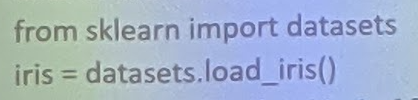
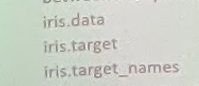
MACHINE LEARNING WITH SKICIT-LEARN

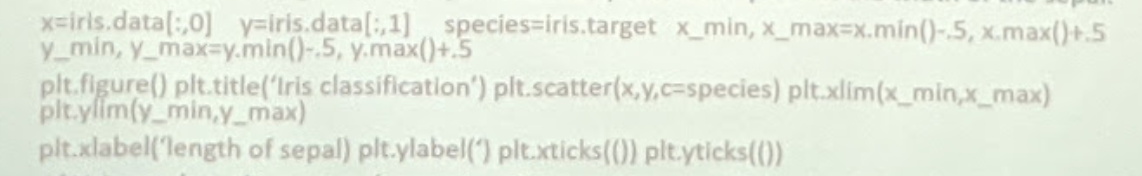
1/ Download iris data and use skicit-learn to import. Try to call the attribute data of the variable iris.

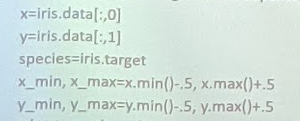


2/ How to know what kind of flower belongs each item? How to know the correspondence between the species and the number?



3/ Create a scatter plot that displays three different species in three different colors; X-axis will represent the length of the sepal while the y-axis will represent the width of the sepal.





4/ Using reduce dimension, here using PCA, create a new dimension (=3, called principle component).

from sklearn.decomposition import PCA

from mpl\_tooikits.mplot3d import Axes3D

x\_reduced=PCA(n\_components=3).fit\_transform(iris.data)

fig=plt.figure() ax=Axes3D(fig) ax.set\_title(')

ax.scatter(x\_reduced[:,0],x\_reduced[:,1],x\_reduced[:,2],c=species)

ax.set\_xlabel(‘) ax.set\_ylable() ax.set\_zlabel(')

5/ Using k-nearest neighbor to classify the group that each species belongs to. First, create a training set and test set; with 140 will be used as a training set, and 10 remaining will be used as test set.

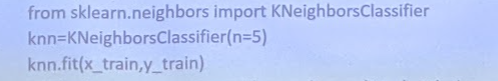
import numpy as np

np.random.seed(0)

x= iris.data y= iris.target i=np.random.permutation(len(iris.data))

x\_train=x[i[:-10l] y\_train=y[i[ :-10]] x\_test=x[i[-10:]] y\_test=y[i[-10:]]

6/ Next, apply the K-nearest neighbor, try with K=5.



7/ Finally, you can compare the results predicted with the actual observed contained in the y\_test.

8/ Now, you can visualize all this using decision boundaries in a space represented by the 2D scatterplot of sepals.

9/ Download diabete dataset. To predict the model, we use the linear regression.

10/ First, you will need to break the dataset into a training dataset (composed of the first 422 patients) and a test set (the last 20 patients).

11/ Now, apply the training set to predict the model?

12/ How can get the ten b coefficient calculated once the model is trained.

13/ If you apply the test set to the linear regression prediction, you will get a series of a target to be compared with the value actually observed.

14/ How to check the optimum of the prediction.

15/ Now, you will start with the linear regression taking into account a single physiological factor, for example, you can start with the age.

16/ Actually, you have 10 physiological factors within the diabetes dataset. Therefore, to have a more complete picture of all the training set, you can make a linear regression for every physiological feature, creating 10 models and seeing the result for each of them through a linear chart.

17/ Using skicit-learn download the breast cancer dataset of Winconstn university. Print the key of this dictionary from sklearn.datasets Import load breast cancer

cancer=load\_breast\_cancer()

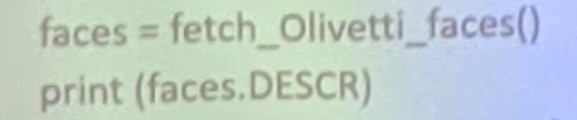
print("cancer.keys(): \n()".format(cancer.keys()))

18/ Check the shpae of the data. Count the number of "benign" tumor and "maglinant" tumor.

19/ Split the data into the training and a test set. After that, evaluate the training and test set performance with the different number of neighbors (from 1 to 10). Make a visualization.

20/ Download mglearn library. Using make\_forge dataset. Compare Logistic Regression and Linear SVC

21/ We will apply SVM to image recognition. Our learning set will be a group of labelled images of peoples' faces. Now let's start by importing and printing its description.



22/ Looking at the content of the faces object, we get the following properties: images, data and target. print (faces.keys())

print faces.images.shape print faces.data.shape print faces.target.shape

23/ Before learning, let's plot some faces. Please define a helper function. define

plot\_faces(image,target,top\_n):

fig = plt.figure(figsize=(12,12))

for i in range(top\_n):

p=fig.add\_suplot(20,20,i+1)

p.imshow(image[i),cmap=...)

24/ The SVC implementation has different important parameters; probably the most relevant is kernel. To start, we will use the simplest kernel, the linear one.

25/ Before continuing, we will split our dataset into training and testing datasets.

26/ And we will define a function to evaluate K-fold cross-validation.

