EECS 4404 Machine Learning: Assignment 3 Perceptron Algorithm

Christopher Boyd, 216 869 356

Step 1

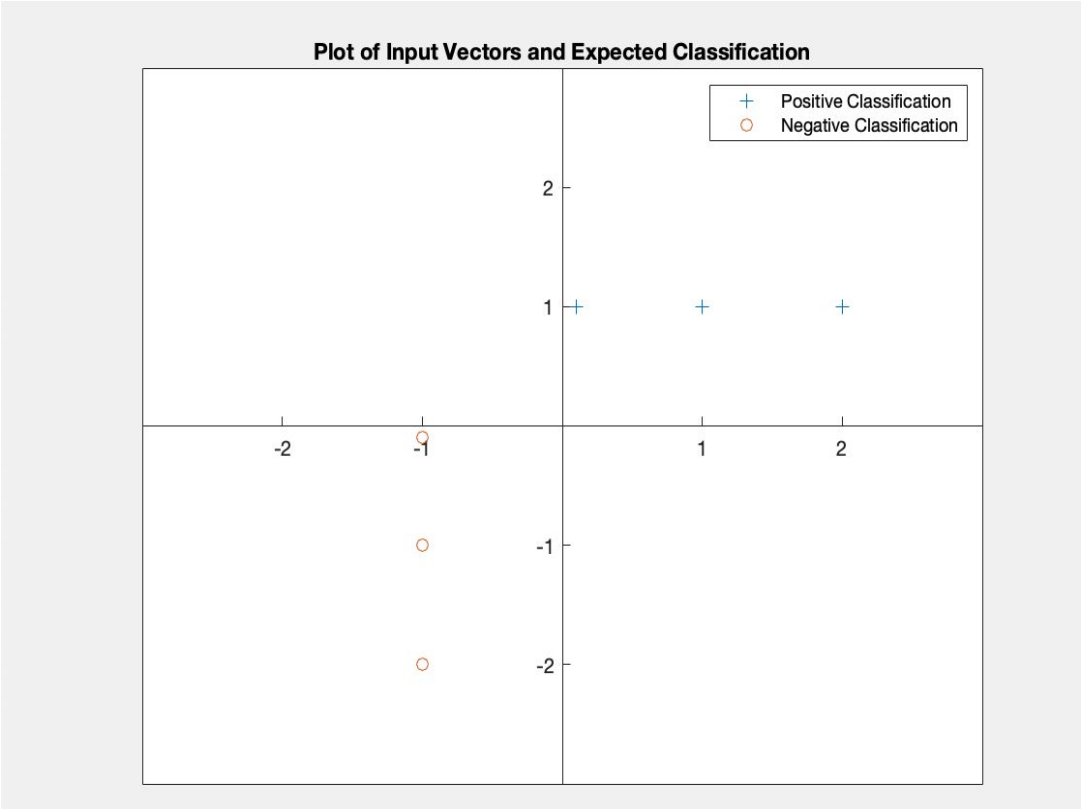

Fig 1: Input Data and Expected Labels

The Perceptron Algorithm does always find a predictor that successfully categorizes all 6 data points. From this, and a visual examination of Fig 1, we can say that the data is linearly separated. However, now that the homogeneous linear classifier is of degree D +1, the algorithm does not always after one update. Most of the time one update is sufficient but in certain scenarios a second update will be required. If the first two rows of the data matrix are alternating positive and negative labels the algorithm will take an additional step to terminate. The algorithm never required a third update to terminate, regardless of the sequence of inputs.

