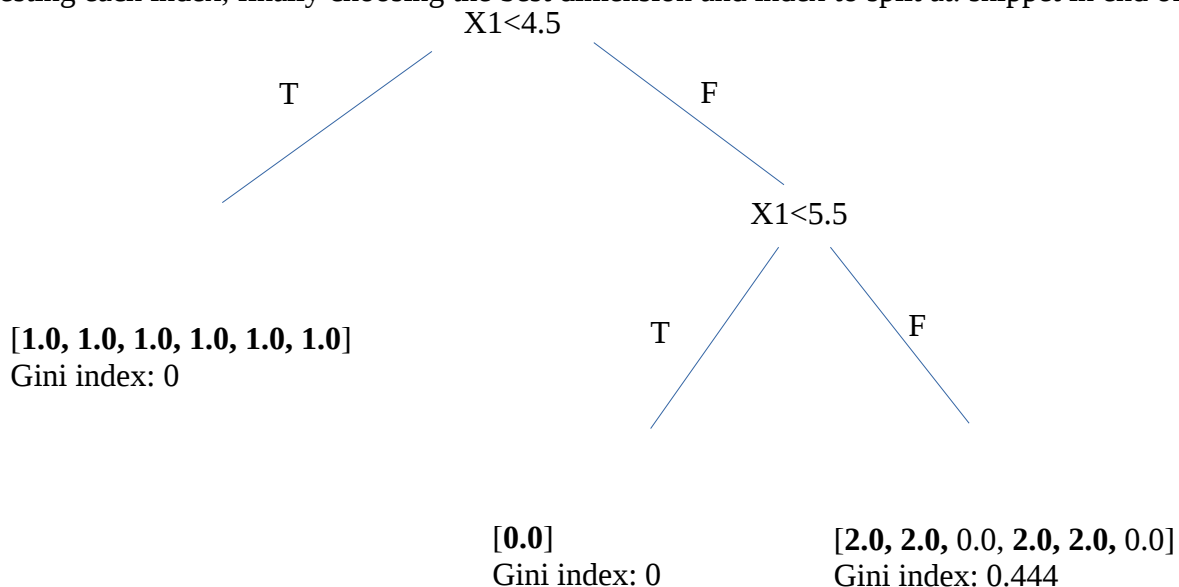


Machine learning: homework 1
Fredrik Opeide. Matrikelnummer: 03699829
1.

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

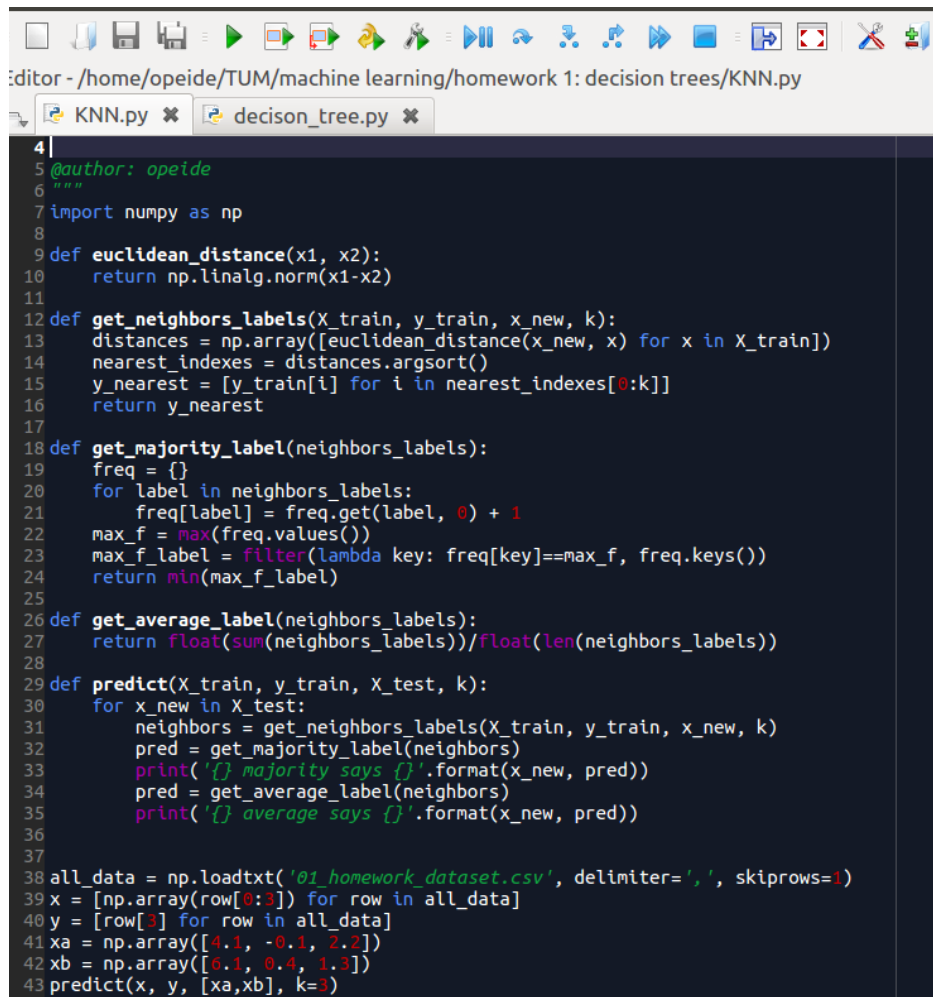
5.
This is achieved by modifying the `get_response` function in the notebook to return the average of the labels, instead of the mode. Code in end of document.
 x_a : [4.1 -0.1 2.2] average says 1.0
 x_b : [6.1 0.4 1.3] average says 1.333333333333

