k-Nearest Neighbors and Decision Trees

Machine Learning - Prof. Dr. Stephan Günnemann

Leonardo Freiherr von Lerchenfeld

October 29, 2017

Contents

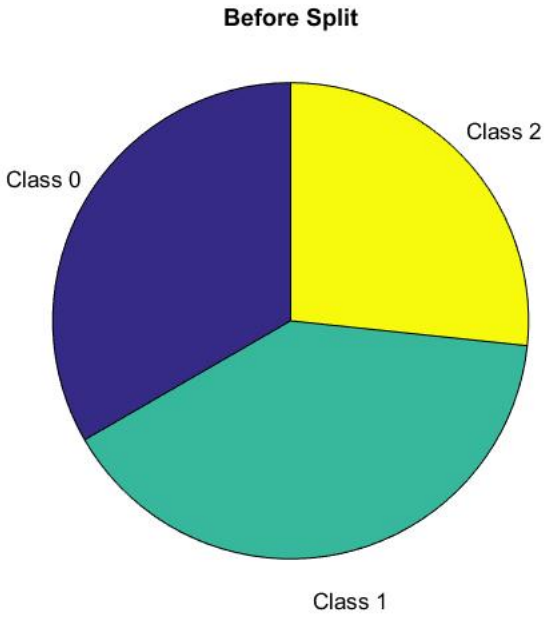
1 Decision Trees

1.1 Dataset

The table below gives you a feature matrix X together with the targets y, where each row i represents one sample. This data is also available on Piazza as 01_*homework_dataset.csv*. The column with names for each datapoint i can help you reference specific data points.

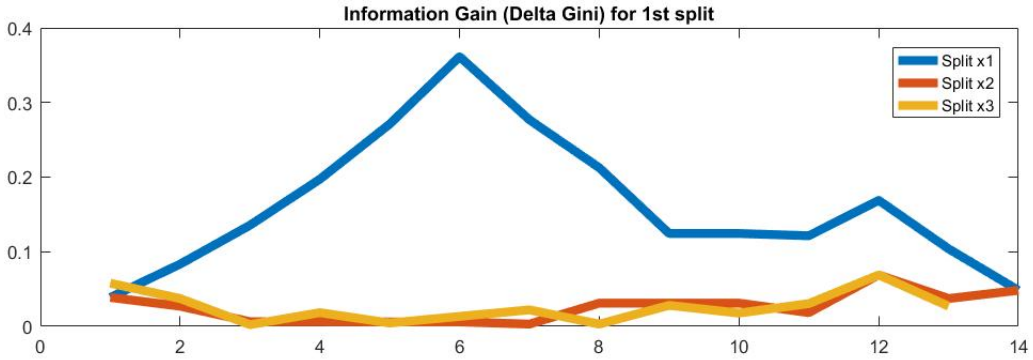
i	$x_{i,1}$	$x_{i,2}$	$x_{i,3}$	y_i
A	5.5	0.5	4.5	2
B	7.4	1.1	3.6	0
C	5.9	0.2	3.4	2
D	9.9	0.1	0.8	0
E	6.9	-0.1	0.6	2
F	6.8	-0.3	5.1	2
G	4.1	0.3	5.1	1
H	1.3	-0.2	1.8	1
I	4.5	0.4	2.0	0
J	0.5	0.0	2.3	1
K	5.9	-0.1	4.4	0
L	9.3	-0.2	3.2	0
M	1.0	0.1	2.8	1
N	0.4	0.1	4.3	1
O	2.7	-0.5	4.2	1

In class 0 are 5 samples, in class 1 are 6 samples and in class 2 are 4 samples.
At the beginning the distribution is as follows



1.2 Results

So the Gini-Index can be calculated as follows
$\Rightarrow i_G(t) = 1 - \frac{5}{15}^2 - \frac{6}{15}^2 - \frac{4}{15}^2 \approx 0.6578$
With Δi_G the maximum information gain can be determined