Step 1: Code

```
inputs = load('6pointsinputs.txt');
labels = load('6pointsoutputs.txt');
for j=1:100
    num_updates = prove_perceptron(inputs,labels);
    if num_updates ~= 1
        fprintf('WARNING: more than one update for weight vector\n');
    end
end
function num_updates = prove_perceptron(inputs,labels)
    [rows, cols] = size(inputs);
    %randomize order of inputs and respective labels
    des_matr = zeros(rows, cols+1);
    des_matr(:,1:2) = inputs;
    des_matr(:,3) = labels;
    des_matr = des_matr(randperm(size(des_matr, 1)), :);
    inputs = des_matr(:,1:2);
    labels = des_matr(:,3);
    updates = 0;
    errors = -1;
    weight = zeros(1,2);
    j = 1;
    while j<=100 && errors == -1
        %assume no more training required
        errors = 0;
        for i=1:6
            sign = labels(i) * dot(weight, inputs(i,:));
            if sign <= 0
                weight = weight + labels(i) * inputs(i,:);
                updates = updates + 1;
                %more training is required
                errors = -1;
            end
        end
        j = j+1;
    end
    num_updates = updates;
end
```

Function prove_perceptron does not return anything other than 1 for the number of updates completed, regardless of data point order.

Step 2

Adding an extra column of 1's so that the perceptron algorithm will learn a homogeneous linear system does affect how many updates are required to find a solution. After randomizing the order of data points seen by the algorithm, if the sign of the label of the first two points seen are opposite the algorithm will require two updates to find a successful solution. The algorithm does always find a predictor that classifies all six data points. The correction of the predictor can be seen by examining the exit condition of the while loop shown in the code below. It always exits due to error ~= -1. If there were misclassified points the algorithm would keep updating the weight vector until the exit condition of more than 100 iterations of the loop was reached (variable 'j'). Further, in Step 3, it can be seen that there is no Binary Loss

Step 2: Perceptron Implementation

```
function weight = perceptron(inputs, labels)
   [rows,cols] = size(inputs);
   updates = 0;
   error = -1;
   weight = zeros(1,cols);
   j = 1;
   while j<=100 && error == -1
      %assume no more training required
      error = 0;
      for i=1:rows
         sign = labels(i) * dot(weight, inputs(i,:));
         if sign <= 0
            weight = weight + labels(i) * inputs(i,:);
            updates = updates + 1;
            %more training is required
            error = -1;
         end
      end
      j = j+1;
   end
   weight = weight ./ norm(weight);
end
```

