MACHINE LEARNING HOMEWORK SHEET-1

k-NN & DT.

Name: MAHALAKSHMI SABANAYAGAM

TUM ID: ge73yuw

DUE DATE: 29.10.2018.

Problem 1:

Build DT T for data {x, y}, using Gini Index, depth - 2.

Arranging the given data {x, y} in ascending order with respect to x_1 and then x_2 (if x_1 values are equal) and x_3 (if x_2 values are equal)

i	x_1	x_2	x_3	y	
N	0.4	0.1	4.3	1	
J	0.5	0	2.3	1	
M	1	0.1	2.8	1	
H	1.3	-0.2	1.8	1	
O	2.7	-0.5	4.2	1	
G	4.1	0.3	5.1	1	(possible 1st level cut)
I	4.5	0.4	2	0	(possible 1st level cut)
A	5.5	0.5	4.5	2	
K	5.9	-0.1	4.4	0	
C	5.9	0.2	3.4	2	
F	6.8	-0.3	5.1	2	
E	6.9	-0.1	0.6	2	(possible 2nd level cut)
B	7.4	1.1	3.6	0	
L	9.3	-0.2	3.2	0	
D	9.9	0.1	0.8	0	

$$\text{Gini Index } i_G(t) = 1 - \sum_{c_i \in C} \pi_{c_i}^2$$

@ depth 0 : (5, 6, 4)

$$\text{Gini } (5,6,4) = 1 - \left[\left(\frac{5}{15}\right)^2 + \left(\frac{6}{15}\right)^2 + \left(\frac{4}{15}\right)^2\right] = 0.6578$$

The trees generated by splitting at $x_4 \le 4.1$ and $x_4 \le 4.5$ are as follows,

T_1 : split @ $x_4 \le 4.1$

(5, 6, 4) $i_G(t) = 0.6578$

$x_4 \le 4.1$

(0,6,0) — (5,0,4)

$i_G(t_L)$ = Gini (0,6,0)
$= 1 - 1 = 0$

$i_G(t_R)$ = Gini (5,0,4)
$= 1 - \left(\frac{25+16}{81}\right)$
$= 0.4938$

T2 : split @ $x_4 \le 4.5$

(5, 6, 4) $i_G(t) = 0.6578$

$x_4 \le 4.5$

(1, 6, 0) — (4, 0, 4)

$i_G(t_L)$ Gini (1,6,0)
$= 1 - \left(\frac{1+36}{49}\right)$
$= 0.2449$

$i_G(t_R)$ Gini (4,0,4)
$= 1 - \left(\frac{16+16}{64}\right)$
$= 0.5$

Calculating improvement at level 1 to decide between T_1 & T2,

Improvement for split s of t into t_R & t_L,

$$\Delta i\,(s,t) = i\,(t) - p_L\, i(t_L) - p_R\, i(t_R)$$

Improvement of T_1

$$\Delta i\,(x_4 \le 4.1, T_1) = 0.6578 - \frac{6}{15}(0) - \frac{9}{15}(0.4938) = 0.36152$$

Improvement of T2

$$\Delta i\,(x_4 \le 4.5, T2) = 0.6578 - \frac{7}{15}(0.2449) - \frac{8}{15}(0.5) = 0.27685$$

$$\Delta i\,(x_4 \le 4.1, T_1) > \Delta i\,(x_4 \le 4.5, T2)$$

So T1 is better than T2.

For 2nd level,

(Final Tree)

Splitting @ $x_4 \le 6.9$

(5, 6, 4) $i_G(t) = 0.6578$

$x_4 \le 4.1$

(0,6,0) $i_G(t_L) = 0$ — (5,0,4) $i_G(t_R) = 0.4938$

$x_4 \le 6.9$

(2,0,4) — (3,0,0)

$i_G(t_{RL}) = 1 - \frac{4+16}{36} = 0.4444$

$i_G(t_{RR}) = 1 - 1 = 0$

Problem 2:

$\vec{x}_a = (4.1, -0.1, 2.2)^T$

Using the generated T in problem 1, x_a will be classified as $\hat{y}_a = 1$

$p(c = \hat{y}_a | \vec{x}_a, T) = \frac{6}{6} = 1$

$\vec{x}_b = (6.1, 0.4, 1.3)^T$

$\hat{y}_b = 2$

$p(c = \hat{y}_b | \vec{x}_b, T) = \frac{4}{6} = 0.6667$

k- NEAREST NEIGHBORS,

Problem 4, and 5:

Please check the ipynb. Both the problems are solved programatically at the end of the given notebook.

