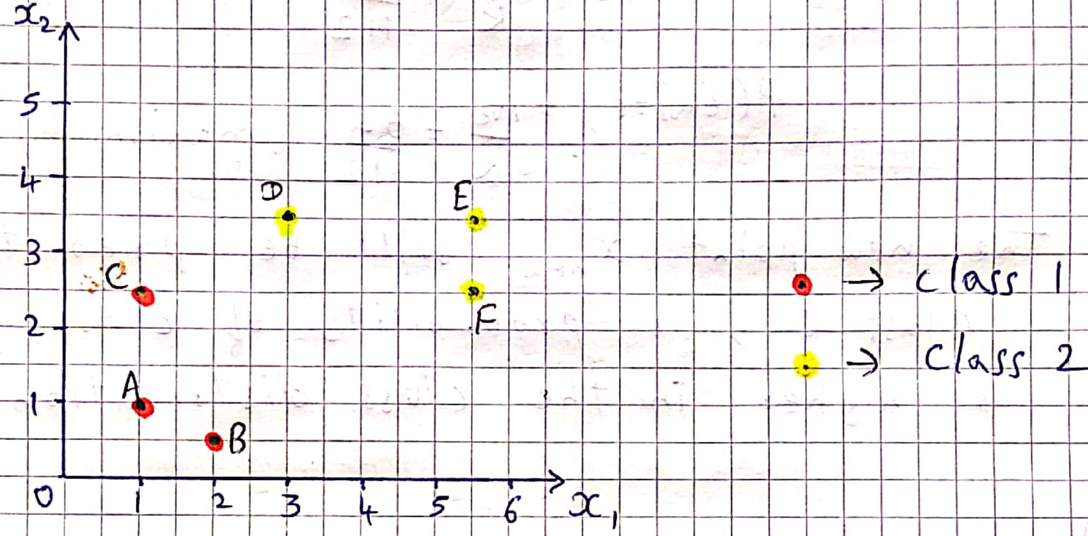
MACHINE LEARNING
(IN 2064)
EXERCISE SHEET-2

COLLABORATOR
AKHIL NASSER
03713232

1)



a) Distance between each point and its nearest neighbor using L_1-norm as distance measure:

Name	distance (L_1-norm)
A	1.5
B	1.5
C	1.5
D	2.5
E	1
F	1

L_1 norm $= \sum_i |u_i - v_i|$

b) Distance between each point and its nearest neighbor using L_2-norm as distance measure

Name	distance (L_2-norm)
A	$\sqrt{1.25}$
B	$\sqrt{1.25}$
C	$\sqrt{2.25}$
D	$\sqrt{5}$
E	1
F	1

L_2-norm $= \sqrt{\sum_i (u_i - v_i)^2}$

<u>Note</u>
D gets class 1 using L_2-norm

c) The classification is performed between each point and its nearest neighbor using <u>L_1-norm</u> as distance measure.

Scanned by CamScanner

2 - a)

$$K = N_A + N_B + N_C$$

$$P(\text{Class} = A) = \frac{N_A}{K} = \frac{16}{112}$$

$$P(\text{Class} = B) = \frac{N_B}{K} = \frac{32}{112}$$

$$P(\text{class} = C) = \frac{N_C}{K} = \frac{64}{112} \text{ (maximum)}$$

The new point x_{new} will be predicted as Class C, as probability of class having 'C' is higher in the class distribution

b)

As weight is inversely proportional to the distance, the prediction for a new point x_{new} is dependent on its location.

x_{new} gets the label of the class, which is closest to it.

In other words, the distances between x_{new} and data points of a class should be smaller compared to distances between x_{new} and data points of other classes.

Then, x_{new} get class label as such a class.