Step 3

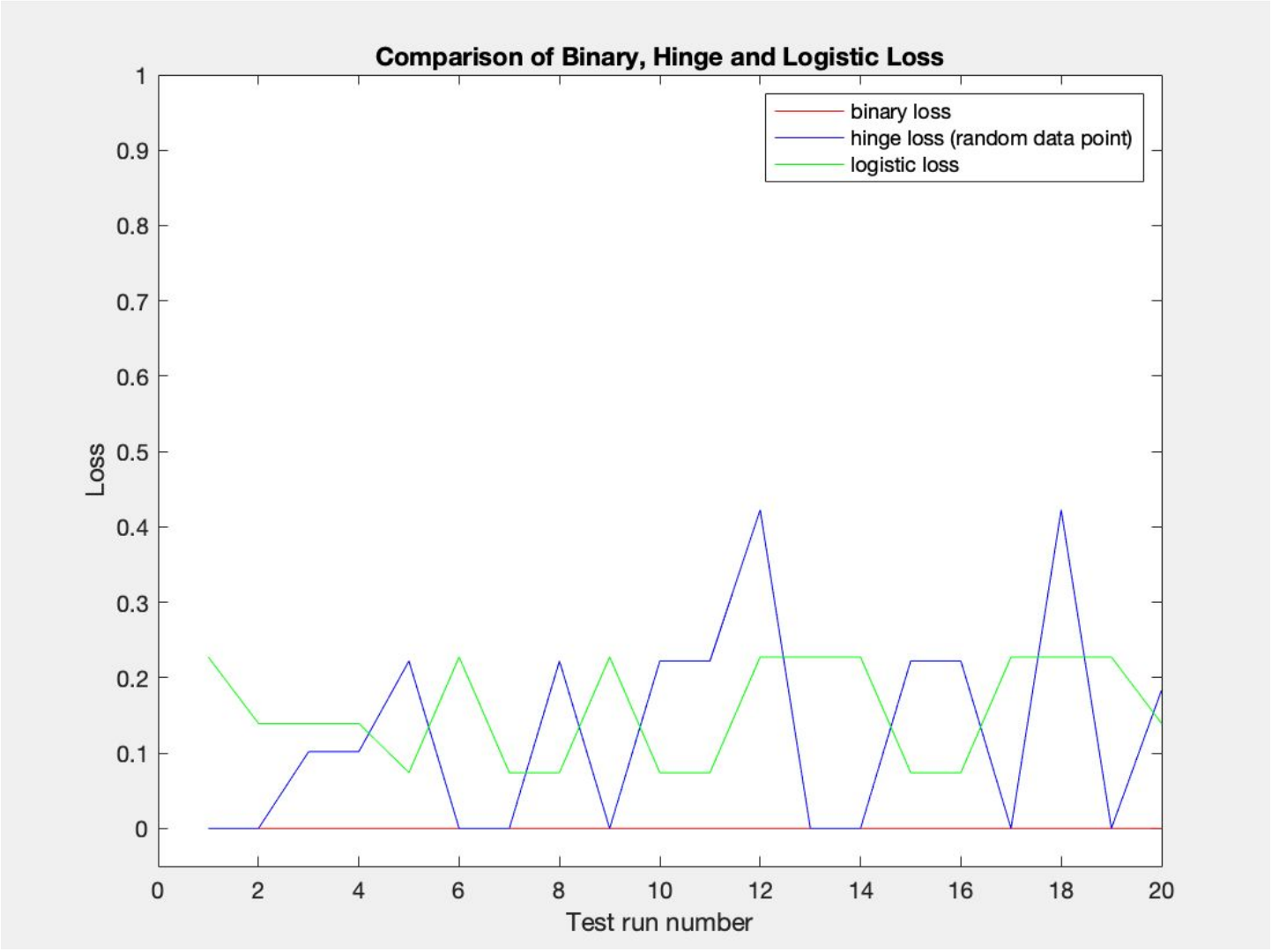

Fig 2. Binary, Hinge and Logistic Loss for fg_inputs/fg_outputs Dataset

As expected, the binary loss is 0 because the 6 points in dataset 1 are linearly separable. The hinge loss is the most variable because it is based off of a single, randomly selected point, for each test run. Whereas, the logistic loss is averaged over the whole dataset.

Both hinge and logistic loss are expected to have variance as they quantify how close the linear classifier is to predicting the exact value output for a given input or set of inputs, respectively. The binary loss is simpler in that it only quantifies correct/incorrect classification.

Step 3: Loss Functions

```
function bloss = binary_loss(weight, data, labels)
%compute average binary loss for given weight vector over all
%inputs and respective labels
bloss = 0;
for i=1:size(labels)
    sign = dot(weight,data(i,:)) * labels(i);
    sign = sign / abs(sign);
    bloss = bloss + (.5 * sign - .5);
end

bloss = bloss / i;
end
```

```
function hloss = hinge_loss(weight, data, labels)
%hinge loss = max(0, 1 - t<w,x>)
hloss = max(0,1 - labels * dot(weight,data));

end
```

```
function logloss = logistic_loss(weight, data, labels)
%logistic loss is average hinge loss over all data points
logloss = 0;
for i=1:size(labels)
    loss = 1 - labels(i) * dot(weight, data(i,:));
```

```
    loss = max(0, loss);
    logloss = logloss + loss;
end
logloss = logloss / i;
end
```

