Lab Exercises - Pattern Recognition

Ing. Viktor Kocur, PhD.

Faculty of Mathematics, Physics and Informatics

Comenius University in Bratislava

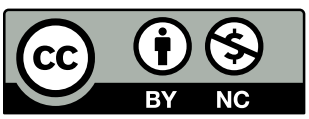
Názov:
Lab Exercises - Pattern Recognition

Autor:
Ing. Viktor Kocur, PhD.
Katedra Aplikovanej Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského v Bratislave

Vydavateľ:
Fakulta matematiky, fyziky a informatiky UK
Knižničné a edicné centrum
Bratislava 2024
1. vydanie

ISBN:
978-80-8147-139-1

Fotografie a obrázky: vlastná tvorba, public domain fotografie z USFWS National Digital Library



Contents

Introduction

This document contains a set of solved exercises for the course 2-AIN-204/10 Pattern Recognition taught at Faculty of Mathematics, Physics and Informatics of the Comenius University in Bratislava. The exercises are presented in the form of a Jupyter Notebook. In the notebook format there are two types of cells. Text cells such as this one present a formatted text. The other type of cell is the code cell where the user can write python code and execute it. After execution any output of the given cell code is presented below the code cell. Sometimes the output of a cell is too long. In such cases some of the output is ommited to make the document more readable. Such instances are denoted in the outputs.

Originally the exercises were prepared for each week individually. They can be accessed in this form here: `https://github.com/kocurvik/edu/tree/master/RO/cvicenia/notebooky`. The individual notebooks as well as this one can be run locally or using the Google Colab service.

The chapters in this document roughly correspond to the original invidiual notebooks. For this reason the imports of the relevant packages as well as downloads of the data will be repeated for each chapter to make them easier to run individually. An online version of this document is also available at https://github.com/kocurvik/edu/blob/master/pubs/RO_all_en.ipynb. A button is placed at the beginning of the notebook which enables the readers to open the notebook in the Google Colab service.

In the current pdf version of the notebook the solutions to the exercises are provided at the end of the document with links included for convenience. It is also possible to toggle the display of the solution code direcly in the block right after the exercise by clicking "Click to Show/Hide Here", thus making it unnecessary to jump to the end of the notebook. This functionality only works in some pdf viewers and was tested using Acrobat Version 2023.006.20380.

The photographic images used in this document are in the public domain and were provided by the USFWS National Digital Library.

Chapter 1

NumPy

NumPy is a very useful library that enables us to work efficiently with multidimensional arrays in python. Multidimensional arrays are useful in many computer science applications. In this chapter we will cover the basics of using NumPy.

The documentation for numpy can be found here: https://numpy.org/doc/stable/.

1.1 NumPy Arrays

To conform to standards we `import numpy as np`. A vector can be created from a simple list using the `np.array` constructor. It might be important to initialize the array with a specific data type which can be done by using the `dtype` keyword in the constructor.

[1]:
```
import numpy as np

a = np.array([3, 2, 3, 4])
print(a)
b = np.array([1,5,7], dtype=np.float32)
print(b)
```

```
[3 2 3 4]
[1. 5. 7.]
```

A matrix can be created from a list of lists, but it is necessary for the lengths of the inner lists to be consitent. If they are not consistent we will obtain a vector of lists instead of a matrix of values.

[2]:
```
A = np.array([[1,5,8],[50,60,84]])
print(A)
B = np.array([[7,8],[6,7],[0,9,4]])
print(B)
```

```
[[ 1  5  8]
 [50 60 84]]
[list([7, 8]) list([6, 7]) list([0, 9, 4])]
```

```
<ipython-input-2-575e0ddf66be>:3: VisibleDeprecationWarning: Creating an ndarray
from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or
ndarrays with different lengths or shapes) is deprecated. If you meant to do
this, you must specify 'dtype=object' when creating the ndarray.
  B = np.array([[7,8],[6,7],[0,9,4]])
```

The array object has a shape method which tells us its shape (dimensions) and method `dtype` which returns the data type of the values. It also has a method `astype` which returns an array with the specified type.

[3]:
```
print(a.shape)
print(A.shape)
print(A.dtype)
C = A.astype(np.float32)
print(C.dtype)
```

```
(4,)
(2, 3)
int64
float32
```

Numpy has a few functions to generate basic arrays. The most commnly used onse are: `np.zeros`, `np.ones` and `np.empty`.

[4]:
```
z = np.zeros([5,10])
print(z)
o = np.ones([3,4,5])
print(o)
e = np.empty([6])
print(e)
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[[1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]]

 [[1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]]

 [[1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]]]
```

```
[4.80226774e-310 0.00000000e+000 0.00000000e+000 0.00000000e+000
 0.00000000e+000 0.00000000e+000]
```

It is also possible to create arrays with random numbers by using np.random.random.

[5]:
```
r = np.random.random([6,10,3])
print(r)
```

```
[[[0.03927402 0.25997187 0.12050436]
  [0.22788362 0.99401703 0.73891044]
  [0.70776797 0.98027064 0.5608338 ]
  [0.14826953 0.63750554 0.01358893]
  [0.30155577 0.1737418  0.02625739]
  [0.80391756 0.21859913 0.4713636 ]
  [0.84064278 0.81182762 0.41160772]
  [0.67596575 0.62617364 0.00658714]
  [0.14767122 0.14422655 0.25105898]
  [0.10734818 0.67028693 0.6744029 ]]

 [[0.33654621 0.53477279 0.37479022]
  [0.57196043 0.43298301 0.59938077]
  [0.89820847 0.41622304 0.35835855]
  [0.13385084 0.45610268 0.11319234]
  [0.36970368 0.05891678 0.32715475]
  [0.0832755  0.37044039 0.81251564]
  [0.48264324 0.52318288 0.95310172]
  [0.58550107 0.03407844 0.01396058]
  [0.45930699 0.91181918 0.74284367]
  [0.29610823 0.38589167 0.44894221]]

 [[0.72333274 0.95029282 0.77356166]
  [0.94113196 0.09746667 0.71708081]
  [0.53237485 0.16153711 0.70895207]
  [0.82323089 0.70809912 0.99706874]
  [0.62508163 0.21008885 0.38195205]
  [0.62169499 0.82209616 0.33027538]
  [0.55436987 0.06374548 0.81707047]
  [0.09180143 0.96966071 0.8451665 ]
  [0.07811964 0.93258348 0.94550044]
  [0.59742108 0.94799824 0.97929486]]

 [[0.792741   0.07876859 0.4006137 ]
  [0.0359317  0.01097506 0.65782585]
  [0.47936689 0.55814856 0.45013732]
  [0.0562282  0.69710772 0.47756422]
  [0.25392366 0.81906964 0.22034452]
  [0.72214868 0.31608735 0.09032434]
  [0.96278365 0.6727497  0.66687074]
  [0.19635573 0.08078596 0.16489624]
```

```
  [0.73040113 0.40141052 0.58033061]
  [0.4867363  0.9551071  0.18285063]]

 [[0.34191587 0.19884808 0.3331601 ]
  [0.30178991 0.05589621 0.40595726]
  [0.33785696 0.92865667 0.42812781]
  [0.1442013  0.78284654 0.44590454]
  [0.08747589 0.961954   0.8049251 ]
  [0.3725808  0.90889039 0.99645125]
  [0.10387132 0.91414762 0.5114261 ]
  [0.2581524  0.06769299 0.87888835]
  [0.39755972 0.90474845 0.57987556]
  [0.18814986 0.71542441 0.68866328]]

 [[0.52374467 0.37287343 0.63505648]
  [0.10753066 0.76419508 0.64286767]
  [0.42293865 0.07485525 0.08195524]
  [0.38496033 0.03548383 0.97481697]
  [0.78662416 0.12826309 0.24548125]
  [0.58980218 0.08188186 0.25231437]
  [0.12312953 0.3514306  0.55861394]
  [0.24797484 0.96914889 0.93665439]
  [0.11456477 0.76838557 0.05460265]
  [0.12700855 0.28440565 0.06367693]]]
```

We can now use these random arrays to work with indices. Indexing is similar to Matlab, but the python conventions are applied. We can use multiple indices (based on tensor order). If we omit one of the indices it is implicitly used as :, which means that all of the elements along that dimesnsions are used.

[6]:
```
print(r[3,4,1])
print(r[:,:,-1])
print(r[:,:,1].shape)
print(r[0:4,5:6,:])
print(r[0:4,5:6,:].shape)
```

```
0.8190696437510293
[[0.12050436 0.73891044 0.5608338  0.01358893 0.02625739 0.4713636
  0.41160772 0.00658714 0.25105898 0.6744029 ]
 [0.37479022 0.59938077 0.35835855 0.11319234 0.32715475 0.81251564
  0.95310172 0.01396058 0.74284367 0.44894221]
 [0.77356166 0.71708081 0.70895207 0.99706874 0.38195205 0.33027538
  0.81707047 0.8451665  0.94550044 0.97929486]
 [0.4006137  0.65782585 0.45013732 0.47756422 0.22034452 0.09032434
  0.66687074 0.16489624 0.58033061 0.18285063]
 [0.3331601  0.40595726 0.42812781 0.44590454 0.8049251  0.99645125
  0.5114261  0.87888835 0.57987556 0.68866328]
 [0.63505648 0.64286767 0.08195524 0.97481697 0.24548125 0.25231437
  0.55861394 0.93665439 0.05460265 0.06367693]]
```

```
(6, 10)
[[[0.80391756 0.21859913 0.4713636 ]]

 [[0.0832755  0.37044039 0.81251564]]

 [[0.62169499 0.82209616 0.33027538]]

 [[0.72214868 0.31608735 0.09032434]]]
(4, 1, 3)
```

We can also use steps when indexing this is also called slicing. That can be done by another colon. The format is then `[start:stop:step]`. If any of that is left empty the start is implicitly assumed to be 0, the end -1 and step 1.

[7]:
```
p = np.arange(25)
print(p)
print(p[4:16:2])
print(p[2:-4:6])
print(p[:10:])
print(p[::3])
print(p[1::6])
print(p[-6:])
print(r[1::2,0::3,:])
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24]
[ 4  6  8 10 12 14]
[ 2  8 14 20]
[0 1 2 3 4 5 6 7 8 9]
[ 0  3  6  9 12 15 18 21 24]
[ 1  7 13 19]
[19 20 21 22 23 24]
[[[0.33654621 0.53477279 0.37479022]
  [0.13385084 0.45610268 0.11319234]
  [0.48264324 0.52318288 0.95310172]
  [0.29610823 0.38589167 0.44894221]]

 [[0.792741   0.07876859 0.4006137 ]
  [0.0562282  0.69710772 0.47756422]
  [0.96278365 0.6727497  0.66687074]
  [0.4867363  0.9551071  0.18285063]]

 [[0.52374467 0.37287343 0.63505648]
  [0.38496033 0.03548383 0.97481697]
  [0.12312953 0.3514306  0.55861394]
  [0.12700855 0.28440565 0.06367693]]]
```

We will often need to create a so-called singleton dimension. This can be done by adding a `None` index, or in a more verbose way by using `np.newaxis`.

[8]:
```
print(r[None,:,:,:].shape)
print(r[None].shape)
print(r[np.newaxis,:,:,:].shape)
print(r[:,:,:,np.newaxis].shape)
print(r[np.newaxis,:,0,:].shape)
```

```
(1, 6, 10, 3)
(1, 6, 10, 3)
(1, 6, 10, 3)
(6, 10, 3, 1)
(1, 6, 3)
```

Similar to Matlab broadcasting is done implicitly (though different rules are applied in Matlab).

[9]:
```
r += 10
print(r)
r[0,:,:] = np.random.random([10,3])
print(r.shape)
r[0] = np.random.random([10,3])
print(r.shape)
r[0] = np.zeros([10,1])
print(r.shape)
r/=500
print(r)
```

```
[[[10.03927402 10.25997187 10.12050436]
  [10.22788362 10.99401703 10.73891044]
  [10.70776797 10.98027064 10.5608338 ]
  [10.14826953 10.63750554 10.01358893]
  [10.30155577 10.1737418  10.02625739]
  [10.80391756 10.21859913 10.4713636 ]
  [10.84064278 10.81182762 10.41160772]
  [10.67596575 10.62617364 10.00658714]
  [10.14767122 10.14422655 10.25105898]
  [10.10734818 10.67028693 10.6744029 ]]

 [[10.33654621 10.53477279 10.37479022]
  [10.57196043 10.43298301 10.59938077]
  [10.89820847 10.41622304 10.35835855]
  [10.13385084 10.45610268 10.11319234]
  [10.36970368 10.05891678 10.32715475]
  [10.0832755  10.37044039 10.81251564]
  [10.48264324 10.52318288 10.95310172]
  [10.58550107 10.03407844 10.01396058]
  [10.45930699 10.91181918 10.74284367]
  [10.29610823 10.38589167 10.44894221]]

 [[10.72333274 10.95029282 10.77356166]
  [10.94113196 10.09746667 10.71708081]
```

```
  [10.53237485 10.16153711 10.70895207]
  [10.82323089 10.70809912 10.99706874]
  [10.62508163 10.21008885 10.38195205]
  [10.62169499 10.82209616 10.33027538]
  [10.55436987 10.06374548 10.81707047]
  [10.09180143 10.96966071 10.8451665 ]
  [10.07811964 10.93258348 10.94550044]
  [10.59742108 10.94799824 10.97929486]]

 [[10.792741   10.07876859 10.4006137 ]
  [10.0359317  10.01097506 10.65782585]
  [10.47936689 10.55814856 10.45013732]
  [10.0562282  10.69710772 10.47756422]
  [10.25392366 10.81906964 10.22034452]
  [10.72214868 10.31608735 10.09032434]
  [10.96278365 10.6727497  10.66687074]
  [10.19635573 10.08078596 10.16489624]
  [10.73040113 10.40141052 10.58033061]
  [10.4867363  10.9551071  10.18285063]]

 [[10.34191587 10.19884808 10.3331601 ]
  [10.30178991 10.05589621 10.40595726]
  [10.33785696 10.92865667 10.42812781]
  [10.1442013  10.78284654 10.44590454]
  [10.08747589 10.961954   10.8049251 ]
  [10.3725808  10.90889039 10.99645125]
  [10.10387132 10.91414762 10.5114261 ]
  [10.2581524  10.06769299 10.87888835]
  [10.39755972 10.90474845 10.57987556]
  [10.18814986 10.71542441 10.68866328]]

 [[10.52374467 10.37287343 10.63505648]
  [10.10753066 10.76419508 10.64286767]
  [10.42293865 10.07485525 10.08195524]
  [10.38496033 10.03548383 10.97481697]
  [10.78662416 10.12826309 10.24548125]
  [10.58980218 10.08188186 10.25231437]
  [10.12312953 10.3514306  10.55861394]
  [10.24797484 10.96914889 10.93665439]
  [10.11456477 10.76838557 10.05460265]
  [10.12700855 10.28440565 10.06367693]]]
(6, 10, 3)
(6, 10, 3)
(6, 10, 3)
[[[0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
```

```
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]

 [[0.02067309 0.02106955 0.02074958]
  [0.02114392 0.02086597 0.02119876]
  [0.02179642 0.02083245 0.02071672]
  [0.0202677  0.02091221 0.02022638]
  [0.02073941 0.02011783 0.02065431]
  [0.02016655 0.02074088 0.02162503]
  [0.02096529 0.02104637 0.0219062 ]
  [0.021171   0.02006816 0.02002792]
  [0.02091861 0.02182364 0.02148569]
  [0.02059222 0.02077178 0.02089788]]

 [[0.02144667 0.02190059 0.02154712]
  [0.02188226 0.02019493 0.02143416]
  [0.02106475 0.02032307 0.0214179 ]
  [0.02164646 0.0214162  0.02199414]
  [0.02125016 0.02042018 0.0207639 ]
  [0.02124339 0.02164419 0.02066055]
  [0.02110874 0.02012749 0.02163414]
  [0.0201836  0.02193932 0.02169033]
  [0.02015624 0.02186517 0.021891  ]
  [0.02119484 0.021896   0.02195859]]

 [[0.02158548 0.02015754 0.02080123]
  [0.02007186 0.02002195 0.02131565]
  [0.02095873 0.0211163  0.02090027]
  [0.02011246 0.02139422 0.02095513]
  [0.02050785 0.02163814 0.02044069]
  [0.0214443  0.02063217 0.02018065]
  [0.02192557 0.0213455  0.02133374]
  [0.02039271 0.02016157 0.02032979]
  [0.0214608  0.02080282 0.02116066]
  [0.02097347 0.02191021 0.0203657 ]]

 [[0.02068383 0.0203977  0.02066632]
  [0.02060358 0.02011179 0.02081191]
  [0.02067571 0.02185731 0.02085626]
  [0.0202884  0.02156569 0.02089181]
  [0.02017495 0.02192391 0.02160985]
  [0.02074516 0.02181778 0.0219929 ]
  [0.02020774 0.0218283  0.02102285]
  [0.0205163  0.02013539 0.02175778]
```

```
  [0.02079512 0.0218095  0.02115975]
  [0.0203763  0.02143085 0.02137733]]

 [[0.02104749 0.02074575 0.02127011]
  [0.02021506 0.02152839 0.02128574]
  [0.02084588 0.02014971 0.02016391]
  [0.02076992 0.02007097 0.02194963]
  [0.02157325 0.02025653 0.02049096]
  [0.0211796  0.02016376 0.02050463]
  [0.02024626 0.02070286 0.02111723]
  [0.02049595 0.0219383  0.02187331]
  [0.02022913 0.02153677 0.02010921]
  [0.02025402 0.02056881 0.02012735]]]
```

An array can be reshaped using np.reshape

[10]:
```
q = np.reshape(r, [6,30])
print(q.shape)
```

```
(6, 30)
```

Arrays can be joined. One way is to use `np.concatenate([arr1, arr2, ...], axis = i)` which connects the arrays `arr1` and `arr2` through a given axis `i`. Note that is usually better (for code comprehension) to explicitly use the keyword axis even if it is redundant.

[11]:
```
a = np.ones([3,4])
b = np.zeros([6,4])
c = np.concatenate([a,b], axis = 0)
print(c)
```

```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

A different option is to use `np.stack([arr1, arr2, ...], axis= i)`. The difference is that this creates increases the tensor order by one.

[12]:
```
a = np.ones([6,4,3])
b = np.zeros([6,4,3])
d = np.stack([a,b], axis = 0)
print(d.shape)
f = np.stack([a,b], axis = -1)
print(f.shape)
g = np.stack([a,b], axis = 1)
```

```
print(g.shape)
```

```
(2, 6, 4, 3)
(6, 4, 3, 2)
(6, 2, 4, 3)
```

We can also index using conditions.

[13]:
```
r = np.random.random([5,5])
print(r)
r[r < 0.5] = -500
print(r)
```

```
[[0.69235829 0.07094077 0.55038478 0.99776445 0.39053001]
 [0.31765605 0.36999447 0.69777561 0.44595678 0.35492434]
 [0.39412525 0.94930968 0.55739847 0.72763678 0.18216753]
 [0.75681375 0.77504905 0.04851184 0.3163624  0.17033864]
 [0.61876904 0.73954674 0.12856628 0.10270831 0.83073472]]
[[   0.69235829 -500.          0.55038478    0.99776445 -500.        ]
 [-500.         -500.          0.69777561 -500.         -500.        ]
 [-500.            0.94930968    0.55739847    0.72763678 -500.        ]
 [   0.75681375    0.77504905 -500.         -500.         -500.        ]
 [   0.61876904    0.73954674 -500.         -500.            0.83073472]]
```

The conditions can also be using in `np.where(cond, a1, a2)`, which returns an array which contans elemnts from `a1` where the condition is true and elements from `a2` where condition is false. We can also use `np.arange(i)` which is equivalent to `range(i)` from python.

[14]:
```
a = np.arange(10)
print(a)
b = np.where(a < 5, a, a**2)
print(b)
```

```
[0 1 2 3 4 5 6 7 8 9]
[ 0  1  2  3  4 25 36 49 64 81]
```

1.1.1 Exercise - Generating a chessboard

Create a funciton `chessboard(rows, cols)` which returns a numpy array with dimensions `rows` × `cols`. The array will contain 1 in spots where a chessboard tile would be white and 0 where a chessboard tile would be black.

The simplest way is to use slicing. If you feel adventurous you can check the NumPy documentation and use conditions with `np.mgrid` or repetition and `np.tile`.

Solution 1.1.1

Click to Show/Hide Here or Navigate to Solution 1.1.1 on page 135

[ ]:
```

```

[16]:
```
print(chessboard(8,8))
print(chessboard(5,12))
```

```
[[0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]]
[[0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1.]]
```

1.2 Numpy operations

It is possible to perform various operations on Numpy arrays. It is possible to add up the elements with `np.sum`. There are also other similar methods such as `np.min`, `np.max`, `np.mean`, `np.median`, `np.std` and `np.var`.

[17]:
```
import numpy as np

a = np.random.rand(4, 3)
print(a)
print(np.mean(a))
print(np.median(a))
print(np.sum(a))
print(np.std(a))
```

```
[[0.80081112 0.71100519 0.58970208]
 [0.83916361 0.77021306 0.46115738]
 [0.17934264 0.24352752 0.29182282]
 [0.91800577 0.15061649 0.26248953]]
0.5181547674647632
0.5254297301635253
6.217857209577158
0.27291688915671997
```

1.2.1 The axis keyword

We will quite often intend to only perform a certain operation along a given dimension. In that case it is suitbale to use the `axis` keyword. As the argument you can select the dimension along which

to perform the opration. It is also possible to use a tuple to perform the operation along multiple axes.

[18]:
```
print(a)
print(np.sum(a, axis=0))
print(np.sum(a, axis=0).shape)
print(np.sum(a, axis=1))
print(np.sum(a, axis=1).shape)

b = np.random.rand(4,3,16,16)
print(np.sum(b, axis=(1,2,3)))
print(np.sum(b, axis=(1,2,3)).shape)
```

```
[[0.80081112 0.71100519 0.58970208]
 [0.83916361 0.77021306 0.46115738]
 [0.17934264 0.24352752 0.29182282]
 [0.91800577 0.15061649 0.26248953]]
[2.73732315 1.87536225 1.60517181]
(3,)
[2.10151839 2.07053405 0.71469298 1.33111179]
(4,)
[391.10675683 371.32755097 389.65614014 380.90368525]
(4,)
```

1.2.2 Exercise - Euclidian measure

Implement the function `euclidian_distance_single(x, y)`, which takes two vecotrs x and y from $\mathbb{R}^n$ on input and calculates:

$$\rho_e(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Solution 1.2.2

Click to Show/Hide Here or Navigate to Solution 1.2.2 on page 136

[ ]:
```

```

We can test the code using the `np.linalg.norm`, which essentially calculates the same thing.

[20]:
```
for i in range(10):
  n = np.random.randint(100)
  a = np.random.rand(n)
  b = np.random.rand(n)

  d = euclidian_distance_single(a, b)
  d_gt = np.linalg.norm(a - b)
```

```
  if np.abs(d_gt - d) < 1e-8:
    print("Pass!")
  else:
    print("Fail")
```

```
Pass!
Pass!
Pass!
Pass!
Pass!
Pass!
Pass!
Pass!
Pass!
Pass!
```

1.3 Matrix multiplication

We can also multiply matrices. We can use `@` to perform standard matrix multiplication and `*` to perform elementwise multiplication. Another alternative for matrix multiplication is the `np.dot` function which behaves similarly to matrix multiplication, but if arrays of higher order are considered then it performs the operation along a certain axes.

We can also transpose an array by using the `.T` method.

[21]:
```
A = np.array([[3, 4], [1, 8]])
B = np.array([[1, 2], [5, 6]])
C = np.array([[1, 2, 3], [4, 5, 6]])

print(A * B)
print(A @ B)
print(A @ C)
print(C.T @ A)


print("Matrix vector multiplication")
x = np.array([1, 2, 3])
print(C @ x)
print(x.T @ C.T)
print(C * x)
```

```
[[ 3  8]
 [ 5 48]]
[[23 30]
 [41 50]]
[[19 26 33]
 [33 42 51]]
[[ 7 36]
```

```
 [11 48]
 [15 60]]
Matrix vector multiplication
[14 32]
[14 32]
[[ 1  4  9]
 [ 4 10 18]]
```

Chapter 2

Pandas

We will use the Pandas library to store our data. If we only used numerical features we would be fine with just NumPy, but this is usually not sufficient for categorical data or more complex structures. More info on Pandas data structures can be found in this user guide.

[22]:
```
import pandas as pd
import numpy as np
```

2.1 Series

One of the basic datatypes is `Series`. It is essentially a more sophisticated 1-d array of data, which can include named indices.

[23]:
```
s = pd.Series(np.random.randn(5))
print(s)

s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])
print(s)

d = {"a": 0.0, "b": 1.0, "c": 2.0}
print(pd.Series(d))
print(pd.Series(d, index=["b", "c", "d", "a"]))
```

```
0    2.748390
1   -1.590515
2    1.307520
3   -1.537803
4   -0.274861
dtype: float64
a    0.128369
b   -0.382535
c   -1.720448
d   -0.087222
e    0.588637
```

```
dtype: float64
a    0.0
b    1.0
c    2.0
dtype: float64
b    1.0
c    2.0
d    NaN
a    0.0
dtype: float64
```

In many cases Series can be used in place of standard NumPy array.

[24]:
```
print(s.dtype)
print(s.shape)
print(s[0])
print(s[:3])
print(np.sum(s))
print(np.exp(s))
```

```
float64
(5,)
0.12836879982168492
a    0.128369
b   -0.382535
c   -1.720448
dtype: float64
-1.473198182755516
a    1.136972
b    0.682130
c    0.178986
d    0.916474
e    1.801531
dtype: float64
```

We can convert Series to a NumPy array.

[25]:
```
print(s.to_numpy())
```

```
[ 0.1283688  -0.3825345  -1.72044778 -0.08722183  0.58863713]
```

We can also perform indexing and standard operations on Series.

[26]:
```
print(2.5 * s + s)
print(s[s > s.median()])
print(s[[4, 3, 1]])
```

```
a    0.449291
b   -1.338871
c   -6.021567
d   -0.305276
```

```
e    2.060230
dtype: float64
a    0.128369
e    0.588637
dtype: float64
e    0.588637
d   -0.087222
b   -0.382535
dtype: float64
```

But beware! Operations rely on the defined indexes! This is different from NumPy.

[27]:
```
print(s[1:] + s[:-1])
n = s.to_numpy()
print(n[1:] + n[:-1])
```

```
a         NaN
b   -0.765069
c   -3.440896
d   -0.174444
e         NaN
dtype: float64
[-0.2541657  -2.10298229 -1.80766961  0.5014153 ]
```

A Series also behaves as a dict!

[28]:
```
print(s["a"])
s["c"] = 0.2
print(s)
```

