HW2 solution

1. K-NN

Sample Matlab code for getting leave-one-out error of KNN for a given list of k values:

```matlab
function err=knn_loo_error(traindata,ks)
%data: contains the data matrix, labels are stored as the first column, other columns are features
%ks: a list k values to consider for knn

Y=data(:,1); % retrieve labels of all training examples
X=data(:, 2:end); %retrieve feature vectors of all training examples
for i=1:length(Y) %for each example
    differences = X-repmat(X(i,:), length(Y),1); %compute difference from example i
    distances(i,:) = sum(differences.^2,2)'; %calculate norm^2 of difference vectors
    distances(i, i) = max(distances(i,:))+1; %for loo, set distance to self to super-large
    [ordered_distance, ordered_list]=sort(distances(i,:)); %sort distances
    for j=1:length(ks) %for each possible k value
        k=ks(j);
        topys = Y(ordered_list(1:k)); % grab all the votes
        if sum(topys)>0 predicted(i,j) = 1; % majority vote
        else    predicted(i,j)=-1;
        end
    end
    end
for j=1:length(ks)
    err(j)=sum(predicted(:,j)~=Y)/length(Y); %measure error rate
end
```

Sample matlab code for computing training and testing errors of different k values for KNN

```matlab
function err=knn_error(traindata,testdata, ks)
%data: contains the data matrix, labels are stored as the first column, other columns are features
%ks: a list k values to consider for knn

Y=traindata(:,1);%grab the labels of training data
X=traindata(:, 2:end); %grab the feature vectors of training data
Yt=testdata(:,1);%grab the labels of test data
Xt=testdata(:,2:end); %grab the feature vectors of test data
for i=1:length(Yt) %for each test example
    differences = X-repmat(Xt(i,:),size(X,1),1); %compute difference from training examples to test example i
    distances= sum(differences.^2,2)'; %calculate norm^2 of the difference vectors
    [ordered_distance, ordered_list]=sort(distances); % sort the distances
    for j=1:length(ks) %for each possible k value
        k=ks(j);
        topys = Y(ordered_list(1:k)); % grab all the votes
        if sum(topys)>0 predicted(i,j) = 1; % majority vote
        else    predicted(i,j)=-1;
        end
    end
    end
for j=1:length(ks)
    err(j)=sum(predicted(:,j)~=Yt)/length(Yt); %measure error rate
end
```
When calling the second function, if we specify the testdata to be the same as the training data, we get the training error. Using a separate testdata, we get the testing error. I tested k values 1,3, ..., 29.

The errors that we get for these k values are (first row is training, the second row is loo, and the last row is testing):

\[
\begin{array}{cccccccccccc}
0 & 0.0423 & 0.0317 & 0.0387 & 0.0387 & 0.0423 & 0.0599 & 0.0599 & 0.0563 & 0.0704 & 0.0704 & 0.0704 \\
0.0704 & 0.0704 & 0.0704 & 0.0739 & \\
0.0704 & 0.0528 & 0.0423 & 0.0493 & 0.0458 & 0.0599 & 0.0669 & 0.0669 & 0.0704 & 0.0739 & 0.0739 & 0.0704 \\
0.0704 & 0.0704 & 0.0775 & 0.0810 & \\
0.1021 & 0.0915 & 0.0775 & 0.0739 & 0.0704 & 0.0880 & 0.0915 & 0.0986 & 0.1021 & 0.1056 & 0.1056 & 0.1021 \\
0.1021 & 0.1162 & 0.1232 & 0.1232 & \\
\end{array}
\]

Based on the LOO error, we would choose k=5.

The following figure plots the different errors:

From the figure, we can observe:

1. Training error generally increases with increasing k. It is usually lower than LOO and testing error.
2. LOO error resembles the test error closely (not in terms of the exact values, but the trend of changing as we change K), indicating that using LOO error gives us somewhat reliable choice of K that work reasonably well for testing.
3. The difference between LOO and training error become much smaller when we increase K. This is expected as the set of points these two methods use to make predictions only differ by one point. As we increase K, the influence of this one point difference becomes neglectable.
Decision tree

This is the full decision tree learned from the training data.

The decision stump will have the same root node and if $x_6 = 1$ then predict 1, and otherwise predict 0.

The error rate of the decision stump is 13.67% for training data, and 18.35% for testing data.

The error rate of the full decision tree is 0 for the training data, and 10.07%

Here is a compact decision tree that represents the class concept:

The top-down greedy algorithm will not likely learn such a tree because it would require looking two decisions ahead to determine that testing head-shape then body-shape jointly determines the class label. Just like the data we have here, given a random set of training data, we will likely decide that head-shape or body-shape alone does not provide much information gain, and select other features to test instead.