NAME: THEODOR STOICAN

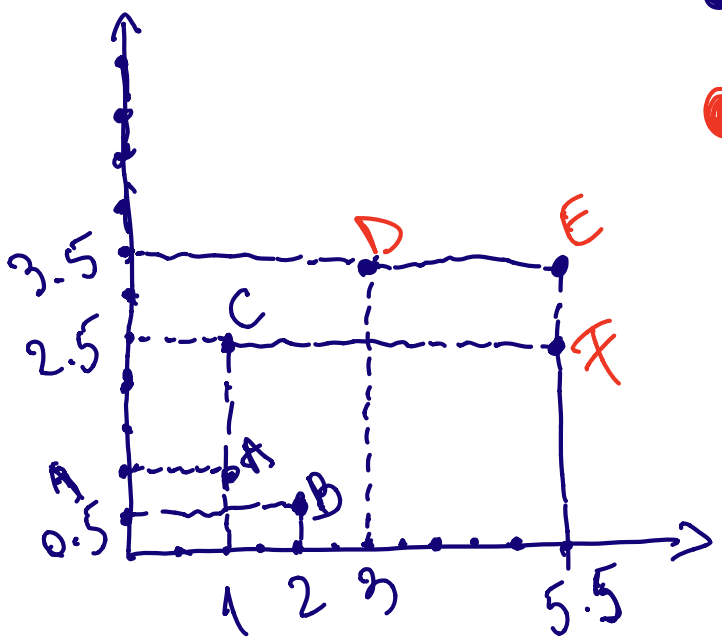
ID : ge93pas

MATRICULATION NUMBER : 03725732

ASSIGNMENT 2

- class 1
- class 2

Problem 1



a) Point A:

According to the L-1 norm, either B or C are the nearest neighbours for A. In this case, by convention, we choose B.

$$d(A,B) = |2-1| + |0.5-1| = 1.5$$

(class 1)

Point B:

Similarly, $d(B,A) = |1-2| + |1-0.5|$

$= 1.5$ (class 1)

Point C:

$d(C,A) = |2.5-1| + |1-1| = 1.5$ (class 1)

Point D:

$d(D,E) = |5.5-3| + |3.5-3.5| = 2.5$ (class 2)

Point E:

$d(E,F) = |3.5-2.5| + |5.5-5.5|$

$= 1$ (class 2)

Point F:

$d(F,E) = |2.5-3.5| + |5.5-5.5|$

$= 1$ (class 2)

b) Point A:

$$d(A,B) = \sqrt{(2-1)^2 + (1-0.5)^2}$$

$$= \sqrt{1 + (0.5)^2}$$

$$= \sqrt{1 + \frac{1}{4}} = \sqrt{\frac{5}{4}} = \frac{\sqrt{5}}{2}$$

Point B: (class 1)

$$d(B,A) = \frac{\sqrt{5}}{2}$$ (as before)

(class 1)

Point C:

$$d(C,A) = \sqrt{(2.5-1)^2 + (1-1)^2}$$

$$= \sqrt{(1.5)^2} = 1.5$$

Point D: (class 1)

$$d(D,C) = \sqrt{(3-1)^2 + (3.5-2.5)^2}$$

$$= \sqrt{4+1} = \sqrt{5}$$

(class 1)

(class 1)

Point E:

$$d(E,F) = \sqrt{(3.5-2.5)^2}$$

$$= \sqrt{1} = 1$$

(class 2)

Point F:

$$d(F,E) = 1 \quad \text{(as before)}$$

(class 2)

c) As we can see from the previous computations, using the L1 norm yields better performance since it classifies correctly all

the points, whereas using the L_2 norm leads to a misclassification (point B).

Problem 2

a) The probability that x_{new} is assigned to a class x is the following:

$$P(y = X \mid x_{new}, K = N_A + N_B + N_C) =$$

$$= \frac{1}{N_A + N_B + N_C} \sum_{i \in N_K(x_{new})} \mathbb{I}(y_i = X)$$

This value will reach its maximum for class $X = C$ since the instances that belong to class C are the most numerous ones.

($\frac{N_C}{N_A + N_B + N_C}$ is the largest value)

Therefore, a new point $\vec{x}$ will always be assigned to class C.

b) In this case, the classification score will diminish, since new

points which are within one particular cluster and significantly farther from the other clusters will be assigned the same class as the particular cluster they are part of (given that the other clusters are sufficiently "far away" so that their score may not be greater than the correct score).

Long story short, the answer is dependent on the positions of the clusters and the new point.

Problem 3

The diagonal line has equation $x_1 - x_2 = 0$

Since, due to practical reasons, decision trees rely on horizontal and vertical splits only, no matter what we do, we cannot approximate such the diagonal with a single line (depth=1). Therefore, we cannot have 100% accuracy on this dataset with a decision tree of depth 1.