```
0.12836879982168492
a    0.128369
b   -0.382535
c    0.200000
d   -0.087222
e    0.588637
dtype: float64
```

2.2 DataFrames

DataFrame is a 2-d structure of data. We will mostly use this structure!

It can be constructed from:

- Dict of 1D ndarrays, lists, dicts, or Series
- 2-D numpy.ndarray
- Structured or record ndarray
- A Series
- Another DataFrame

[29]:
```
d = {
    "one": pd.Series([1.0, 2.0, 3.0], index=["a", "b", "c"]),
    "two": pd.Series([1.0, 2.0, 3.0, 4.0], index=["a", "b", "c", "d"]),
}
print(pd.DataFrame(d))
print(pd.DataFrame(d, index=["d", "b", "a"]))
print(pd.DataFrame(d, index=["d", "b", "a"], columns=["two", "three"]))
print(10 * '*')
d = {"one": [1.0, 2.0, 3.0, 4.0], "two": [4.0, 3.0, 2.0, 1.0]}
print(pd.DataFrame(d))
print(10 * '*')
data2 = [{"a": 1, "b": 2}, {"a": 5, "b": 10, "c": 20}]
print(pd.DataFrame(data2))
print(10 * '*')
print(pd.DataFrame(np.random.randn(6, 3), index=list('abcdef'),
                   columns=['first', 'second', 'third']))
```

```
   one  two
a  1.0  1.0
b  2.0  2.0
c  3.0  3.0
d  NaN  4.0
   one  two
d  NaN  4.0
b  2.0  2.0
a  1.0  1.0
   two three
d  4.0   NaN
b  2.0   NaN
a  1.0   NaN
**********
   one  two
0  1.0  4.0
1  2.0  3.0
2  3.0  2.0
3  4.0  1.0
**********
   a   b     c
0  1   2   NaN
1  5  10  20.0
**********
      first    second     third
a -0.012544 -0.459674  0.354260
b -0.448890 -2.152327 -0.379489
c  0.646587  0.283817  0.454899
d -1.912000  2.058778 -1.379720
e  0.631358 -0.442478 -1.074486
f  2.404664  0.054204  0.606327
```

We can add new columns.

[30]:
```
df = pd.DataFrame(np.random.randn(6, 3), index=list('abcdef'),
                  columns=['first', 'second', 'third'])
df.insert(1, "bar", df["first"])
df.insert(4, "last", 5.0)
df.insert(5, "actually_last", ['car', 'tram', 'bus', 'car', 'bike', 'car'])
print(df)
```

```
      first       bar    second     third  last actually_last
a -0.109098 -0.109098 -0.255901  1.822014   5.0           car
b  0.049083  0.049083 -0.661713  0.287186   5.0          tram
c -0.377638 -0.377638 -1.142217 -2.417332   5.0           bus
d  0.712614  0.712614 -0.523564 -0.649470   5.0           car
e  0.160007  0.160007  0.941660 -1.197198   5.0          bike
f  0.800899  0.800899 -0.333121 -0.130231   5.0           car
```

We can also create a new column from existing columns.

[31]:
```
print(df.assign(new_column=df["first"] / df["second"]))
```

```
      first       bar    second     third  last actually_last  new_column
a -0.109098 -0.109098 -0.255901  1.822014   5.0           car    0.426328
b  0.049083  0.049083 -0.661713  0.287186   5.0          tram   -0.074176
c -0.377638 -0.377638 -1.142217 -2.417332   5.0           bus    0.330619
d  0.712614  0.712614 -0.523564 -0.649470   5.0           car   -1.361083
e  0.160007  0.160007  0.941660 -1.197198   5.0          bike    0.169920
f  0.800899  0.800899 -0.333121 -0.130231   5.0           car   -2.404230
```

Indexing can be performed in multiple ways: * Select column `df[col]` * Select row by label `df.loc[label]` * Select row by integer location `df.iloc[loc]` * Slice rows `df[5:10]` * Select rows by boolean vector `df[bool_vec]`

[32]:
```
print(df['first'])
print(df.loc['b'])
print(df.iloc[2])
```

```
a   -0.109098
b    0.049083
c   -0.377638
d    0.712614
e    0.160007
f    0.800899
Name: first, dtype: float64
first            0.049083
bar              0.049083
second          -0.661713
third            0.287186
last                  5.0
actually_last        tram
```

```
Name: b, dtype: object
first            -0.377638
bar              -0.377638
second           -1.142217
third            -2.417332
last                   5.0
actually_last          bus
Name: c, dtype: object
```

We can also perform operations on dataframes. Most NumPy functions work as well.

[33]:
```
df2 = df[['first', 'bar', 'second']]
print(df2)
print(2 * df2 + 1.0)
print(df2 - df2.iloc[0])
print(np.exp(df2))
```

```
      first       bar    second
a -0.109098 -0.109098 -0.255901
b  0.049083  0.049083 -0.661713
c -0.377638 -0.377638 -1.142217
d  0.712614  0.712614 -0.523564
e  0.160007  0.160007  0.941660
f  0.800899  0.800899 -0.333121
      first       bar    second
a  0.781804  0.781804  0.488198
b  1.098167  1.098167 -0.323425
c  0.244723  0.244723 -1.284435
d  2.425227  2.425227 -0.047128
e  1.320014  1.320014  2.883319
f  2.601799  2.601799  0.333758
      first       bar    second
a  0.000000  0.000000  0.000000
b  0.158181  0.158181 -0.405812
c -0.268540 -0.268540 -0.886316
d  0.821712  0.821712 -0.267663
e  0.269105  0.269105  1.197561
f  0.909997  0.909997 -0.077220
      first       bar    second
a  0.896643  0.896643  0.774218
b  1.050308  1.050308  0.515967
c  0.685478  0.685478  0.319111
d  2.039314  2.039314  0.592406
e  1.173519  1.173519  2.564233
f  2.227543  2.227543  0.716683
```

We can also load a csv. We will first download one. We will use the data from the dataset known as Census Income by authors Barry Becker and Ronny Kohavi.

[34]:
```
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.names
!tail -n 16 adult.names
```

```
--2024-01-11 14:25:45--  https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: ‘adult.data’

adult.data              [   <=>              ]   3.79M  7.17MB/s    in 0.5s

2024-01-11 14:25:46 (7.17 MB/s) - ‘adult.data’ saved [3974305]

--2024-01-11 14:25:46--  https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.names
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: ‘adult.names’

adult.names             [ <=>                ]   5.11K  --.-KB/s    in 0s

2024-01-11 14:25:46 (99.1 MB/s) - ‘adult.names’ saved [5229]

>50K, <=50K.

age: continuous.
workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov,
State-gov, Without-pay, Never-worked.
fnlwgt: continuous.
education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm,
Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th,
Preschool.
education-num: continuous.
marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed,
Married-spouse-absent, Married-AF-spouse.
occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial,
Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-
fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
relationship: Wife, Own-child, Husband, Not-in-family, Other-relative,
Unmarried.
race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
```

```
sex: Female, Male.
capital-gain: continuous.
capital-loss: continuous.
hours-per-week: continuous.
native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany,
Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran,
Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal,
Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia,
Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador,
Trinadad&Tobago, Peru, Hong, Holand-Netherlands.
```

[56]:
```
data = pd.read_csv('adult.data', header=None)
print(data)
print(data.iloc[0])
```

```
       0                  1       2            3   4                    5   \
0      39          State-gov   77516    Bachelors  13        Never-married
1      50   Self-emp-not-inc   83311    Bachelors  13   Married-civ-spouse
2      38            Private  215646      HS-grad   9             Divorced
3      53            Private  234721         11th   7   Married-civ-spouse
4      28            Private  338409    Bachelors  13   Married-civ-spouse
...    ..                ...     ...          ...  ..                  ...
32556  27            Private  257302   Assoc-acdm  12   Married-civ-spouse
32557  40            Private  154374      HS-grad   9   Married-civ-spouse
32558  58            Private  151910      HS-grad   9              Widowed
32559  22            Private  201490      HS-grad   9        Never-married
32560  52       Self-emp-inc  287927      HS-grad   9   Married-civ-spouse

                       6              7       8        9      10  11  12  \
0           Adm-clerical   Not-in-family   White     Male   2174   0  40
1         Exec-managerial         Husband   White     Male      0   0  13
2       Handlers-cleaners   Not-in-family   White     Male      0   0  40
3       Handlers-cleaners         Husband   Black     Male      0   0  40
4          Prof-specialty            Wife   Black   Female      0   0  40
...                   ...             ...     ...      ...    ...  ..  ..
32556       Tech-support            Wife   White   Female      0   0  38
32557   Machine-op-inspct         Husband   White     Male      0   0  40
32558        Adm-clerical       Unmarried   White   Female      0   0  40
32559        Adm-clerical       Own-child   White     Male      0   0  20
32560     Exec-managerial            Wife   White   Female  15024   0  40

                   13      14
0       United-States   <=50K
1       United-States   <=50K
2       United-States   <=50K
3       United-States   <=50K
4                Cuba   <=50K
...               ...     ...
```

```
32556   United-States   <=50K
32557   United-States    >50K
32558   United-States   <=50K
32559   United-States   <=50K
32560   United-States    >50K

[32561 rows x 15 columns]
0                 39
1          State-gov
2              77516
3          Bachelors
4                 13
5      Never-married
6       Adm-clerical
7      Not-in-family
8              White
9               Male
10              2174
11                 0
12                40
13     United-States
14             <=50K
Name: 0, dtype: object
```

Chapter 3

Information Theory Measures

In this chapter we will implement two information theory measures.

3.0.1 Exercise - Shannon Entropy

Shannon entropy is defined as: $H(Y) = -\sum_{y \in \omega} P(Y = y) \cdot log_2(P(Y = y))$

Implement the Shannon Entropy as the function entropy. Its only input will be a Pandas Series object.

Note: You can calculate the probabilities by calculating how many times a given value occurs in the given series. In order to do that you may want to use the `value_counts` method of the `Series` class.

Solution 3.0.1

Click to Show/Hide Here or Navigate to Solution 3.0.1 on page 137

[ ]:
```
```

If you implemented these correctly you should get the same values as in the saved cell output.

[37]:
```
print(entropy(data[0]))
print(entropy(data[13]))
```

```
5.68332439640033
0.9437954138017222
```

3.0.2 Exercise - Conditional Entropy

We can recap the definition of the conditinal entropy.

Specific conditional entropy: $H(Y|X = v) = H(Y)$, only for values of Y, where $X = x$

Conditional Entropy: $H(Y|X) = \sum_{x \in \Omega_x} P(X = x) \cdot H(Y|X = x)$

Now we will implement the function `conditional_entropy` which will take two `Series` objects as inputs.

Note: You want to sum over the categories of y. Therefore, it is useful to convert that `Series` object into `category` type using the `astype` method. You can then obtain a list of the categories in the `Series`, try to google how.

Solution 3.0.2

Click to Show/Hide Here or Navigate to Solution 3.0.2 on page 138

[ ]:
```
```

These tests should run successfully if your implementation is correct.

[57]:
```
print(conditional_entropy(data[13], data[14]))
print(conditional_entropy(data[1], data[14]))
print(conditional_entropy(data[8], data[14]))
print(conditional_entropy(data[3], data[14]))
```

```
0.9351000717808609
1.626405337311804
0.7903627046854481
2.837760056715097
```

Chapter 4

Dimension Reduction Algorithms

In this chapter we will cover dimension reduction algorithms such as PCA and LDA.

4.1 PCA

We will now implement PCA [13] on our own. Later we will use an existing version from a library. First let us recap some of the mathematics involved.

Eigenvalues and eigenvectors: Let $\mathbb{A}$ be a matrix $n \times n$, then a nonzero vector $\vec{v} \in \mathbb{R}^n$ is an eigenvector of $\mathbb{A}$ with eigenvalue $\lambda \in \mathbb{C}$ if: $\mathbb{A}\vec{v} = \lambda\vec{v}$.

Hermitian Matrices: A is hermitian iff $A^H = A$, if A is real then $A^T = A$ is sufficient. Hermitian matrices have only real eigenvalues. It is also possible to find an eigenvector for each eigenvalue.

Covariance: $cov(X, Y) = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{n}$

Covariance matrix: $COV(X)_{i,j} = cov(X_i, X_j)$, Covariance matrix is Hermitian and positively semi-definite.

Eigenvectors as a new base: We can create a new base by using the normalized eigenvectors of the covariance matrix. We can represent it as a matrix W where each column is one normalized eigenvector. PCA is based on the fact that eigenvectors of covariance matrix represent an orthogonal base in which the covariance matrix is diagonal.

PCA Process: We start by centering our data as $\vec{x}' = \vec{x} - \overline{x}$. Then we compute W. The new data are obtained by matrix multiplication $y = \vec{x}'W$, if $\vec{x}$ is a row vector. Eigenvalues correspond to the portion of the variance explained by given direction.

4.1.1 Exercise - Custom PCA algorithm

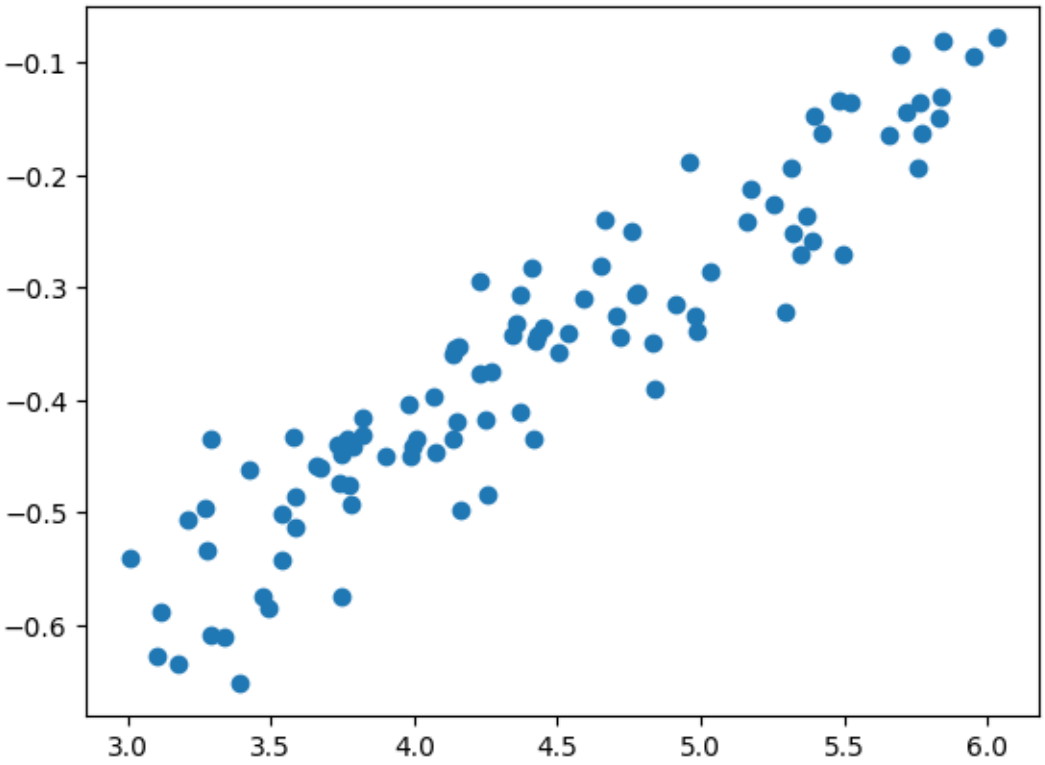
We will now create our own PCA algorithm. We will work with a synthetic dataset.

[38]:
```
!wget https://github.com/kocurvik/edu/raw/master/RO/cvicenia/materialy/data.mat
from scipy.io import loadmat
data = loadmat('data.mat')['data']
```

```
--2024-01-11 14:26:54--
https://github.com/kocurvik/edu/raw/master/RO/cvicenia/materialy/data.mat
Resolving github.com (github.com)... 140.82.114.4
Connecting to github.com (github.com)|140.82.114.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/kocurvik/edu/master/RO/cvicenia/mate
rialy/data.mat [following]
--2024-01-11 14:26:54--  https://raw.githubusercontent.com/kocurvik/edu/master/R
O/cvicenia/materialy/data.mat
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1765 (1.7K) [application/octet-stream]
Saving to: ‘data.mat’

data.mat            100%[===================>]   1.72K  --.-KB/s    in 0s

2024-01-11 14:26:54 (23.3 MB/s) - ‘data.mat’ saved [1765/1765]
```

[40]:
```
from matplotlib import pyplot as plt

print(data.shape)
plt.scatter(data[:, 0], data[:, 1])
plt.show()
```

```
(100, 2)
```



Now try to calculate PCA on your own.

In first step you need to subtract the means for both features from the data. Then you will need to calculate the covariance matrix. Next, you will need to calculate the eigenvectors for the covariance matrix. Using these eigenvectors you can then transform the original features into a new feature space where axes should be aligned with the principal components. You should then plot the result. Also try to find out what portion of variance is explained by the data.

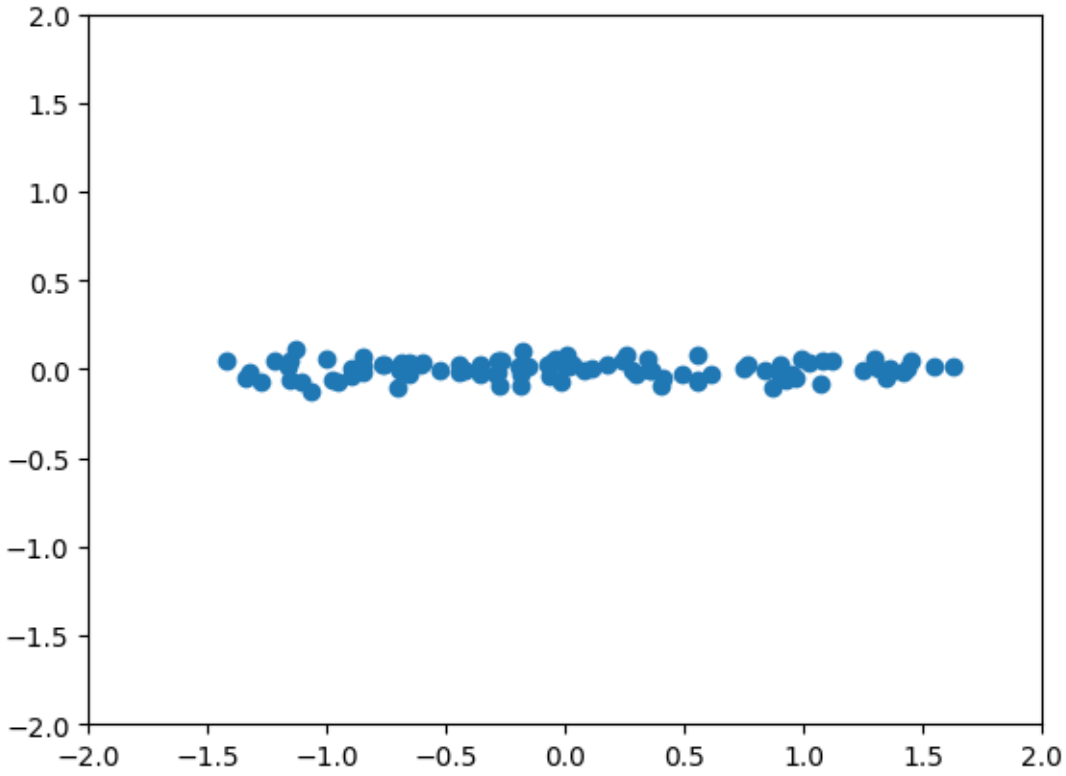
You will need the functions `np.cov` which returns the covariance matrix. Note that the function expects the input to be such that each row of m represents a variable, and each column a single observation of all those variables. Which is different from our current representation of data.

You will also need `np.linalg.eig` which returns the eigenvalues and the normalized eigenvectors (check the documentation). You could also use `np.linalg.eigh` since our matrix is Hermitian.

Solution 4.1.1

Click to Show/Hide Here or Navigate to Solution 4.1.1 on page 139

[ ]:



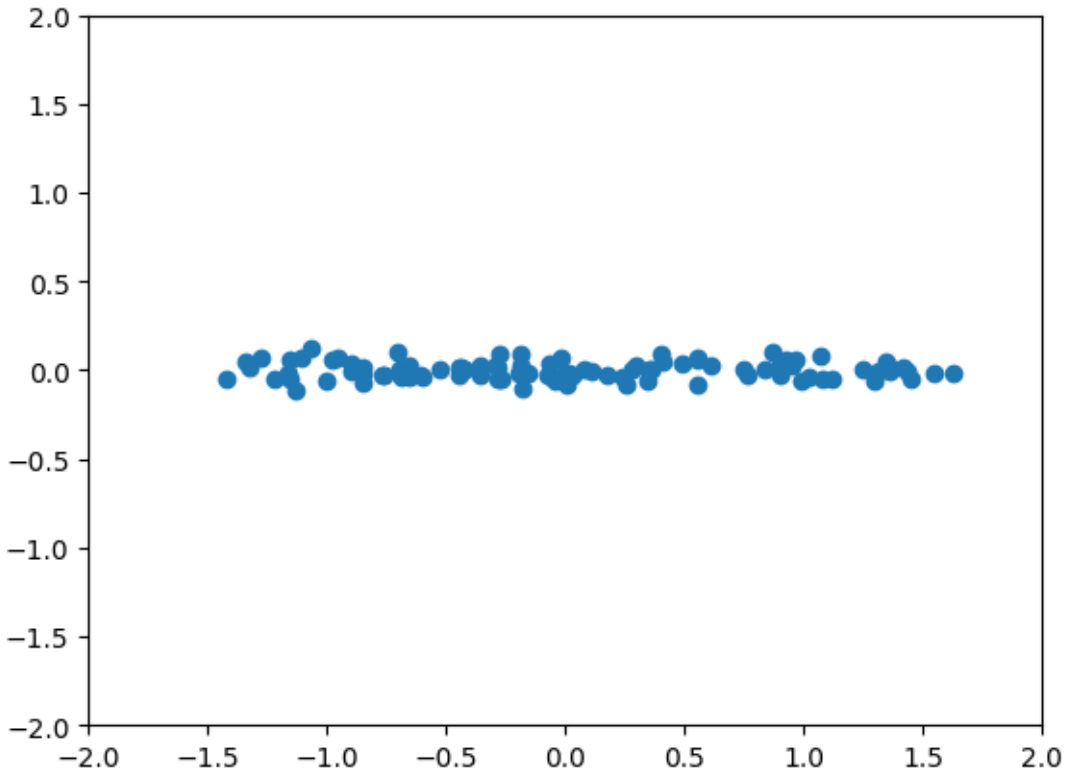

```
Portion of variance explained:  0.9965218526177005
```

4.1.2 Scikit Learn PCA

Now we will use the PCA provided in the Scikit learn library.

[42]:
```
from sklearn.decomposition import PCA

pca = PCA()

pca.fit(data)
XXX = pca.transform(data)

plt.scatter(XXX[:, 0], XXX[:, 1])
plt.xlim([-2, 2])
```

```
plt.ylim([-2, 2])
plt.show()
```



We can select the PCA algorithm to reduce the dimension of the feature space. There are two options. We can either select the portion of variance explained by the data or the dimension of the new feature space. Check the documentation for details.

4.1.3 Exercise - Ovarian cancer dataset

Use the PCA from scikit learn to reduce the dimensionality of mass spectrometry data used to determine whether cancer is present (downloaded in the cell below) to either just two dimensions or dimensions covering the whole space. We will work with the ARCENE dataset by Isabelle Guyon et al [10].

Determine what portion of variance is explained by the first two features (check out the `explained_variance_ratio_` property of the PCA object).

[43]:
```
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/arcene/ARCENE/
↪arcene_train.data

from pandas import read_csv

cancer_data = read_csv('arcene_train.data', sep=' ', header=None,
```

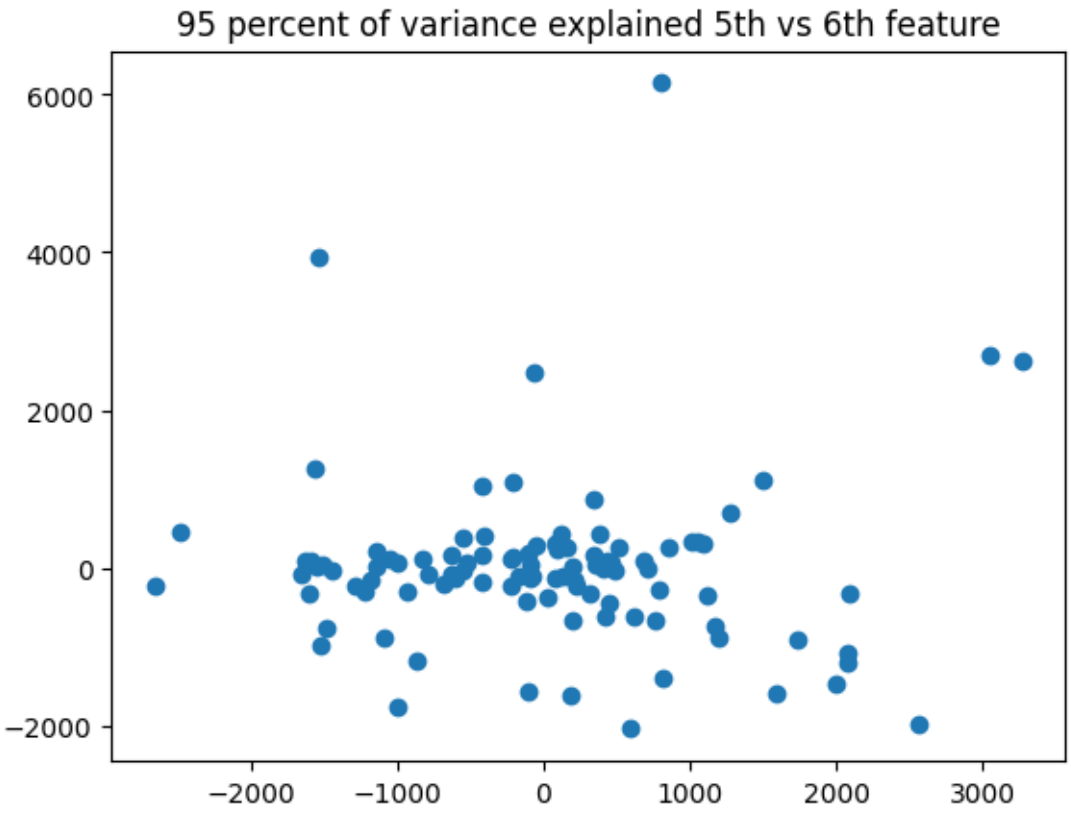
```
                      index_col=False).iloc[: , :-1]
print(cancer_data)
print(cancer_data.shape)
```

```
--2024-01-11 14:28:53--  https://archive.ics.uci.edu/ml/machine-learning-
databases/arcene/ARCENE/arcene_train.data
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: ‘arcene_train.data’

arcene_train.data       [   <=>              ]   2.59M  5.48MB/s    in 0.5s

2024-01-11 14:28:54 (5.48 MB/s) - ‘arcene_train.data’ saved [2715582]

    0     1     2     3     4     5     6     7     8     9     ...  9990  \
0     0    71     0    95     0   538   404    20     0     0  ...   255
1     0    41    82   165    60   554   379     0    71     0  ...   213
2     0     0     1    40     0   451   402     0     0     0  ...   235
3     0    56    44   275    14   511   470     0     0     0  ...    91
4   105     0   141   348     0   268   329     0     0     1  ...   813
..  ...   ...   ...   ...   ...   ...   ...   ...   ...   ...  ...   ...
95   24    26     0   461     0   545     0    17   159   177  ...   275
96   40     0     0   419    71   502     0    39    93   163  ...   277
97    2    15    48   677     0   434   442     0    43     0  ...   211
98    8     0    38   205    69   419   454     0   113     3  ...   193
99    0     0     0   148     0   583     0     6   130   112  ...   271