Step 3: Test Code

```
inputs = load('6pointsinputs.txt');
labels = load('6pointsoutputs.txt');
[rows, cols] = size(inputs);
data_matrix = zeros(rows, cols+1);

test_runs = 20;
weights = zeros(test_runs, cols+1);
bloss = zeros(1, test_runs);
hloss = zeros(1, test_runs);
logloss = zeros(1, test_runs);
for i=1:test_runs
    %randomize order of inputs and respective labels
    data_matrix(:,1:cols) = inputs;
    data_matrix(:,end) = labels;
    data_matrix = data_matrix(randperm(size(data_matrix, 1)), :);
    data_labels = data_matrix(:,end);
    data_matrix(:,end) = 1;

    %train the perceptron
    weights(i,:) = perceptron(data_matrix,data_labels);
    bloss(i) = binary_loss(weights(i,:),data_matrix,data_labels);
    %randomly select a point from data_matrix and data_labels to find
    %hinge error for
    hpoint = randi(rows);
    hloss(i) =
hinge_loss(weights(i,:),data_matrix(hpoint,:),data_labels(hpoint,:));
    logloss(i) = logistic_loss(weights(i,:),data_matrix,data_labels);
end

figure;
plot([1:1:test_runs],bloss,'r');
hold on;
plot([1:1:test_runs],hloss,'b');
plot([1:1:test_runs],logloss,'g');
legend('binary loss','hinge loss (random data point)','logistic loss');
title('Comparison of Binary, Hinge and Logistic Loss');
ylabel('Loss');
xlabel('Test run number');
ylim([-0.05,1]);
```

Step 4

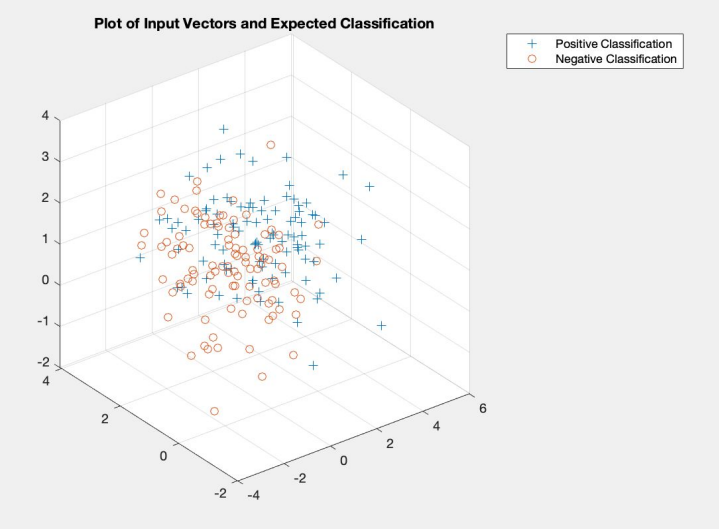

Fig 3: Visualization of fg_inputs and fg_outputs Classifications

From the plot, it can be seen that the data is not linearly separable. A higher degree of error from the perceptron algorithm is expected.