Problem 6:

The scales of the features (x_1, x_2, x_3) of X are different.

Mean and variance of,

	μ	σ^2
x_1	$\simeq 5$	$\simeq 9$
x_2	$\simeq 0.1$	$\simeq 0.14$
x_3	$\simeq 3$	$\simeq 2$

As the scale of x_2 is very less compared to x_1 and x_3, x_2 is insignificant when kNN is used. So, the useful information held by x_2 will be lost.

To avoid this, we have to scale each feature to zero mean and unit variance. So, standardising / normalising the data will help.

We can also use other distance metric like Mahalanobis.

This problem will not come up when using Decision Tree as it is not dependent on scales of the features.

Programming assignment 1: k-Nearest Neighbors classification

In [1]:
```python
import numpy as np
import math
from sklearn import datasets, model_selection
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
```

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/

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).

Exporting the results to PDF

Once you complete the assignments, export the entire notebook as PDF and attach it to your homework solutions. The best way of doing that is

1. Run all the cells of the notebook.
2. Download the notebook in HTML (click File > Download as > .html)
3. Convert the HTML to PDF using e.g. https://www.sejda.com/html-to-pdf or `wkhtmltopdf` for Linux (tutorial)
4. Concatenate your solutions for other tasks with the output of Step 3. 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.

This way is preferred to using `nbconvert`, since `nbconvert` clips lines that exceed page width and makes your code harder to grade.

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`.

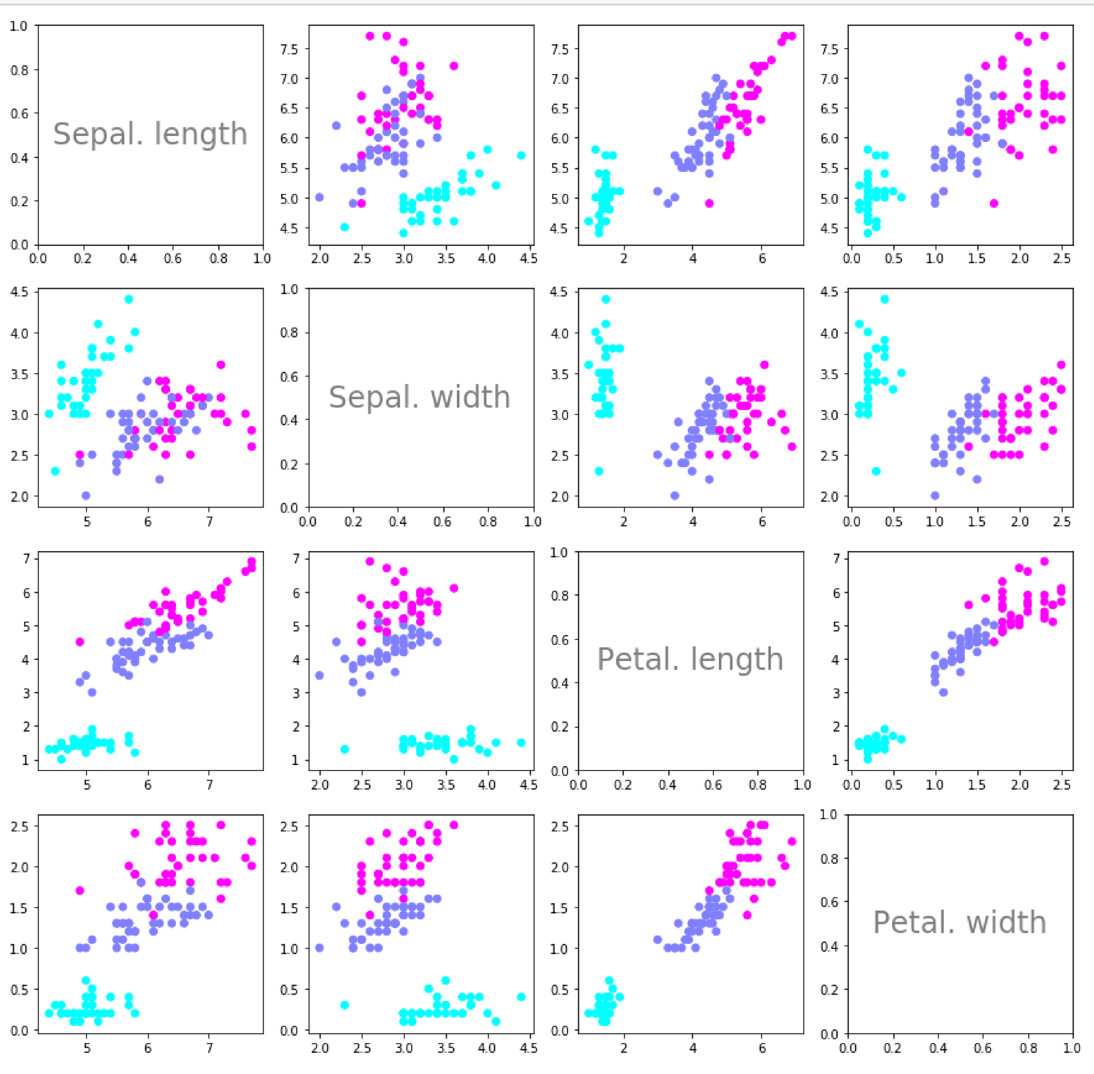
In [2]:
```python
def load_dataset(split):
    """Load and split the dataset into training and test parts.

    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_state=123, test
_size=(1 - split))
    return X_train, X_test, y_train, y_test
```