    9991  9992  9993  9994  9995  9996  9997  9998  9999
0    570    86     0    36     0    80     0     0   524
1    605    69     7   473     0    57     0   284   423
2    593    28     0    24     0    90     0    34   508
3    600     0    26    86     0   102     0     0   469
4      0     0     0     0   190   301     0     0   354
..   ...   ...   ...   ...   ...   ...   ...   ...   ...
95   460    22    26   130   306   182     0    94   336
96   436     0    68    61   295   133     0     0   292
97   628     0     7   228     0   105     0     0   453
98   587   148    27   656     0   133     0   189   403
99   477     0     0   111   414   210     0    10   365

[100 rows x 10000 columns]
(100, 10000)
```

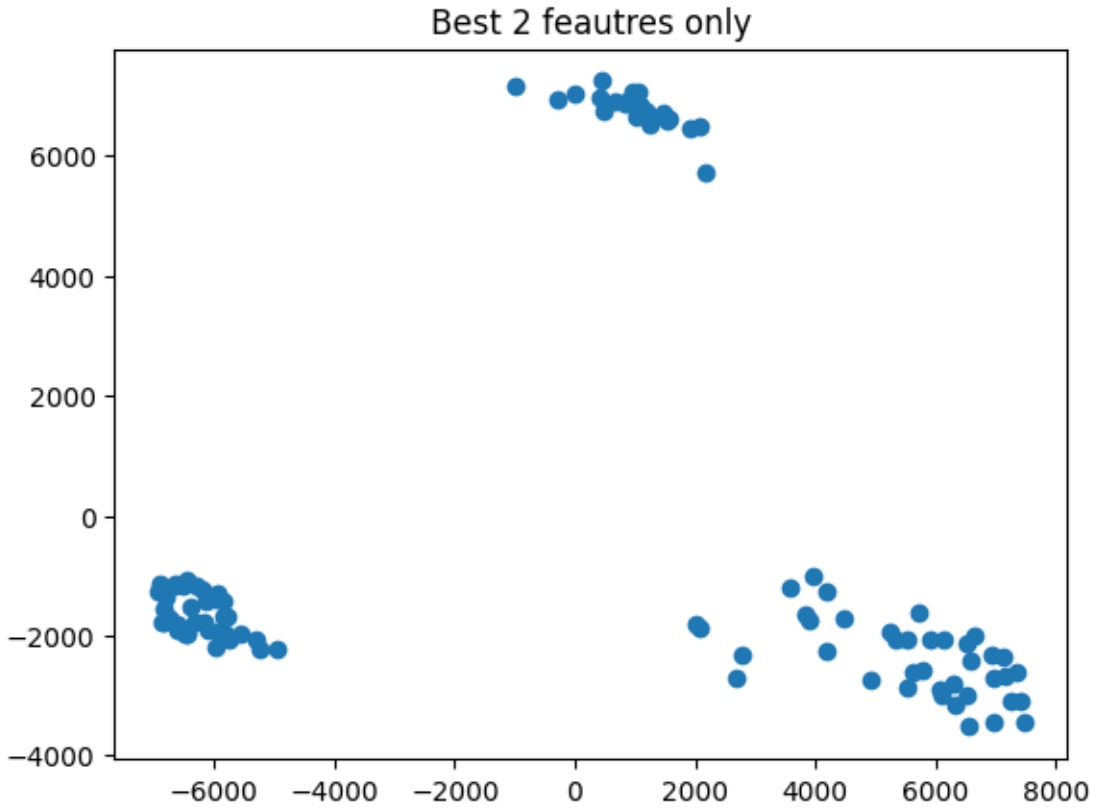
Solution 4.1.3

Click to Show/Hide Here or Navigate to Solution 4.1.3 on page 140

[ ]:



```
We got 42 features
Percent of variance explained for each feature:  [0.44859537 0.22397886
0.05459793 0.03176634 0.02357209 0.01995894
 0.01858637 0.01254881 0.01131351 0.00920648 0.00760523 0.00670265
 0.00648032 0.00547148 0.00521921 0.00484254 0.00437686 0.00406957
 0.00360946 0.00330621 0.00324208 0.00287959 0.00272499 0.00262583
 0.00251016 0.0024182  0.00226027 0.00213232 0.00210926 0.00203499
 0.00194431 0.00186583 0.00176077 0.00174059 0.00168485 0.00163076
 0.00161406 0.00154571 0.00148174 0.00146314 0.00143758 0.00137239]
```




```
Percent of variance explained for each feature:  [0.44859537 0.22397886]
```

4.1.4 Exercise - Displaying the principal components

We will now turn back to our original synthetic data. Your task will now be to diplay the original features in a scatter plot and add lines showing the principal components (e.g. the new axes). You can also try doing the reverse by showing the new space and showing where the old axes were.

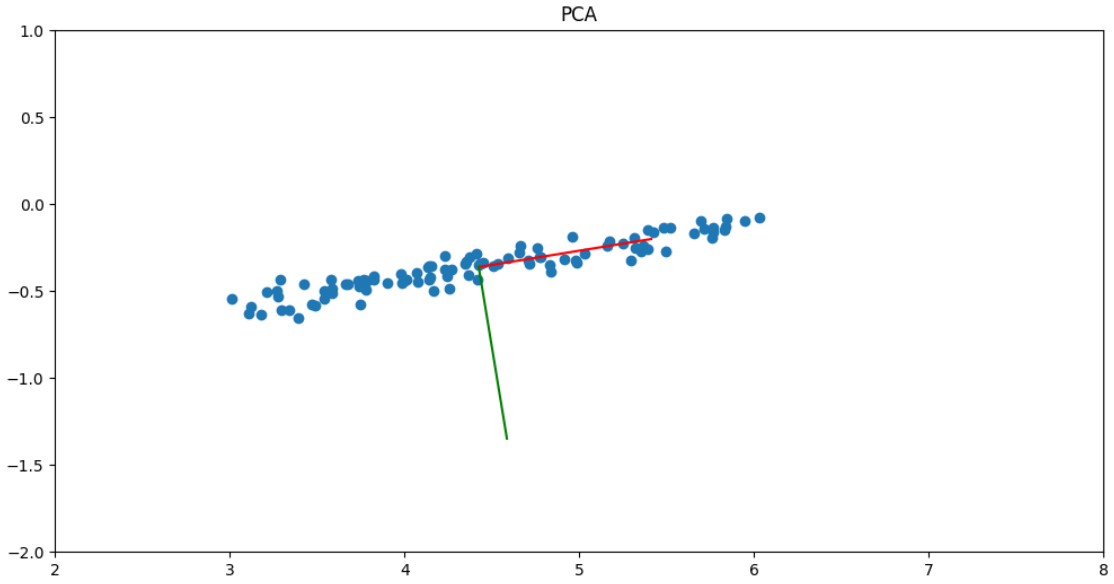
You will need to extract the principal components and also obtaining the mean of the data this can be done by accessing the `components_` and `mean_` properties of the PCA object.

As a bonus you can try to do this, but instead of PCA use ICA from `sklearn.decomposition.FastICA`

Solution 4.1.4

Click to Show/Hide Here or Navigate to Solution 4.1.4 on page 141

[ ]:



4.1.5 Exercise - PCA Color Space reduction

We can also try to reduce the dimensionality of the color space. Normally, we represent an image with pixels as an array of shape `height * width * 3`. E.g. each pixel is a 3-dimensional vector representing the RGB colors. We can try to use PCA to reduce this representation to PCA.

Try to do this on your own.

Hint: You will need to reshape the image from its original representation so you get a two-dimensional array. You will also need to use the `inverse_transform` method of the PCA object.

The downloaded images were released into public domain by USFWS. They were created by Thomas Leeman, Ryan Hagerty and Thomas G. Barnes.

[47]:
```
# loading and image for now we use scipy load image
!wget https://github.com/kocurvik/edu/raw/master/pubs/images/lizard.jpg
!wget https://github.com/kocurvik/edu/raw/master/pubs/images/building.jpg
!wget https://github.com/kocurvik/edu/raw/master/pubs/images/plant.jpg

from PIL import Image

img_liz = np.asarray(Image.open('lizard.jpg')) / 255
plt.imshow(img_liz)
plt.axis("off")
plt.show()

img_building = np.asarray(Image.open('building.jpg')) / 255
plt.imshow(img_building)
plt.axis("off")
plt.show()

img_plant = np.asarray(Image.open('plant.jpg')) / 255
plt.imshow(img_plant)
plt.axis("off")
plt.show()
```

```
--2024-01-11 14:35:37--
https://github.com/kocurvik/edu/raw/master/pubs/images/lizard.jpg
Resolving github.com (github.com)... 140.82.114.3
Connecting to github.com (github.com)|140.82.114.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location:
https://raw.githubusercontent.com/kocurvik/edu/master/pubs/images/lizard.jpg
[following]
--2024-01-11 14:35:37--
https://raw.githubusercontent.com/kocurvik/edu/master/pubs/images/lizard.jpg
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443... connected.
```

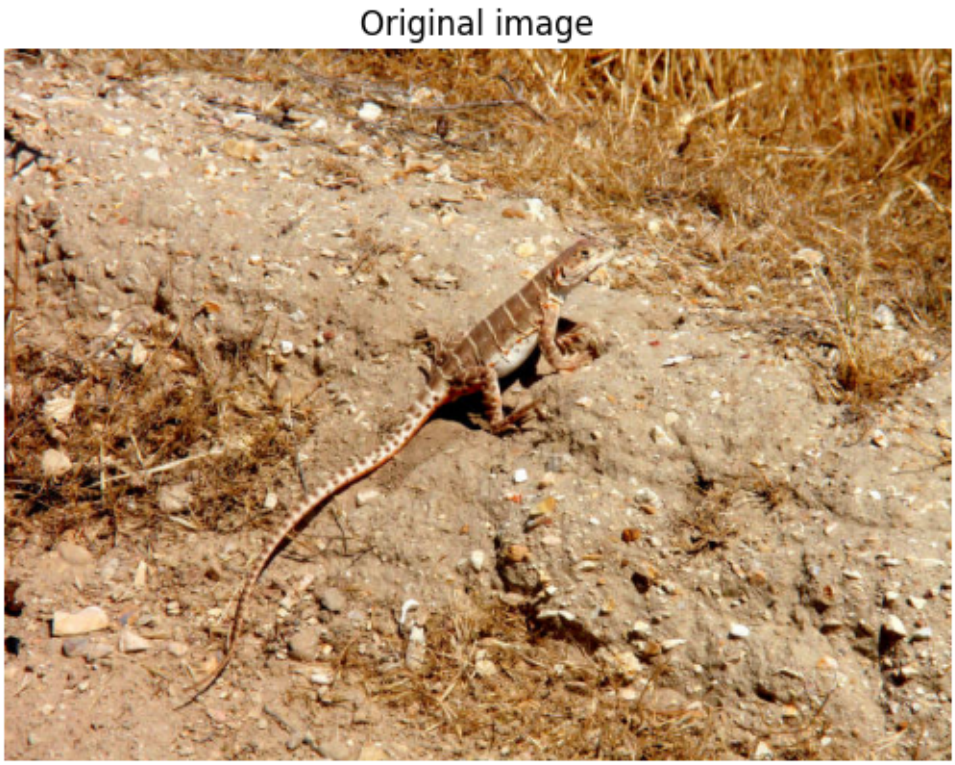
```
HTTP request sent, awaiting response... 200 OK
Length: 120711 (118K) [image/jpeg]
Saving to: ‘lizard.jpg’

lizard.jpg          100%[===================>] 117.88K  --.-KB/s    in 0.004s

2024-01-11 14:35:37 (30.4 MB/s) - ‘lizard.jpg’ saved [120711/120711]

--2024-01-11 14:35:37--
https://github.com/kocurvik/edu/raw/master/pubs/images/building.jpg
Resolving github.com (github.com)... 140.82.114.3
Connecting to github.com (github.com)|140.82.114.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location:
https://raw.githubusercontent.com/kocurvik/edu/master/pubs/images/building.jpg
[following]
--2024-01-11 14:35:37--
https://raw.githubusercontent.com/kocurvik/edu/master/pubs/images/building.jpg
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 57259 (56K) [image/jpeg]
Saving to: ‘building.jpg’

building.jpg        100%[===================>]  55.92K  --.-KB/s    in 0.001s

2024-01-11 14:35:37 (45.4 MB/s) - ‘building.jpg’ saved [57259/57259]

--2024-01-11 14:35:37--
https://github.com/kocurvik/edu/raw/master/pubs/images/plant.jpg
Resolving github.com (github.com)... 140.82.112.4
Connecting to github.com (github.com)|140.82.112.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location:
https://raw.githubusercontent.com/kocurvik/edu/master/pubs/images/plant.jpg
[following]
--2024-01-11 14:35:37--
https://raw.githubusercontent.com/kocurvik/edu/master/pubs/images/plant.jpg
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 104311 (102K) [image/jpeg]
Saving to: ‘plant.jpg’
```

```
plant.jpg           100%[===================>] 101.87K  --.-KB/s    in 0.002s

2024-01-11 14:35:38 (50.6 MB/s) - ‘plant.jpg’ saved [104311/104311]
```

Solution 4.1.5

Click to Show/Hide Here or Navigate to Solution 4.1.5 on page 142

[ ]:


Original image


Reduced colorspace image

Original image

Reduced colorspace image

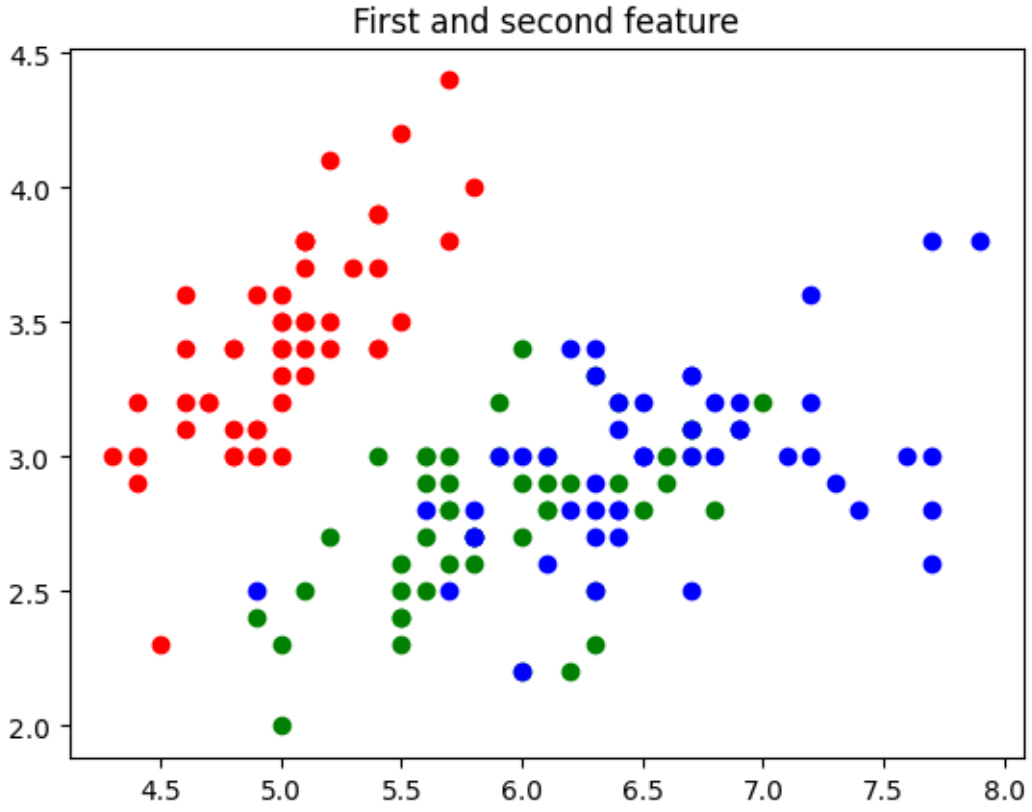

Original image

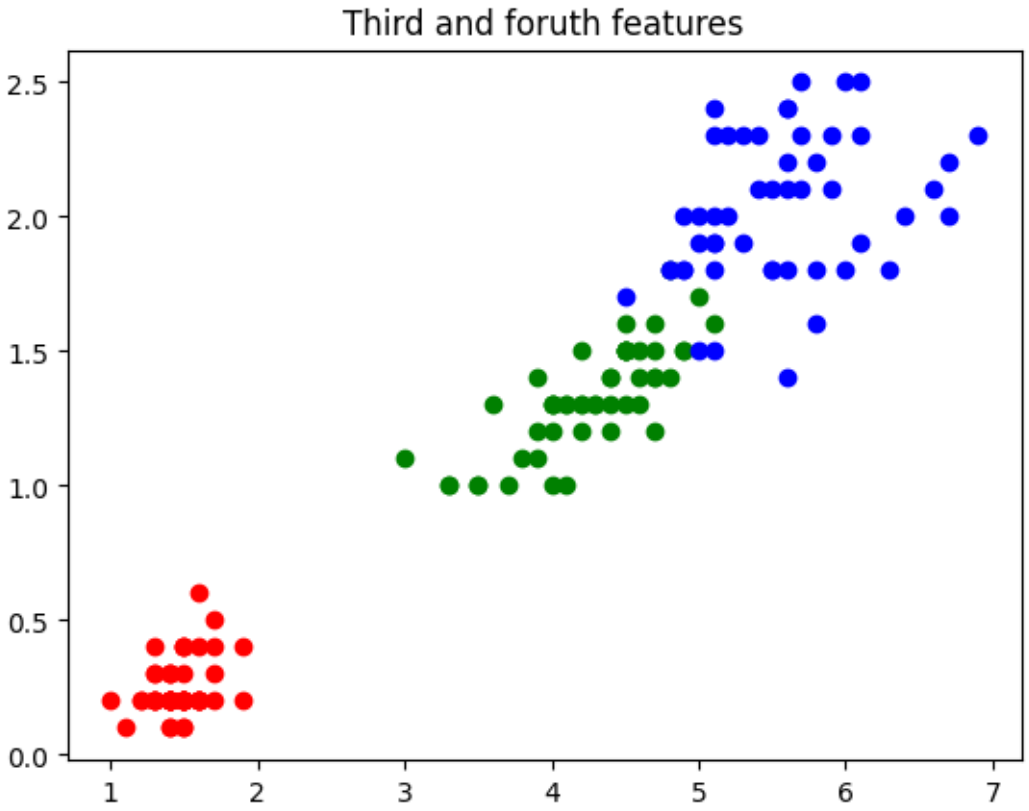

Reduced colorspace image

4.2 LDA

We will now try to use LDA [3]. For that we also need labels. We will use the standard Fisher Iris dataset by R.A. Fisher [7].

[49]:
```
from sklearn import datasets
iris = datasets.load_iris()

X = iris.data
y = iris.target
target_names = iris.target_names

plt.scatter(X[y == 0, 0], X[y == 0, 1], color='red')
plt.scatter(X[y == 1, 0], X[y == 1, 1], color='green')
plt.scatter(X[y == 2, 0], X[y == 2, 1], color='blue')
plt.title("First and second feature")
plt.show()

plt.scatter(X[y == 0, 2], X[y == 0, 3], color='red')
plt.scatter(X[y == 1, 2], X[y == 1, 3], color='green')
plt.scatter(X[y == 2, 2], X[y == 2, 3], color='blue')
plt.title("Third and foruth features")
plt.show()
```





4.2.1 Exercise - LDA

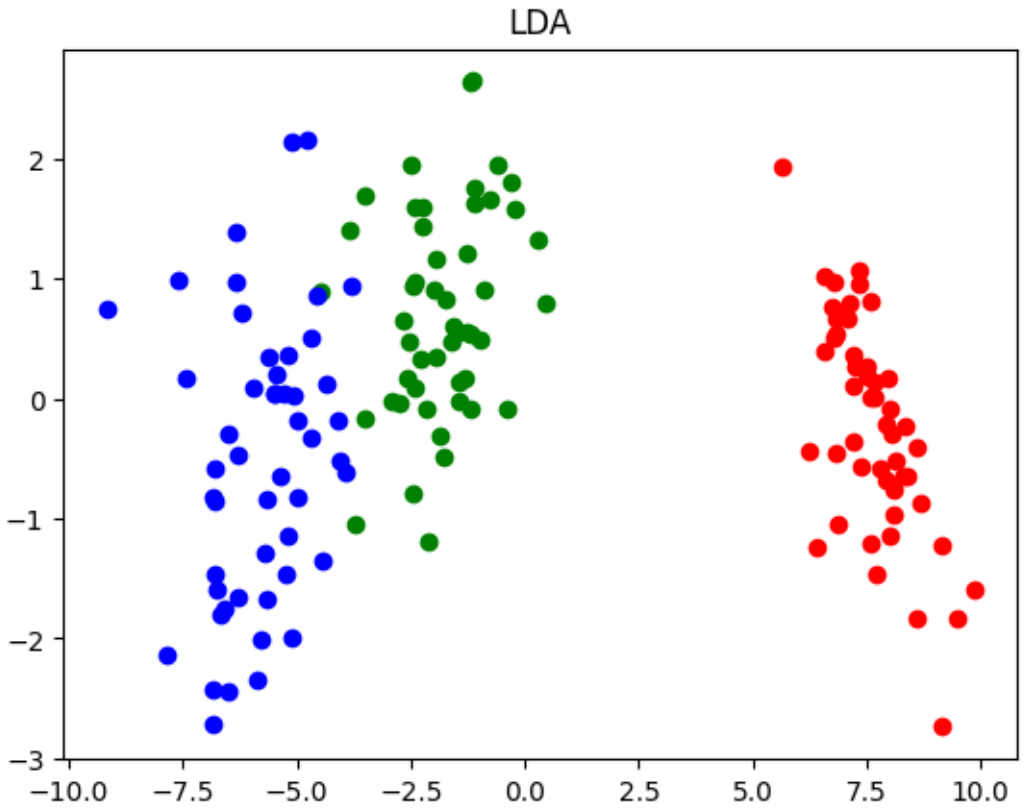
Try to use LDA to transform the data. LDA takes on input also the annotation labels stored in y. Compare the differences between reducing the dimensionality of the dataset to 2 with LDA vs PCA.

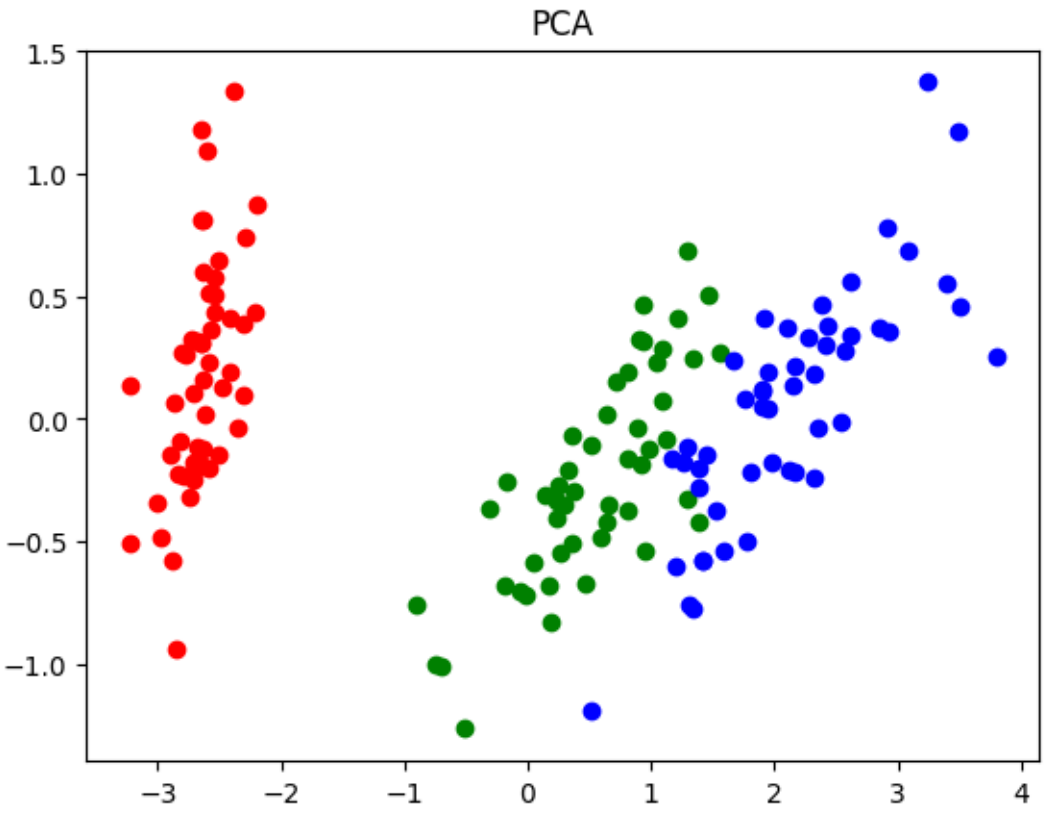
As a Bonus you can try the ICA algorithm from sklearn.decomposition.FastICA

Solution 4.2.1

Click to Show/Hide Here or Navigate to Solution 4.2.1 on page 143

[ ]:


LDA
2
1
0
−1
−2
−3
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


PCA
1.5
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3
4

Chapter 5

Linear Classifier, SVM

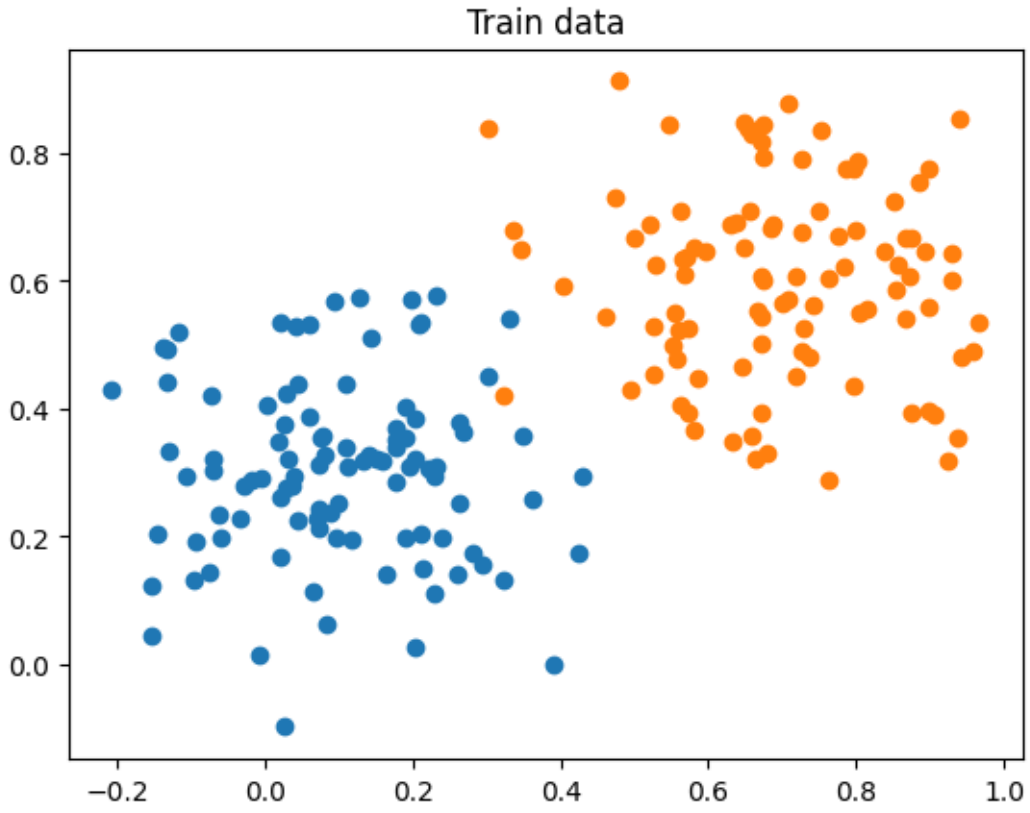
In this chapter we will cover the linear classifier as well as the SVM classifier [6]. We will first implement a linear classifier using our own loss function and then we will utilize existing libraries.

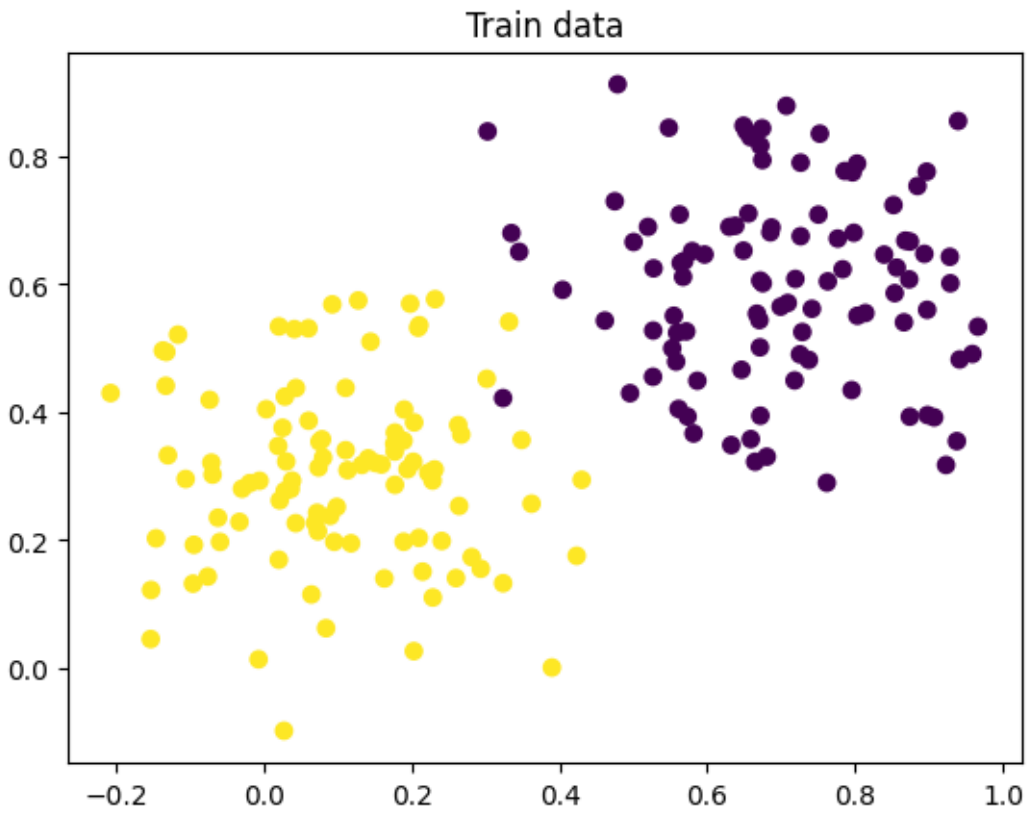
5.1 Synthetic data

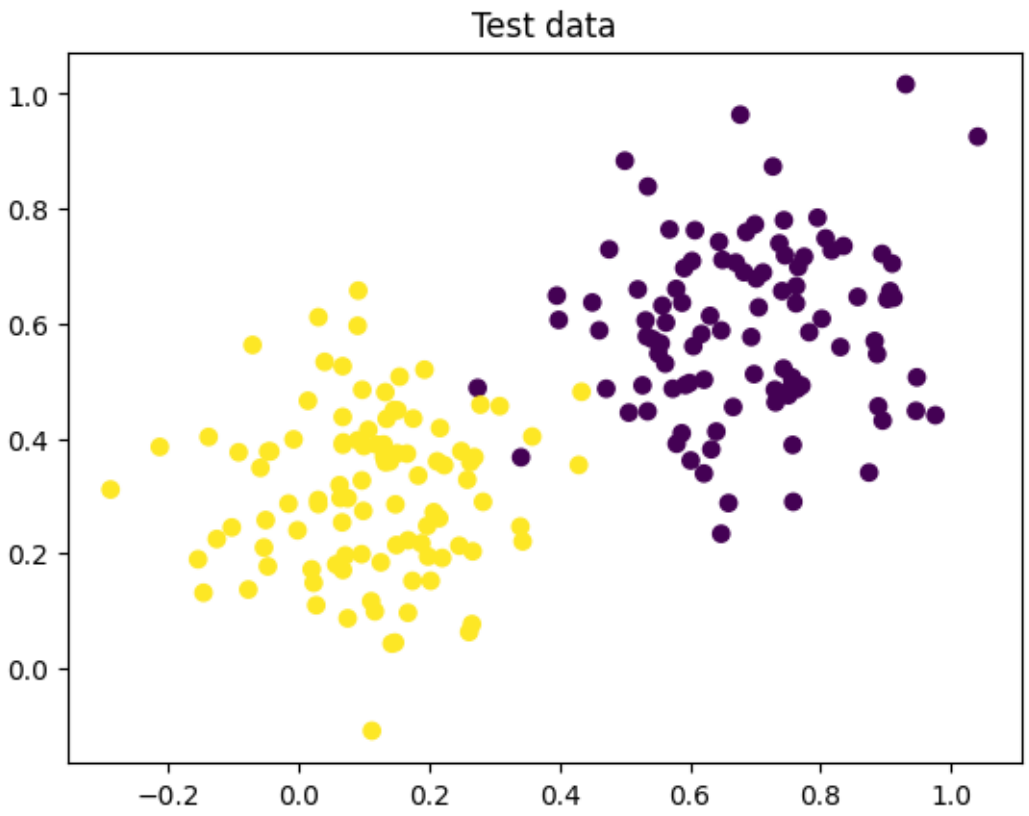
We will first work with some synthetic data. We will first start with simple data of two classes each from a distinct normal distribution in 2D space.

[58]:
```
import numpy as np
from matplotlib import pyplot as plt

