Machine Learning Worksheet 02

Shang-Hsin Yu – 03681048 – shanghsin.yu@tum.de

Problem 1

I wrote a python code myself to solve this problem.
For every node with id x, its left child will have id $x * 2$ and its right child will have id $x * 2 + 1$
The root has id 1
Node 1 splits at $x = 4.3$
Node 2 does not need to split anymore
Node 3 splits at $x = 7.15$
For each split those smaller than the split value goes the the left child and vice versa

The GINI index and class distribution of all the nodes are as follows:
Node 1 – GINI : 0.657777777778, class 0 : 5, class 1 : 6, class 2 : 4
Node 2 – GINI : 0.000000000000, class 0 : 0, class 1 : 6, class 2 : 0
Node 3 – GINI : 0.493827160494, class 0 : 5, class 1 : 0, class 2 : 4
Node 4 – Does not exist
Node 5 – Does not exist
Node 6 – GINI : 0.444444444444, class 0 : 2, class 1 : 0, class 2 : 4
Node 7 – GINI : 0.000000000000, class 0 : 3, class 1 : 0, class 2 : 0

Problem 2

According to the results from problem 1,
x_a will be classified as class 1, with probability 100%
x_b will be classified as class 2, with probability 66.67%

Problem 3

Done!

Problem 4

x_a is classified as class 0
x_b is classified as class 2

Problem 5

x_a is labeled as 0.561016425974
x_b is labeled as 1.4467507092

Problem 6

The problem is that the three features for this set of data have quite a different variance.
This causes euclidean distance to favor some features while that is not what we really want.
The easiest way to solve this problem is to standardize the data set by performing some linear transformation to give them all the same variance first, then start training. This problem does not arise in decision tree because we don't perform multiplication and addition on the data. We only did comparison. Comparison is not sensitive to variance at all.

Implementation exercise: k-NN

In [59]:

```
import random
import numpy as np
import operator
import math
from sklearn import datasets
import matplotlib.pyplot as plt
%matplotlib inline
```

Load dataset

The iris data set (https://en.wikipedia.org/wiki/Iris_flower_data_set (https://en.wikipedia.org/wiki/Iris_flower_data_set)) it loaded by the function loadDataset.

Arguments:

- *split*: int: Split rate between test and training set e.g. 0.67 corresponds to 1/3 test and 2/3 validation

Returns:

- *X*: list(array of length 4); Trainig data
- *Z*: list(int); Training labels
- *XT*: list(array of length 4); Test data
- *ZT*: list(int); Test labels

In [60]:

```
def loadDataset(split, X=[] , XT=[], Z = [], ZT = []):
    dataset = datasets.load_iris()
    c = list(zip(dataset['data'], dataset['target']))
    random.seed(224)
    random.shuffle(c)
    x, t = zip(*c)
    sp = int(split*len(c))
    X = x[:sp]
    XT = x[sp:]
    Z = t[:sp]
    ZT = t[sp:]
    return X, XT, Z, ZT
```

In [61]:

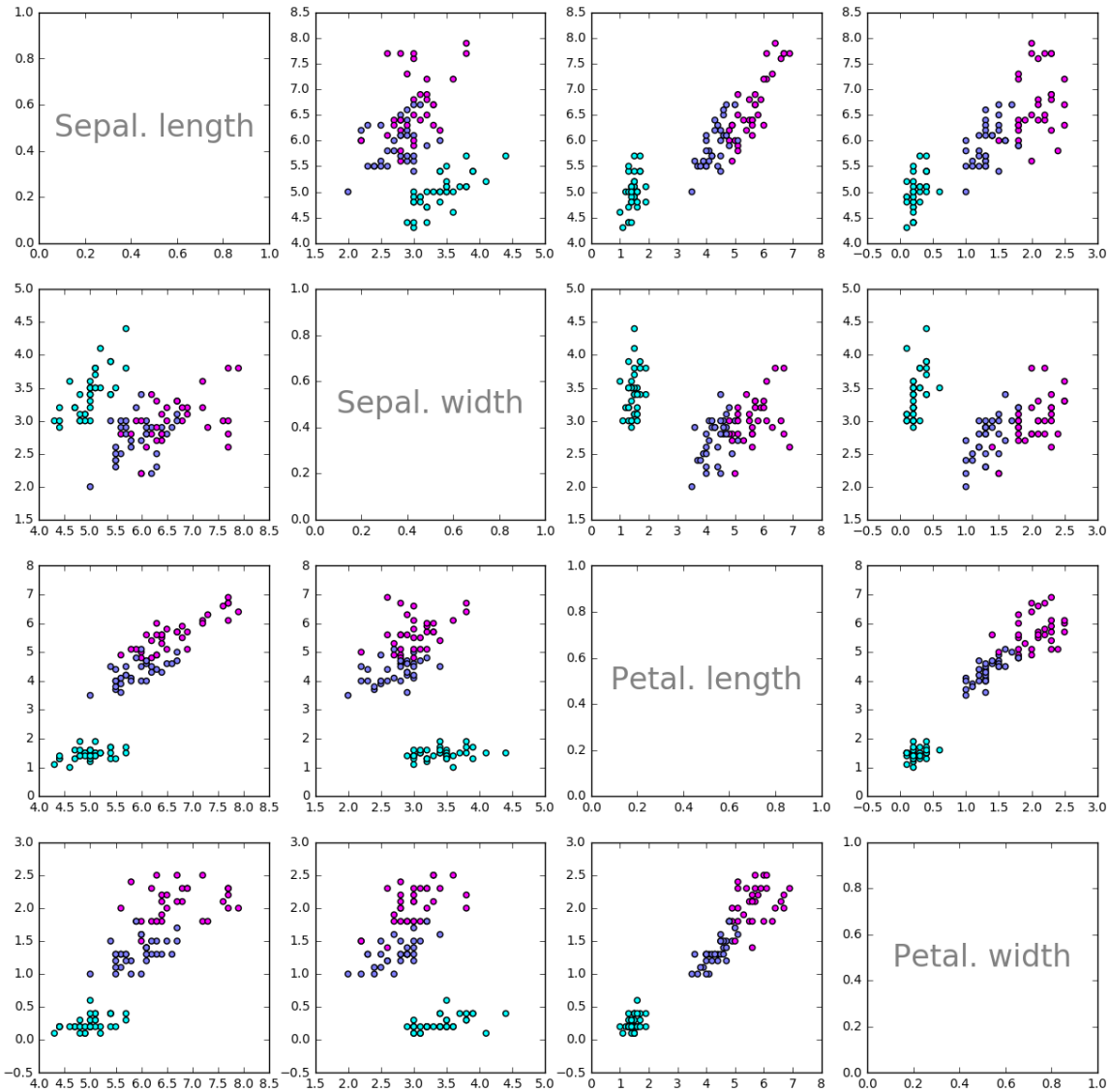
```
# prepare data
split = 0.67
X, XT, Z, ZT = loadDataset(split)
```

