = [ytrain; ydata(a+sizeTest+1:n,:)];

            %convert y values
            y_input = zeros(size(ytrain,1),1);

              for i=1:n-sizeTest
                 if(ytrain(i,1) == 2)
                     y_input(i,1) = 1;
                 else
                     y_input(i,1) = 0;
                 end
              end
```

For each fold I calculated averages based on if I was initializing the weights to zero or initializing the weights to random values. This part obtains the xtrain and ytrain for the current partition. I convert the ytraining data to +1 if it is ‘2’ class or 0 if it is ‘4’ class

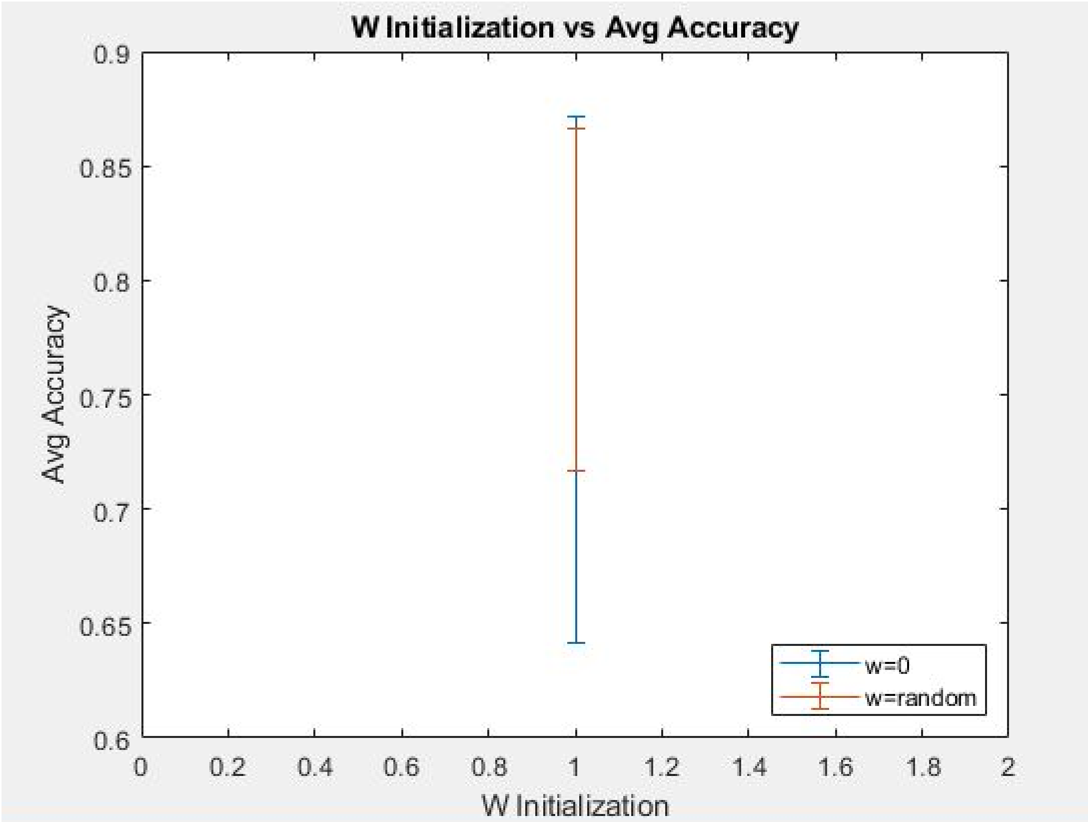
```
            %used for when q=2 so that we can get 10 independent runs of
            %the training so that randomness does not affect it
             avgWeight = zeros(9, 10);
             avgWeight(:,1) = w;
             if(q==2)
                 %initialize w to random values from [-25,25]
                  for u=2:10
                      w = -25 + 50*rand(size(xdata,2),1);
                      avgWeight(:,u) = train_perceptron(xtrain,y_input,w);
                  end
                  for u=1:9
                      actualWeight(u,1) = mean(avgWeight(u,:),2);
                  end
                  w = actualWeight;
             end
            w = train_perceptron(xtrain, y_input, w);