Problem 4

a) $i_H(y) = \sum_c -p(y=c)\log p(y=c)$

$$p(y = \text{Win}) = \frac{4}{10} = \frac{2}{5}$$

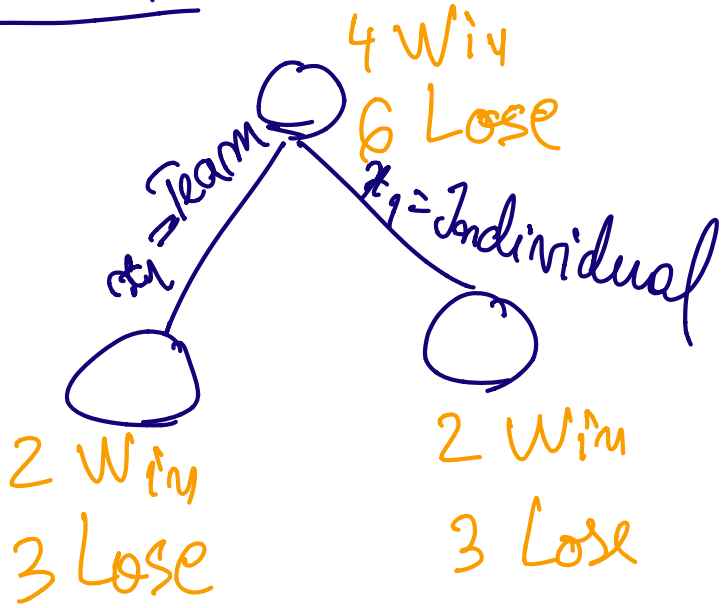
$$p(y = \text{Lose}) = 1 - \frac{2}{5} = \frac{3}{5}$$

$$\Rightarrow i_H(y) = -\frac{2}{5}\log\left(\frac{2}{5}\right) - \frac{3}{5}\log\left(\frac{3}{5}\right) \simeq 0.52 + 0.44$$

$$\simeq 0.96$$

b) We have 3 features and therefore 3 possible splits that we can make.

Split on x_1



$$i_H(t_L) = -p(y=Win)\log p(y=Win) - p(y=Lose)\log p(y=Lose)$$

$$= -\frac{2}{5}\log\frac{2}{5} - \frac{3}{5}\log\frac{3}{5}$$

$$\approx 0.96$$

$i_H(t_R) \approx 0.96$ (same as $i_H(t_L)$ because of the distribution of classes)

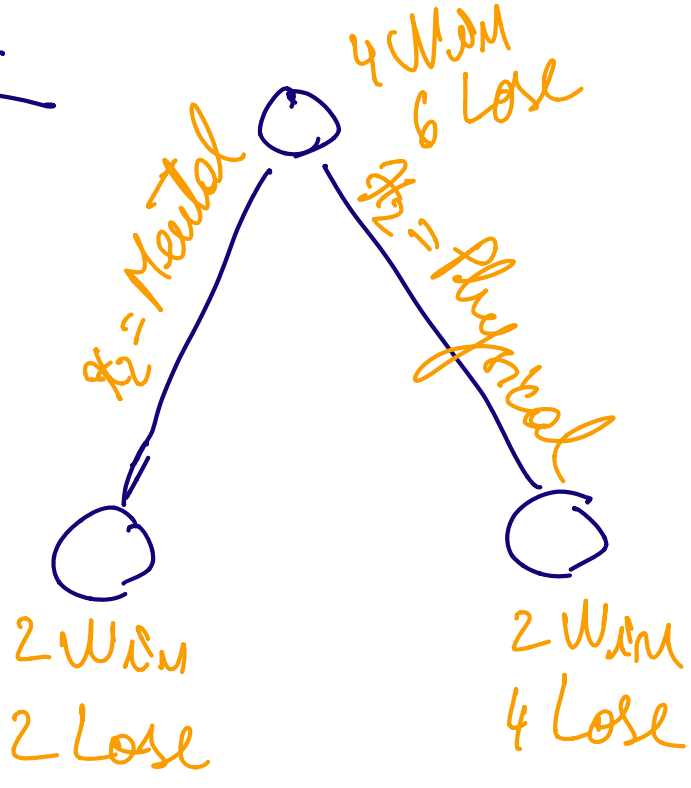
Hence, $\Delta i_H(x_1 = \text{Team}, t) = i_H(t) - \frac{5}{10} i_H(t_L) - \frac{5}{10} i_H(t_R)$

