k-NN classification

Andreas C. Kapourani

(Credit: Hiroshi Shimodaira)

09 February 2018

1 Background

In classification, the data consist of a *training* set and a *test* set. The training set is a set of N feature vectors and their class labels; and a learning algorithm is used to train a classifier using the training set. The test set is a set of feature vectors to which the classifier must assign labels.

An intuitive way to decide how to classify an unlabelled test item is to look at the training data points nearby, and make the classification according to the classes of those nearby labelled data points. This intuition is formalised in a classification approach called *K-nearest neighbour (k-NN)* classification. The k-NN approach looks at the K points in the training set that are closest to the test point; the test point is then classified according to the class to which the majority of the K-nearest neighbours belong.

The training of the K-NN classifier is simple, we just need to store all the training set! However, testing is much slower, since it involves measuring the distance between each test point and every training point. We can write the k-NN algorithm precisely as follows, where X is the training data set with class labels so that $X = \{(\mathbf{x}, c)\}$, Z is the test set, there are C possible classes, and r is the distance metric (typically the Euclidean distance):

- For each test example $z \in Z$:
  - Compute the distance $r(\mathbf{z}, \mathbf{x})$ between $\mathbf{z}$ and each training example $(\mathbf{x}, c) \in X$
  - Select $U_k(\mathbf{z}) \subseteq X$, the set of the k nearest training examples to $\mathbf{z}$.
  - Assign test point $\mathbf{z}$ to class c, where the majority of the K-nearest neighbours belongs.

To illustrate how the k-NN algorithm works, we will use the *Lemon-Orange* dataset described in the lecture slides. Since we do not have class labels for this data set, we will use the output of the K-means algorithm as ground truth labels for the training data; then, based on this training set we will run k-NN algorithm to classify new test points.

Exercise

- Download the `lemon-orange.txt` file from the course website, and load the data in MATLAB. Extract the data in matrix `A` and the metadata (i.e. labels) in variable `col_headers`.
- Run MATLAB's K-means algorithm for K = 5 clusters and plot the data together with the cluster means. The result should look like Figure 4. **Note**: Before running k-means type `rng(2)`. Check what `rng` command does.

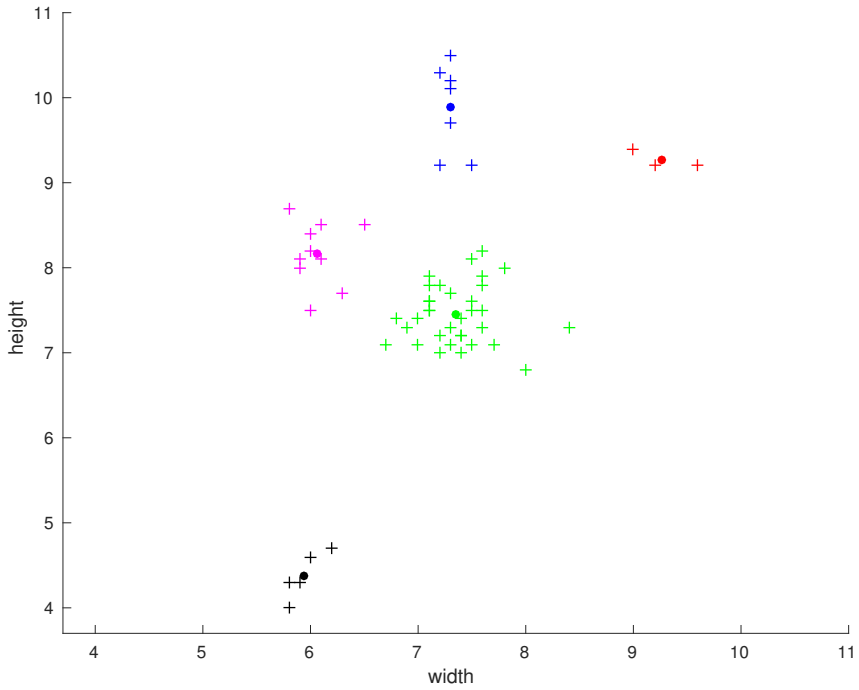


Figure 1: *Lemon-Orange dataset after running k-means algorithm for K = 5.*

Based on k-means output, we will assign each data point to that specific class using the following code.

```
% Orange-lemons data
training_data = A;
% set seed for repeatable results
rng(2);
% Number of clusters
K = 5;
% Corresponding class labels for each training point
C = kmeans(training_data, K); % we use Matlab's built in function
% Concatenate labels to the training data (type: help cat)
training_data = cat(2, training_data, C);
```