3) Yes, there exists a decision tree of depth 1 that classifies this dataset with 100% accuracy

$x_1 \leq x_2$

True — Blue

False — Red

4.a)

$$i_H(y) = -p(y=W)\log_2 p(y=W) - p(y=L)\log_2 p(y=L)$$

$$= -\frac{4}{10}\log_2\frac{4}{10} - \frac{6}{10}\log_2\frac{6}{10}$$

$$= -\frac{4}{10} \times -1.322 - \frac{6}{10} \times -0.737$$

$$= 0.5288 + 0.4422$$

$$= 0.971$$

b) Let us take only x_1 (team or individual) as feature

<u>Case (i)</u>

$x_1 = T$

True

$P(y=W \mid x_1=T) = \frac{2}{5}$

$P(y=L \mid x_1=T) = \frac{3}{5}$

False

$P(y=W \mid x_1=I) = \frac{2}{5}$

$P(y=L \mid x_1=I) = \frac{3}{5}$

$$\Delta i_H(x_1=T) = -\frac{5}{10}\log_2\frac{5}{10} - \frac{5}{10}\log_2\frac{5}{10}$$

$$-\frac{5}{10}\left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right)$$

$$-\frac{5}{10}\left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right)$$

$$= -\log_2\left(\frac{1}{2}\right) - \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right)$$

$$= 1 + 0.4 \times -1.322 + 0.6 \times -0.737$$

$$= 1 - 0.5288 - 0.4422$$

$$\Delta i_H(x_1 = T) = 0.029$$

<u>Case (ii)</u> Taking x_2 (Mental or Physical) only as feature

$$x_2 = M$$

True | False

$P(y=W \mid x_2=M) = \frac{2}{4}$ $\qquad$ $P(y=W \mid x_2=P) = \frac{2}{6}$

$P(y=L \mid x_2=M) = \frac{2}{4}$ $\qquad$ $P(y=L \mid x_2=P) = \frac{4}{6}$

$$\Delta i_H(x_2 = M)$$

$$= -\frac{4}{10}\log_2\frac{4}{10} - \frac{6}{10}\log_2\frac{6}{10}$$

$$- \frac{4}{10}\left(-\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4}\right)$$

$$- \frac{6}{10}\left(-\frac{2}{6}\log_2\frac{2}{6} - \frac{4}{6}\log_2\frac{4}{6}\right)$$

$$= 0.4 \times 1.322 + 0.6 \times 0.737 - 0.4 + 0.2 \times \log_2\left(\tfrac{1}{3}\right) + 0.4\log_2\left(\tfrac{2}{3}\right)$$

$$= 0.5288 + 0.4422 - 0.4 - 0.2 \times 1.585 - 0.4 \times 0.585$$

$$= 0.02$$

<u>Case (iii)</u> Taking x_3 (skill or chance) only as feature

$$x_3 = S$$

True | False

$P(y=W \mid x_3=S) = 3/5$ $\qquad$ $P(y=W \mid x_3=C) = 1/5$

$P(y=L \mid x_3=S) = 2/5$ $\qquad$ $P(y=L \mid x_3=C) = 4/5$

$$\Delta i_H(x_3 = s) = -\frac{5}{10}\log_2\frac{5}{10} - \frac{5}{10}\log_2\frac{5}{10}$$

$$-\frac{5}{10}\left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right)$$

$$-\frac{5}{10}\left(-\frac{1}{5}\log_2\frac{1}{5} - \frac{4}{5}\log_2\frac{4}{5}\right)$$

$$= 1 + 0.3\log_2\frac{3}{5} + 0.2\log_2\frac{2}{5} + 0.1\log_2\frac{1}{5} + 0.4\log_2\frac{4}{5}$$

$$= 1 + 0.3\times -0.737 + 0.2\times -1.322 + 0.1\times -2.322 + 0.4\times -0.322$$

$$= 0.1535$$

$\therefore$ We get maximum improvement for case (iii)

Optimal decision tree of depth 1

$x_3 = s$

True: $P(y=w \mid x_3 = s) = 3/5$, $P(y=L \mid x_3 = s) = \frac{2}{5}$

False: $P(y=w \mid x_3 = c) = 1/5$, $P(y=L \mid x_3 = c) = 4/5$

exercise_02_notebook

October 26, 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.

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. 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.

