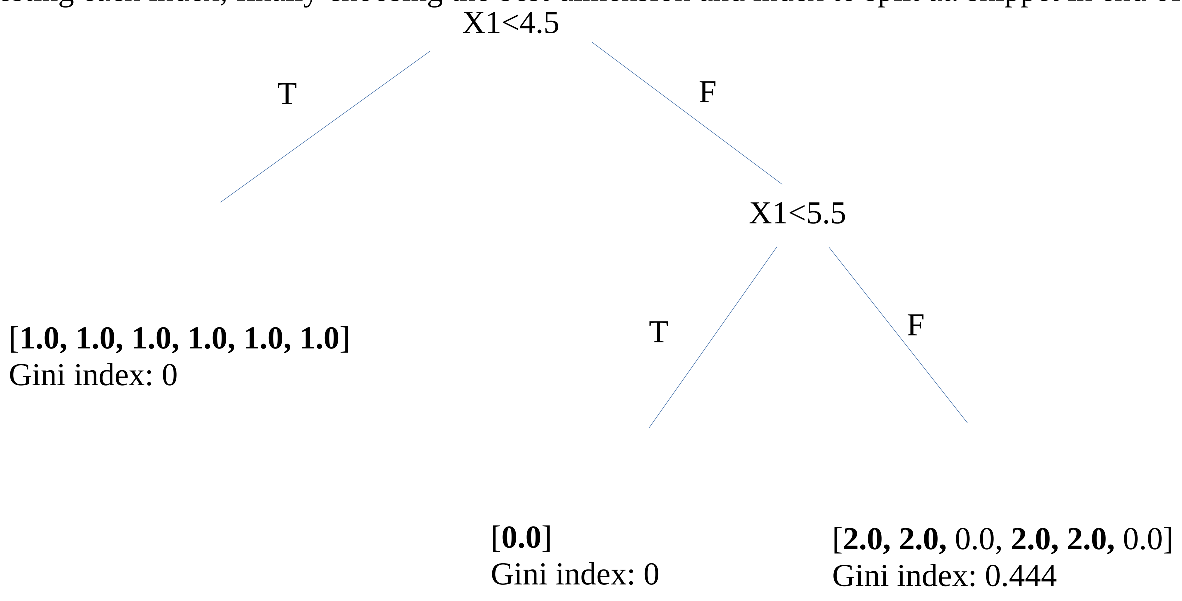
Machine learning: homework 1
Fredrik Opeide. Matrikelnummer: 03699829
1.

Used python to create the decision tree. Finds the split which results in the best change in gini index between start set and the resulting sets (weighted). Tests this by sorting set in each dimension and testing each index, finally choosing the best dimension and index to split at. snippet in end of doc.



2.
x_a would be classified as y_a=1 with p(c=1 | x_a,T)=1
x_b would be classified as y_b=2 with p(c=2 | x_b,T)=4/6=0.667

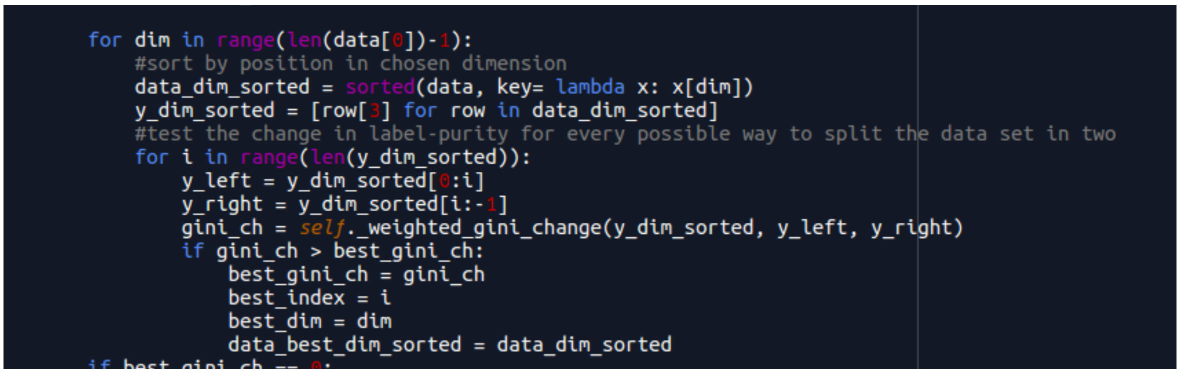
3.
the solution to this task (notebook printout) is found at the end of the document

4.
This is achieved by using x_a and x_b as the test set and reusing most of the code from the notebook exercise solution. Code in end of the document
x_a: [4.1 -0.1 2.2] majority says 0.0
x_b: [6.1 0.4 1.3] majority says 2.0

5.
This is achieved by modifying the get_response function in the notebook to return the average of the labels, instead of the mode. Code in end of document.
x_a: [4.1 -0.1 2.2] average says 1.0
x_b: [6.1 0.4 1.3] average says 1.33333333333