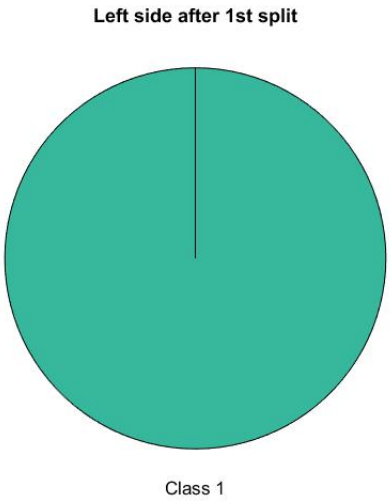


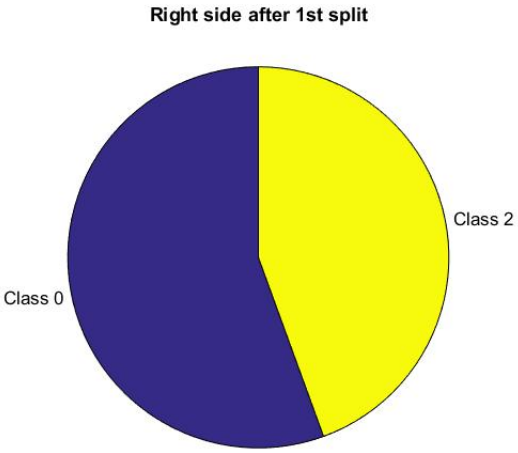
According to this Figure, the most information is won, if the x1 Axis is split between the 6. and 7. lowest value. To determine the value in between of two values, I add 0.05 (which is my chosen tolerance that can distinguish all the values in the dataset) to every value of the dataset. So here the boundary is 4.15. **All the values that are under this boundary will be on the left side after the split.**

So the next step is to divide the dataset according to

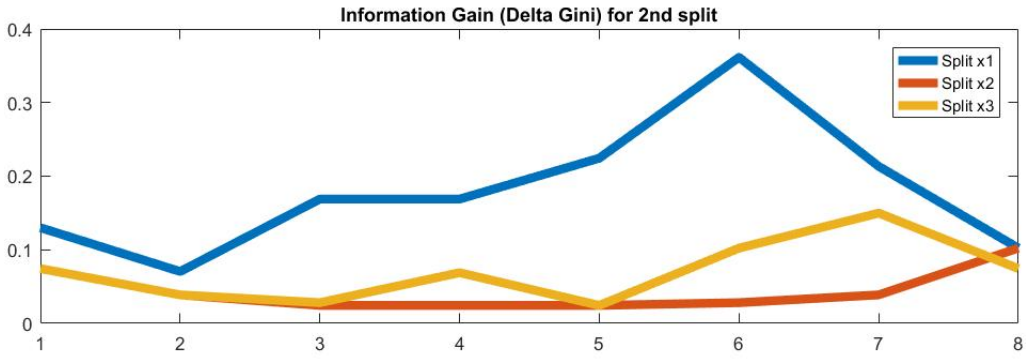
$$x_1 < 4.15$$

After this division, the left side respectively the right of the decision tree have these distributions

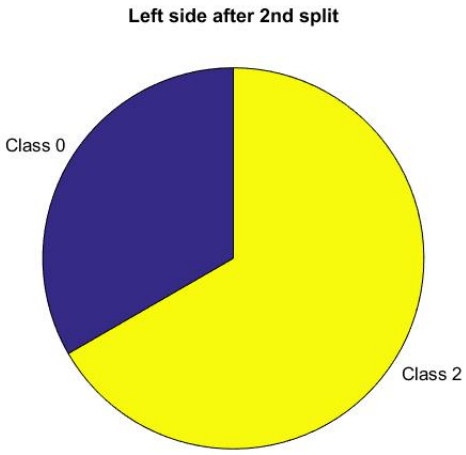


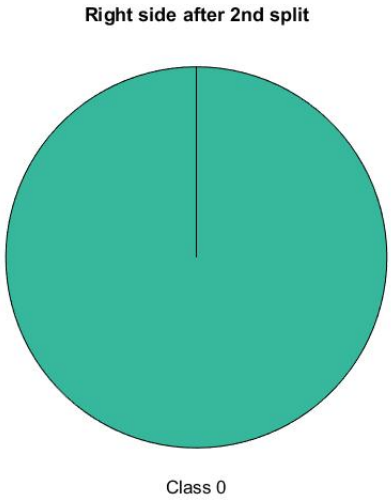


It can be seen that the right side is perfectly pure, so no further split is necessary. To find a good boundary to divide the left side, I use the Gini Index again.



