① a)

	a	b	c	d	e	f	nearest-neighbour L_1
a	0	1,5	1,5	4,5	7	6	$\Rightarrow$ b, c = 1,5
b	1,5	0	3	4	6,5	5,5	$\Rightarrow$ a = 1,5
c	1,5	3	0	3	5,5	4,5	$\Rightarrow$ a = 1,5
d	4,5	4	3	0	2,5	3,5	$\Rightarrow$ e = 2,5
e	7	6,5	5,5	2,5	0	1	$\Rightarrow$ f = 1
f	6	5,5	4,5	3,5	1	0	$\Rightarrow$ e = 1

$\sum_i |u_i - v_i|$

b)

	a	b	c	d	e	f	nearest neighbour L_2
a	0	$\sqrt{1,25}$	1,5	$\sqrt{10,25}$	$\sqrt{26,5}$	$\sqrt{22,5}$	$\Rightarrow b = \sqrt{1,25}$
b	$\sqrt{1,25}$	0	$\sqrt{5}$	$\sqrt{10}$	$\sqrt{21,25}$	$\sqrt{16,25}$	$\Rightarrow a = \sqrt{1,25}$
c	1,5	$\sqrt{5}$	0	$\sqrt{5}$	$\sqrt{21,25}$	$\sqrt{20,25}$	$\Rightarrow$ a = 1,5
d	$\sqrt{10,25}$	$\sqrt{10}$	$\sqrt{5}$	0	$\sqrt{6,25}$	$\sqrt{7,25}$	$\Rightarrow c = \sqrt{5}$
e	$\sqrt{26,5}$	$\sqrt{21,25}$	$\sqrt{21,25}$	$\sqrt{6,25}$	0	1	$\Rightarrow$ f = 1
f	$\sqrt{22,5}$	$\sqrt{20,25}$	$\sqrt{20,25}$	$\sqrt{7,25}$	1	0	$\Rightarrow$ e = 1

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

c) Since we use a different measure for distance in each classification instance, and we don't guarantee the same results (closest neighbour wise). It means that each classification instance is unique and the norm to be used should vary with the problem in hand.

② 3 classes
$$\begin{cases} a \to N_A = 16 \\ b \to N_B = 32 \\ c \to N_C = 64 \end{cases}$$

a) for any new point the classifier will predict c in these condition since if k = N you will get a majority rule and c is the most frequent class

b) with the weighted version we will predict the class in which the datapoints are most similar (distance wise) as long as they are close enough to outweigh the majority of c-class points.

(3) • The units in which the dimensions are represented are not really comparable so the distance measures wouldn't have much success. We could solve this by normalizing every dimension, since we would be comparing relatively similar values

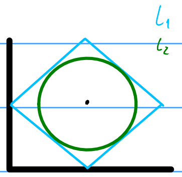
=> regarding decision trees, since we make comparisons for each dimension in isolation, we don't run into this problem

• Say we have atleast 100 data points for each class. This means that, assuming each data point is unique in every dimension. We cover roughly 500 points in a $\mathbb{R}^5$ space, with a total of 100^5 possible points. This means that the space covered equates to something as $500/100^5 = 5 \times 10^{-6}$ % of of the total sample space. This problem can't be fully solved but we can try to maximize the value of the available data by performing k-fold cross validation.

=> This is a common problem for most ML methods, including decision trees as we can't generate information if we don't have enough information to start.

(4)
$$\sum_{i=1}^{d} \sqrt{(x_i - y_i)^2} \leq \sum_{i=1}^{d} |x_i - y_i|$$

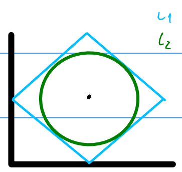
Let's consider a point p in $\mathbb{R}^2$ (generalizable to more dimensions)



if we consider an arbitrary distance d in which another point resides we can see that the L_1 norm forms a rhombus around p, as for L_2 we have a circular region around p.

visually we have that all points at distance d in L_2 of p are $\leq$ than all points of distance d of p in L_1. Proving that the original statement is correct

(5)



should we take the point y in the center, and let's say we have a point x in the green region. Say that this point x is y's closest neighbor.