np.random.seed(11)

x1_center = np.array([0.1, 0.3])
x1 =  x1_center + 0.15 * np.random.randn(100, 2)

x2_center = np.array([0.7, 0.6])
x2 =  x2_center + 0.15 * np.random.randn(100, 2)

# when the data is separated by classes it is easy to plot
plt.scatter(x1[:, 0], x1[:, 1])
plt.scatter(x2[:, 0], x2[:, 1])
plt.title("Train data")
plt.show()

# we try to get a vector of classies and a matrix of all feature vectors
y = np.ones(200)
y[100:] = -1
x = np.concatenate([x1, x2], axis=0)

# now we will shuffle the data
pindxs = np.random.permutation(200)
```

```
x = x[pindxs]
y = y[pindxs]

plt.scatter(x[:, 0], x[:, 1], c=y)
plt.title("Train data")
plt.show()

# we will also generate synthetic data as a test set using the same distribution
x1_test = x1_center + 0.15 * np.random.randn(100, 2)
x2_test =  x2_center + 0.15 * np.random.randn(100, 2)
y_test = np.ones(200)
y_test[100:] = -1
x_test = np.concatenate([x1_test, x2_test], axis=0)
pindxs = np.random.permutation(200)
x_test = x_test[pindxs]
y_test = y_test[pindxs]

plt.scatter(x_test[:, 0], x_test[:, 1], c=y_test)
plt.title("Test data")
plt.show()
```




Train data
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0



5.2 Scikit-learn Classifiers

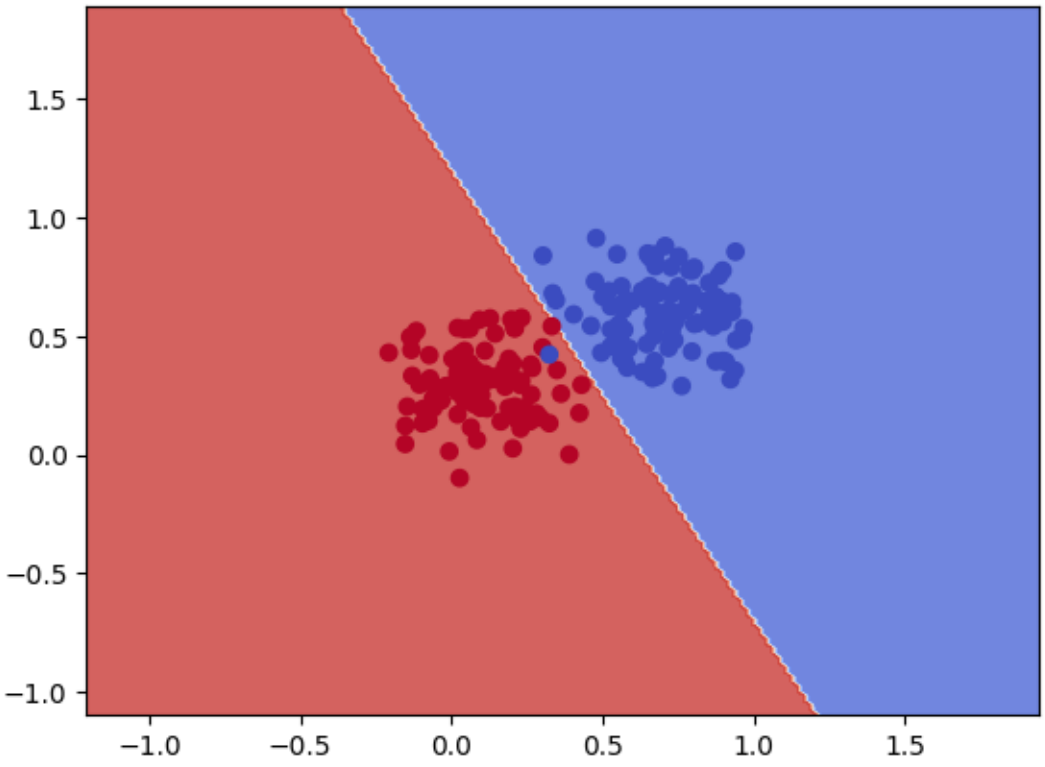
Now we will use the classifiers from the scikit-learn library. Before doing that we will create a function which helps us to show the decision boundary of a classifier.

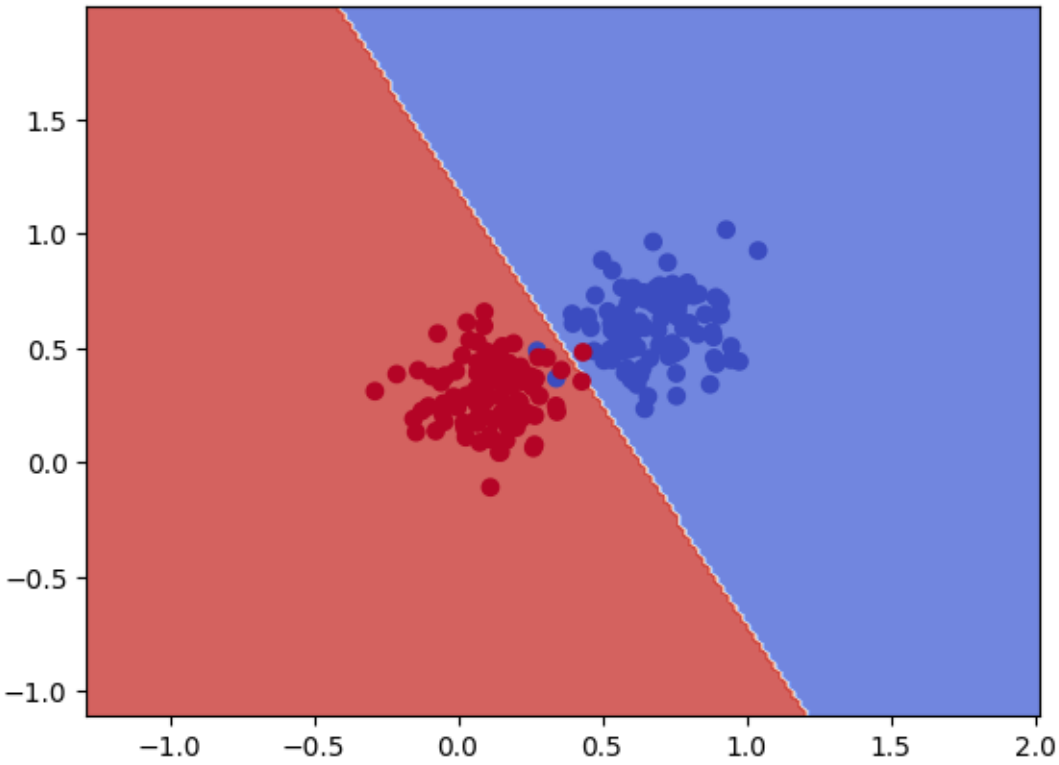
[59]:
```
def plot_clf_boundary(clf, x, y):
  x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
  y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
  h_x = (x_max - x_min) / 150
  h_y = (y_max - y_min) / 150
  xx, yy = np.meshgrid(np.arange(x_min, x_max, h_x),
                       np.arange(y_min, y_max, h_y))
  z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

  # Put the result into a color plot
  z = z.reshape(xx.shape)
  plt.contourf(xx, yy, z, cmap=plt.cm.coolwarm, alpha=0.8)

  # Plot also the training points
  plt.scatter(x[:, 0], x[:, 1], c=y, cmap=plt.cm.coolwarm)
```

```
  plt.xlim(xx.min(), xx.max())
  plt.ylim(yy.min(), yy.max())
  # plt.xticks(())
  # plt.yticks(())
  plt.show()
```

Now we can use a simple linear classifier `sklearn.linear_model.LogisticRegression`. You can check out its documentation for more info.

[60]:
```
from sklearn.linear_model import LogisticRegression

linear = LogisticRegression()
linear.fit(x, y)

plot_clf_boundary(linear, x, y)
plot_clf_boundary(linear, x_test, y_test)
```





5.2.1 Exercise - Calculating accuracy on the training set and the test set

Your task will now be to calculate the accuracy on the test data and also on training data. For this purpose you should check out the method `predict` of the classifier object.

Solution 5.2.1

Click to Show/Hide Here or Navigate to Solution 5.2.1 on page 144

[ ]:

```
Accuracy on train data:  0.995
Accuracy on test data:  0.985
```

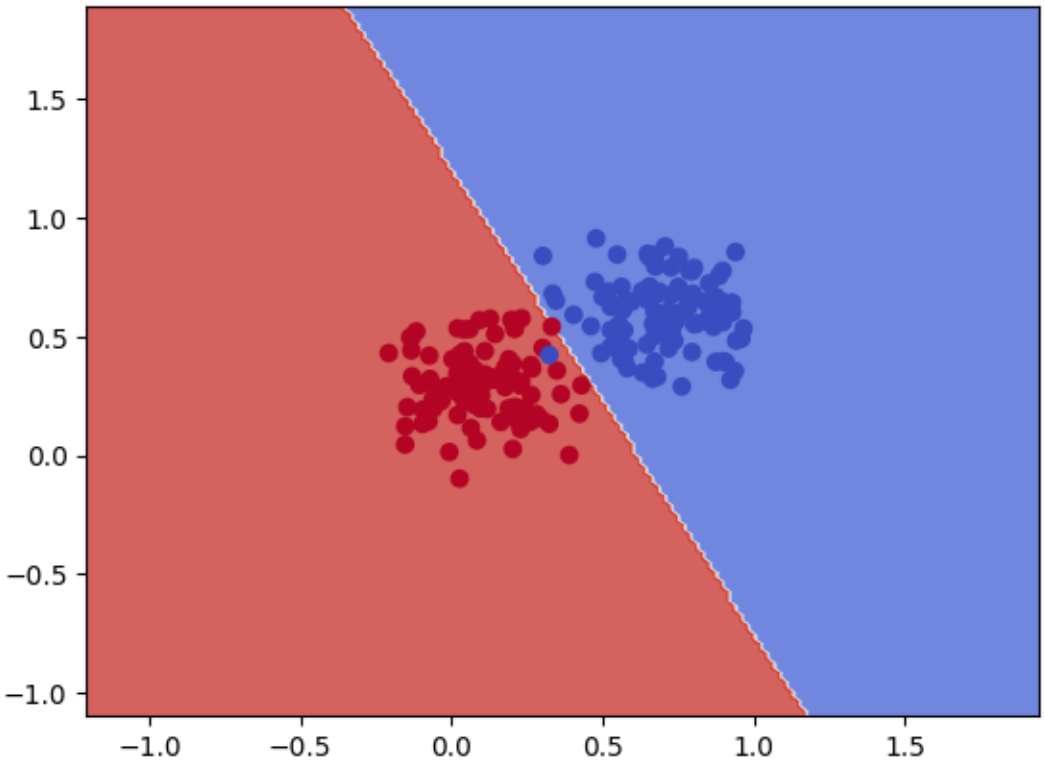
5.2.2 Exercise - SVM

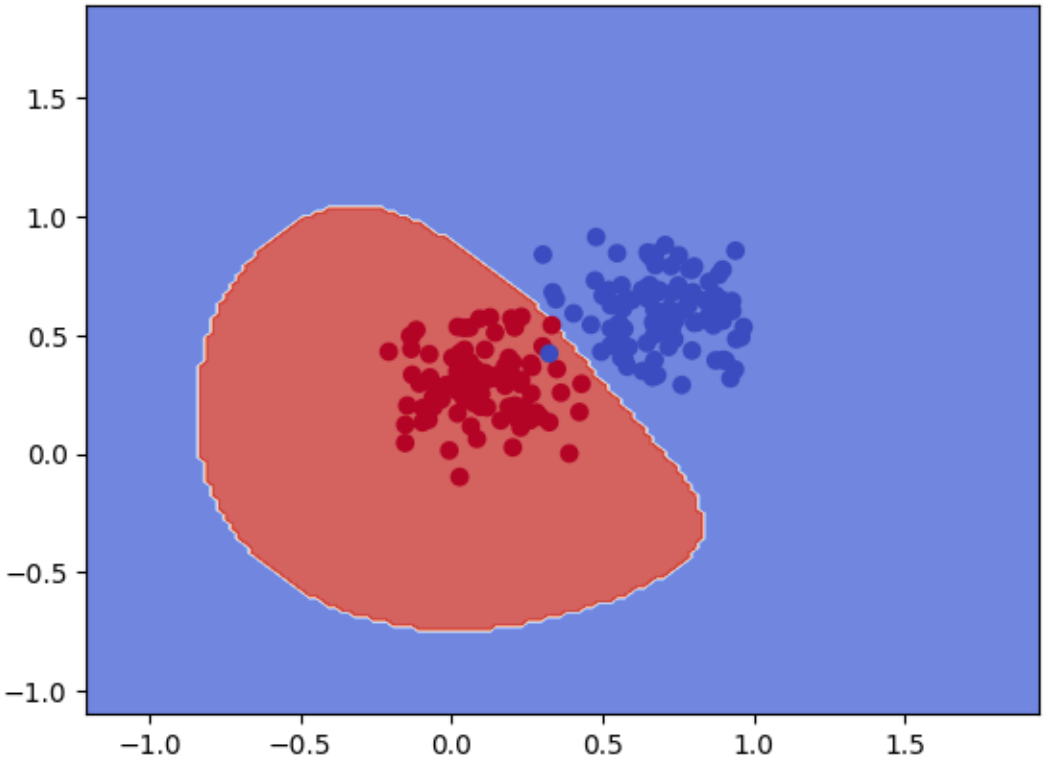
We can find the SVM in scikit-learn as `sklearn.svm.SVC`. Check its documentation and use it. Try change the kernel to emulate the decision boundaries as shown below.

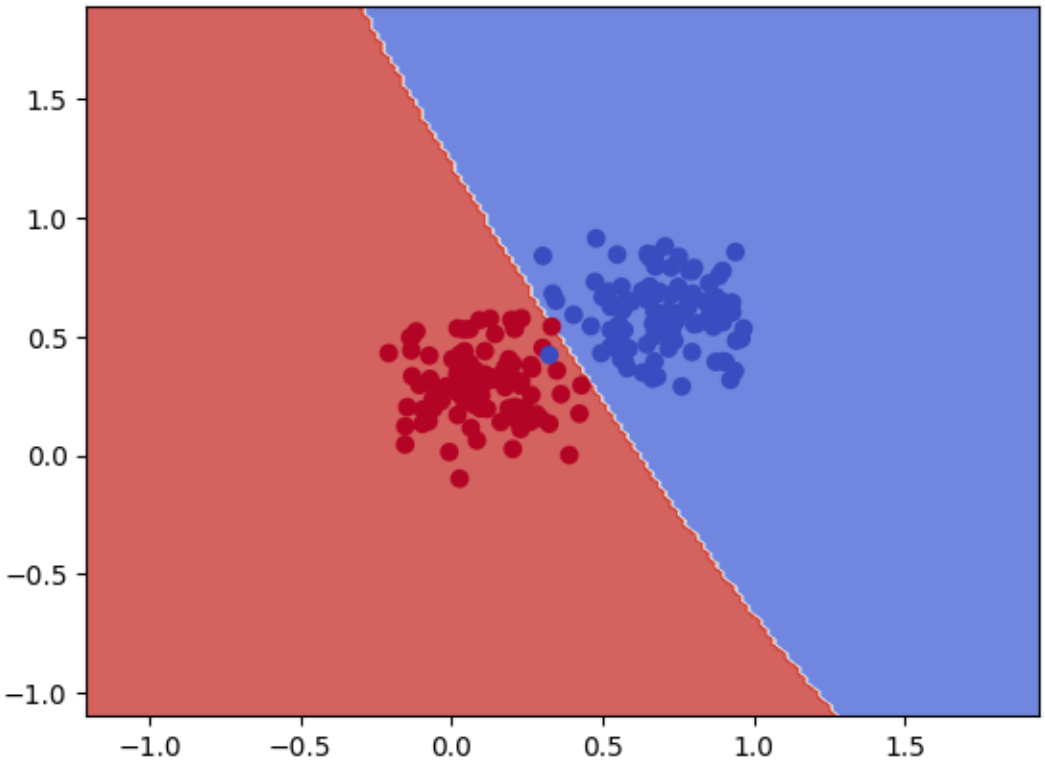
Solution 5.2.2

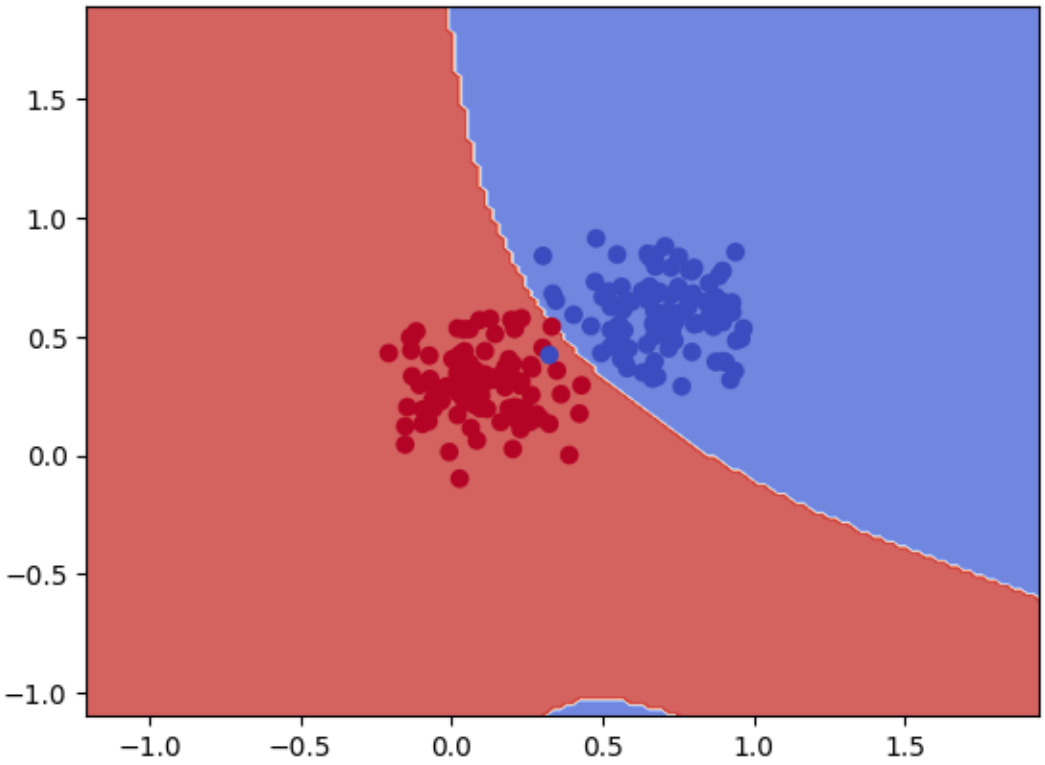
Click to Show/Hide Here or Navigate to Solution 5.2.2 on page 145

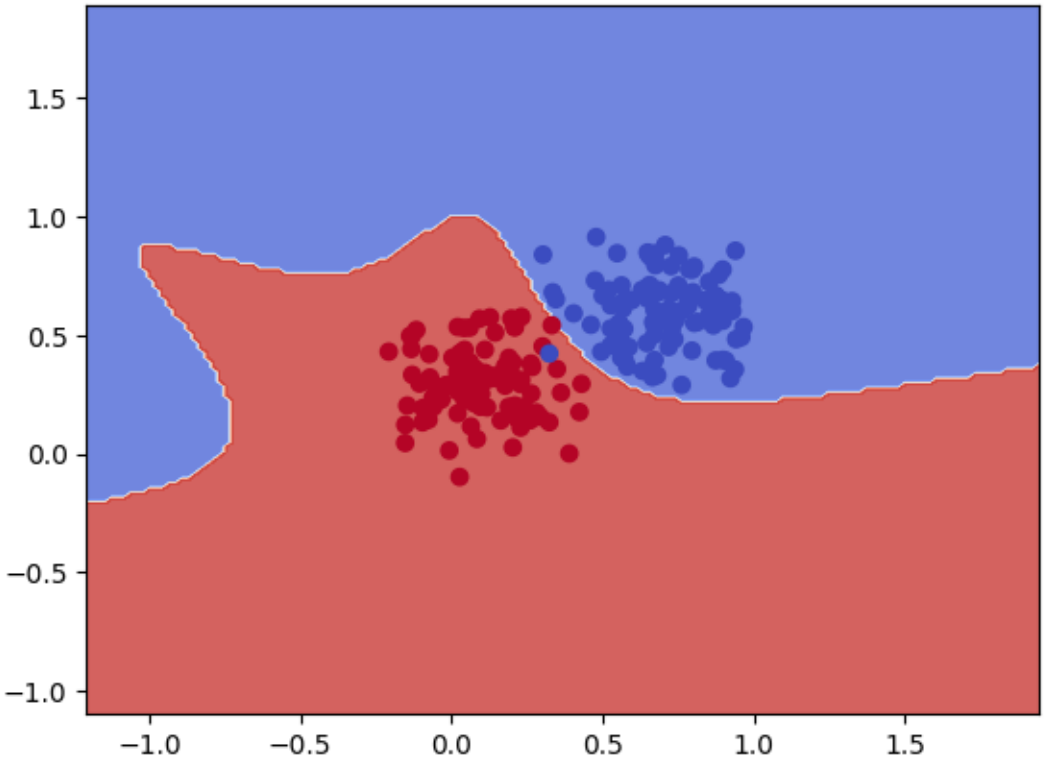
[ ]:

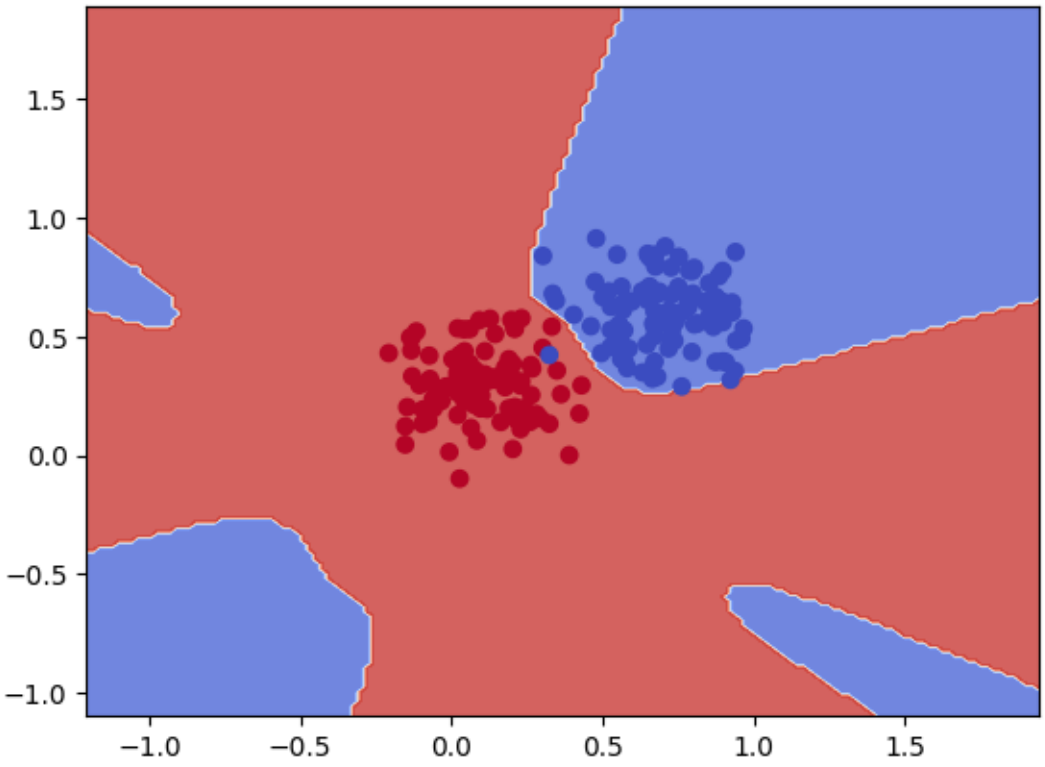

1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

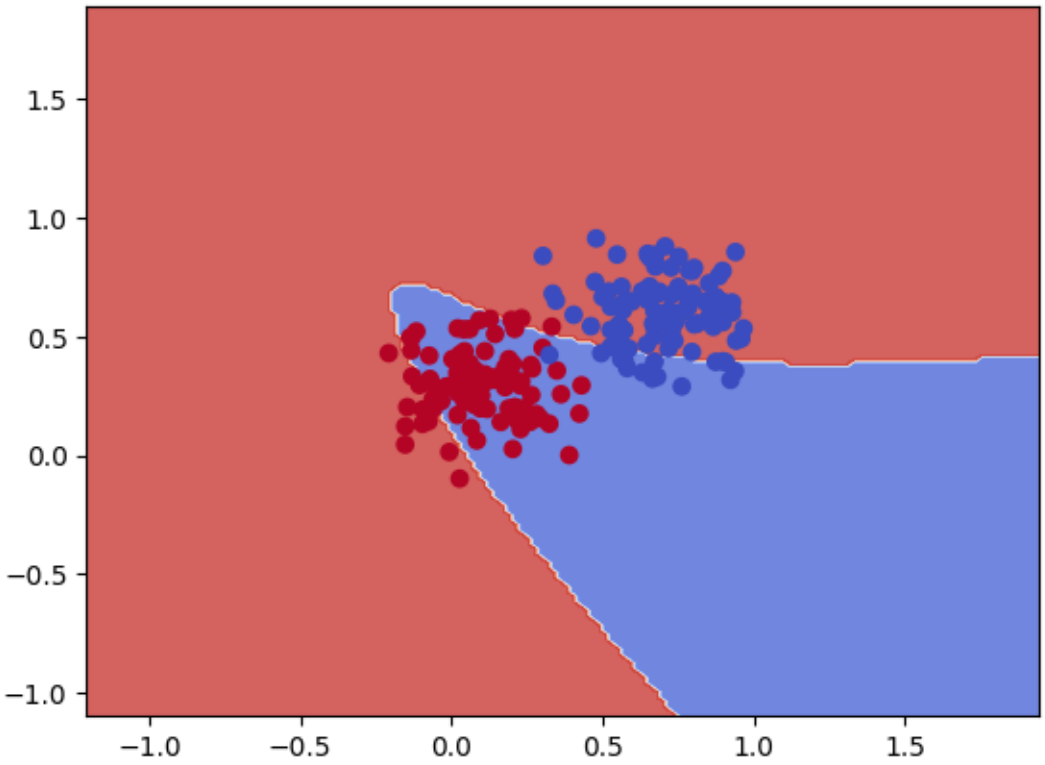

1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5



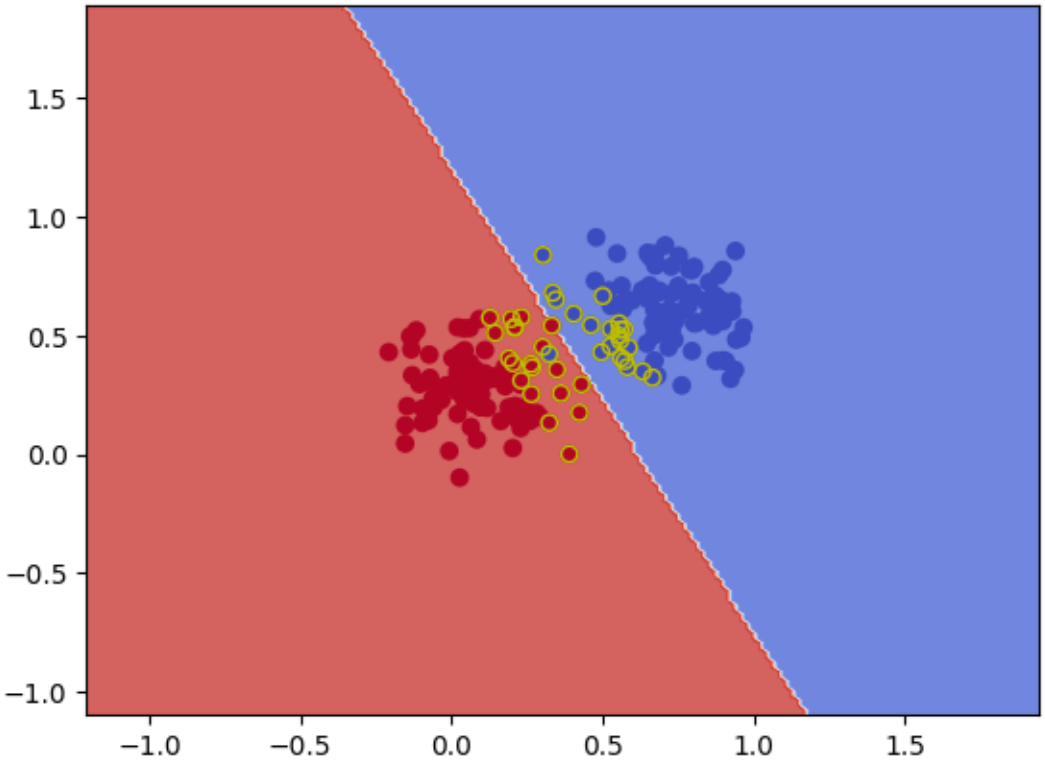
5.2.3 Exercise - Displaying the support vectors

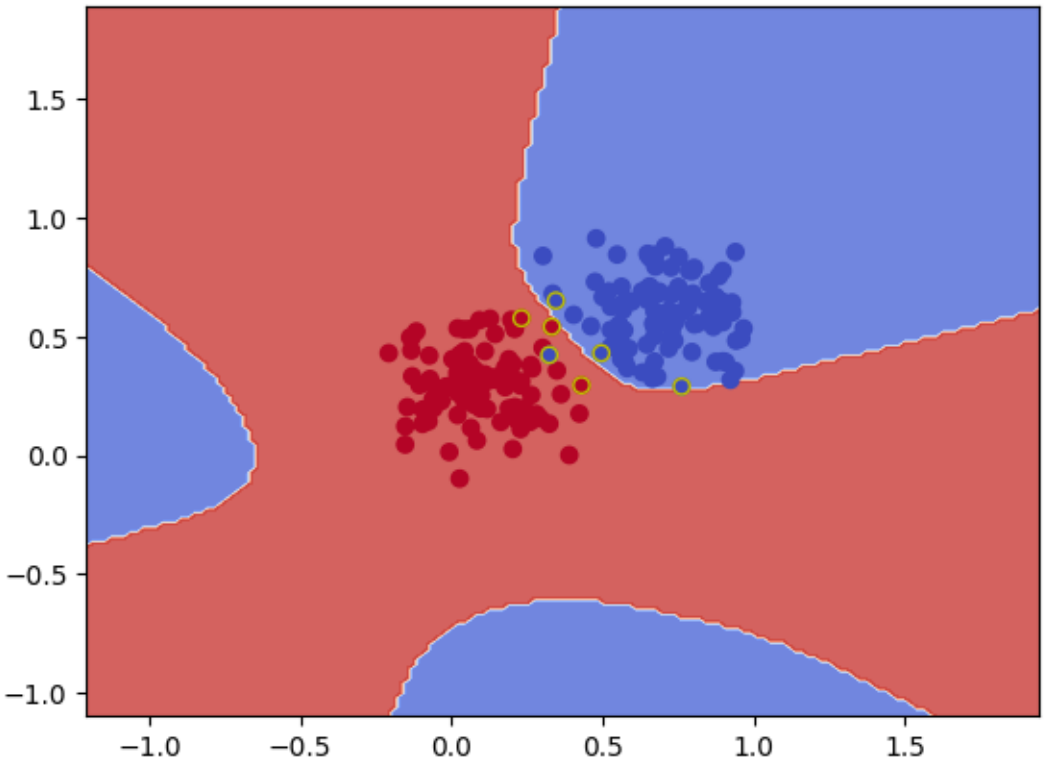
By checking the documentation try to diplay the support vectors for a given classifier. In order to extract the information about the support vectors consult the documentation

Solution 5.2.3

Click to Show/Hide Here or Navigate to Solution 5.2.3 on page 146

[ ]:

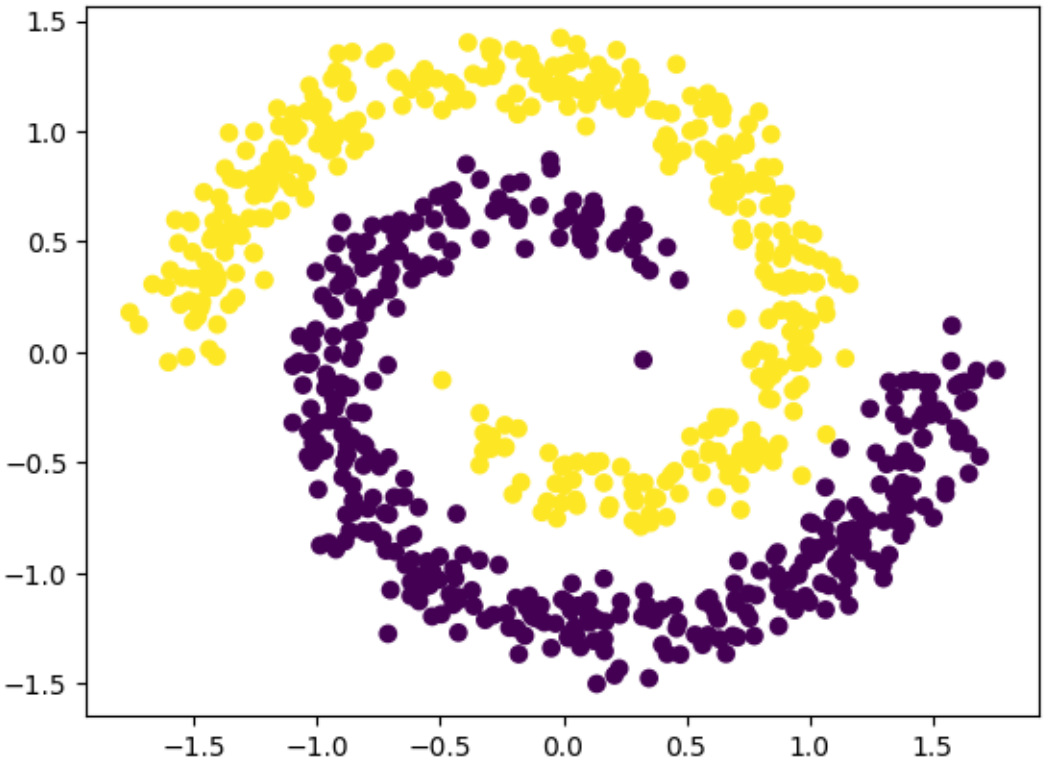

1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5



We will now generate a spiral dataset using code from this gist.

[64]:
```
from numpy import pi

N = 400
theta = np.sqrt(np.random.rand(N))*2*pi
r_a = 2*theta + pi
data_a = np.array([np.cos(theta)*r_a, np.sin(theta)*r_a]).T
x_a = data_a + np.random.randn(N,2)

r_b = -2*theta - pi
data_b = np.array([np.cos(theta)*r_b, np.sin(theta)*r_b]).T
x_b = data_b + np.random.randn(N,2)

res_a = np.append(x_a, np.zeros((N,1)), axis=1)
res_b = np.append(x_b, np.ones((N,1)), axis=1)

res = np.append(res_a, res_b, axis=0)
np.random.shuffle(res)

x_spiral = 0.1 * res[:, :2]
y_spiral = res[:, 2].astype(np.int64)
```

```
plt.scatter(x_spiral[:, 0], x_spiral[:, 1], c=y_spiral)
plt.show()
```



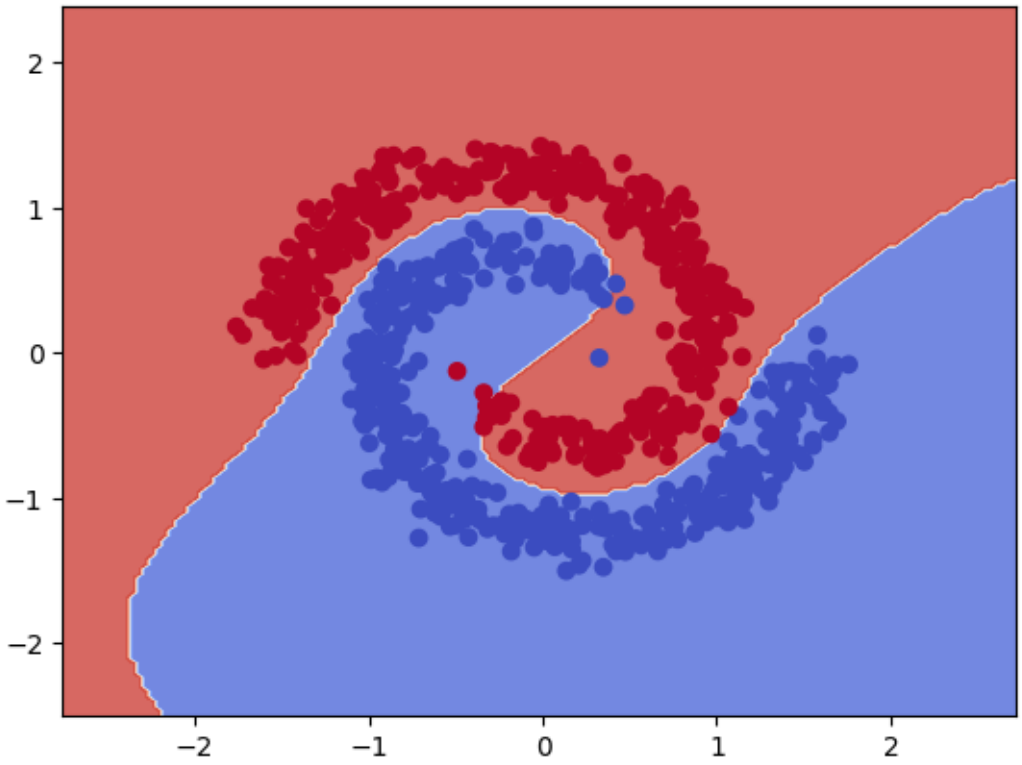
5.2.4 Exercise - Spiral Data

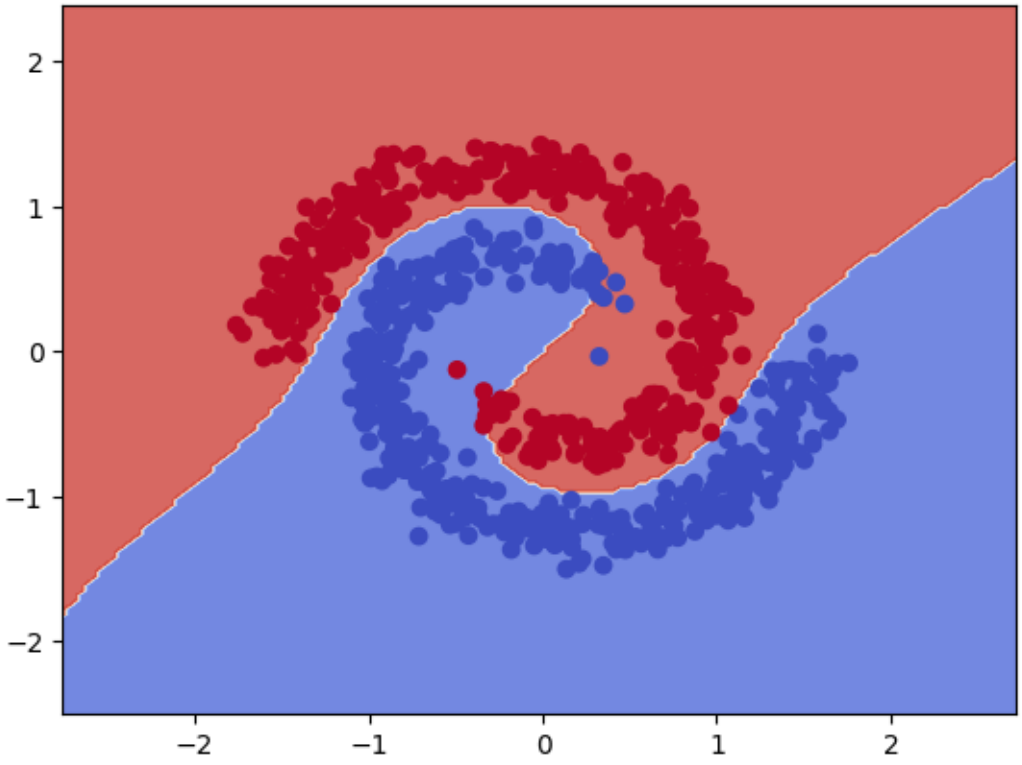
Try to find a good classifier for the spiral data.

Solution 5.2.4

Click to Show/Hide Here or Navigate to Solution 5.2.4 on page 147

```
[ ]:
```


2
1
0
−1
−2
−2
−1
0
1
2


2
1
0
−1
−2
−2
−1
0
1
2

5.3 Custom implementation of Linear Classifier

The core principle of a linear classifier is a linear function $f : \mathbb{R}^n \mapsto \mathbb{R}, f(\vec{x}) = \vec{w}^T\vec{x} + b$, where $\vec{x}$ is the feature vector, $\vec{w}$ is the weight vector a b is the bias term. If we have two classes ω_1 a ω_2, then we add the feature vector $\vec{x}$ to the class ω_1 if $f(\vec{x}) \geq 0$, and to class ω_2 if $f(\vec{x}) < 0$. The function f divides the feature space into two areas divided by a hyperplane. Points where $f(\vec{x}) = 0$ lie exactly on this hyperplane.

To train the classifier we want to find the parameters of the classifier so that the hyperplane separates the training set the best.

We will work with some synthetic data where the individual classes are represented as two separate normal distributions.

5.3.1 Exercise - Prediction

We will first implement a function which takes in a feature vector and predicts the resulting class.

The function will return the class from $\{-1, 1\}$. We will use the definition of standard classifier $f(\vec{x}) = \vec{w}^T\vec{x} + b$. The prediction function should also return the class which can be simply done by applying signum: $\text{sgn}(f(\vec{x}))$.

Your task is now to implement the function predict which takes the vector $\vec{x}$ on its input and outputs the predicted class.

To verify the results we can use the the following code to display the classifier.

[66]:
```
def display_binary_cls(x, y, w, b):
  if x.shape[1] != 2:
    raise TypeError("x has to represent 2-d points of shape (n, 2)")
  if y.shape[0] != x.shape[0]:
    raise TypeError("x and y have to have same number of rows")
  if w.shape[0] != 2:
    raise TypeError("w has to be an array of shape (2,) ")

  xmin = np.min(x[:, 0])
  xmax = np.max(x[:, 0])
  ymin = np.min(x[:, 1])
  ymax = np.max(x[:, 1])

  plt.xlim(xmin=xmin - 0.1, xmax=xmax + 0.1)
  plt.ylim(ymin=ymin - 0.1, ymax=ymax + 0.1)


  if w[1] == 0.0:
    plt.plot([-b / w[0], -b / w[0]], [ymin, ymax], c='black')
  else:
```

```
    ymin = (- w[0] * xmin - b) / w[1]
    ymax = (- w[0] * xmax - b) / w[1]

    plt.plot([xmin, xmax], [ymin, ymax], c='black')