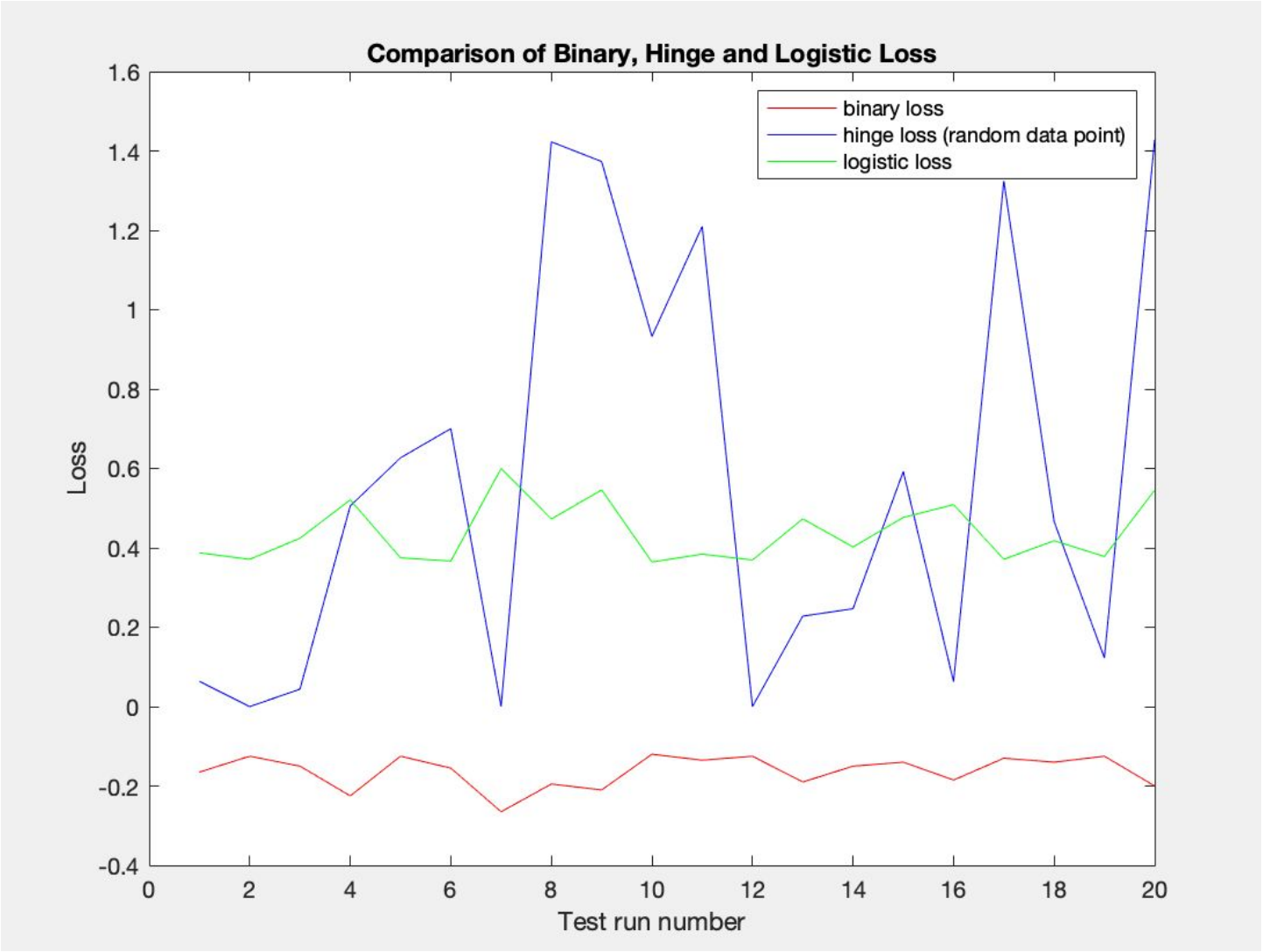

Fig 4: Comparison of Binary, Hinge and Logistic Loss over 20 tests

As predicted, the rate of loss is higher than in the linearly separable dataset from earlier. The negative binary loss comes from, it appears, that the negative classification points are more likely to be incorrectly identified due to their being intermixed with positive classification points. Again, the hinge loss has the most variability relative to both the binary and logistic loss. Though this does vary from one set of 20 tests to the next, depending on the randomly selected point to test.