• by the proof above where $L_2 \leq L_1$, if we have a point in the green region this implies that there can't be another point z in the blue region where $d_{L_1}z \leq d_{L_2}x$. Which is equivalent to saying that if x is the point which is closest to y in L_2, the same applies when measuring distances in L_1.

(6) No since every split of continuous data can only be binary. The lines dividing the input space which are created by decision trees, can only be parallel to the vectors Ox and Oy which means that in order to create a perfect split of this data, we would have to create a sort of ladder with infinitesimal steps that approximate a line with slope 1

$\Rightarrow$ we need a DT with n (large enough) depth in order to classify this dataset with 100% accuracy. If the dataset fully occupied the space available then we would need n to be infinite.

(7) $I_H(t) = -\sum_{c \in C} \pi_c \log_2(c)$ | Note: $I(x,y) = x\log_2(x) + y\log_2(y)$

$$\Rightarrow I_H(y) = -\left(\underbrace{2/5 \lg(2/5)}_{w} + \underbrace{3/5 \lg(3/5)}_{l}\right)$$

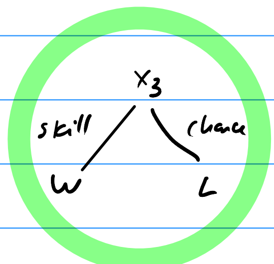
$$= (0{,}5288 + 0{,}4428)$$

$$= (0{,}971)$$

- $\Delta_H(x_1) = I_H(y) - I_{team} - I_{ind} = 0$
  - $\Rightarrow I_{team} = 5/10 \; I(2/5, 3/5) \simeq 0{,}485$
  - $\Rightarrow I_{ind} = 5/10 \; I(3/5, 2/5) = 0{,}485$

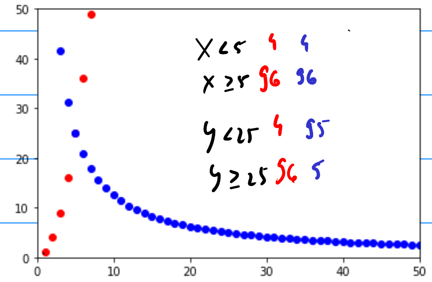
- $\Delta_H(x_2) = I_H(y) - I_{metal} - I_{play} = 0{,}021$
  - $\Rightarrow I_{metal} = 4/10 \; I(1/2, 1/2) = 0{,}4$
  - $\Rightarrow I_{play} = 6/10 \; I(1/3, 2/3) \simeq 0{,}55$

$\Rightarrow$ since a split over x_3 has the highest information gain, an optimal DT with depth 1 is:



- $\Delta_H(x_3) = I_H(y) - I_{skill} - I_{chance} = 0{,}971 - 0{,}485 - 0{,}361 = 0{,}125$
  - $\Rightarrow I_{skill} = 5/10 \; I(3/5, 2/5) \simeq 0{,}485$
  - $\Rightarrow I_{chance} = 5/10 \; I(1/5, 4/5) \simeq 0{,}361$

(8) plotting the data (limiting both coordinates to 50)



C1, C2 coincide when $x=5$ and $y=25$

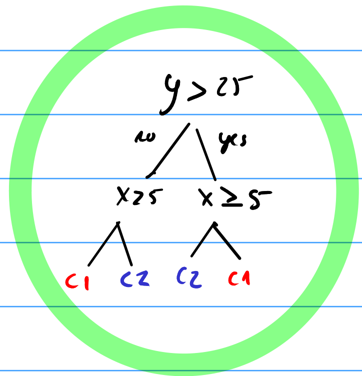
⇒ given the shape of the curve we plotted we can say that an optimum first split would have to occur over this point