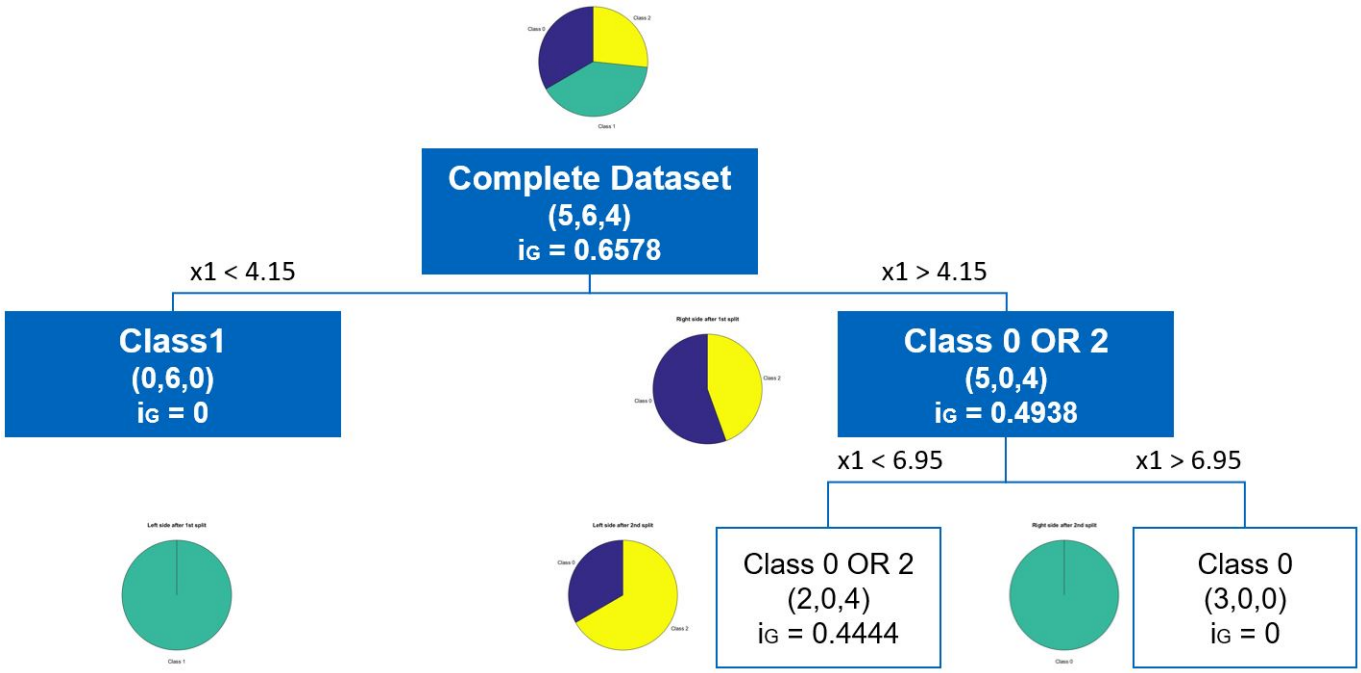
Again the most information is won if we split at the x_1 axis for
$x_1 < 6.95$
Then we obtain the following distributions





We can see that now the left side is perfectly pure.
So now the model for the decision tree is built. For repetition, we first split at $x_1 < 4.15$ and then we split the right side at $x_1 < 6.95$.

1.3 Decision Tree (Problem 1)



1.4 Classification of Vectors (Problem 2)

Now let's classify the vectors
$x_a = (4.1, -0.1, 2.2)^T$
$x_b = (6.1, 0.4, 1.3)^T$
In this decision tree we only split at the x_1 axis. According to the training data
x_a is for 100% class 1.
x_b is for 67% class 2 and for 33% class 0.