$\Rightarrow \Delta i_H(x_1 = \text{Team}, t) = 0.96 - \frac{1}{2}(0.96 + 0.96)$

$= 0.96 - \frac{1}{2} \cdot 2 \cdot 0.96$

$= 0$

Split on x_2



$i_H(t_L) = -P(y = \text{Win}) \log P(y = \text{Win}) - P(y = \text{Lose}) \log P(y = \text{Lose}) =$

$$= -\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2}$$

$$= -\log\frac{1}{2} = -\log 2^{-1} = 1$$

$$i_H(x_R) = -\frac{2}{6}\log\frac{2}{6} - \frac{4}{6}\log\frac{4}{6}$$

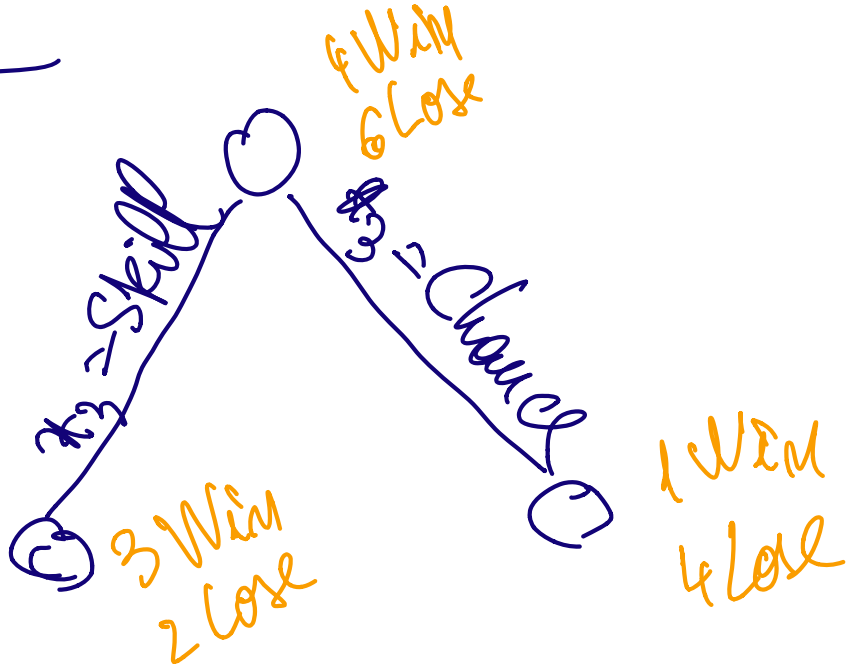
$$= -\frac{1}{3}\log\frac{1}{3} - \frac{2}{3}\log\frac{2}{3}$$

$$\approx 0.91$$

$$\Rightarrow \Delta i_H(x_2 = \text{Mental}, t) = 0.96 - \frac{4}{10}\cdot 1 - \frac{6}{10}\cdot 0.91$$

$$\Rightarrow \Delta i_H(x_2 = \text{Mental}, t) \approx \boxed{0.044}$$

Split on x_3



$$i_H(t_L) = -\frac{3}{5}\log\frac{3}{5} - \frac{2}{5}\log\frac{2}{5}$$

$$\approx 0.96$$

$$i_H(t_R) = -\frac{1}{5}\log\frac{1}{5} - \frac{4}{5}\log\frac{4}{5}$$

$$\approx 0.72$$

$$\Delta i_H(x_3 = \text{Skill}, t) = 0.96 - \frac{5}{10}\cdot 0.96 - \frac{5}{10}\cdot 0.72$$

$$\approx \boxed{0.12}$$

Given all of these results, the largest Δi_H is 0.12 and the optimal decision tree of depth = 1 can be obtained by splitting on x_3.