  plt.scatter(x[:, 0], x[:, 1], c=y)
  plt.show()
```

Implement the function `predict_linear` which will take on input a matrix `x` of shape $m \times n$ where m is the number of feature vectors and n is the dimension of the feature vectors (e.g. each row of the matrix is one feature vector), the weights vector `w` and the bias parameter `b` and returs a vector of length m with the predicted classes.

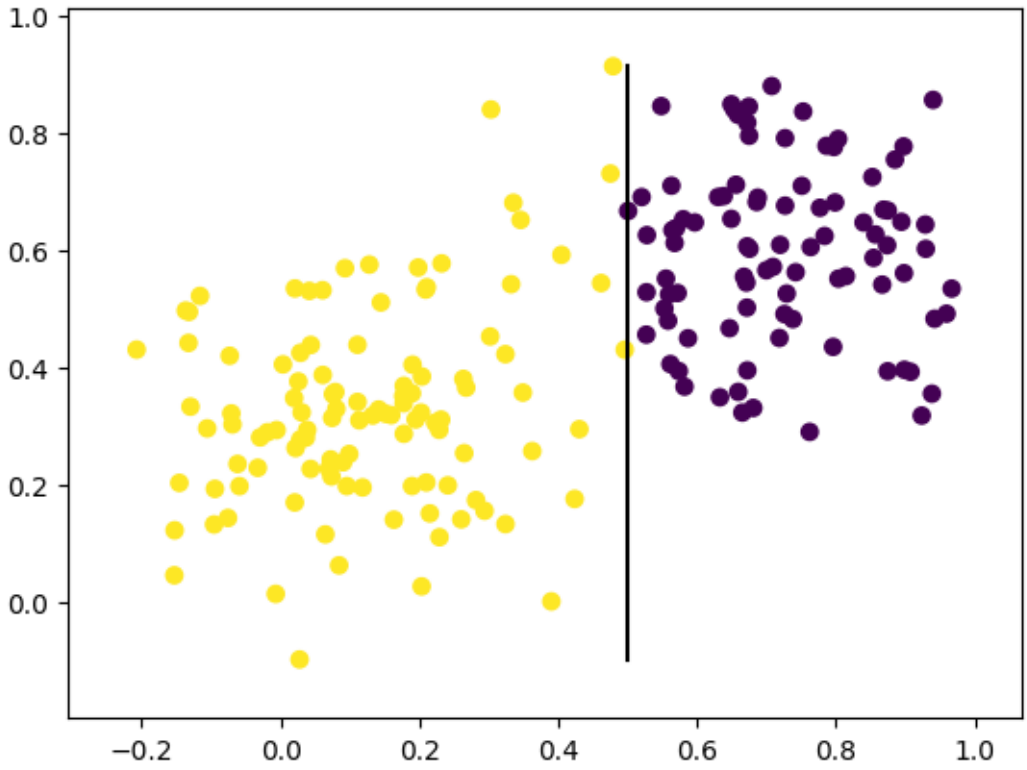
Solution 5.3.1

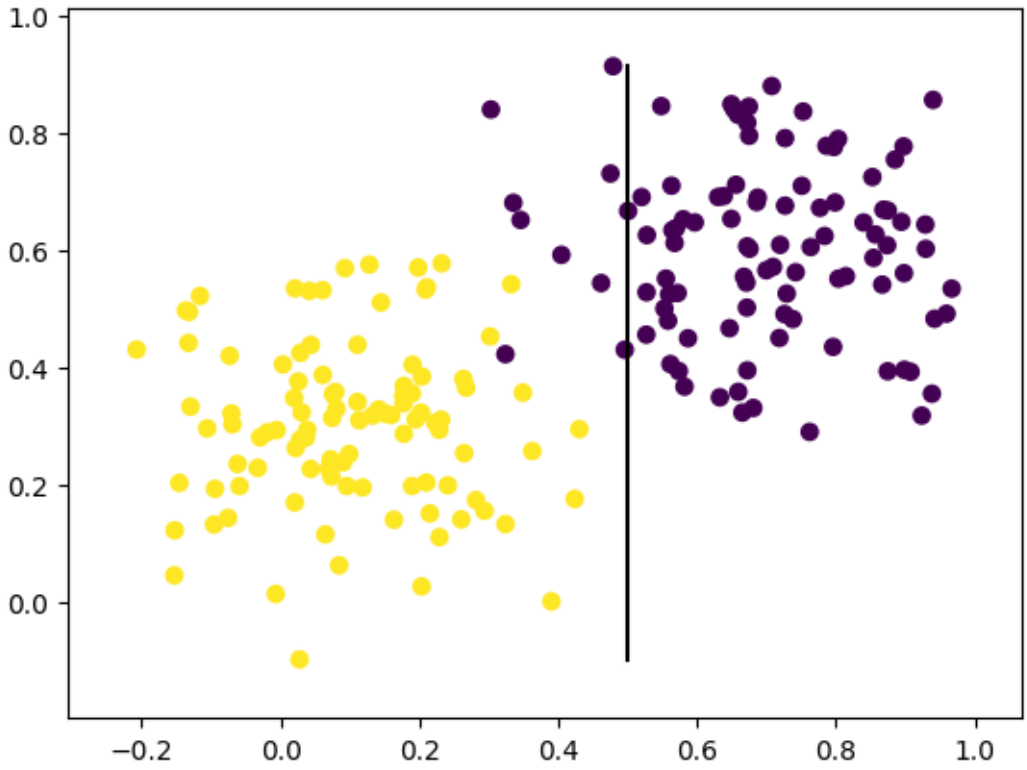
Click to Show/Hide Here or Navigate to Solution 5.3.1 on page 148

[ ]:
```
```

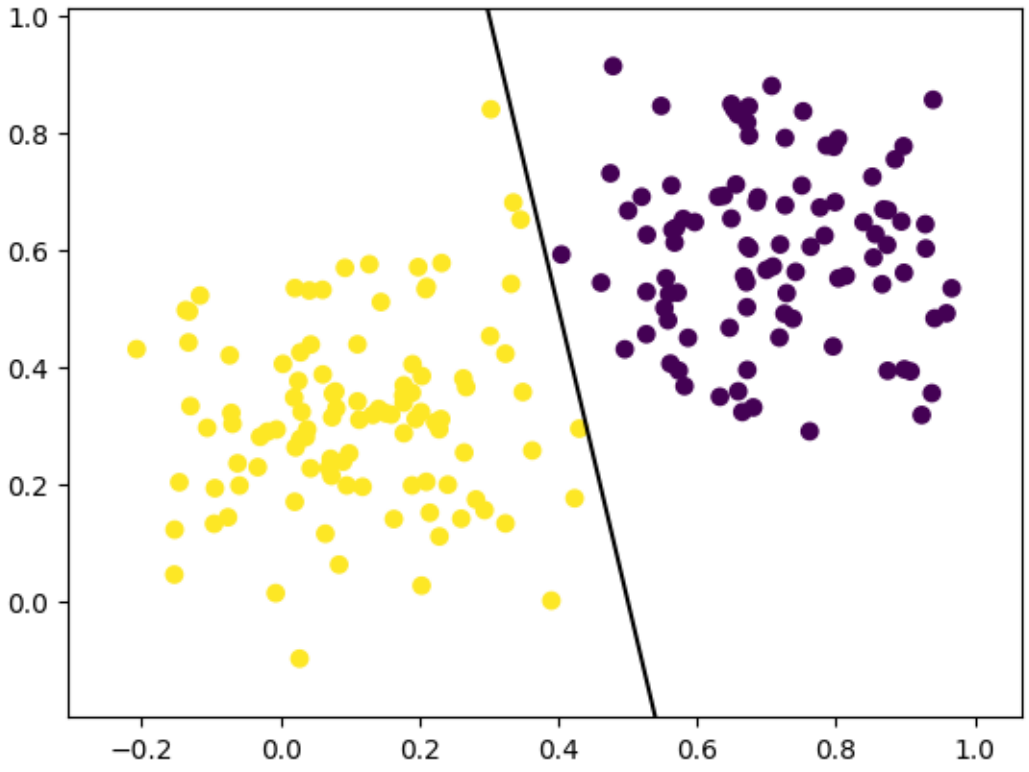
You can test the code on randomly guessed parameters and see if the classes are separated by the give line as plotted in the output.

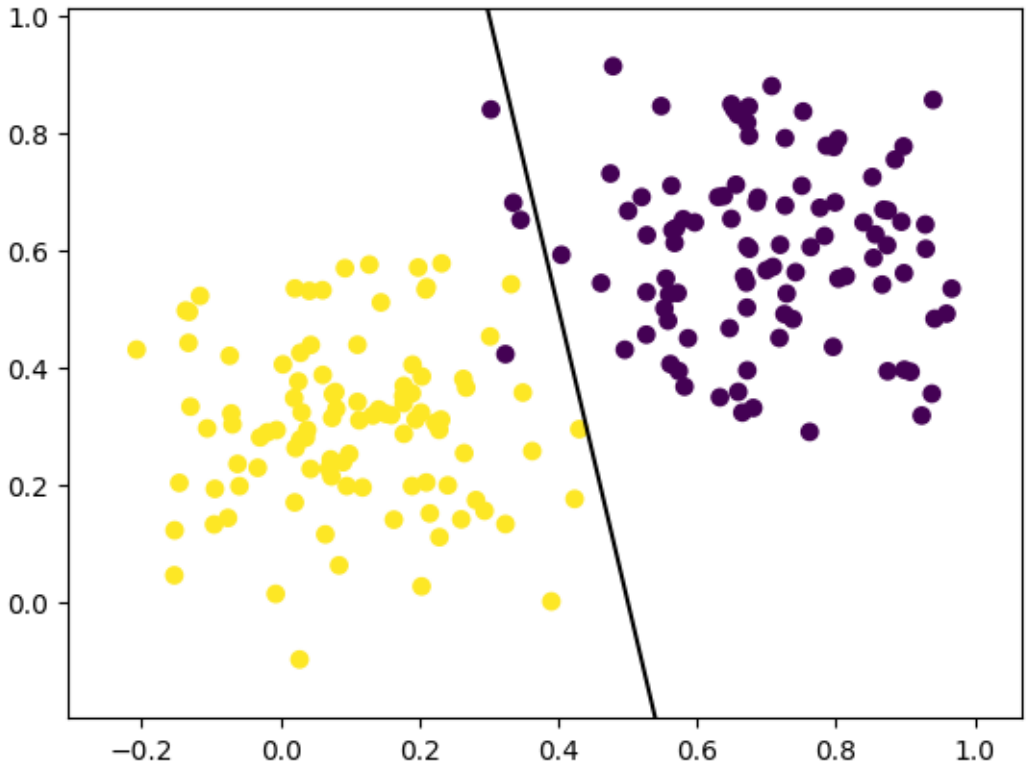
[68]:
```
w = np.array([-1.0,0.0])
b = 0.5

y_pred = predict_linear(x, w, b)
display_binary_cls(x, y_pred, w, b)
display_binary_cls(x, y, w, b)

acc = np.sum(y_pred == y)/len(y)
print("Accuracy of classifier: ", acc)



w = np.array([-1.0, -0.2])
b = 0.5

y_pred = predict_linear(x, w, b)
display_binary_cls(x, y_pred, w, b)
display_binary_cls(x, y, w, b)

acc = np.sum(y_pred == y)/len(y)
print("Accuracy of classifier: ", acc)
```


1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0

```
Accuracy of classifier:  0.955
```






```
Accuracy of classifier:  0.98
```

5.3.2 Exercise - Training

The guessed parameters work fine, but usually we want to determine them automatically (in other words we want the machine to learn - hence machine learning). We wil convert this problem to the standard optimization task. We will try to find parameters $\vec{w}$ and b, such that they minimize a function $L(\vec{w}, b, X, Y)$, where $X = (\vec{x}_1, \vec{x}_2, \ ... \ \vec{x}_m)$ represents the input data and a $Y = (y_1, y_2, \ ..., \ y_m)$ the labeled classes.

L is also called a loss function. Today we will use the Hinge Loss:

$$L(\vec{w}, b, X, Y) = \frac{1}{m} \sum_{i=1}^{m} \max\left(0, 1 - y_i \left(\vec{w}^T \vec{x}_i + b\right)\right).$$

To find the optimal parameters we will use gradient descent. It relies on iterative calculation of the parameters $\vec{w}$ and b using the gradient.The iteration begins with a guesss and then we iterate:

$$b^{n+1} = b^n - \eta \frac{\partial L}{\partial b}(\vec{w}^n, b^n, X, Y),$$

$$w_j^{n+1} = w_j^n - \eta \frac{\partial L}{\partial w_j}(\vec{w}^n, b^n, X, Y),$$

where η is the step size. To do this we need to calculate the gradients:

$$\frac{\partial L}{\partial b}(\vec{w}_j^n, b^n, X, Y) = \frac{1}{m}\sum_{i=1}^{m} -z_i^n y_i,$$

$$\frac{\partial L}{\partial w_j}(\vec{w}_j^n, b^n, X, Y) = \frac{1}{m}\sum_{i=1}^{m} -z_i^n y_i x_{i,j},$$

where $x_{i,j}$ is the j-th element of the i-th vector from the training set X and

$$z_i^n = \begin{cases} 1, \ if \ \ \left(\vec{w}^{nT}\vec{x}_i + b\right) y_i < 1 \\ 0, \ otherwise \end{cases}$$

Implement the function `train_linear_cls(x, y, w_init, b_init, eta, n)` which takes $X, Y, \vec{w}^0, b^0$ and η on input as well as the number of iterations n. On output it returns $\vec{w}^n$ a b^n. At every 100th step it should print out the value of the loss function.

As a an extra exercise you can try to calculate the gradient if we add a regularization term to the loss function:

$$L(\vec{w}, b, X, Y) = \frac{1}{m}\sum_{i=1}^{m} \max\left(0, 1 - y_i\left(\vec{w}^T\vec{x}_i + b\right)\right) + \lambda\sum_{i=1}^{2} w_i^2$$

The parameter λ will then be added as another argument of the `train_linear_cls` function.

Solution 5.3.2

Click to Show/Hide Here or Navigate to Solution 5.3.2 on page 149

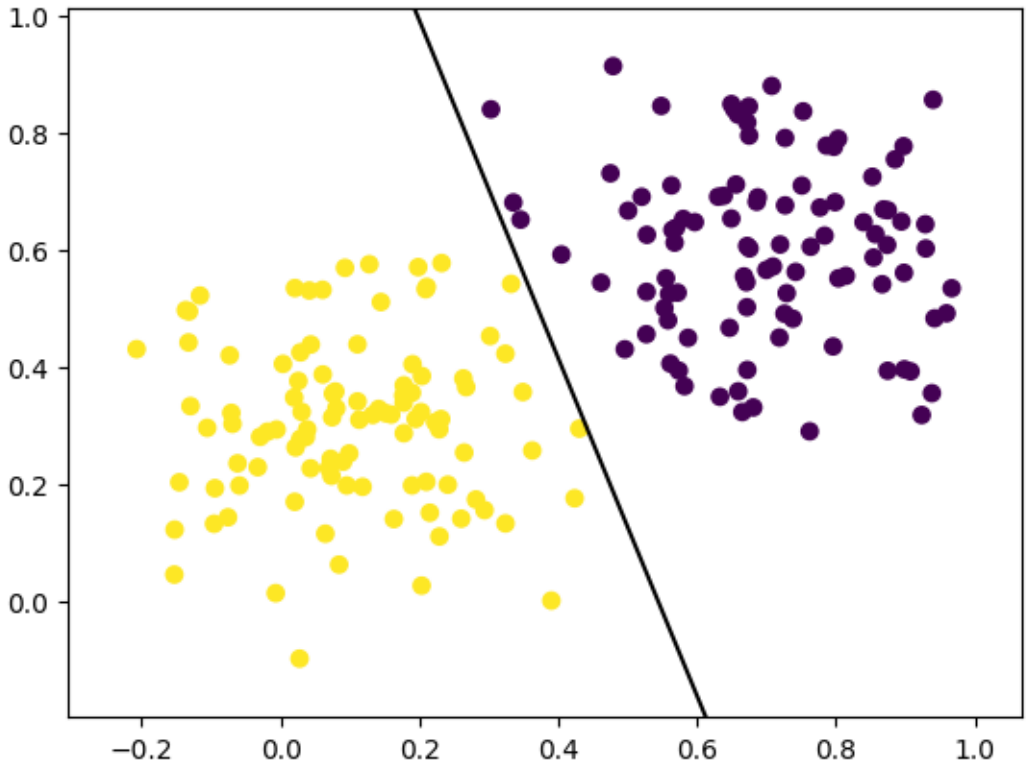
[ ]:

We can test the code.

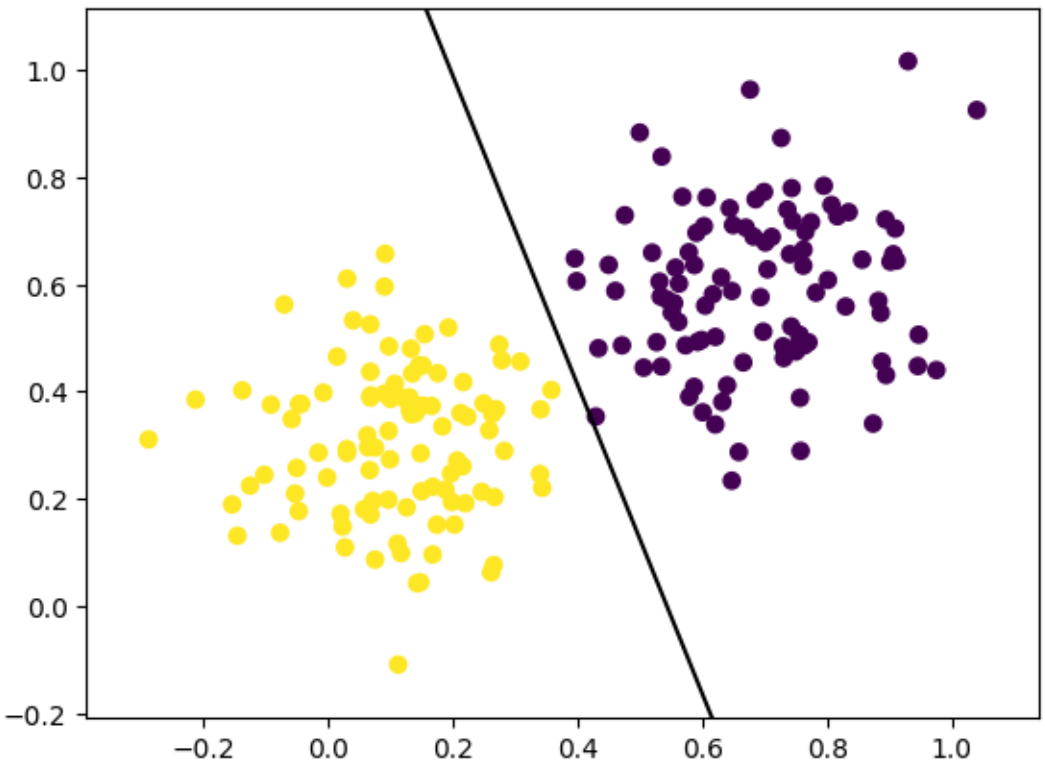
[72]:
```
w_train, b_train = train_binary_cls(x, y, np.array([0.0, 0.5]), -1.0, 1e-2, 5000)
print(w_train, b_train)
```

```
y_pred = predict_linear(x, w_train, b_train)
display_binary_cls(x, y_pred, w_train, b_train)
acc = np.sum(y_pred == y)/len(y)
print("Accuracy of classifier on training data: ", acc)


y_pred_test = predict_linear(x_test, w_train, b_train)
display_binary_cls(x_test, y_pred_test, w_train, b_train)
acc_test = np.sum(y_pred_test == y_test)/len(y)
print("Accuracy of classifier on test data data: ", acc_test)
```

```
At step 0 loss: 1.074400979741701
At step 1000 loss: 0.48882695163693013
At step 2000 loss: 0.1963507697301243
At step 3000 loss: 0.1398970157745743
At step 4000 loss: 0.10855273262331816
[-4.19693291 -1.45659813] 2.281849999999942
```




```
Accuracy of classifier on training data:  0.995
```




```
Accuracy of classifier on test data data:  0.98
```

Chapter 6

Feature Selection and Normalization, Naive Bayes Classifier

In this chapter we will cover some methods of features selection and normalization and some transformations of data from categorical type to a numeric version. We will use these methods in conjunction with the Bayes Classifier [11].

6.1 Feature Selection

We can easily perform feature selection using scikit-learn. We will work with a database of wines and their respective qualities for simplicity.

We can use the `sklearn.feature_selection.SelectKBest` to perform filter-type approach to feature selection. We will also need to select a measure such as `chi2`, `f_classif`, `mutual_info_classif` to evaluate the individual features.

[73]:
```
from sklearn.datasets import load_wine
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, f_classif, mutual_info_classif
X, y = load_wine(return_X_y=True)
bunch = load_wine()
print("Loaded wine dataset with features: ", bunch.feature_names, " and classes:␣
 ↪", bunch.target_names)
print(X.shape)
print("First vector")
print(X[0, :])

for stat in (chi2, f_classif, mutual_info_classif):
  print("Using stat: ", stat)
  X_new = SelectKBest(stat, k=8).fit_transform(X, y)
  print(X_new.shape)
  print(X_new[0, :])
```

```
Loaded wine dataset with features:  ['alcohol', 'malic_acid', 'ash',
```

```
'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids',
'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue',
'od280/od315_of_diluted_wines', 'proline']  and classes:  ['class_0' 'class_1'
'class_2']
(178, 13)
First vector
[1.423e+01 1.710e+00 2.430e+00 1.560e+01 1.270e+02 2.800e+00 3.060e+00
 2.800e-01 2.290e+00 5.640e+00 1.040e+00 3.920e+00 1.065e+03]
Using stat:  <function chi2 at 0x7943c31271c0>
(178, 8)
[   1.71   15.6   127.      2.8     3.06    5.64    3.92 1065.  ]
Using stat:  <function f_classif at 0x7943c3126a70>
(178, 8)
[1.423e+01 1.710e+00 2.800e+00 3.060e+00 5.640e+00 1.040e+00 3.920e+00
 1.065e+03]
Using stat:  <function mutual_info_classif at 0x7943c0e30e50>
(178, 8)
[1.423e+01 2.800e+00 3.060e+00 2.290e+00 5.640e+00 1.040e+00 3.920e+00
 1.065e+03]
```

6.1.1 Exercise - Which features were used

Modify the code above so that it also outputs the names of the features which were kept. Check the documentation on how to do this.

Solution 6.1.1

Click to Show/Hide Here or Navigate to Solution 6.1.1 on page 150

[ ]:

```
Loaded wine dataset with features:  ['alcohol', 'malic_acid', 'ash',
'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids',
'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue',
```

```
'od280/od315_of_diluted_wines', 'proline']  and classes:  ['class_0' 'class_1'
'class_2']
(178, 13)
First vector
[1.423e+01 1.710e+00 2.430e+00 1.560e+01 1.270e+02 2.800e+00 3.060e+00
 2.800e-01 2.290e+00 5.640e+00 1.040e+00 3.920e+00 1.065e+03]
Using stat:  <function chi2 at 0x7943c31271c0>
Features kept:  ['malic_acid' 'alcalinity_of_ash' 'magnesium' 'total_phenols'
'flavanoids'
 'color_intensity' 'od280/od315_of_diluted_wines' 'proline']
(178, 8)
[   1.71   15.6   127.      2.8     3.06    5.64    3.92 1065.  ]
Using stat:  <function f_classif at 0x7943c3126a70>
Features kept:  ['alcohol' 'malic_acid' 'total_phenols' 'flavanoids'
'color_intensity'
 'hue' 'od280/od315_of_diluted_wines' 'proline']
(178, 8)
[1.423e+01 1.710e+00 2.800e+00 3.060e+00 5.640e+00 1.040e+00 3.920e+00
 1.065e+03]
Using stat:  <function mutual_info_classif at 0x7943c0e30e50>
Features kept:  ['alcohol' 'total_phenols' 'flavanoids' 'proanthocyanins'
 'color_intensity' 'hue' 'od280/od315_of_diluted_wines' 'proline']
(178, 8)
[1.423e+01 2.800e+00 3.060e+00 2.290e+00 5.640e+00 1.040e+00 3.920e+00
 1.065e+03]
```

6.1.2 Exercise - Wrapper

Now you should try to use the wrapper approach by checking out the `sklearn.feature_selection.SequentialFeatureSelector`. Try to get 10 features by using an SVM classifier.

Solution 6.1.2

Click to Show/Hide Here or Navigate to Solution 6.1.2 on page 151

[ ]:

```
Features kept going forward: ['alcohol' 'malic_acid' 'ash' 'total_phenols'
'flavanoids'
 'nonflavanoid_phenols' 'proanthocyanins' 'color_intensity' 'hue'
 'od280/od315_of_diluted_wines']
Features kept going backward: ['alcalinity_of_ash' 'magnesium' 'total_phenols'
'flavanoids'
 'nonflavanoid_phenols' 'proanthocyanins' 'color_intensity' 'hue'
 'od280/od315_of_diluted_wines' 'proline']
```

6.2 Feature normalization and Bayes classifier

We can use the naive Gaussian Bayes classifier from sklearn as `sklearn.naive_bayes.GaussianNB`. We can try it on the Wine dataset by S. Aeberhard and M. Forina [1].

[76]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
import numpy as np

X, y = load_wine(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)

gnb = GaussianNB()
gnb.fit(X_train, y_train)

y_test_pred = gnb.predict(X_test)

print('Accuracy: ', np.sum(y_test_pred == y_test)/len(y_test))
```

```
Accuracy:  0.9166666666666666
```

6.2.1 Exercise - Importance of feature normalization

Now lets use the same classifier, but before it we will apply PCA. First try to only do PCA on the unchanged data and reduce the dimensionality to 2. Check the accuracy. We will see that the accuracy will be way worse.

However, if we first normalize the data and then apply PCA we will achieve better accuracy accuracy. You can normalize the data using `sklearn.preprocessing.StandardScaler`.

Try both of these methods. Since we transform to 2 dimensions you can also visualize the results.

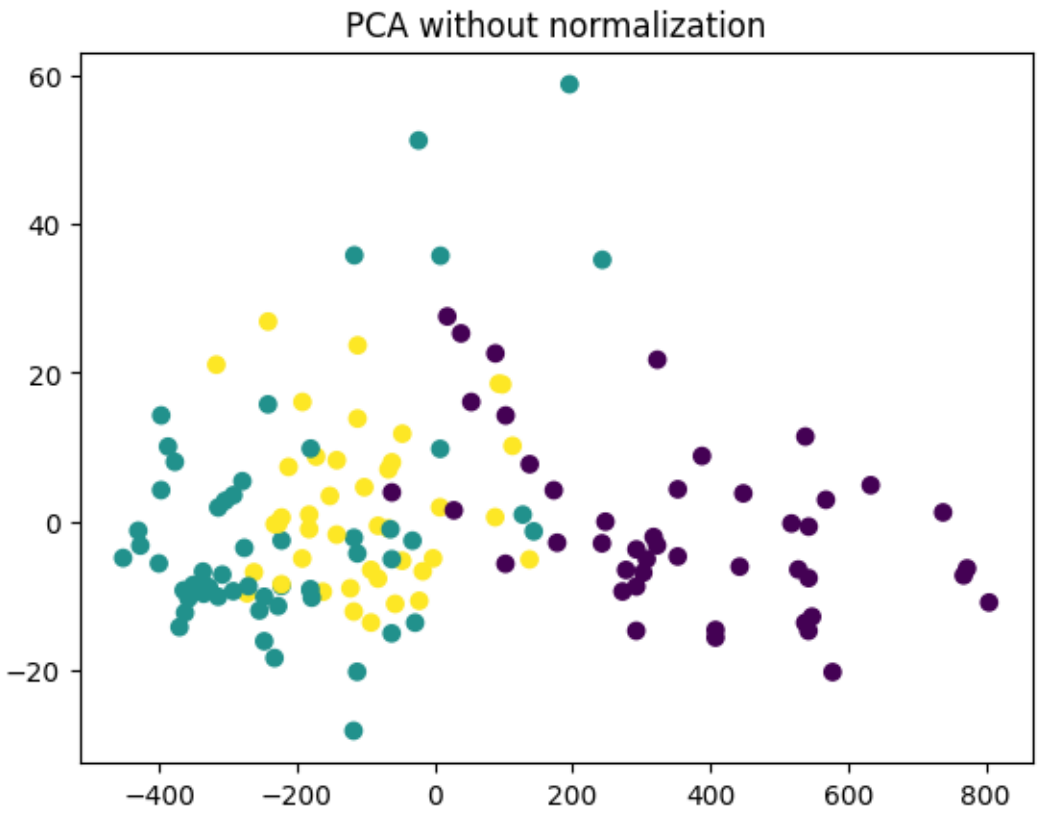
Note: This exercise was inspired by this sklearn example. Do not check before working on this as the solution to this exercise is written there directly in the code.

Solution 6.2.1

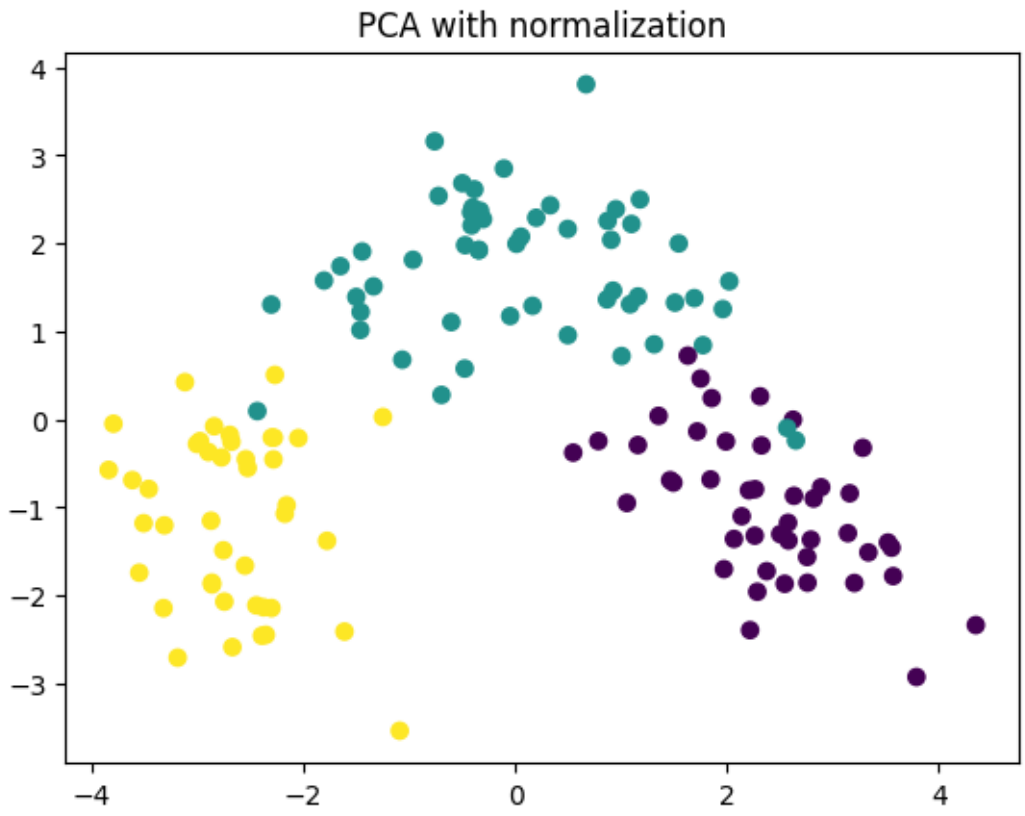
Click to Show/Hide Here or Navigate to Solution 6.2.1 on page 152

[ ]:

```
Accuracy of PCA without normalization:  0.7777777777777778
```




```
Accuracy of PCA with normalization:  0.9722222222222222
```

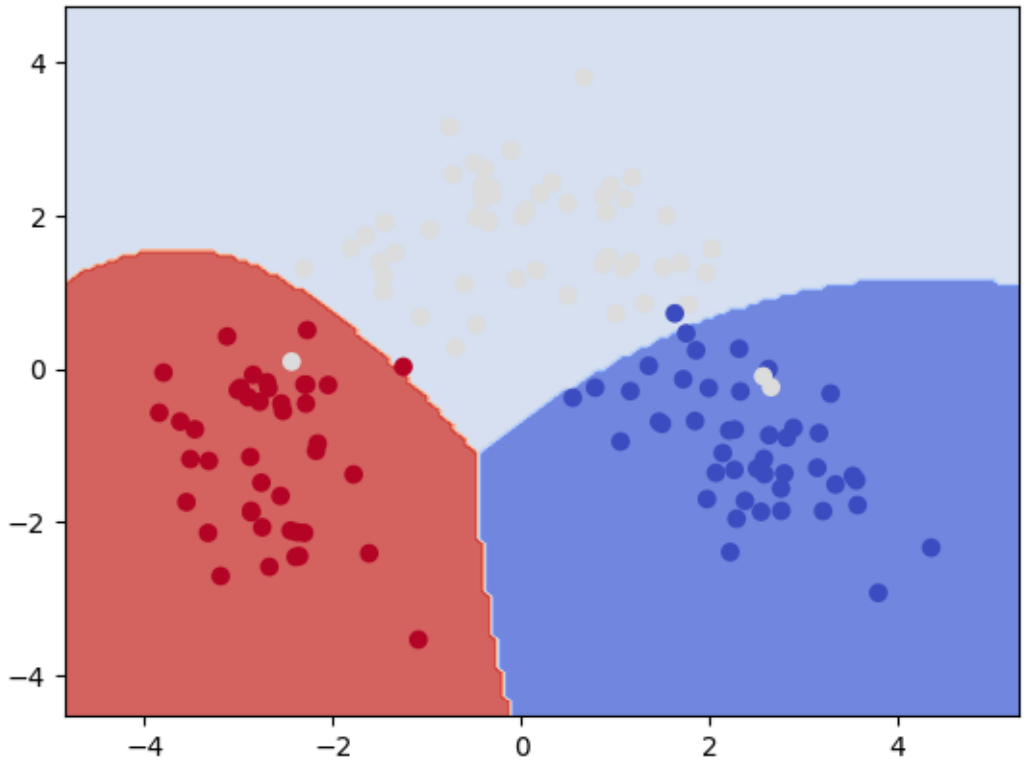


6.2.2 Gaussian classifier decision boundary

We can plot the boundaries using the function from the previous chapter.