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:



The screenshot shows a code editor window titled "editor - /home/opeide/TUM/machine learning/homework 1: decision trees/KNN.py". The editor has two tabs: "KNN.py" and "decison_tree.py". The code in the "KNN.py" tab is as follows:

```
4 |
5 | @author: opeide
6 | """
7 | import numpy as np
8 |
9 | def euclidean_distance(x1, x2):
10 |     return np.linalg.norm(x1-x2)
11 |
12 | def get_neighbors_labels(X_train, y_train, x_new, k):
13 |     distances = np.array([euclidean_distance(x_new, x) for x in X_train])
14 |     nearest_indexes = distances.argsort()
15 |     y_nearest = [y_train[i] for i in nearest_indexes[0:k]]
16 |     return y_nearest
17 |
18 | def get_majority_label(neighbors_labels):
19 |     freq = {}
20 |     for label in neighbors_labels:
21 |         freq[label] = freq.get(label, 0) + 1
22 |     max_f = max(freq.values())
23 |     max_f_label = filter(lambda key: freq[key]==max_f, freq.keys())
24 |     return min(max_f_label)
25 |
26 | def get_average_label(neighbors_labels):
27 |     return float(sum(neighbors_labels))/float(len(neighbors_labels))
28 |
29 | def predict(X_train, y_train, X_test, k):
30 |     for x_new in X_test:
31 |         neighbors = get_neighbors_labels(X_train, y_train, x_new, k)
32 |         pred = get_majority_label(neighbors)
33 |         print('{} majority says {}'.format(x_new, pred))
34 |         pred = get_average_label(neighbors)
35 |         print('{} average says {}'.format(x_new, pred))
36 |
37 |
38 | all_data = np.loadtxt('01_homework_dataset.csv', delimiter=',', skiprows=1)
39 | x = [np.array(row[0:3]) for row in all_data]
40 | y = [row[3] for row in all_data]
41 | xa = np.array([4.1, -0.1, 2.2])
42 | xb = np.array([6.1, 0.4, 1.3])
43 | predict(x, y, [xa,xb], k=3)
```

01_homework_knn

October 30, 2017

1 Programming assignment 1: k-Nearest Neighbors classification

```
In [378]: 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 `pdffunite`, 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 [379]: 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, y, random_s

return X_train, X_test, y_train, y_test

```
In [380]: # prepare data
```

```
split = 0.75
```

```
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 [381]: 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='center', size=24)
```

```
        elif j == 1 and i == 1:
```

```
            axes[i,j].text(0.5, 0.5, 'Sepal. width', ha='center', va='center', size=24)
```