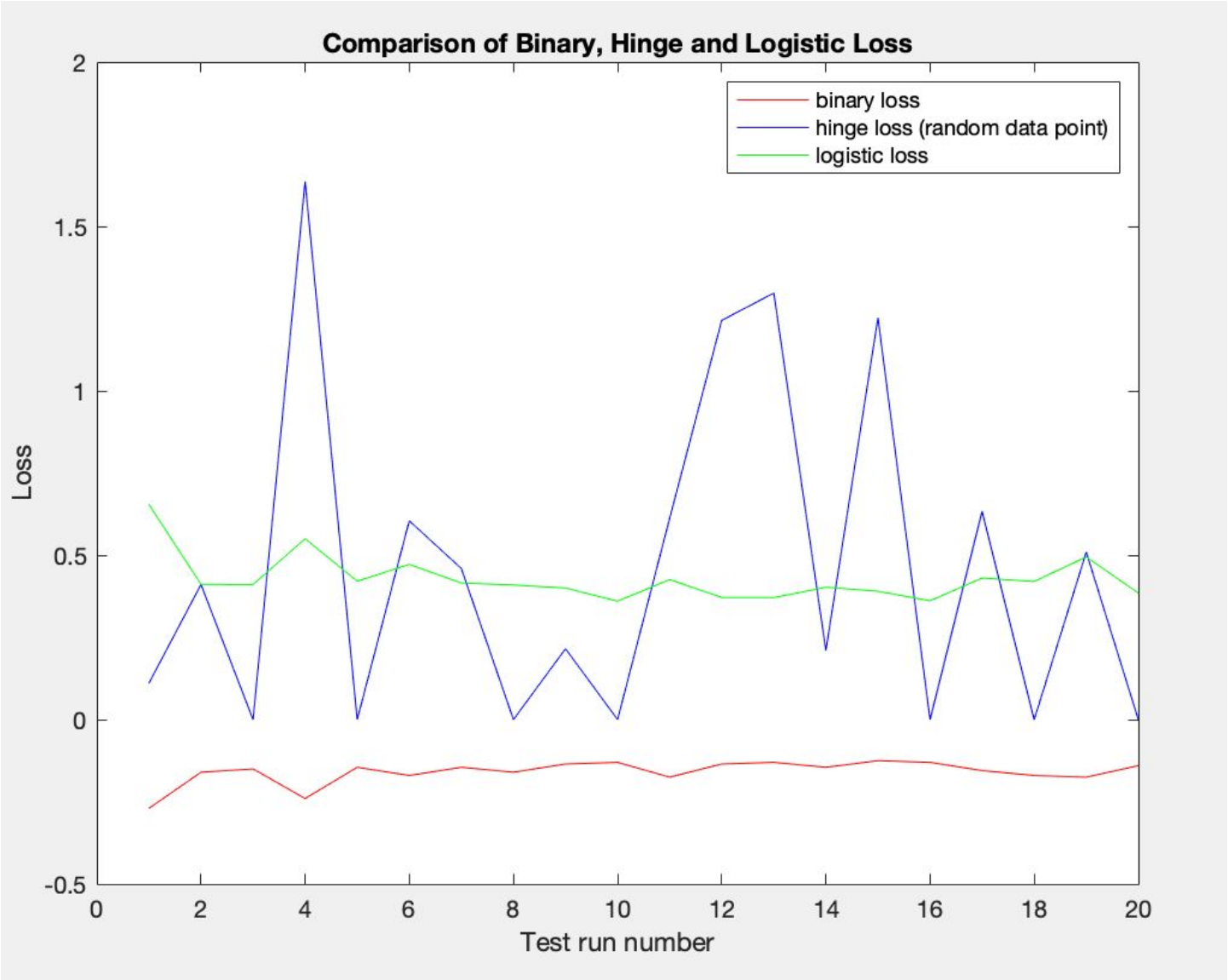

Fig 5: Second set of 20 tests showing variability of Binary, Hinge and Logistic Loss

Here is another set of 20 tests that shows the hinge loss having similar variability, while the logistic and binary losses stay within the same overall ranges. This is as expected due to the errors averaging out across many data points over each test run.

Sample minimum losses and the respective weight vectors:
Note 1: Offset is last parameter of vector
Note 2: These numbers correlate to Fig 5.

Binary Loss
Minimum = -0.1250
Minimum Test Number = 15
Minimum Loss Weight Vector = 0.5807 0.1427 0.2806 -0.7508

Hinge Loss
Minimum = 0
Minimum Test Number = 3
Minimum Loss Weight Vector = 0.5945 0.0358 0.3081 -0.7419

Logistic Loss
Minimum = 0.3602
Minimum Test Number = 10
Minimum Loss Weight Vector = 0.7244 0.1852 0.2995 -0.5927

The use of various surrogate loss functions is important for each classifier. The decision of which loss function to favour can have some potentially powerful impacts on the results of model selection. If it is determined that it is more important to get the highest number of points classified correctly then choosing the model that minimizes the binary loss would be ideal. However, if it is more important to find a classifier that is close to correct, even if the classifier is incorrect more often, one would select the model that minimizes the logistic loss.