1.5 Used Functions and Script

All the functions and the script are implemented in MATLAB. When the dataset is load, you can directly execute the files.

```
function [Delta_Gini] =
Delta_Gini(Gini,x,z,threshold,nclasses)
%% Input: Gini Index, x Vector of Values, z (or y) output /
class, threshold-value, amount of classes
% Output: Information Gain

%% Begin Code
len=length(x);
ileft=zeros(nclasses,1);
iright=zeros(nclasses,1);
for i=1:len
    if x(i) <= threshold
        ileft(z(i)+1)=ileft(z(i)+1)+1;
        % (z(i)+1) > because of class 0
    else
        iright(z(i)+1)=iright(z(i)+1)+1;
    end
end

Gini_left=zeros(3,1);
Gini_right=zeros(3,1);
for i=1:nclasses
        Gini_left(i)=(ileft(i)/(sum(ileft)))^2;
        Gini_right(i)=(iright(i)/(sum(iright)))^2;
end
Gini_Left=1-sum(Gini_left);
Gini_Right=1-sum(Gini_right);
Delta_Gini = Gini -
(sum(ileft)/(sum(ileft)+sum(iright)))*Gini_Left -
(sum(iright)/(sum(ileft)+sum(iright)))*Gini_Right;
end
```

```
function [X2L,X2R,Z2R,Z2L] =
dividedataset(X,z,splitbound,axis)

len=size(X,1);
lrow=0;
rrow=0;
for j=1:len
    if X(j,axis) < splitbound
        lrow=lrow+1;
        X2L(lrow,:)=X(j,:);
        Z2L(lrow)=z(j);
```

```
else
        rrow=rrow+1;
        X2R(rrow,:)=X(j,:);
        Z2R(rrow)=z(j);
    end
end

end
```

```
function[axis,thresholdvalue] = dosplit
(X,z,nclasses,boundary)
%% Input:
% Output:

Index_Gini = Gini_index(z,nclasses)
Gini_Delta=zeros(size(boundary,1),nclasses);
figure
for j=1:nclasses
    % check all values for these variables
    for i=1:length(boundary)
        %boundary=boundary1(i);
        Gini_Delta(i,j) =
Delta_Gini(Index_Gini,X(:,j),z,boundary(i,j),nclasses);
        %Gini_Delta(i)=d_G;
    end
plot (1:size(boundary,1), Gini_Delta(:,j), 'LineWidth',5)
hold on
end
legend('Split x1','Split x2','Split x3')
title('Information Gain (Delta Gini) for split')
hold off
% Split xJ at value boundary(J,I)
[M, Jot] = max(Gini_Delta);
[m, I] = max(M);
J=Jot(I);
Textausgabe = 'Split X for < '
axis = I
thresholdvalue=boundary(J,I)
clear i j m M Textausgabe

end
```

```
function [boundary]=findboundary(X,tolerance)
X=sort(X);
classes=size(X,2);
len=size(X,1);
for col=1:classes
    boundary(:,col)=X(1:len-1,col)+tolerance;
end
end
```

```
function [Gini_index] = Gini_index(z,nclasses)
    z=sort(z);
    class=0;
    n=zeros(nclasses,1);
    for row=1:length(z)
        if z(row)==class
            % n amount of samples in class
            % Class + 1, because lowest class is 0
            n(class+1)=n(class+1)+1;
        else
            while(z(row)~=class)
            class=class+1;
            end
            n(class+1)=n(class+1)+1;
        end
    end
    Gini_index=1;
    for row=1:nclasses
        Gini_index=Gini_index-(n(row)/length(z))^2;
    end
end
```

```
function []=show_distribution(z,titel)
ntype0=0;
ntype1=0;
ntype2=0;
z=sort(z);
len=length(z);
for i=1:len
    if z(i)==0
        ntype0=ntype0+1;
    end
```

```
    if z(i)==1
        ntype1=ntype1+1;
    end
    if z(i)==2
        ntype2=ntype2+1;
    end
end

    labels = {'Class 0', 'Class 1', 'Class 2'};


figure
pie([ntype0 ntype1 ntype2],labels)
title(titel)
%labels = {classlabel};
end
```

```
% Using script to load datasets
%Load_dataset_01 %Dataset can also be load manually
nclasses=3;
%% Find best threshold for splitting
% check all variables x1, x2, x3
X=[x1 x2 x3];
% Distribution before split
show_distribution(z,'Before Split')
[boundary]=findboundary(X,0.05);
[axis,thresholdvalue] = dosplit (X,z,nclasses,boundary);

%% Divide Datasets
[X2L,X2R,Z2R,Z2L] = dividedataset(X,z,thresholdvalue,axis);
show_distribution(Z2L,'Left side after 1st split')
show_distribution(Z2R,'Right side after 1st split')
% Now I see everything from class 1 is on the left, no
further split left

%% Second Split
[boundary]=findboundary(X2R,0.05);
[axis,thresholdvalue] = dosplit (X2R,Z2R,nclasses,boundary);
[X3L,X3R,Z3R,Z3L] =
dividedataset(X2R,Z2R,thresholdvalue,axis);
Gini_index = Gini_index(Z3L,nclasses)
show_distribution(Z3L,'Left side after 2nd split')
show_distribution(Z3R,'Right side after 2nd split')
```