We can create some random test data using the following code:

```
% Random test data
test_data = [6.5 7.6;
       7.5 8.7;
       8.5 9];
% show test data on the same plot with the training data
plot(training_data(:,1), training_data(:,2), '+'); hold on;
xlabel(col_headers{1}); ylabel(col_headers{2});

% show test data as red circles
plot(test_data(:,1), test_data(:,2), 'ro');
axis([3.7 11 3.7 11]);
hold off;
```

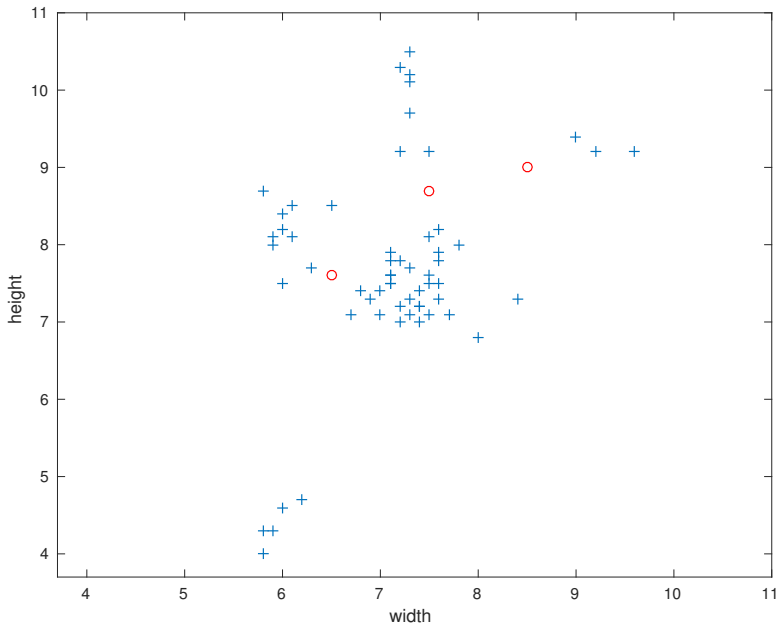


Figure 2: *Lemon-Orange training dataset together with some random test data shown in red circles.*

2 k-NN algorithm

Now we can implement k-NN algorithm for different K and observe on which class each test point will be assigned. The following code illustrates the k-NN algorithm for the first test point:

```
% distance between first test point and each training observation
% NOTE: we need the 'square_dist' function from previous labs
r_zx = square_dist(A, test_data(1,:));
% Sort the distances in ascending order
[r_zx,idx] = sort(r_zx, 2, 'ascend');

Knn = 3;              % K nearest neighbors, e.g. Knn = 3
r_zx = r_zx(1:Knn); % keep the first 'Knn' distances
idx = idx(1:Knn);    % keep the first 'Knn' indexes

% majority vote only on those 'Knn' indexes
prediction = mode(C(idx))
% class label 2, belongs to the green colour
prediction =
     2
```

Exercise

- For the same test point, use different number of k-nearest neighbours and check if the class label changes.

- Write a function `simpleKnn` which will implement a simple k-NN algorithm, similar to what was shown in the lab session.
- Classify the other two test samples using your k-NN algorithm for K-nearest neighbours = 1, 4, 6, 10.