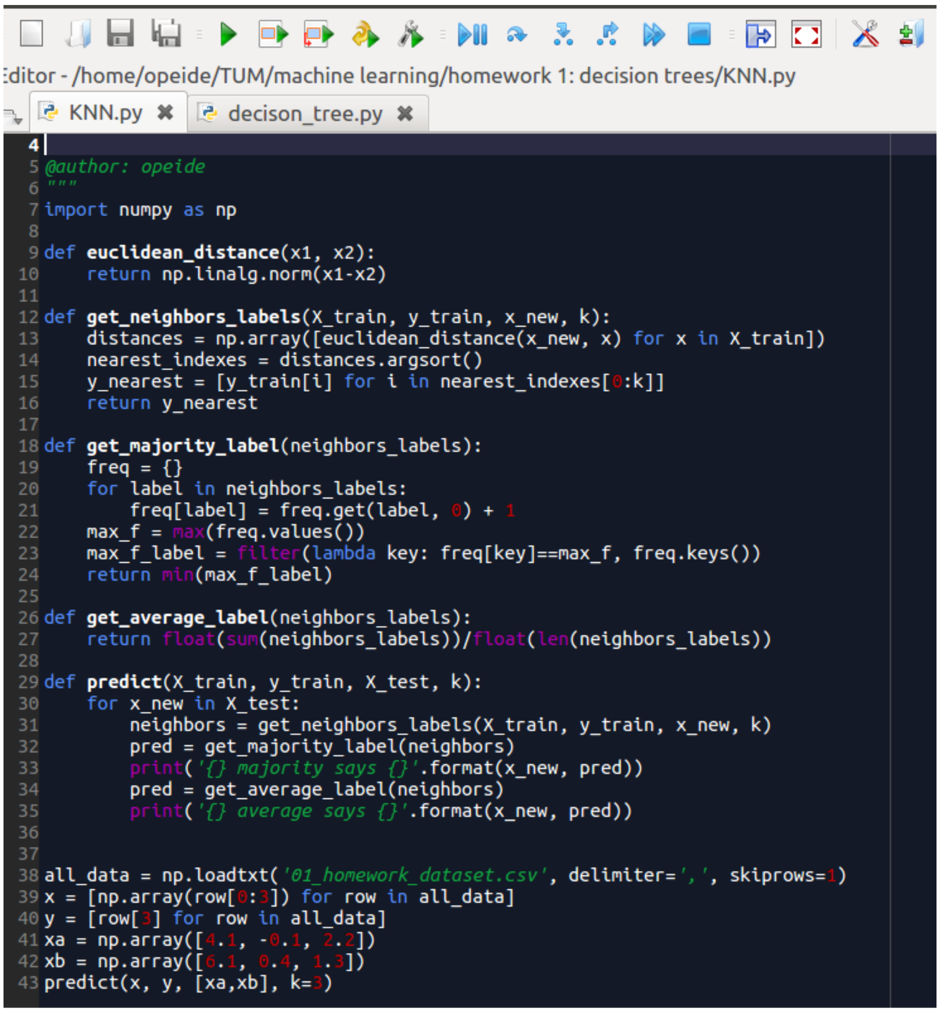
6. Perhaps this euclidean based solution has trouble recognizing groups of elongated shapes.

Snippet from task 1:



```
    for dim in range(len(data[0])-1):
        #sort by position in chosen dimension
        data_dim_sorted = sorted(data, key= lambda x: x[dim])
        y_dim_sorted = [row[3] for row in data_dim_sorted]
        #test the change in label-purity for every possible way to split the data set in two
        for i in range(len(y_dim_sorted)):
            y_left = y_dim_sorted[0:i]
            y_right = y_dim_sorted[i:-1]
            gini_ch = self._weighted_gini_change(y_dim_sorted, y_left, y_right)
            if gini_ch > best_gini_ch:
                best_gini_ch = gini_ch
                best_index = i
                best_dim = dim
                data_best_dim_sorted = data_dim_sorted
    if best_gini_ch == 0:
```

Code for task 4 and 5:



ditor - /home/opeide/TUM/machine learning/homework 1: decision trees/KNN.py

KNN.py ✖ decison_tree.py ✖

```

@author: opeide
"""
import numpy as np

def euclidean_distance(x1, x2):
    return np.linalg.norm(x1-x2)

def get_neighbors_labels(X_train, y_train, x_new, k):
    distances = np.array([euclidean_distance(x_new, x) for x in X_train])
    nearest_indexes = distances.argsort()
    y_nearest = [y_train[i] for i in nearest_indexes[0:k]]
    return y_nearest

def get_majority_label(neighbors_labels):
    freq = {}
    for label in neighbors_labels:
        freq[label] = freq.get(label, 0) + 1
    max_f = max(freq.values())
    max_f_label = filter(lambda key: freq[key]==max_f, freq.keys())
    return min(max_f_label)

def get_average_label(neighbors_labels):
    return float(sum(neighbors_labels))/float(len(neighbors_labels))

def predict(X_train, y_train, X_test, k):
    for x_new in X_test:
        neighbors = get_neighbors_labels(X_train, y_train, x_new, k)
        pred = get_majority_label(neighbors)
        print('{} majority says {}'.format(x_new, pred))
        pred = get_average_label(neighbors)
        print('{} average says {}'.format(x_new, pred))


all_data = np.loadtxt('01_homework_dataset.csv', delimiter=',', skiprows=1)
x = [np.array(row[0:3]) for row in all_data]
y = [row[3] for row in all_data]
xa = np.array([4.1, -0.1, 2.2])
xb = np.array([6.1, 0.4, 1.3])
predict(x, y, [xa,xb], k=3)
```