27/ We will also define a function to perform training on the training set and evaluate the performance on the testing set.

28/ If we train and evaluate, the classifier performs the operation with almost no errors. Check

29/ Then we'll define a function that from those segments returns a new target array that marks with 1 for the faces with glasses and 0 for the faces without glasses (our new target classes).

def create\_target(segments):

y=np.zeros(faces.target.shape(O))

for (start,end) in segments:

y[start:end+1]=1

return y

target\_glasses=create\_target(glasses)

30/So we must perform the training/testing split again. Now let's create a new SVC classifier, and train it with the new target vector.

X\_train,X\_test,y\_train,y\_test=train\_test\_split(faces.data,target\_glasses,

test\_size=.25,random\_state=0)

31/ Check the performance with cross-validation. We obtain a mean accuracy of 0.967 with cross-validation if we evaluate on our testing set.

32/ Let's separate all the images of the same person, sometimes wearing glasses and sometimes not. We will also separate all the images of the same person, the ones with indexes from 30 to 39, train by using the remaining instances, and evaluate on our new IO instances set. With this experiment we will try to discard the fact that it is remembering faces, not glassed-related features.

X\_test=faces.data(30:40), y\_test=target\_glasses[30:40]

select=np.ones(target\_glasses.shape[0])

select[30:40]=0, X\_train=faces.data(select==1),y\_train=target\_glasses(select==1)

svc\_3 = SVC(kernel=’linear’),

train \_and\_evaluate(svc\_3,X\_train,X\_test,y\_train,y\_test)(defined before)

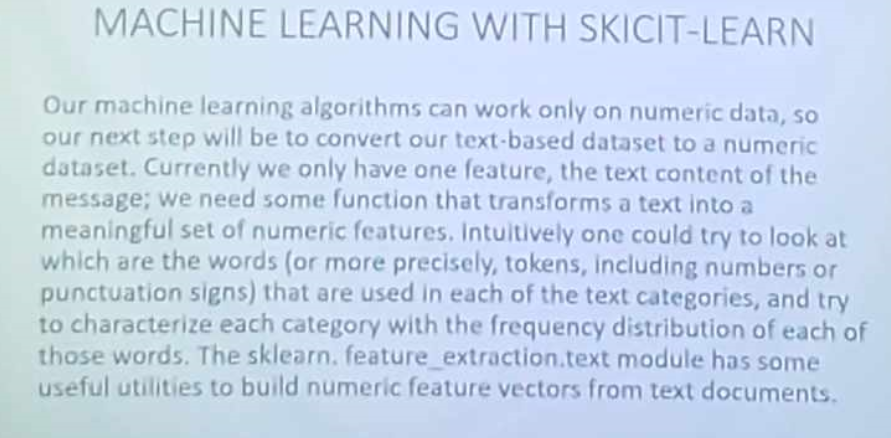
34/ Our dataset can be obtained by Importing the fetch\_20newgroups function from the sklearn.datasets module. We have to specify If we want to import a part or all of the set of instances (we Will import all of them).

from sklearn.datasets import fetch\_20newgroups

news=fetch\_20newgroups(subset='all')

35/ If we jook at the properties of the datasct, we will find that we have the usual ones: DESCR, data, target. and target\_names

36/ Preprocessing the data: before starting the transformation. we Will have to partition our data into training and testines set, The loaded data is already in a random order, so we only hove to split the data into, for example, 75 percent for training and the rest 25 percent for testing.



UNSUPERVISED LEARNING

1. As working example, in this section we willl use a dataset of handwritten digits digitalized in matrices of 8x8 pixels, so each instance will consist initially of 64 attributes. We start by loading our dataset and print it’s key

( MARKOV DECISION PROCESS - DATASET)

Sklearn.satasets import load\_digits

Digits=load\_digit()

X\_digits, y\_digits=digits.data, digits.target

1. We will use the data matrix that has the instances of 64 attributes. Each and the target vector that has the corresponding digit number. Let us define a function for printing the digits to take a look at how the instances will appear

import matplotlib.pyplot as plt

from sklearn.datasets import load\_digits

digits = load\_digits()

x\_digits, y\_digits = digits.data, digits.target

n\_row, n\_col = 2, 5

def print\_digits(images, y, max\_n=10):

fig = plt.figure(figsize=(2 \* n\_col, 2 \* n\_row))

i = 0

while i < max\_n and images.shape[0]:

p = fig.add\_subplot(n\_row, n\_col, i + 1, xticks=[], yticks=[])

p.imshow(images[i].reshape((8, 8)), cmap=plt.cm.bone, interpolation='nearest')

p.text(0, -1, str(y[i]))

i = i + 1

print\_digits(digits.images, digits.target)

plt.show()

1. Define a function that will plot a scatter with the two-dimensional points that will also be colored according to their classes.

From sklearn.decomposition import PCA

Estimator = PCA(n\_components=10)

X\_pca=estimator.fit\_transform(X\_digits)

Plot\_pca\_scatter

1. Now we perform the transformation and plot the results.
2. We will plot all the components in the same shape as the original data (digits)
3. We will show in this section how k-means work using a motivating example, the problem of clustering handwritten digits. So, let us first import our dataset into our Python environment and show how handwritten digits look.
4. As usual, we must separate train and testing sets.

Markov decision process

Multilayter network

relations