exercise_02_notebook

October 27, 2019

1 Programming assignment 1: k-Nearest Neighbors classification

[1]:
```
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.

You are only allowed to use the imported packages. Importing anything else is NOT allowed.

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).

1.3 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. Export/download the notebook as PDF (File -> Download as -> PDF via LaTeX (.pdf)). 3. Concatenate your solutions for other tasks with the output of Step 2. 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.

Make sure you are using `nbconvert` Version 5.5 or later by running `jupyter nbconvert --version`. Older versions clip lines that exceed page width, which makes your code harder to grade.

1.4 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`.

[2]:
```
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
```

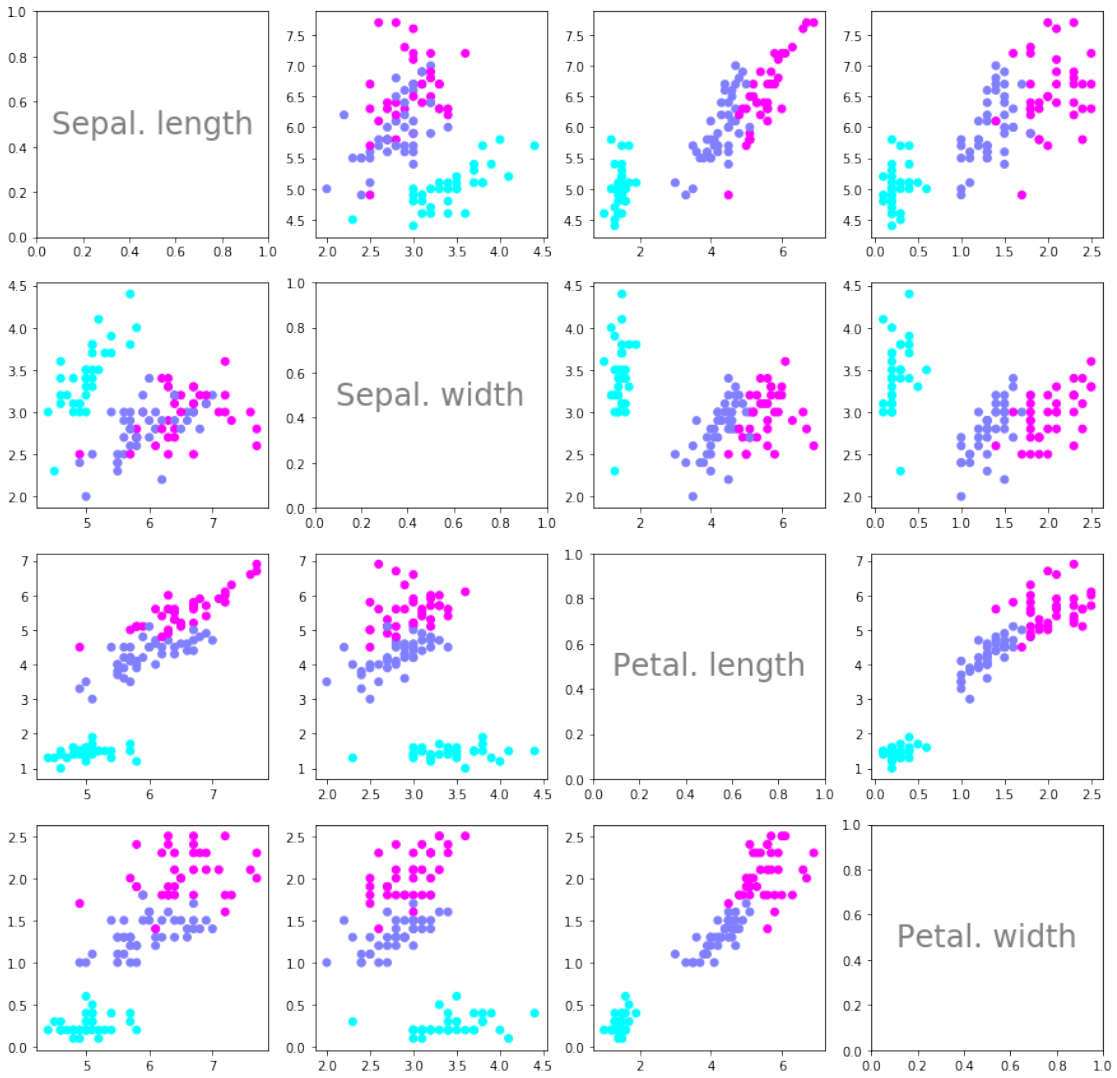
[3]:
```
# prepare data
split = 0.75
X_train, X_test, y_train, y_test = load_dataset(split)
```

1.5 Plot dataset

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

[4]:
```
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)
```



1.6 Task 1: Euclidean distance

Compute Euclidean distance between two data points.

[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
    return np.linalg.norm(x1-x2)
```

1.7 Task 2: get k nearest neighbors' labels

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

[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
    neighbours = []

    for x, y in zip(X_train, y_train):
      neighbours.append((euclidean_distance(x, x_new), y))
    neighbours.sort(key = lambda x: x[0])
```

```
    return list(map(lambda x: x[1], neighbours[0:k]))
```

1.8 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 $0 > 1 > 2$).

[7]:
```
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)

    for label in neighbors_labels:
      class_votes[label] += 1

    maj_label = 0

    for i in range(len(class_votes)):
      if class_votes[maj_label] < class_votes[i]:
        maj_label = i
      elif class_votes[maj_label] == class_votes[i] and i < maj_label:
        maj_label = i

    return maj_label
```

1.9 Task 4: compute accuracy

Compute the accuracy of the generated predictions.

[8]:
```
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 = 0

    for i in range(len(y_test)):
      if y_test[i] == y_pred[i]:
        correct += 1

    return correct / len(y_test)
```

[9]:
```
# 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))
    y_pred = np.array(y_pred)
    return y_pred
```

1.10 Testing

Should output an accuracy of 0.9473684210526315.

[10]:
```
# 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
```