2 k-Nearest Neighbors

2.1 Dataset

The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in his 1936 paper *The use of multiple measurements in taxonomic problems as an example of linear discriminant analysis*. The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimetres. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

Iris Setosa

Iris Versicolor

Iris Virginica

$[Source : http : //en.wikipedia.org/wiki/Iris_flower_data_set]$

2.2 Jupyter Notebook (Problem 3)

01_homework_knn

October 25, 2017

1 Programming assignment 1: k-Nearest Neighbors classification

```
In [18]: import numpy as np
         from sklearn import datasets, model_selection
         import matplotlib.pyplot as plt
         %matplotlib inline
```

1.1 Introduction

For those of you new to Python, there are lots of tutorials online, just pick whichever you like best :)

If you never worked with Numpy or Jupyter before, you can check out these guides * https://docs.scipy.org/doc/numpy-dev/user/quickstart.html * http://jupyter.readthedocs.io/en/latest/

1.2 Your task

In this notebook code to perform k-NN classification is provided. However, some functions are incomplete. Your task is to fill in the missing code and run the entire notebook.

In the beginning of every function there is docstring, which specifies the format of input and output. Write your code in a way that adheres to it. You may only use plain python and `numpy` functions (i.e. no scikit-learn classifiers).

Once you complete the assignments, export the entire notebook as PDF using nbconvert and attach it to your homework solutions. On a Linux machine you can simply use `pdfunite`, there are similar tools for other platforms too. You can only upload a single PDF file to Moodle.

1.3 Load dataset

The iris data set (https://en.wikipedia.org/wiki/Iris_flower_data_set) is loaded and split into train and test parts by the function `load_dataset`.

```
In [19]: def load_dataset(split):
             """Load and split the dataset into training and test parts.

             Parameters
             ----------
             split : float in range (0, 1)
                 Fraction of the data used for training.
```

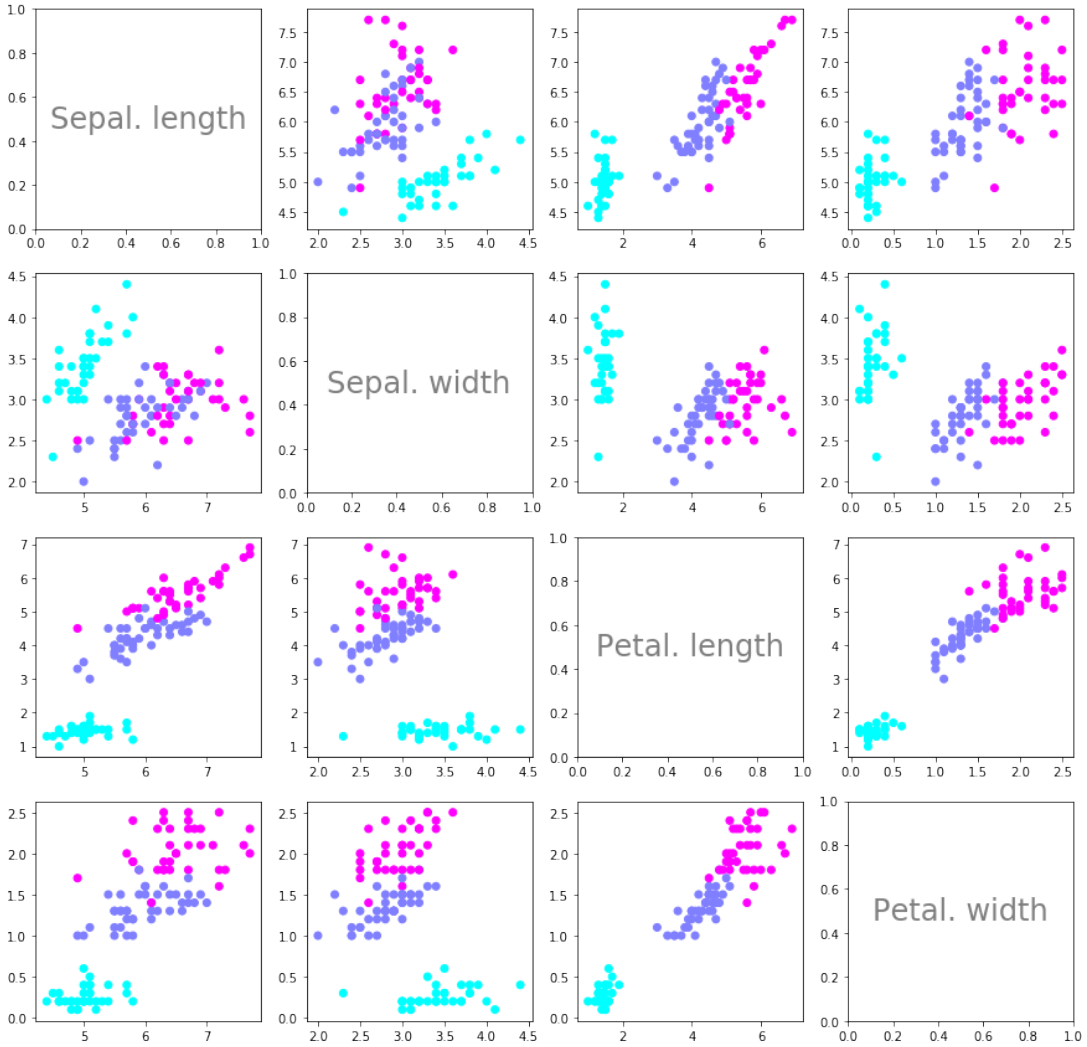
```
        Returns
        -------
        X_train : array, shape (N_train, 4)
            Training features.
        y_train : array, shape (N_train)
            Training labels.
        X_test : array, shape (N_test, 4)
            Test features.
        y_test : array, shape (N_test)
            Test labels.
        """
        dataset = datasets.load_iris()
        X, y = dataset['data'], dataset['target']
        X_train, X_test, y_train, y_test = model_selection.train_test_split(X,
        return X_train, X_test, y_train, y_test
```

```
In [20]: # prepare data
         split = 0.67
         X_train, X_test, y_train, y_test = load_dataset(split)
```