2.1 Plot decision boundary

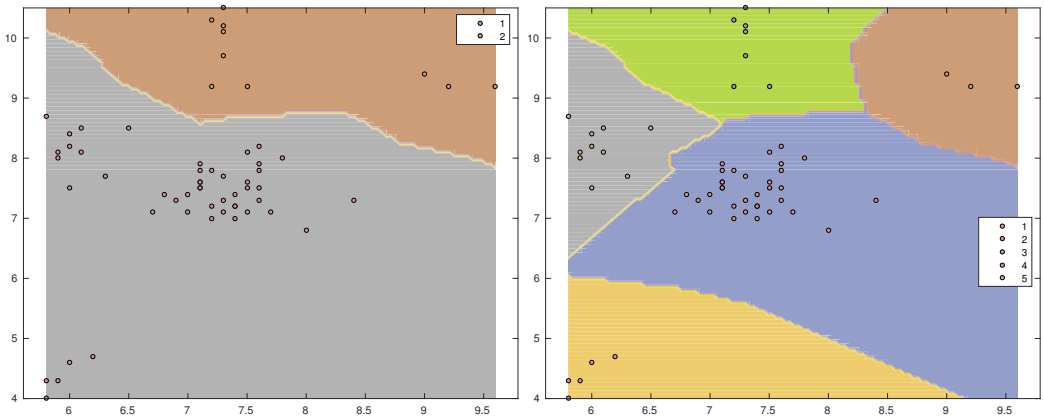
We can draw a line such that one side of it corresponds to one class and the other side to the other. Such a line is called a *decision boundary*. In the case where we need to classify more than two classes a more complex decision boundary will be created.

For the Lemon-Orange dataset, we will create decision boundaries for 1-nearest neighbour using K = 2 and K = 5 classes.

```
%  Colormap we will use to colour each classes.
cmap = [0.80369089,  0.61814689,  0.46674357;
        0.81411766,  0.58274512,  0.54901962;
        0.58339103,  0.62000771,  0.79337179;
        0.83529413,  0.5584314 ,  0.77098041;
        0.77493273,  0.69831605,  0.54108421;
        0.72078433,  0.84784315,  0.30039217;
        0.96988851,  0.85064207,  0.19683199;
        0.93882353,  0.80156864,  0.4219608 ;
        0.83652442,  0.74771243,  0.61853136;
        0.7019608 ,  0.7019608 ,  0.7019608];
rng(2);     % Set seed
Knn = 1;    % K- nearest neighbours
K = 2;      % Number of classes
C = kmeans(A, K); % Class labels for each training point

Xplot = linspace(min(A(:,1)), max(A(:,1)), 100)';
Yplot = linspace(min(A(:,2)), max(A(:,2)), 100)';
% Obtain the grid vectors for the two dimensions
[Xv Yv] = meshgrid(Xplot, Yplot);
gridX = [Xv(:), Yv(:)]; % Concatenate to get a 2-D point.

classes = length(Xv(:));
for i = 1:length(gridX)  % Apply k-NN for each test point
    dists = square_dist(A, gridX(i, :))'; % Compute distances
    [d I] = sort(dists, 'ascend');
    classes(i) = mode(C(I(1:Knn)));
end

figure;
% This function will draw the decision boundaries
[CC,h] = contourf(Xplot(:), Yplot(:), reshape(classes, length(Xplot
   ), length(Yplot)));
set(h,'LineColor','none');
colormap(cmap); hold on;
```

```
% Plot the scatter plots grouped by their classes
scatters = gscatter(A(:,1), A(:,2), C, [0,0,0], 'o', 4);
% Fill in the color of each point according to the class labels.
for n = 1:length(scatters)
    set(scatters(n), 'MarkerFaceColor', cmap(n,:));
end
```

Running the above code for 2 and 5 classes with `kNN = 1` we obtain Figure 3.



(a) (b)

Figure 3: *Decision boundaries for (a) C = 2 and (b) C = 5 using kNN = 1.*

Exercise

Show the decision boundaries using `kNN` = 1, 2, 5, 10, when we have two clusters.

3 Optional extension

To practice more on k-NN you can use the `Iris` data set (https://archive.ics.uci.edu/ml/datasets/iris) which we used in Lab 3. The `Iris` data set consists of three different classes and the goal would be to perform prediction on new test data.

Using this dataset:

- First apply PCA to perform dimensionality reduction and keep the first two principal components.
- Randomly remove 5 observations and keep them as test set.
- Run k-NN on the remaining training set using C = 3 classes and show the decision boundaries using kNN = 1, 5, 10.
- Classify the test set data with the same settings as above.

Note that these are optional questions and no feedback will be provided, mainly are provided so you get familiar with the new methods introduced in the lectures and be able to explore new data using machine learning / statistical models.