Plot dataset

Since *X* is dimentionality 4, 16 scatterplots (4x4) are plotted showing the dependencies of each two features.

In [62]:

```
Xa = np.asarray(X)
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, alpha=.5)
        elif j == 1 and i == 1:
            axes[i,j].text(0.5, 0.5, 'Sepal. width', ha='center', va='center', s
ize=24, alpha=.5)
        elif j == 2 and i == 2:
            axes[i,j].text(0.5, 0.5, 'Petal. length', ha='center', va='center',
size=24, alpha=.5)
        elif j == 3 and i == 3:
            axes[i,j].text(0.5, 0.5, 'Petal. width', ha='center', va='center', s
ize=24, alpha=.5)
        else:
            axes[i,j].scatter(Xa[:,j],Xa[:,i], c = Z, cmap=plt.cm.cool)
```



Exercise 1: Euclidean distance

Compute euclidean distance between two data points.

arguments:

- *x1*: array of length 4; data point
- *x2*: array of length 4; data point

returns:

- *distance*: float; euclidean distance between *x1* and *x2*

In [63]:

```
def euclideanDistance(x1, x2):
    result = 0.0
    for i in xrange(len(x1)):
        result += (x1[i]-x2[i])*(x1[i]-x2[i])
    return math.sqrt(result)
```

Exercise 2: get k nearest neighbors

For one data point xt compute all k nearest neighbors.

arguments:

- *X*: list(array of length 4); Trainig data
- *Z*: list(int); Training labels
- *xt*: array of length 4; Test data point

returns:

- neighbors: list of length *k* of tuples (X_neighbor, Z_neighbor, distance between neighbor and xt); **this is the list of k nearest neighbors to xt**

In [64]:

```
def getNeighbors(X, Z, xt, k):
    input = zip(X, Z)
    input = sorted(input, key=lambda item: euclideanDistance(item[0], xt))
    result = []
    for i in xrange(k):
        result.append((input[i][0], input[i][1], euclideanDistance(input[i][0], x

    return result
```

Exercise 3: get neighbor response

For the previously computed k nearest neighbors compute the actual response. I.e. give back the class of the majority of nearest neighbors. What do you do with a tie?

arguments:

- neighbors: list((array, int, float)
- c: int; number of classes in the dataset, for the iris dataset c=3

returns

- y: int; majority target

In [65]:

```
def getResponse(neighbors, c=3):
    counter = [0, 0, 0]
    neighbors.reverse()
    for stuff in neighbors:
        counter[stuff[1]] += 1
    for stuff in neighbors:
        for i in xrange(3):
            a = (i+1)%3
            b = (i+2)%3
            if counter[i]>counter[a] and counter[i]>counter[b]:
                return i
        counter[stuff[1]] -= 1
```

Exercise 4: Compute accuracy

arguments:

- YT: list(int); predicted targets
- ZT: list(int); actual targets

returns:

- accuracy: float; percentage of correctly classified test data points

In [66]:

```
def getAccuracy(YT, ZT):
    result = 0.0
    for i in xrange(len(YT)):
        if YT[i] == ZT[i]:
            result += 1
    return result / len(YT)
```

In [67]:

```
def predict(X, Z, XT, k):
    Y=[]
    for xt in XT:
        neighbors = getNeighbors(X, Z, xt, k)
        Y.append(getResponse(neighbors))
    return Y
```

Testing

Should output an accuracy of 0.95999999999999996.

In [68]:

```
# prepare data
split = 0.67
X, XT, Z, ZT = loadDataset(split)
print 'Train set: ' + repr(len(X))
print 'Test set: ' + repr(len(XT))
# generate predictions
k = 3
YT = predict(X, Z, XT, k)
accuracy = getAccuracy(YT, ZT)
print('Accuracy: ' + repr(accuracy))
```

```
Train set: 100
Test set: 50
Accuracy: 0.96
```

In [ ]:

```

```