```
        elif j == 2 and i == 2:
```

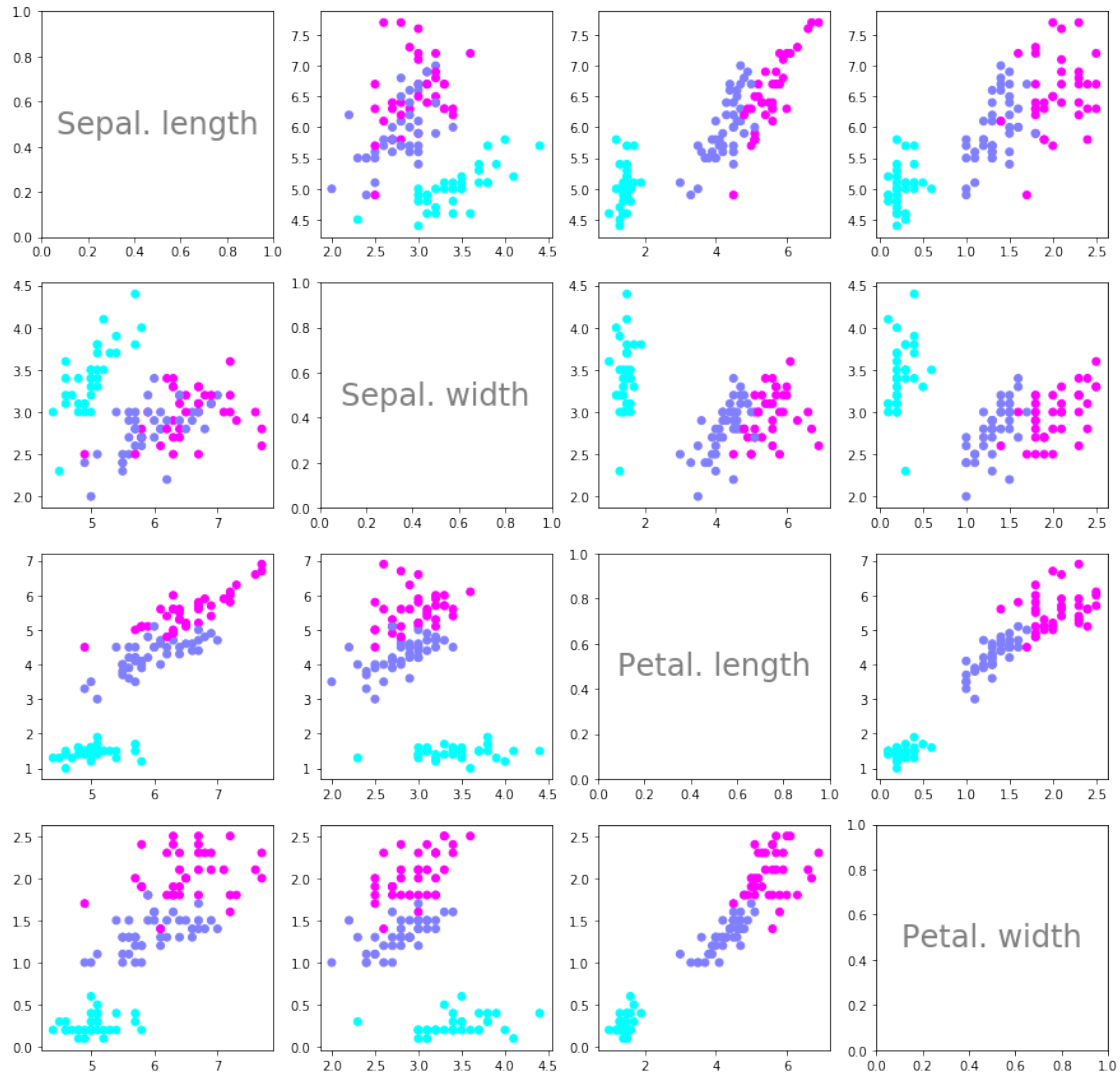
```
            axes[i,j].text(0.5, 0.5, 'Petal. length', ha='center', va='center', size=24)
```

```
        elif j == 3 and i == 3:
```

```
            axes[i,j].text(0.5, 0.5, 'Petal. width', ha='center', va='center', size=24)
```

```
        else:
```

```
            axes[i,j].scatter(X_train[:,j], X_train[:,i], c=y_train, cmap=plt.cm.cool)
```



1.5 Task 1: Euclidean distance

Compute Euclidean distance between two data points.

```
In [382]: 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  $x_1$  and  $x_2$ .
"""
return np.linalg.norm(x1-x2)

```

1.6 Task 2: get k nearest neighbors' labels

Get the labels of the k nearest neighbors of the datapoint x_{new} .

```

In [383]: 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.
    """
    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

```

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. In case of a tie, choose the "lowest" label (i.e. the order of tie resolutions is $0 > 1 > 2$).

```

In [384]: def get_response(neighbors_labels, 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.
"""
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)

```

1.8 Task 4: compute accuracy

Compute the accuracy of the generated predictions.

```

In [385]: 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.
    """
    n_correct = len([p for p,t in zip(y_pred,y_test) if p==t])
    accuracy = float(n_correct)/float(len(y_pred))
    return accuracy

```

```

In [386]: # 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 [387]: # prepare data
split = 0.75
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
Accuracy = 0.9473684210526315

```