⇒ choosing either splitting over $x<5$ or $y<25$

- the total entropy is $-\left(1/2 \log(1/2) + 1/2 \log(1/2)\right) = 1$
- $\Delta_H(x<5) = 1 - I_{x<5} - I_{x\geq5} = 0$
  - $I_{x<5} = 8/100 \; I(4/8, 4/8) = 0{,}04$
  - $I_{x\geq5} = 192/200 \; I(1/2, 1/2) = 0{,}96$
- $\Delta_H(y<25) = 1 - I_{y\geq25} - I_{y<25} = 0{,}7356$
  - $I_{y<25} = 99/200 \; I(4/99, 95/99) \simeq 0{,}1209$
  - $I_{y\geq25} = 101/200 \; I(96/101, 5/101) \simeq 0{,}1436$

⇒ this result was intuitive after looking at the plotted data

⇒ this means that we need to split over $x<5$ in the next leaf for both branches



⇒ after computing the optimal tree, we can see that 199/200 points are correctly classified. With the one error being the overlapping point of both classes. Since it shares the same coordinates, there is no possible decision tree that predicts two different classes for the same datapoint.

⇒ We can also see that there are equivalent trees, namely by changing the split values to the opposite region $(x>5, y>25)$ or, in hindsight, since we end up splitting over $x<5$ in both branches anyway it would result in a tree with the same accuracy and depth if we chose it as the first split.

exercise_02_notebook

November 3, 2021

1 Programming assignment 1: k-Nearest Neighbors classification

[ ]:
```
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).

In addition, we strongly recommend you to solve this task **without a single for loop**, i.e., only via vectorized (`numpy`) operations.

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`.

[ ]:
```
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
```

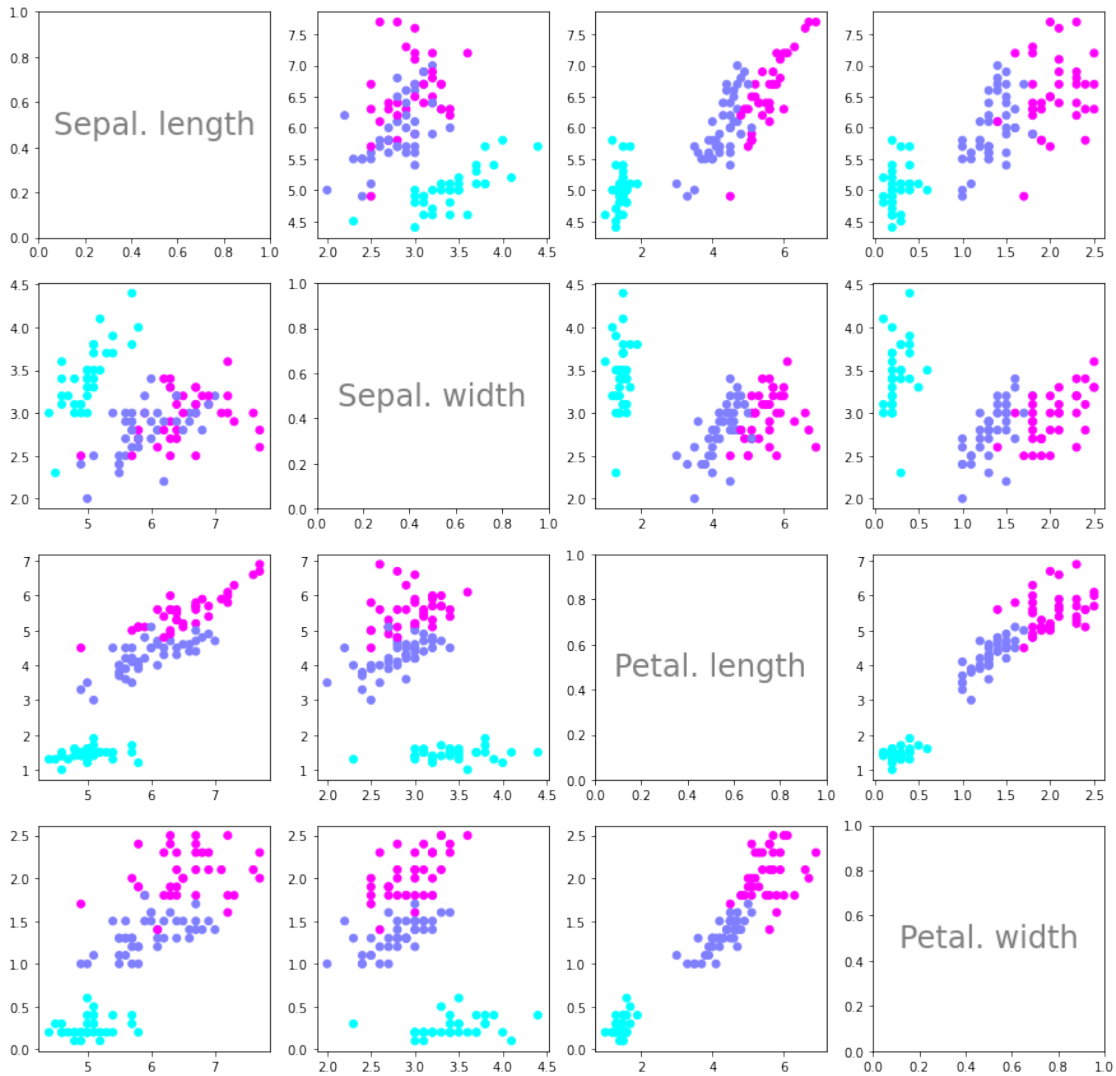
[ ]:
```
# prepare data
split = 0.75
X_train, X_test, y_train, y_test = load_dataset(split)
```

1.5 Plot dataset

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

[ ]:
```
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.