[78]:
```
def plot_clf_boundary(clf, x, y):
  x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
  y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
  h_x = (x_max - x_min) / 150
  h_y = (y_max - y_min) / 150
  xx, yy = np.meshgrid(np.arange(x_min, x_max, h_x),
                       np.arange(y_min, y_max, h_y))
  z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

  # Put the result into a color plot
  z = z.reshape(xx.shape)
  plt.contourf(xx, yy, z, cmap=plt.cm.coolwarm, alpha=0.8)

  # Plot also the training points
  plt.scatter(x[:, 0], x[:, 1], c=y, cmap=plt.cm.coolwarm)
  plt.xlim(xx.min(), xx.max())
  plt.ylim(yy.min(), yy.max())
```

```
  # plt.xticks(())
  # plt.yticks(())
  plt.show()

plot_clf_boundary(gnb, X_train_scaled_transformed, y_train)
```



6.2.3 Exercise - Sklearn Pipelines

Sometimes we group together multiple operations into one by using from `sklearn.pipeline.make_pipeline`.

Rewrite the code in the block below using the pipeline.

Solution 6.2.3

Click to Show/Hide Here or Navigate to Solution 6.2.3 on page 153

[ ]:

```
Accuracy with feature selection, normalization and PCA:  1.0
```

[80]:
```
from sklearn.pipeline import make_pipeline

pipeline = make_pipeline(SelectKBest(f_classif, k=8), StandardScaler(),
                         PCA(6), GaussianNB())

pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)

print("Accuracy with feature selection, normalization and PCA: ",
      np.sum(y_pred == y_test) / len(y_test))
```

```
Accuracy with feature selection, normalization and PCA:  1.0
```

6.3 Categorical Naive Bayes Classifier

Now we will work with the Bayes classifier, but we will work with categorical data.

We will load the Car dataset by M. Bohanec and V. Rajkovic [4].

[81]:
```
!wget http://archive.ics.uci.edu/ml/machine-learning-databases/car/car.data

import pandas as pd

car_data = pd.read_csv('car.data', names=['buying','maint','doors','persons',
                                          'lug_boot','safety','class'], sep=",")
print(car_data.head())
print(55 * '*')
print(car_data.describe())
```

```
--2024-01-11 14:49:28--  http://archive.ics.uci.edu/ml/machine-learning-
databases/car/car.data
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:80...
connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: ‘car.data’

car.data                [ <=>                ]  50.65K  --.-KB/s    in 0.1s
```

```
2024-01-11 14:49:28 (393 KB/s) - 'car.data' saved [51867]
```

```
  buying  maint doors persons lug_boot safety  class
0  vhigh  vhigh     2       2    small    low  unacc
1  vhigh  vhigh     2       2    small    med  unacc
2  vhigh  vhigh     2       2    small   high  unacc
3  vhigh  vhigh     2       2      med    low  unacc
4  vhigh  vhigh     2       2      med    med  unacc
**************************************************
       buying  maint doors persons lug_boot safety  class
count    1728   1728  1728    1728     1728   1728   1728
unique      4      4     4       3        3      3      4
top     vhigh  vhigh     2       2    small    low  unacc
freq      432    432   432     576      576    576   1210
```

6.3.1 Exercise - Categorical Naive Bayes

We cannot use the categorical naive bayes directly. We have to first convert the data to the appropriate format where each contains contains only integers 0 to $n_{categories} - 1$ representing the various categories with given integers.

In order to do that we can use the `sklearn.preprocessing.LabelEncoder` on the target classes and `sklearn.preprocessing.OrdinalEncoder`.

Note: `LabelEncoder` and `OrdinalEncoder` do essentially the same thing, but the former only works on data with shape (n_samples) and the latter works on data of shape (n_samples, n_features). You could use `LabelEncoder` only, but you would need to apply the transform column by column. To do this you could use the method `apply` of the pandas `DataFrame` class. It applies a given function to every column.

[82]:
```
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder

data_X = car_data.drop('class', axis='columns')
data_y = car_data['class']

oe = OrdinalEncoder()
X = oe.fit_transform(data_X)

le = LabelEncoder()
y = le.fit_transform(data_y)


print(X[:5, :])
print(y[:5])
```

```
[[3. 3. 0. 0. 2. 1.]
 [3. 3. 0. 0. 2. 2.]
 [3. 3. 0. 0. 2. 0.]
```

```
 [3. 3. 0. 0. 1. 1.]
 [3. 3. 0. 0. 1. 2.]]
[2 2 2 2 2]
```

Now you can train the `sklearn.naive_bayes.CategoricalNB`.

Solution 6.3.1

Click to Show/Hide Here or Navigate to Solution 6.3.1 on page 154

[ ]:

```
Categorical NB accuracy:  0.8757225433526011
```

6.4 Converting data from categorical to numeric

So far we have mostly worked with numerical data, but what if we have categorical data? We can transform them using different approaches. In case the categorical data has a direct numerical representation e.g. grades A,B,C,D,E,Fx can be converted to integers. We can manually make the change.

However, in other cases such as categories: dog, cat and hamster. Assigning them values 0, 1 and 2 would not make sense as that would also mean that in our representation a cat is an average of a dog and a hamster, which is not meaningful for most purposes.

Instead, we can use one-hot encoding. Which creates new features. One for each category and the values will only be 0 and 1. 1 will be used when the given category is the one representing the original feature. So in our new representation a dog would be $(1, 0, 0)$, cat would be $(0, 1, 0)$ and a hamster would be $(0, 0, 1)$.

We will now try to do this with the car dataset.

We can then call the function `pd.get_dummies` on a DataFrame while specifying which columnt to use to get the one-hot coded categories. The `prefix` keyword creates a column with a more representative name.

[84]:
```
one_hot = pd.get_dummies(car_data['maint'], prefix='maint')

print(one_hot.head())
```

```
   maint_high  maint_low  maint_med  maint_vhigh
0           0          0          0            1
1           0          0          0            1
2           0          0          0            1
3           0          0          0            1
4           0          0          0            1
```

6.4.1 Exercise - Converting the car dataset to one-hot encoding

Given how the get_dummies function work you should create a new dataframe which will contain all of the original data, but the categorical columns will be transformed to one-hot encoding.

Check out the methods `join` and `drop` of the DataFrame class.

Alternatively, you can use the `sklearn.preprocessing.OneHotEncoder`.

The variable `cat_feats` below contains a list of the names of columns which are categorical in nature. You can use this in your code.

Solution 6.4.1

Click to Show/Hide Here or Navigate to Solution 6.4.1 on page 155

[ ]:
```

```

```
(1728, 21)
      buying_high  buying_low  buying_med  buying_vhigh  maint_high  \
0               0           0           0             1           0
1               0           0           0             1           0
2               0           0           0             1           0
3               0           0           0             1           0
4               0           0           0             1           0
...           ...         ...         ...           ...         ...
1723            0           1           0             0           0
1724            0           1           0             0           0
1725            0           1           0             0           0
1726            0           1           0             0           0
```

```
1727            0           1           0            0           0

      maint_low  maint_med  maint_vhigh  doors_2  doors_3  ...  doors_5more  \
0             0          0            1        1        0  ...            0
1             0          0            1        1        0  ...            0
2             0          0            1        1        0  ...            0
3             0          0            1        1        0  ...            0
4             0          0            1        1        0  ...            0
...         ...        ...          ...      ...      ...  ...          ...
1723          1          0            0        0        0  ...            1
1724          1          0            0        0        0  ...            1
1725          1          0            0        0        0  ...            1
1726          1          0            0        0        0  ...            1
1727          1          0            0        0        0  ...            1

      persons_2  persons_4  persons_more  lug_boot_big  lug_boot_med  \
0             1          0             0             0             0
1             1          0             0             0             0
2             1          0             0             0             0
3             1          0             0             0             1
4             1          0             0             0             1
...         ...        ...           ...           ...           ...
1723          0          0             1             0             1
1724          0          0             1             0             1
1725          0          0             1             1             0
1726          0          0             1             1             0
1727          0          0             1             1             0

      lug_boot_small  safety_high  safety_low  safety_med
0                  1            0           1           0
1                  1            0           0           1
2                  1            1           0           0
3                  0            0           1           0
4                  0            0           0           1
...              ...          ...         ...         ...
1723               0            0           0           1
1724               0            1           0           0
1725               0            0           1           0
1726               0            0           0           1
1727               0            1           0           0

[1728 rows x 21 columns]
```

Now we can try to apply use a linear SVM on the data and we obtain a much better accuracy.

[86]:
```
from sklearn.svm import SVC

X = one_hot_data
```

```
y = LabelEncoder().fit_transform(car_data['class'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=10)

svm = SVC(kernel='linear')
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
print('Accuracy: ', np.sum(y_pred == y_test)/len(y_test))
```

```
Accuracy:  0.953757225433526
```

Chapter 7

kNN, Hyperparameter selection, Evaluation

In this chapter we will cover the k nearest neighbors classifier, hyperparameter tuning and evaluation.

7.1 kNN

First we will work with kNN [8]. For now we will use the sklearn library and use `sklearn.neighbors.KNeighborsClassifier`. In the last (bonus) exercise you can implement your own version kNN.

We will now work with the Fisher Iris database. Below is the standard code for this task. We will use `StandardScaler` in a pipeline as that is an important step for kNN.

[87]:
```
import numpy as np
import sklearn
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

data = datasets.load_wine()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=456789)

knn = make_pipeline(StandardScaler(), KNeighborsClassifier(n_neighbors=3,␣
↪metric='euclidean'))
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
```

```
print("Accuracy: ", np.sum(y_pred == y_test)/ len(y_pred))
```

```
Accuracy:  0.9444444444444444
```

7.1.1 Exercise - Validation

Now we will try to find optimal k and metric to use with out data. First we will again split the training set to a proper training set and a validation set. We will then try various values of k and various metrics fitted on the training data. We will then calculate the accuracy on the validation set. The combination that acieved the best results on the validation set can then be used. Finally, we will evaluate the values on test set.

You should try various values of k (`n_neighbors`) and also various metrics. You can find the information on metrics in this part of the documentation. Alternatively, you can just get a list of usable metrics by calling `sorted(sklearn.neighbors.VALID_METRICS['brute'])`

Note: Some metrics also have additional keywords. You can check them out in the docimentation, but you can also

Solution 7.1.1

Click to Show/Hide Here or Navigate to Solution 7.1.1 on page 156

[ ]:

```
Accuracy for k = 1 and chebyshev metric: 0.9010989010989011
Accuracy for k = 3 and chebyshev metric: 0.9010989010989011
Accuracy for k = 5 and chebyshev metric: 0.9120879120879121
Accuracy for k = 7 and chebyshev metric: 0.9230769230769231
Accuracy for k = 9 and chebyshev metric: 0.945054945054945
Accuracy for k = 11 and chebyshev metric: 0.9340659340659341
Accuracy for k = 15 and chebyshev metric: 0.9230769230769231
Accuracy for k = 19 and chebyshev metric: 0.9230769230769231
Accuracy for k = 23 and chebyshev metric: 0.9120879120879121
Accuracy for k = 29 and chebyshev metric: 0.9010989010989011
Accuracy for k = 1 and euclidean metric: 0.9340659340659341
Accuracy for k = 3 and euclidean metric: 0.967032967032967
Accuracy for k = 5 and euclidean metric: 0.978021978021978
Accuracy for k = 7 and euclidean metric: 0.978021978021978
Accuracy for k = 9 and euclidean metric: 0.978021978021978
Accuracy for k = 11 and euclidean metric: 0.989010989010989
Accuracy for k = 15 and euclidean metric: 0.978021978021978
Accuracy for k = 19 and euclidean metric: 0.978021978021978
Accuracy for k = 23 and euclidean metric: 0.978021978021978
Accuracy for k = 29 and euclidean metric: 0.978021978021978
Accuracy for k = 1 and manhattan metric: 0.9560439560439561
Accuracy for k = 3 and manhattan metric: 0.989010989010989
Accuracy for k = 5 and manhattan metric: 0.989010989010989
Accuracy for k = 7 and manhattan metric: 0.989010989010989
Accuracy for k = 9 and manhattan metric: 0.989010989010989
Accuracy for k = 11 and manhattan metric: 0.989010989010989
Accuracy for k = 15 and manhattan metric: 0.989010989010989
Accuracy for k = 19 and manhattan metric: 1.0
```

```
Accuracy for k = 23 and manhattan metric: 0.978021978021978
Accuracy for k = 29 and manhattan metric: 0.967032967032967
Best hyperparams: k = 19 with manhattan metric
Accuracy on test data:  0.9649122807017544
```

7.1.2 Exercise K-fold cross validation

Especially with smaller datsets we may not have enough data to perform validation reliably. In that case we can use K-fold cross-validation which splits the data into K subsets. Then the classifier is trained K times always with one of the subsets left for validation and others used for testing. The accuracy is then averaged over all of the validation subsets. This is more indicative of the generalization ability of the network. In odred to perfom this you can use `sklearn.model_selection.KFold`.

Solution 7.1.2

Click to Show/Hide Here or Navigate to Solution 7.1.2 on page 157

[ ]:

```
Accuracy for k = 1 and chebyshev metric: 0.9120879120879121 for fold 0
Accuracy for k = 1 and chebyshev metric: 0.967032967032967 for fold 1
Accuracy for k = 1 and chebyshev metric: 0.8901098901098901 for fold 2
Accuracy for k = 1 and chebyshev metric: 0.9230769230769231 for fold 3
Accuracy for k = 1 and chebyshev metric: 0.9560439560439561 for fold 4
Accuracy for k = 1 and chebyshev metric: 0.9296703296703297 for all folds
Accuracy for k = 3 and chebyshev metric: 0.9010989010989011 for fold 0
Accuracy for k = 3 and chebyshev metric: 0.989010989010989 for fold 1
Accuracy for k = 3 and chebyshev metric: 0.9230769230769231 for fold 2
Accuracy for k = 3 and chebyshev metric: 0.945054945054945 for fold 3
Accuracy for k = 3 and chebyshev metric: 0.9340659340659341 for fold 4
Accuracy for k = 3 and chebyshev metric: 0.9384615384615385 for all folds
... Some output omitted for clarity ...
Accuracy for k = 29 and manhattan metric: 0.945054945054945 for fold 0
Accuracy for k = 29 and manhattan metric: 0.978021978021978 for fold 1
Accuracy for k = 29 and manhattan metric: 0.9340659340659341 for fold 2
Accuracy for k = 29 and manhattan metric: 0.945054945054945 for fold 3
Accuracy for k = 29 and manhattan metric: 0.9340659340659341 for fold 4
Accuracy for k = 29 and manhattan metric: 0.9472527472527472 for all folds
Best hyperparams: k = 19 with manhattan metric
Accuracy on test data:  0.9824561403508771
```

7.1.3 Exercise - Custom kNN

We can also implement a custom verion of the kNN classifier. Do this by implementing a function `knn` with arguments `x, k, x_train, y_train`. Where `x` stands for a vector we want to classify, `k` for the number of neighbors, `X_train` is a matrix containing the training data with each row representing one feature vector and `y_train` containing the training labels. You can use the Euclidean metric. You can also add a keyword pair argument to the signature which would enable the use of different metrics in this function.

Try to avoid for loops and use broadcasting.

The function `numpy.unique(y, return_counts=True)` might be useful look it up in the documentation.

Note: In oder to get the k neighbors you will need to perform some sort of sorting. Standard sorts usually have $n\log(n)$ time complexity, but we can use argpartition from NumPy (`np.argpartition`) instead. This partitions the array so that the first k elements are actually the k elements with lowest values.

Solution 7.1.3

Click to Show/Hide Here or Navigate to Solution 7.1.3 on page 159

[ ]:
```
```

We should be able to test your implementation

[91]:
```
data = datasets.load_breast_cancer()
# data = datasets.load_iris()
# data = datasets.load_wine()

X = data.data
y = data.target

# you can see how the results may change a bit if we change the random state
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=0)

y_pred = np.array([knn(x, 5, X_train, y_train) for x in X_test])

print("Accuracy: ", np.sum(y_pred == y_test)/len(y_pred))
```

```
Accuracy:  0.9385964912280702
```

7.2 Evaluation

Now we will dicuss some means of evaluation of the trained classifiers.

There are many ways of evaluating our models. We will only cover some of them now.

We will return to the base linear classifier and perform some evaluations. In order to do thes we will work with the breast cancer dataset, but we will only use the first four columns so the results are not so good.

[92]:
```
from sklearn.linear_model import LogisticRegression
data = datasets.load_breast_cancer()
X = data.data[:, :4]
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=456789)

linear = make_pipeline(StandardScaler(), LogisticRegression())
linear.fit(X_train, y_train)

y_pred = linear.predict(X_test)


from sklearn.metrics import accuracy_score, f1_score, classification_report
print("Accuracy: ", accuracy_score(y_test, y_pred))
print("F1 score: ", f1_score(y_test, y_pred))
print("Report: ")
print(classification_report(y_test, y_pred))
```

```
Accuracy:  0.8947368421052632
F1 score:  0.918918918918919
Report:
              precision    recall  f1-score   support

           0       0.97      0.76      0.85        45
           1       0.86      0.99      0.92        69

    accuracy                           0.89       114
   macro avg       0.92      0.87      0.88       114
weighted avg       0.90      0.89      0.89       114
```

For some types of evaluations we need to get scores not just the prediction. This is not possible with all types of classifiers.

[93]:
```
from sklearn.metrics import roc_auc_score

y_proba = linear.predict_proba(X_test)[:, 1]

print("ROC AUC score: ", roc_auc_score(y_test, y_proba))
```

```
ROC AUC score:  0.9655394524959743
```

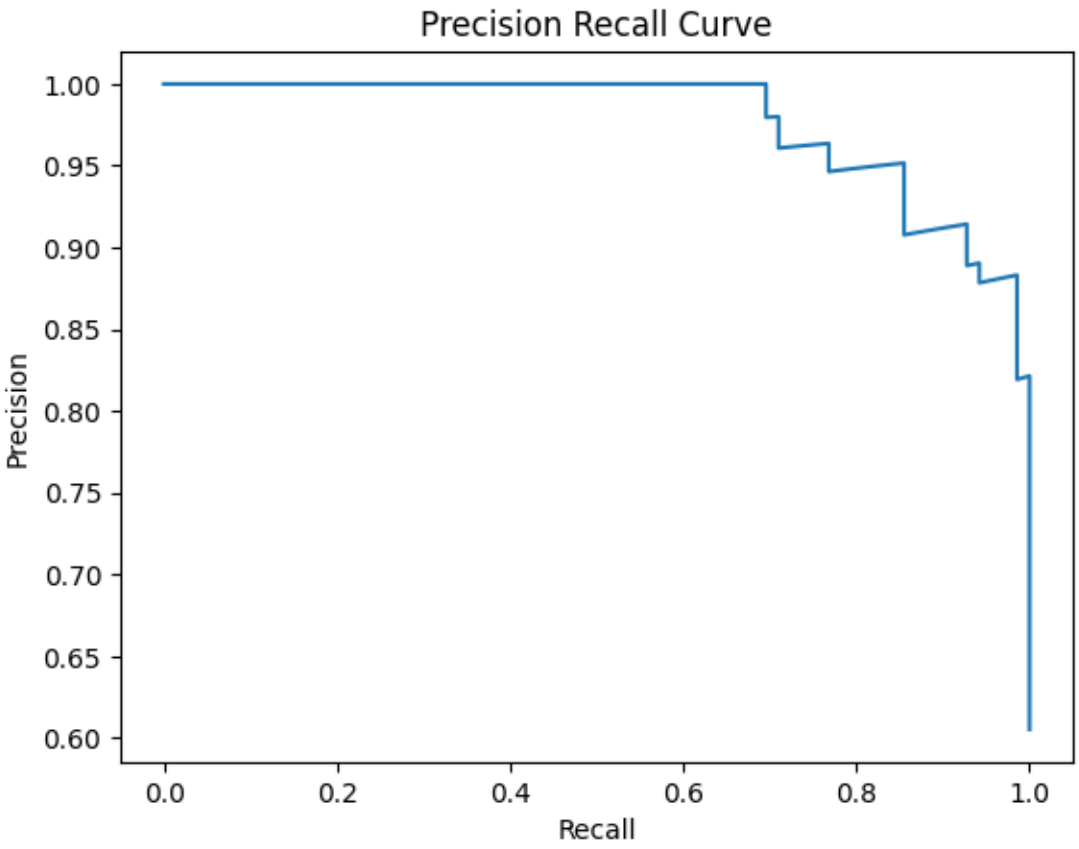
7.2.1 Exercise - ROC and Precision Recall curve

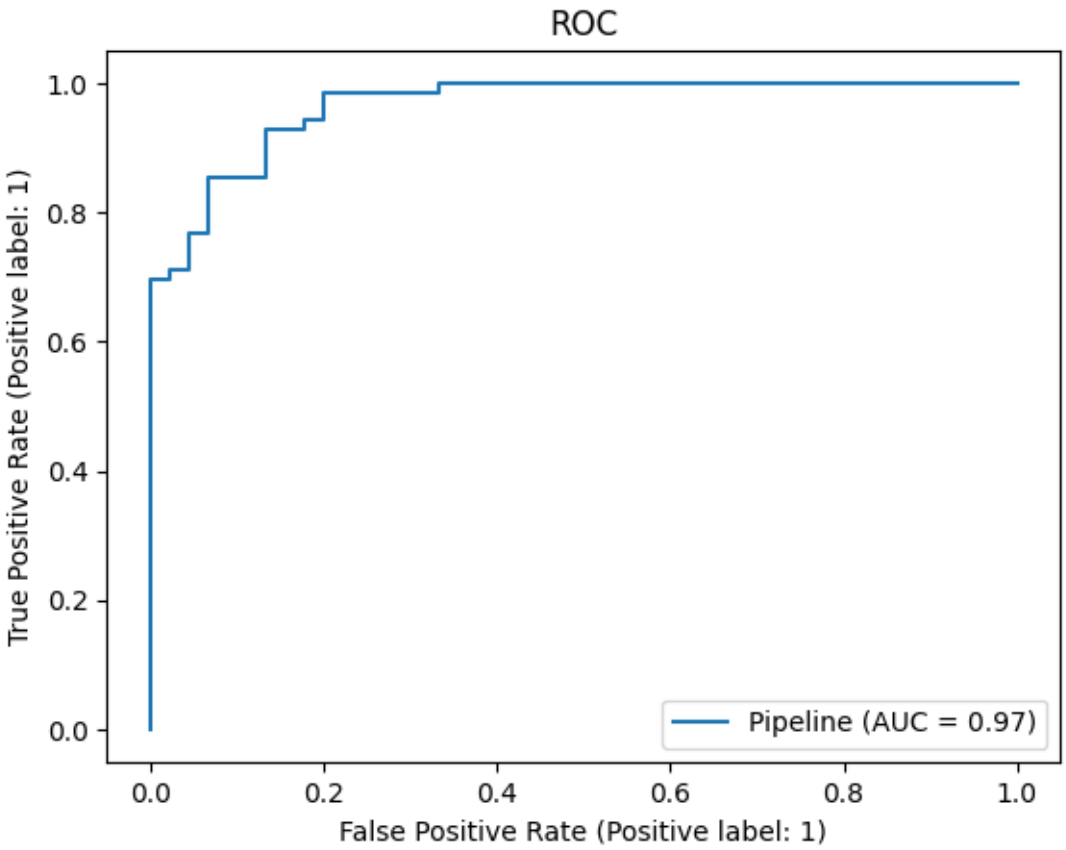
Now try to plot the precision recall or the ROC curve. You can either use `sklearn.metrics.precision_recall_curve`, `sklearn.metrics.roc_curve` and plot them yourself. Or use the `sklearn.metrics.PrecisionRecallDisplay` or use `sklearn.metrics.ROCDisplay`

Solution 7.2.1

Click to Show/Hide Here or Navigate to Solution 7.2.1 on page 160

[ ]:





7.3 Multiple classes

With multiple classes we sometimes have to specify how they are calculated as for example with the F1 score. We can use different ways of averaging the score.

From documentation:

`micro`: Calculate metrics globally by counting the total true positives, false negatives and false positives.

`macro`: Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

`weighted`: Calculate metrics for each label, and find their average weighted by support (the number of true instances for each label). This alters 'macro' to account for label imbalance; it can result in an F-score that is not between precision and recall.