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
```

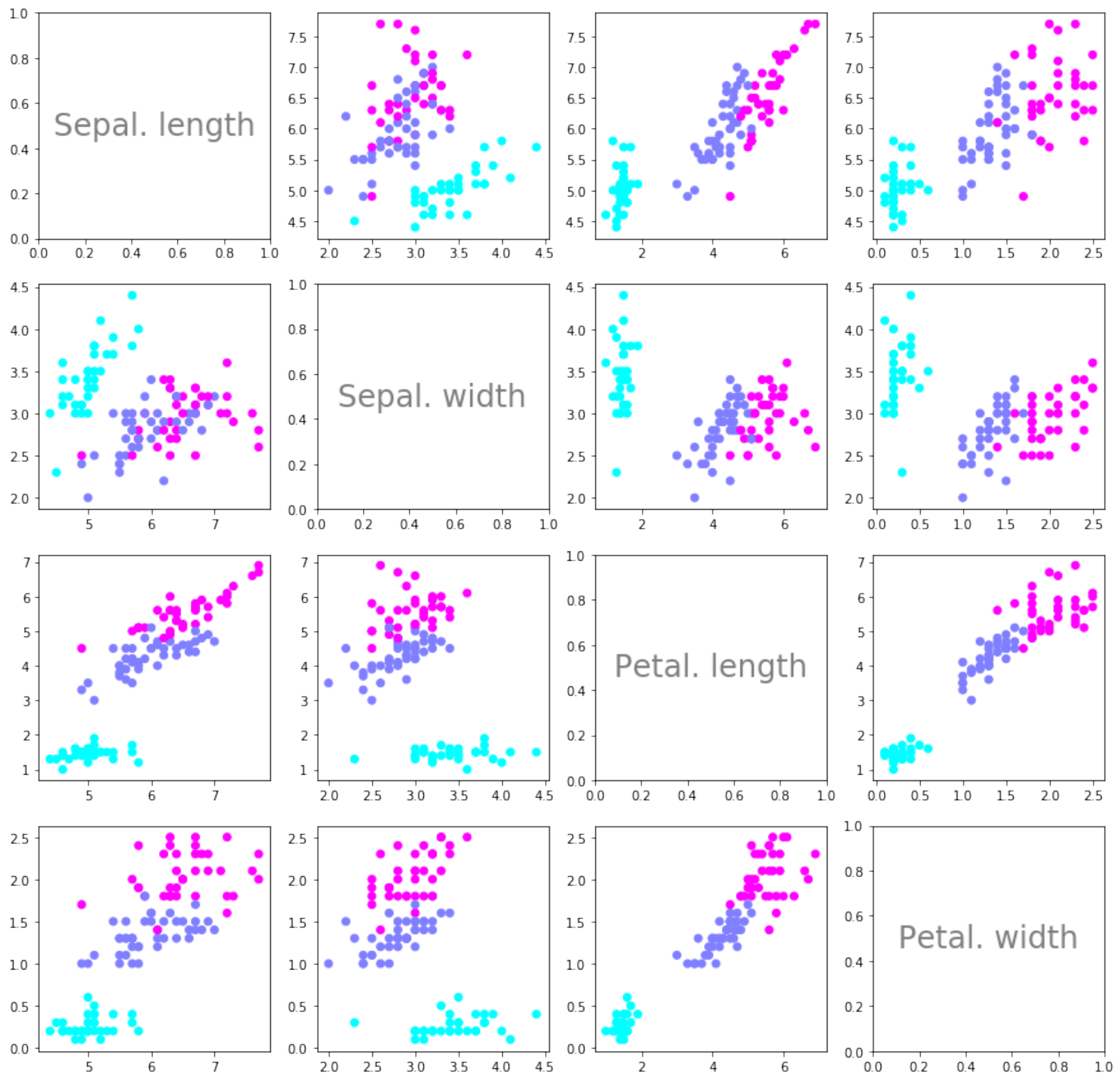
[3]:
```
# prepare data
split = 0.75
X_train, X_test, y_train, y_test = load_dataset(split)
```

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]:
```
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
    dist = np.linalg.norm(x1-x2)
    return dist
```

1.7 Task 2: get k nearest neighbors' labels

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

[6]:
```
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
    distances = []

    i=0
    for x_train in X_train:

        dist = euclidean_distance(x_new,x_train)
        distances.append([dist,y_train[i]])
        i=i+1
        distances=sorted(distances)
    distances = np.array(distances)
    neighbors_labels = distances[0:k,1]
    return neighbors_labels
```

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 c in range(num_classes):
         class_votes[c] = list(neighbors_labels).count(c)
    response = np.argmax(class_votes)
    return response
```

1.9 Task 4: compute accuracy

Compute the accuracy of the generated predictions.

[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
    accuracy = (y_pred == y_test).mean()
    return accuracy
```

[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
```

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
```