Step 5
Classify Iris Setosa versus Iris Versicolour and Iris Virginica

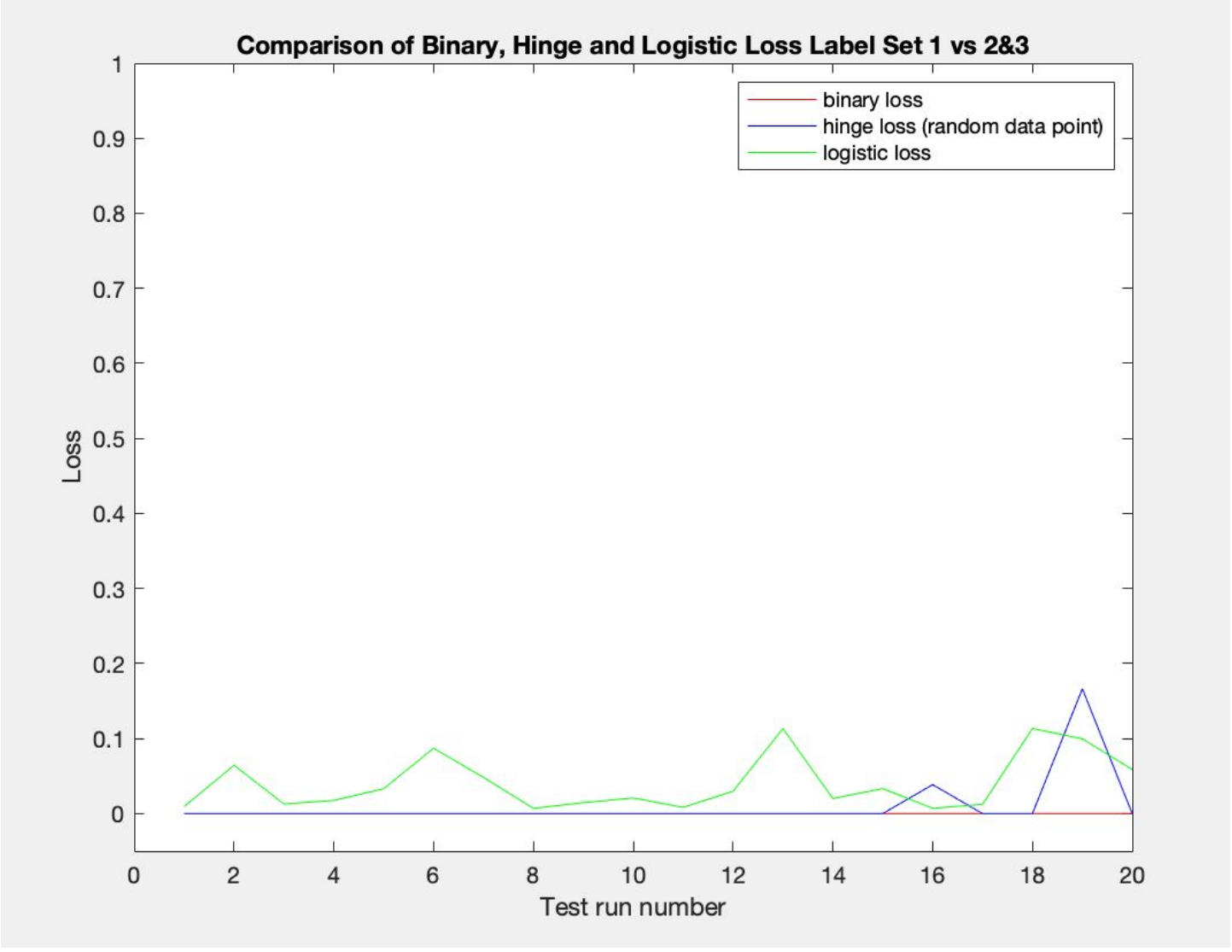

Fig 6: Classify Iris Setosa Versus Not Iris Setosa

Binary Loss
Minimum = 0
Minimum Test Number = 1
Minimum Loss Weight Vector = 0.2412 0.3804 -0.7886 -0.4082 0.0928

Hinge Loss
Minimum = 0
Minimum Test Number = 1
Minimum Loss Weight Vector = 0.2412 0.3804 -0.7886 -0.4082 0.0928