            %compute y label
            ypred = classify_perceptron(xtest,w);
```

If I am initializing the weights to random values then i do 10 iterations of training and find the weight of each iteration then I average it to find the weights to give to the perceptron classifier- this way I can reduce randomness from affecting the results.

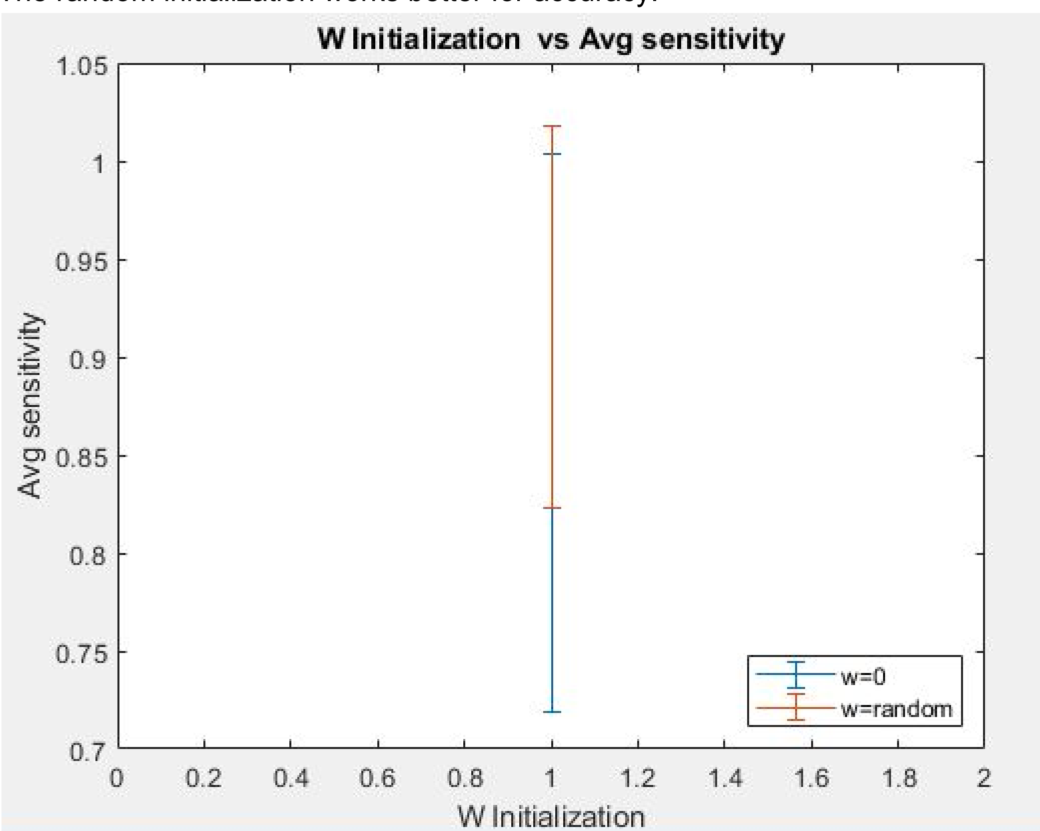
```
        count = 0;
        truePositive = 0;
        trueNegative = 0;
        falsePositive = 0;
        falseNegative = 0;
        for i=1:sizeTest
            if(ypred(i,1) == ydata(a+i,1))
                count = count +1;
            end
            if(ypred(i,1) == 4 && ydata(a+i,1) == 4)
                truePositive = truePositive + 1;
            end
            if(ypred(i,1) == 2 && ydata(a+i,1) == 2)
                trueNegative = trueNegative + 1;
            end
            if(ypred(i,1) == 4 && ydata(a+i,1) == 2)
                falsePositive = falsePositive + 1;
            end
            if(ypred(i,1) == 2 && ydata(a+i,1) == 4)
                falseNegative = falseNegative + 1;
            end

        end
        accuracy(d,q) = count/sizeTest;
        sensitivity(d,q) = truePositive/(truePositive+falseNegative);
        specificity(d,q) = trueNegative/(trueNegative+falsePositive);
```

Then I calculate the accuracy, sensitivity, and specificity of each fold.

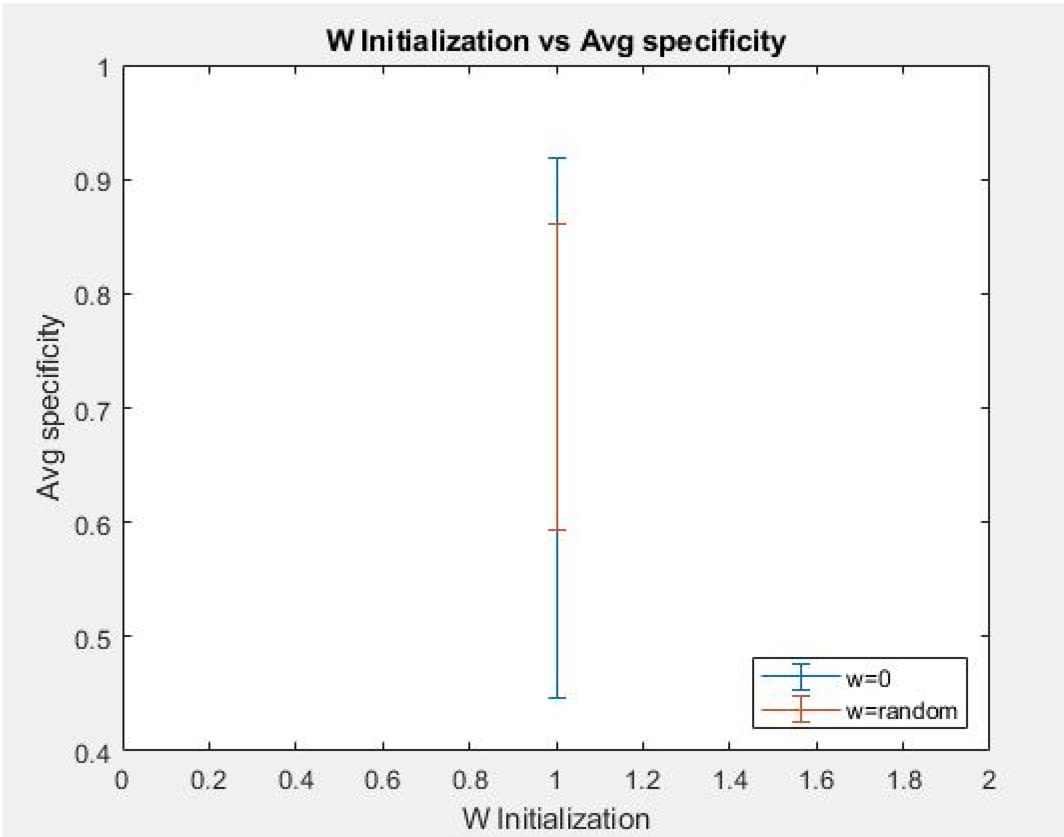
After that I take the average of the performance measures and plot them with respective standard deviations.



The random initialization works better for accuracy.



The random initialization also works better for sensitivity



The random initialization works best for specificity.