1.4 Plot dataset

Since the data has 4 features, 16 scatterplots (4x4) are plotted showing the dependencies between each pair of features.

```
In [21]: f, axes = plt.subplots(4, 4,figsize=(15, 15))
         for i in range(4):
             for j in range(4):
                 if j == 0 and i == 0:
                     axes[i,j].text(0.5, 0.5, 'Sepal. length', ha='center', va='cen
                 elif j == 1 and i == 1:
                     axes[i,j].text(0.5, 0.5, 'Sepal. width', ha='center', va='cent
                 elif j == 2 and i == 2:
                     axes[i,j].text(0.5, 0.5, 'Petal. length', ha='center', va='cen
                 elif j == 3 and i == 3:
                     axes[i,j].text(0.5, 0.5, 'Petal. width', ha='center', va='cent
                 else:
                     axes[i,j].scatter(X_train[:,j],X_train[:,i], c=y_train, cmap=p
```



1.5 Task 1: Euclidean distance

Compute Euclidean distance between two data points.

```
In [22]: def euclidean_distance(x1, x2):
             """Compute Euclidean distance between two data points.

             Parameters
             ----------
             x1 : array, shape (4)
                 First data point.
             x2 : array, shape (4)
                 Second data point.
```

```
    Returns
    -------
    distance : float
        Euclidean distance between x1 and x2.
    """
    # TODO
    distance=0.0
    sum=0.0
    if len(x1)!=len(x2):
        return "Vectors need to have the same length"
    else:
        for index in range(len(x1)):
            sum=x1[index]-x2[index]
            distance=distance+np.power(sum,2)

        return np.sqrt(distance)
```

1.6 Task 2: get k nearest neighbors' labels

Get the labels of the *k* nearest neighbors of the datapoint *x_new*.

```
In [23]: def get_neighbors_labels(X_train, y_train, x_new, k):
    """Get the labels of the k nearest neighbors of the datapoint x_new.

    Parameters
    ----------
    X_train : array, shape (N_train, 4)
        Training features.
    y_train : array, shape (N_train)
        Training labels.
    x_new : array, shape (4)
        Data point for which the neighbors have to be found.
    k : int
        Number of neighbors to return.

    Returns
    -------
    neighbors_labels : array, shape (k)
        Array containing the labels of the k nearest neighbors.
    """
    # TODO
    dist = np.zeros(np.size(y_train))
    for i in range(np.size(y_train)):
        dist[i] = euclidean_distance(X_train[i,:],x_new)

    dist=np.array(dist)
    sortorder=np.argsort(dist)
    label=y_train[sortorder]
```

```
            return label[:k]
```