[ ]:
```python
def euclidean_distance(x1, x2):
    """Compute pairwise Euclidean distances between two data points.

    Parameters
    ----------
    x1 : array, shape (N, 4)
        First set of data points.
    x2 : array, shape (M, 4)
        Second set of data points.

    Returns
    -------
    distance : float array, shape (N, M)
        Pairwise Euclidean distances between x1 and x2.
    """
    return np.linalg.norm(x1[:, None, :]-x2, axis=-1)
```

1.7 Task 2: get k nearest neighbors' labels

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

[ ]:
```python
def get_neighbors_labels(X_train, y_train, X_new, k):
    """Get the labels of the k nearest neighbors of the datapoints x_new.

    Parameters
    ----------
    X_train : array, shape (N_train, 4)
        Training features.
    y_train : array, shape (N_train)
        Training labels.
    X_new : array, shape (M, 4)
        Data points for which the neighbors have to be found.
    k : int
        Number of neighbors to return.

    Returns
    -------
    neighbors_labels : array, shape (M, k)
        Array containing the labels of the k nearest neighbors.
    """

    distances = euclidean_distance(X_new, X_train)
    idx = np.argpartition(distances, k, axis=1) [:,:k]
```

```
    return y_train[idx]
```

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 $0 > 1 > 2$).

[ ]:
```
def get_response(neighbors_labels, num_classes=3):
    """Predict label given the set of neighbors.

    Parameters
    ----------
    neighbors_labels : array, shape (M, k)
        Array containing the labels of the k nearest neighbors per data point.
    num_classes : int
        Number of classes in the dataset.

    Returns
    -------
    y : int array, shape (M,)
        Majority class among the neighbors.
    """

    # since bincount sorts the bins per value the tie resolution order is␣
↪preserved
    freq = np.apply_along_axis(lambda x: np.bincount(x, minlength=num_classes),␣
↪axis=1, arr=neighbors_labels)
    majority = np.argmax(freq, axis=1)

    return majority
```

1.9 Task 4: compute accuracy

Compute the accuracy of the generated predictions.

[ ]:
```
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.
    """

    return y_pred[y_pred == y_test].size / y_pred.size
```

[ ]:
```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.
    """

    neighbors = get_neighbors_labels(X_train, y_train, X_test, k)
    y_pred = get_response(neighbors)
    return y_pred
```

1.10 Testing

Should output an accuracy of 0.9473684210526315.

[ ]:
```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
```