Logistic Loss
Minimum = 0.0070
Minimum Test Number = 8
Minimum Loss Weight Vector = 0.1414 0.5019 -0.7563 -0.3888 0.0707

Classify Iris Versicolour versus Iris Setosa and Iris Virginica

Fig 7: Iris Versicolour Versus Not Iris Versicolour

Binary Loss
Minimum = -0.2267
Minimum Test Number = 8
Minimum Loss Weight Vector = 0.1333   -0.5018   0.1709   -0.5457   0.6351

Hinge Loss
Minimum = 0
Minimum Test Number = 3
Minimum Loss Weight Vector = 0.0262   -0.4422   0.2662   -0.6480   0.5595

Logistic Loss
Minimum = 0.7139
Minimum Test Number = 20
Minimum Loss Weight Vector =   0.1369   -0.5594   0.0378   -0.4453   0.6845

Classify Iris Virginica versus Iris Setosa and Iris Versicolour

Fig 8: Classify Iris Virginica Versus Not Iris Virginica

Binary Loss
Minimum = -0.0133
Minimum Test Number = 3
Minimum Loss Weight Vector = -0.3379 -0.4218 0.5265 0.6000 -0.2661

Hinge Loss
Minimum = 0
Minimum Test Number = 20
Minimum Loss Weight Vector = -0.3425 -0.3566 0.5079 0.6219 -0.3329

Logistic Loss
Minimum = 0.3909
Minimum Test Number = 2
Minimum Loss Weight Vector = -0.4305 -0.2664 0.5817 0.5957 -0.2248

Step 5: Analysis

From Fig 6, it can be seen that the Iris Setosa is linearly separable from the set of Iris Versicolour and Iris Virginica. It is when trying to classify both the Iris Versicolour and Iris Virginica data that we lose the linear separability of the Iris Setosa. Of the two, the Iris Virginica can be more accurately predicted with a minimum binary loss of -0.0133 versus the Iris Versicolour's minimum binary loss of -0.2267.

One technique to develop the three one-versus-all predictor into a muti-class predictor for a single new, unlabeled data point, would be to use the predictor models for each of the Iris Setosa, Iris Versicolour and Iris Virginica for the new data point. Then calculate the hinge loss for each flower type and select the result with the minimal loss.

Step 5: Code

```
inputs = load('irisnum.txt');

[rows, cols] = size(inputs);
test_runs = 20;
weights = zeros(test_runs, cols);
bloss = zeros(1, test_runs);
hloss = zeros(1, test_runs);
logloss = zeros(1, test_runs);

%label1 versus label2 and label3
for i=1:test_runs
   %randomize order
   data_matrix = inputs;
   data_matrix = data_matrix(randperm(size(data_matrix, 1)), :);
   data_labels = data_matrix(:,end);
   data_labels(data_labels(:,end)==1) = -1;
   data_labels(data_labels(:,end)==2) = -1;
   data_labels(data_labels(:,end)==3) = 1;
   data_matrix(:,end) = 1;

   %train the perceptron
   weights(i,:) = perceptron(data_matrix,data_labels);
   bloss(i) = binary_loss(weights(i,:),data_matrix,data_labels);
   %randomly select a point from data_matrix and data_labels to find
   %hinge error for
   hpoint = randi(rows);
   hloss(i) =
hinge_loss(weights(i,:),data_matrix(hpoint,:),data_labels(hpoint,:));
   logloss(i) = logistic_loss(weights(i,:),data_matrix,data_labels);
end
%get minimum loss weight vector for each type of loss
[b_min, b_i] = max(bloss)
weights(b_i,:)
[h_min, h_i] = min(hloss)
weights(h_i,:)
[l_min, l_i] = min(logloss)
weights(l_i,:)

figure;
plot([1:1:test_runs],bloss,'r');
hold on;
plot([1:1:test_runs],hloss,'b');
plot([1:1:test_runs],logloss,'g');
legend('binary loss','hinge loss (random data point)','logistic loss');
title('Comparison of Binary, Hinge and Logistic Loss Label Set 1 vs 2&3');
ylabel('Loss');
xlabel('Test run number');
```