In [3]:
```python
# prepare data
split = 0.75
X_train, X_test, y_train, y_test = load_dataset(split)
```

Plot dataset

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

In [4]:
```python
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', size=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', size=24, alpha=.5)
        else:
            axes[i,j].scatter(X_train[:,j],X_train[:,i], c=y_train, cmap=plt.cm.cool)
```



Task 1: Euclidean distance

Compute Euclidean distance between two data points.

In [5]:
```python
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
    d = sum(np.array(x1-x2)**2)
    distance = math.sqrt(d)
    return distance
```

Task 2: get k nearest neighbors' labels

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

In [6]:
```python
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
    distance_new = []
    #Calculate Euclidean distance of the new data point with all the training data
    for x in X_train:
        distance_new.append(euclidean_distance(x,x_new))
    #Create a sorted list of dist,label
    distance_label = list(zip(distance_new,y_train))
    sorted_distance = sorted(distance_label, key=lambda distance_label: distance_label[0])
    #Return the first k labels (second column of the created list)
    neighbours_labels = np.array(sorted_distance)[:,1]
    labels = [ int(x) for x in neighbours_labels ] #Making sure the labels are int
    return labels[:k]
```

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 [7]:
```python
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.
    """
    # TODO
    class_votes = np.zeros(num_classes)
    #Count the number of instances of each class label in the neighbors_label
    for i in range(num_classes):
        class_votes[i] = list(neighbors_labels).count(i)
    #Get the max in that and return the class it belongs to
    majority_vote = max(class_votes)
    majority_class = np.where(class_votes==majority_vote)
    #There could be tie cases, so the majority_class sometimes have a set of numbers.
    #So choose the first one in that as that is what is needed (lowest label)
    return int(majority_class[0][0])
```

Task 4: compute accuracy

Compute the accuracy of the generated predictions.

In [8]:
```python
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
    correct_pred = 0
    #Subtract each element in the array, the 0 ones in the result are the correct predicitions
    #So count those and divide by the total number of tested instances
    pred = y_pred - y_test
    correct_pred = list(pred).count(0)
    accuracy = correct_pred/len(y_pred)
    return accuracy
```

In [9]:
```python
# 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
```

Testing

Should output an accuracy of 0.9473684210526315.