1.7 Task 3: get the majority label

For the previously computed labels of the k nearest neighbors, compute the actual response. I.e. give back the class of the majority of nearest neighbors. Think about how a tie is handled by your solution.

```
In [24]: def get_response(neighbors, num_classes=3):
            """Predict label given the set of neighbors.

            Parameters
            ----------
            neighbors_labels : array, shape (k)
                Array containing the labels of the k nearest neighbors.
            num_classes : int
                Number of classes in the dataset.

            Returns
            -------
            y : int
                Majority class among the neighbors.
            """
            # TODO
            class_votes = np.zeros(num_classes)
            neighbors=sorted(neighbors)
            currentclass=neighbors[0]
            i=0
            j=0
            remember=[]
            remember.append(neighbors[i])
            while (i<len(neighbors)):
                if neighbors[i]==currentclass:
                    class_votes[j] += 1
                    i += 1
                else:
                    currentclass=neighbors[i]
                    remember.append(neighbors[i])
                    j += 1
                    class_votes[j] += 1
                    i += 1

            return remember[np.argmax(class_votes)]
```

1.8 Task 4: compute accuracy

Compute the accuracy of the generated predictions.

In [25]:
```
def compute_accuracy(y_pred, y_test):
    """Compute accuracy of prediction.

    Parameters
    ----------
    y_pred : array, shape (N_test)
        Predicted labels.
    y_test : array, shape (N_test)
        True labels.
    """
    # TODO
    # TODO
    n_right = 0
    for i, elem in enumerate(y_pred):
        if elem == y_test[i]:
            n_right += 1
    print(y_pred)
    return n_right/np.size(y_pred)
```

In [26]:
```
# This function is given, nothing to do here.
def predict(X_train, y_train, X_test, k):
    """Generate predictions for all points in the test set.

    Parameters
    ----------
    X_train : array, shape (N_train, 4)
        Training features.
    y_train : array, shape (N_train)
        Training labels.
    X_test : array, shape (N_test, 4)
        Test features.
    k : int
        Number of neighbors to consider.

    Returns
    -------
    y_pred : array, shape (N_test)
        Predictions for the test data.
    """
    y_pred = []
    for x_new in X_test:
        neighbors = get_neighbors_labels(X_train, y_train, x_new, k)
        y_pred.append(get_response(neighbors))
    return y_pred
```

1.9 Testing

Should output an accuracy of 0.9473684210526315.

In [27]:
```
# prepare data
split = 0.67
X_train, X_test, y_train, y_test = load_dataset(split)
print('Training set: {0} samples'.format(X_train.shape[0]))
print('Test set: {0} samples'.format(X_test.shape[0]))

# generate predictions
k = 3
y_pred = predict(X_train, y_train, X_test, k)
accuracy = compute_accuracy(y_pred, y_test)
print('Accuracy = {0}'.format(accuracy))
```

```
Training set: 112 samples
Test set: 38 samples
[2, 2, 2, 1, 0, 2, 1, 0, 0, 1, 2, 0, 1, 2, 2, 2, 0, 0, 1, 0, 0, 2, 0, 2, 0, 0, 0, 2
Accuracy = 0.9473684210526315
```

In [ ]:

In [ ]:

2.3 Classification of Vectors with *k*NN (Problem 4 and 5)

Now let's classify the vectors with the kNN algorithm and compare it to the result with the decision tree

	kNN	Decision Tree	kNN (real values)
$x_a = (4.1, -0.1, 2.2)^T$	0	1 (100 %)	0.5610
$x_b = (6.1, 0.4, 1.3)^T$	2	2 (67%) / 0 (33%)	1.3959

2.4 The Problem (6)

The Problem

Which problem do you see w.r.t. building a Euclidean distance-based k-NN model on X? How can you compensate for this problem? Does this problem also arise when training a decision tree?

The Solution

The mean values of the different features deviate strongly from each other. To compensate this problem, feature scaling is necessary. One way to do this is the Mahalanobis distance.

For decision trees, one does not have to do feature scaling.