In [10]:
```python
# 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
```

Problem 4 and 5

In [11]:
```python
#Load the dataset from the csv file and create an array of x1, x2, x3 and another array of y (cls)
data = pd.read_csv("01_homework_dataset.csv")
cls = data[' z'].tolist()
x1 = data['x1'].tolist()
x2= data[' x2'].tolist()
x3 = data[' x3'].tolist()
x = []
for i in range(len(x1)):
    l = [x1[i],x2[i],x3[i]]
    x.append(l)
x, cls
```

Out[11]:
```
([[5.5, 0.5, 4.5],
  [7.4, 1.1, 3.6],
  [5.9, 0.2, 3.4],
  [9.9, 0.1, 0.8],
  [6.9, -0.1, 0.6],
  [6.8, -0.3, 5.1],
  [4.1, 0.3, 5.1],
  [1.3, -0.2, 1.8],
  [4.5, 0.4, 2.0],
  [0.5, 0.0, 2.3],
  [5.9, -0.1, 4.4],
  [9.3, -0.2, 3.2],
  [1.0, 0.1, 2.8],
  [0.4, 0.1, 4.3],
  [2.7, -0.5, 4.2]],
 [2, 0, 2, 0, 2, 2, 1, 1, 0, 1, 0, 0, 1, 1, 1])
```

In [12]:
```python
k = 3
X_test = [[4.1,-0.1,2.2],[6.1,0.4,1.3]]
y_pred = predict(x, cls, np.asarray(X_test), k)
y_pred
```

Out[12]:
```
[0, 2]
```

Answer Problem 4:

(Lowest class label is chosen in case of tie)

xa is classified as 0 and xb is classified as 2

In [13]:
```python
#This function returns the distance with the class label
def get_neighbors_distance_regression(X_train, y_train, x_new, k):
    distance_new = []
    x_new = np.asarray(x_new)
    #Calculate Eucledian distance between the new point and the training data
    for x in X_train:
        x = np.asarray(x)
        distance_new.append(euclidean_distance(x,x_new))
    #Add the class label with the distance, sort and return the first k values
    distance_label = list(zip(distance_new,y_train))
    sorted_distance = sorted(distance_label, key=lambda distance_label: distance_label[0])
    return np.asarray(sorted_distance[:k])
```

In [14]:
```python
#This function computes the normalization term (1/Z in the kNN regression equation
def get_normalization(dist):
    return np.reciprocal(sum(np.reciprocal(dist)))
```

In [15]:
```python
#This function computes sum of yi/d(x,xi)
def get_sum(dist_label):
    dist_inv = np.reciprocal(dist_label[:,0])
    lab = dist_label[:,1]
    return sum(dist_inv * lab)
```

In [16]:
```python
#Calculates y
def get_response_regression(distance_label):
    term1 = get_normalization(distance_label[:,0])
    term2 = get_sum(distance_label)
    return term1*term2
```

Answers for Problem 5

In [17]:
```python
xa = [4.1,-0.1,2.2]
dist_lab = get_neighbors_distance_regression(x, cls, xa, k)
print("3 Nearest Neighbors distances and labels \n", dist_lab)
print("The predicted class is", get_response_regression(dist_lab))
```

```
3 Nearest Neighbors distances and labels
 [[0.67082039 0.        ]
 [2.18403297 2.        ]
 [2.47386338 1.        ]]
The predicted class is 0.5610164259744004
```

In [18]:
```python
xb = [6.1,0.4,1.3]
dist_lab = get_neighbors_distance_regression(x, cls, xb, k)
print("3 Nearest Neighbors distances and labels \n", dist_lab)
print("The predicted class is", get_response_regression(dist_lab))
```

```
3 Nearest Neighbors distances and labels
 [[1.17473401 2.        ]
 [1.74642492 0.        ]
 [2.11896201 2.        ]]
The predicted class is 1.39592451328945
```