[95]:
```
from sklearn.linear_model import LogisticRegression
data = datasets.load_iris()
X = data.data[:, :2]
y = data.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=456789)

linear = make_pipeline(StandardScaler(), LogisticRegression())
linear.fit(X_train, y_train)

y_pred = linear.predict(X_test)


from sklearn.metrics import accuracy_score, f1_score, classification_report
print("Accuracy: ", accuracy_score(y_test, y_pred))
print("F1 score weighted: ", f1_score(y_test, y_pred, average='weighted'))
print("F1 score micro: ", f1_score(y_test, y_pred, average='micro'))
print("F1 score macro: ", f1_score(y_test, y_pred, average='macro'))
print("Report: ")
print(classification_report(y_test, y_pred))
```

```
Accuracy:  0.8
F1 score weighted:  0.8
F1 score micro:  0.8000000000000002
F1 score macro:  0.8461538461538461
Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       0.77      0.77      0.77        13
           2       0.77      0.77      0.77        13

    accuracy                           0.80        30
   macro avg       0.85      0.85      0.85        30
weighted avg       0.80      0.80      0.80        30
```

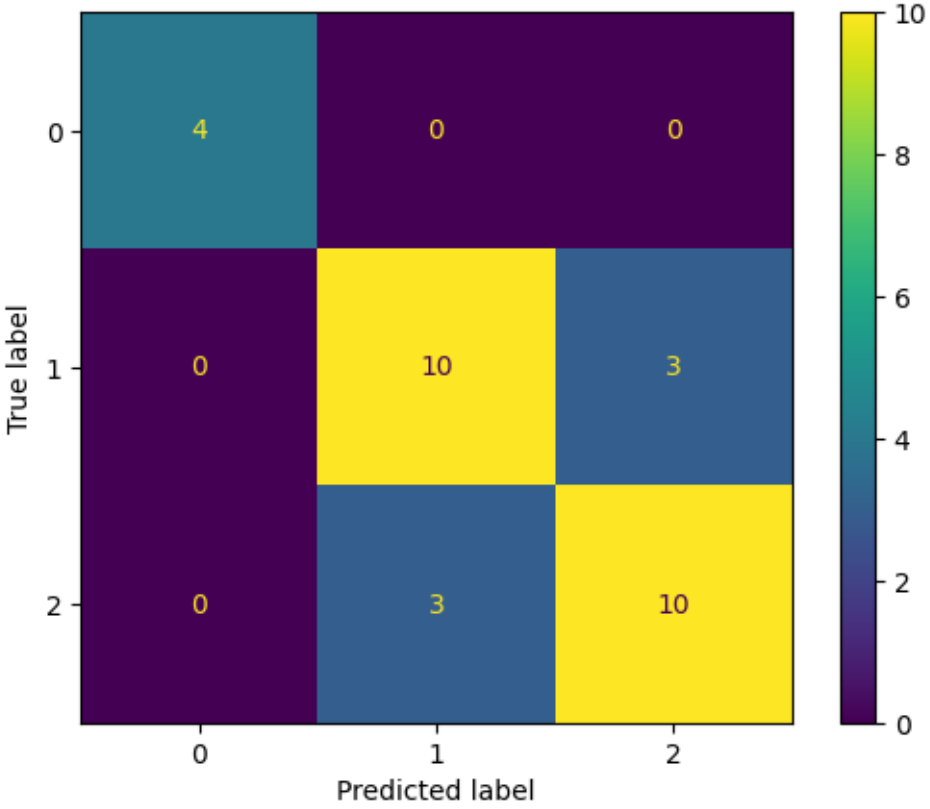
7.3.1 Exercise - Confusion Matrix

From the results it is possible to calculate the confusion matrix. We can use `sklearn.metrics.confusion_matrix`. Or if we also want to diplay it we can use `sklearn.metrics.ConfusionMatrixDisplay`. Try to use these to show the confusion matrix.

Solution 7.3.1

Click to Show/Hide Here or Navigate to Solution 7.3.1 on page 161

[ ]:


4
0
0
0
10
3
0
3
10
0
1
2
True label
0
1
2
Predicted label
10
8
6
4
2
0

Chapter 8

Decision Trees, Random Forests, Boosting

In this chapter we will cover decision trees [5] and related methods [12, 9]. We will work with the same Census dataset [2] as before.

[97]:
```
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.names
!tail -n 16 adult.names
```

```
--2024-01-11 14:52:18--  https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: ‘adult.data.1’

adult.data.1            [   <=>              ]   3.79M  7.19MB/s    in 0.5s

2024-01-11 14:52:19 (7.19 MB/s) - ‘adult.data.1’ saved [3974305]

--2024-01-11 14:52:19--  https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.names
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: ‘adult.names.1’

adult.names.1           [ <=>                ]   5.11K  --.-KB/s    in 0s
```

```
2024-01-11 14:52:19 (50.0 MB/s) - 'adult.names.1' saved [5229]

>50K, <=50K.

age: continuous.
workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov,
State-gov, Without-pay, Never-worked.
fnlwgt: continuous.
education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm,
Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th,
Preschool.
education-num: continuous.
marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed,
Married-spouse-absent, Married-AF-spouse.
occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial,
Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-
fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
relationship: Wife, Own-child, Husband, Not-in-family, Other-relative,
Unmarried.
race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
sex: Female, Male.
capital-gain: continuous.
capital-loss: continuous.
hours-per-week: continuous.
native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany,
Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran,
Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal,
Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia,
Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador,
Trinadad&Tobago, Peru, Hong, Holand-Netherlands.
```

[98]:
```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder

data = pd.read_csv('adult.data', names=['age', 'workclass', 'fnlwgt',
                                        'education', 'education-num',
                                        'marital-status', 'occupation',
                                        'relationship', 'race', 'sex',
                                        'capital-gain', 'capital-loss',
                                        'hours-per-week', 'native-country',
                                        'salary'], sep=",")
print(data.head())
print(55 * '*')
print(data.describe())

data_X = data.drop('salary', axis='columns')
data_y = data['salary']
```

```
cat_ix = data_X.select_dtypes(include=['object', 'bool']).columns

for cat in cat_ix:
  one_hot = pd.get_dummies(data_X[cat], prefix=cat)
  data_X = data_X.drop(cat, axis='columns')
  data_X = data_X.join(one_hot)

print(data_X.shape)
print(data_X)

le = LabelEncoder()
data_y = le.fit_transform(data_y)
```

```
   age          workclass  fnlwgt   education  education-num  \
0   39          State-gov   77516   Bachelors             13
1   50   Self-emp-not-inc   83311   Bachelors             13
2   38            Private  215646     HS-grad              9
3   53            Private  234721        11th              7
4   28            Private  338409   Bachelors             13

        marital-status          occupation    relationship    race      sex  \
0        Never-married        Adm-clerical   Not-in-family   White     Male
1   Married-civ-spouse     Exec-managerial         Husband   White     Male
2             Divorced   Handlers-cleaners   Not-in-family   White     Male
3   Married-civ-spouse   Handlers-cleaners         Husband   Black     Male
4   Married-civ-spouse      Prof-specialty            Wife   Black   Female

   capital-gain  capital-loss  hours-per-week  native-country  salary
0          2174             0              40   United-States   <=50K
1             0             0              13   United-States   <=50K
2             0             0              40   United-States   <=50K
3             0             0              40   United-States   <=50K
4             0             0              40            Cuba   <=50K
**************************************************
                age        fnlwgt  education-num  capital-gain  capital-loss  \
count  32561.000000  3.256100e+04   32561.000000  32561.000000  32561.000000
mean      38.581647  1.897784e+05      10.080679   1077.648844     87.303830
std       13.640433  1.055500e+05       2.572720   7385.292085    402.960219
min       17.000000  1.228500e+04       1.000000      0.000000      0.000000
25%       28.000000  1.178270e+05       9.000000      0.000000      0.000000
50%       37.000000  1.783560e+05      10.000000      0.000000      0.000000
75%       48.000000  2.370510e+05      12.000000      0.000000      0.000000
max       90.000000  1.484705e+06      16.000000  99999.000000   4356.000000

       hours-per-week
count    32561.000000
mean        40.437456
```

```
std          12.347429
min           1.000000
25%          40.000000
50%          40.000000
75%          45.000000
max          99.000000
(32561, 108)
       age  fnlwgt  education-num  capital-gain  capital-loss  hours-per-week  \
0       39   77516             13          2174             0              40
1       50   83311             13             0             0              13
2       38  215646              9             0             0              40
3       53  234721              7             0             0              40
4       28  338409             13             0             0              40
...    ...     ...            ...           ...           ...             ...
32556   27  257302             12             0             0              38
32557   40  154374              9             0             0              40
32558   58  151910              9             0             0              40
32559   22  201490              9             0             0              20
32560   52  287927              9         15024             0              40

       workclass_ ?  workclass_ Federal-gov  workclass_ Local-gov  \
0                 0                       0                     0
1                 0                       0                     0
2                 0                       0                     0
3                 0                       0                     0
4                 0                       0                     0
...             ...                     ...                   ...
32556             0                       0                     0
32557             0                       0                     0
32558             0                       0                     0
32559             0                       0                     0
32560             0                       0                     0

       workclass_ Never-worked  ...  native-country_ Portugal  \
0                            0  ...                         0
1                            0  ...                         0
2                            0  ...                         0
3                            0  ...                         0
4                            0  ...                         0
...                        ...  ...                       ...
32556                        0  ...                         0
32557                        0  ...                         0
32558                        0  ...                         0
32559                        0  ...                         0
32560                        0  ...                         0

       native-country_ Puerto-Rico  native-country_ Scotland  \
0                                0                         0
```

```
1                                0                          0
2                                0                          0
3                                0                          0
4                                0                          0
...                            ...                        ...
32556                            0                          0
32557                            0                          0
32558                            0                          0
32559                            0                          0
32560                            0                          0

       native-country_ South  native-country_ Taiwan  \
0                          0                        0
1                          0                        0
2                          0                        0
3                          0                        0
4                          0                        0
...                      ...                      ...
32556                      0                        0
32557                      0                        0
32558                      0                        0
32559                      0                        0
32560                      0                        0

       native-country_ Thailand  native-country_ Trinadad&Tobago  \
0                             0                                 0
1                             0                                 0
2                             0                                 0
3                             0                                 0
4                             0                                 0
...                         ...                               ...
32556                         0                                 0
32557                         0                                 0
32558                         0                                 0
32559                         0                                 0
32560                         0                                 0

       native-country_ United-States  native-country_ Vietnam  \
0                                  1                         0
1                                  1                         0
2                                  1                         0
3                                  1                         0
4                                  0                         0
...                              ...                       ...
32556                              1                         0
32557                              1                         0
32558                              1                         0
32559                              1                         0
```

```
32560                              1                          0

       native-country_ Yugoslavia
0                                0
1                                0
2                                0
3                                0
4                                0
...                            ...
32556                            0
32557                            0
32558                            0
32559                            0
32560                            0

[32561 rows x 108 columns]
```

8.1 Decision Tree

We will use `sklearn.tree.DecisionTreeClassifier`.

[100]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import numpy as np

X_train, X_test, y_train, y_test = train_test_split(data_X, data_y,
                                                    test_size=0.2,
                                                    random_state=0)

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)

print(classification_report(y_test, y_pred))
print("Depth of tree: ", dtc.get_depth())
```

```
              precision    recall  f1-score   support

           0       0.88      0.87      0.87      4918
           1       0.61      0.62      0.62      1595

    accuracy                           0.81      6513
   macro avg       0.74      0.75      0.74      6513
weighted avg       0.81      0.81      0.81      6513

Depth of tree:  52
```

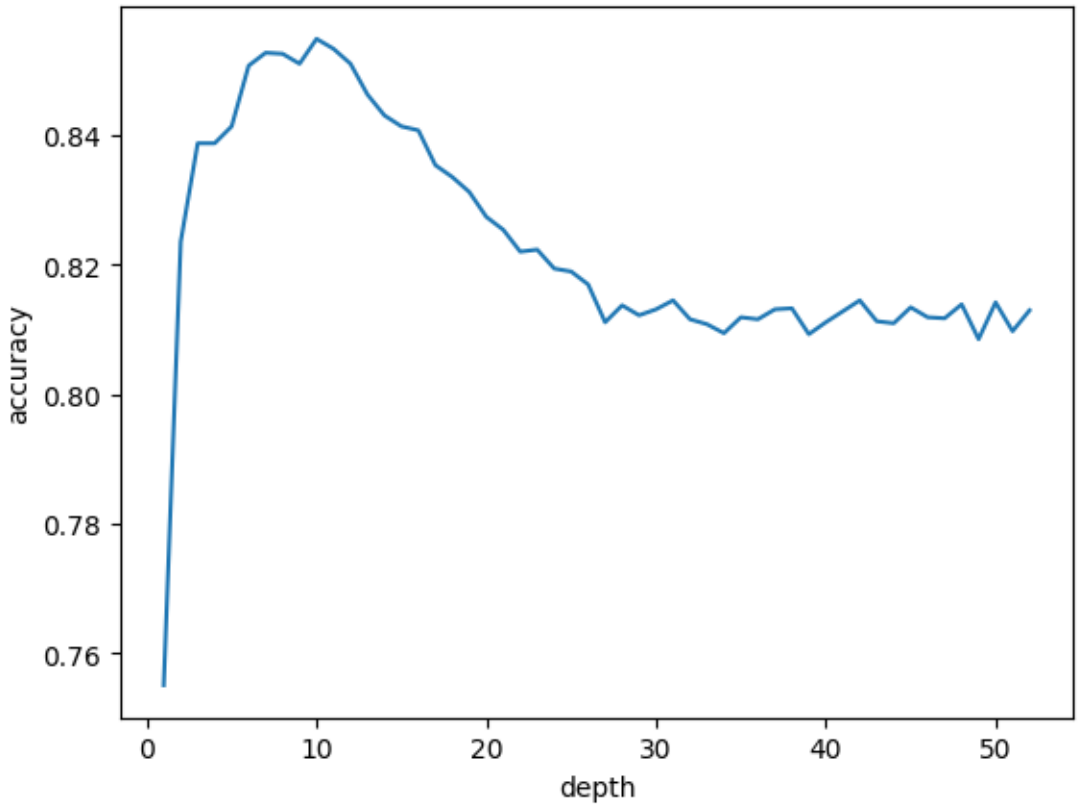
8.1.1 Exercise - Pruning

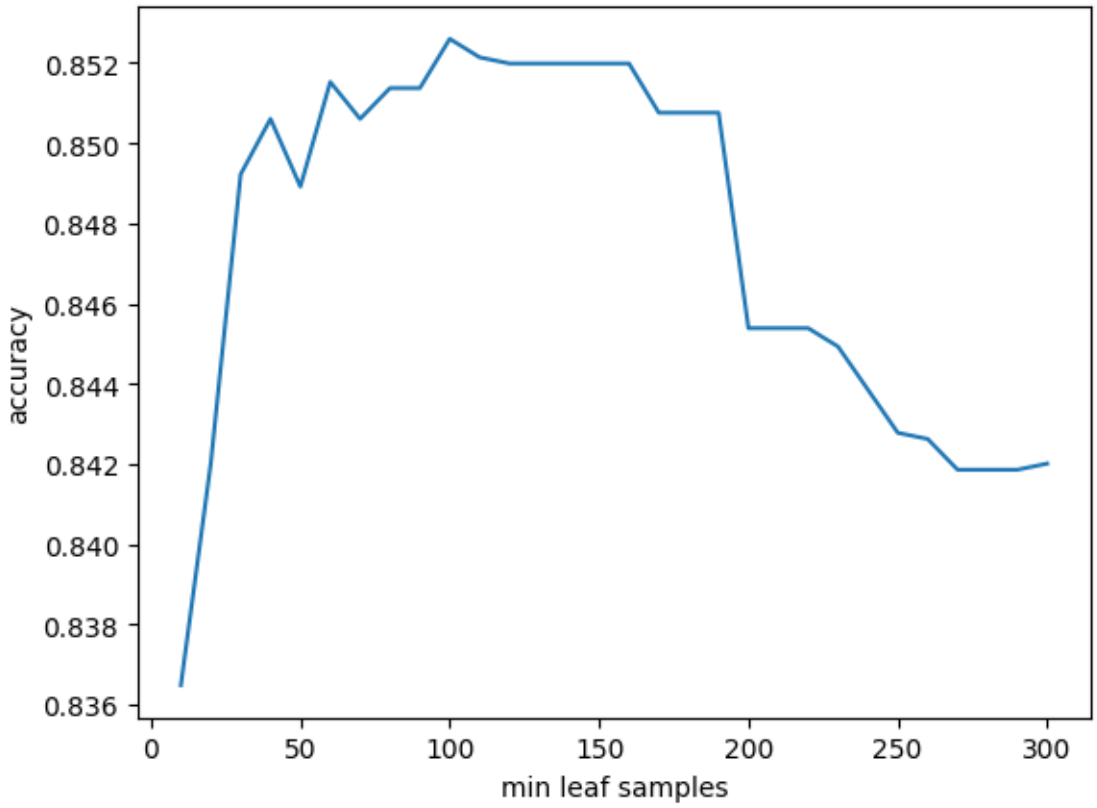
Now you should try two different ways of preventing overfitting. Plot two graphs which show either the maximum depth or the minimum number of samples per leaf on the x-axis and the test accuracy on the y axis.

Solution 8.1.1

Click to Show/Hide Here or Navigate to Solution 8.1.1 on page 162

[ ]:

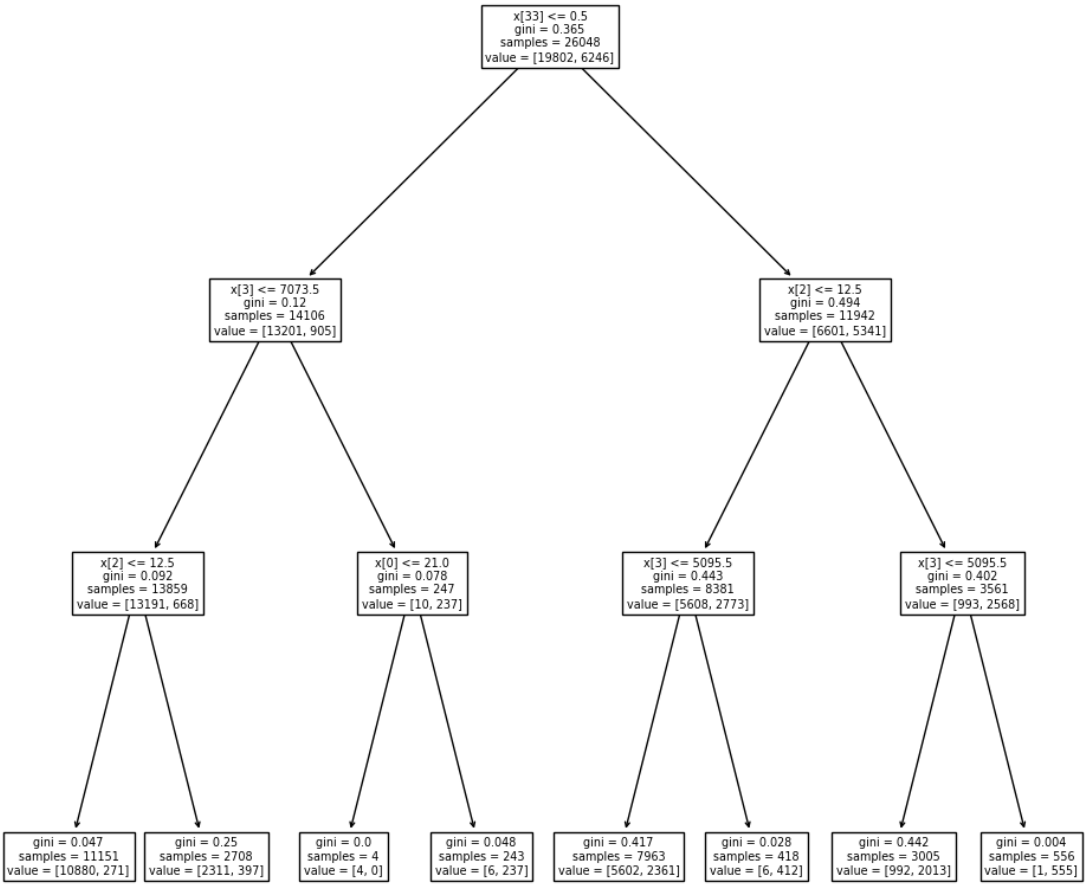

0.84
0.82
0.80
0.78
0.76
accuracy
0
10
20
30
40
50
depth



8.2 Visualizing trees

We can visualize a tree by usgin `sklearn.tree.plot_tree`. In order to do that we should use som smaller depth so the tree is understandable.

[102]:
```
from sklearn.tree import plot_tree

dtc = DecisionTreeClassifier(max_depth=3)
dtc.fit(X_train, y_train)

plt.figure(figsize=(12, 12))
plot_tree(dtc)
plt.show()
```

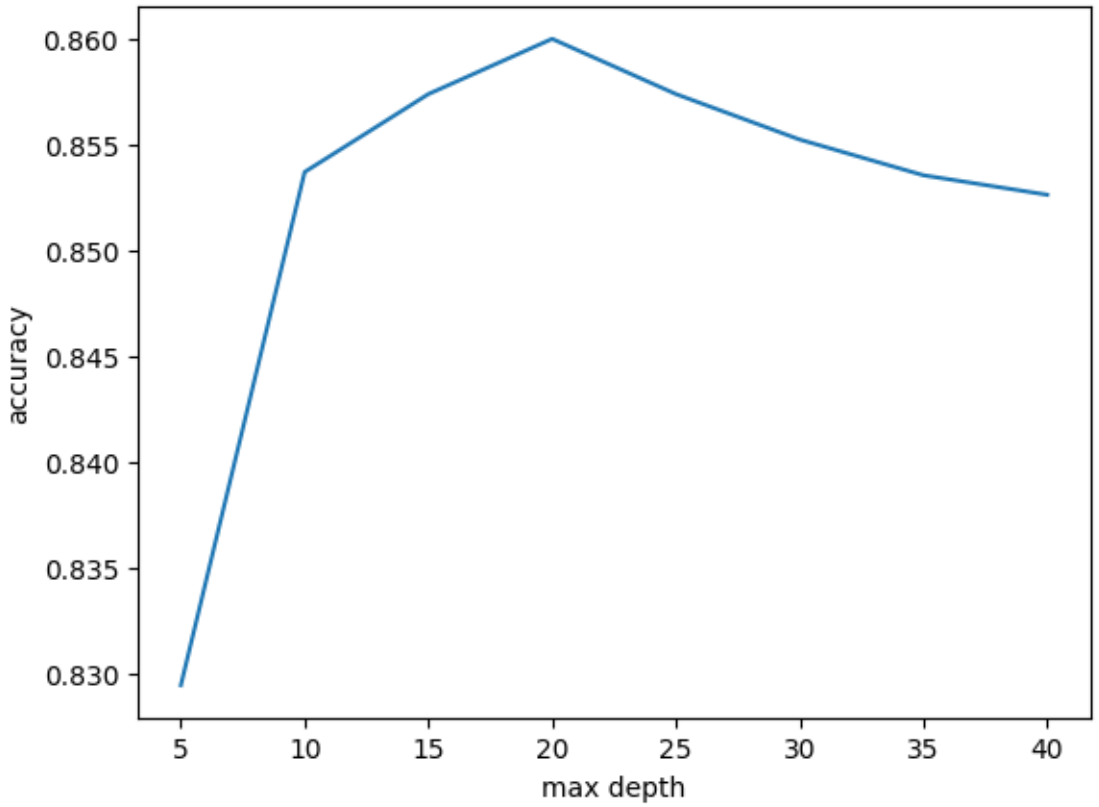


8.3 Random Forests

We can use Random Forests from `sklearn.ensemble.RandomForestClassifier`.

[103]:
```
from sklearn.ensemble import RandomForestClassifier

depths = np.arange(5, 41, 5)
accs = []

for d in depths:
  forest = RandomForestClassifier(max_depth=d)
  forest.fit(X_train, y_train)
  y_pred = forest.predict(X_test)
```

```
  accs.append(np.sum(y_pred == y_test)/len(y_test))

plt.plot(depths, accs)
plt.xlabel('max depth')
plt.ylabel('accuracy')
plt.show()
```



8.4 Boosting

Similarly, we can use AdaBoost from `sklearn.ensemble.AdaBoostClassifier`.

[105]:
```
from sklearn.ensemble import AdaBoostClassifier

adaboost = AdaBoostClassifier()
adaboost.fit(X_train, y_train)

y_pred = dtc.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.95      0.90      4918
           1       0.75      0.51      0.61      1595

    accuracy                           0.84      6513
   macro avg       0.80      0.73      0.75      6513
weighted avg       0.83      0.84      0.83      6513
```

8.4.1 Exercise - Accuracies of individual trees

Now try to print out the accuracies of the individual trees. From a trained Random Forest and separately using AdaBoost. Check out the documentation on how to do this.

Solution 8.4.1

Click to Show/Hide Here or Navigate to Solution 8.4.1 on page 163

[ ]:

```
********************
Random forest
********************
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has
feature names, but RandomForestClassifier was fitted without feature names
  warnings.warn(
              precision    recall  f1-score   support

           0       0.88      0.94      0.91      4918
           1       0.76      0.61      0.68      1595

    accuracy                           0.86      6513
   macro avg       0.82      0.78      0.79      6513
weighted avg       0.85      0.86      0.85      6513

For 0-th forest train acc: 0.8743857493857494, test acc: 0.8271150007676954,
depth: 20
For 1-th forest train acc: 0.8768811425061425, test acc: 0.8254260709350529,
depth: 20
For 2-th forest train acc: 0.8754222972972973, test acc: 0.8188238906801781,
depth: 20
For 3-th forest train acc: 0.8841753685503686, test acc: 0.826040227237832,
depth: 20
For 4-th forest train acc: 0.8687807125307125, test acc: 0.8114540150468295,
depth: 20
For 5-th forest train acc: 0.8774953931203932, test acc: 0.8180561953017043,
depth: 20
... Some output omitted for clarity ...
For 95-th forest train acc: 0.8701243857493858, test acc: 0.8188238906801781,
depth: 20
For 96-th forest train acc: 0.8758829852579852, test acc: 0.8254260709350529,
depth: 20
For 97-th forest train acc: 0.8793765356265356, test acc: 0.8232765238753262,
depth: 20
For 98-th forest train acc: 0.8636363636363636, test acc: 0.8185168125287886,
depth: 20
For 99-th forest train acc: 0.8740786240786241, test acc: 0.8238906801781053,
depth: 20
********************
Random forest
********************
              precision    recall  f1-score   support

           0       0.88      0.93      0.91      4918
           1       0.75      0.61      0.68      1595

    accuracy                           0.86      6513
```

```
   macro avg       0.82      0.77      0.79      6513
weighted avg       0.85      0.86      0.85      6513

For 0-th forest train acc: 0.7602119164619164, test acc: 0.7551051742668509,
depth: 1
For 1-th forest train acc: 0.7530328624078624, test acc: 0.7455857515737755,
depth: 1
For 2-th forest train acc: 0.8040540540540541, test acc: 0.7968678028558268,
depth: 1
For 3-th forest train acc: 0.48790694103194104, test acc: 0.5029940119760479,
depth: 1
For 4-th forest train acc: 0.7040079852579852, test acc: 0.6929218486104713,
depth: 1
... Some output omitted for clarity ...
For 45-th forest train acc: 0.2862023955773956, test acc: 0.289881774911715,
depth: 1
For 46-th forest train acc: 0.5001919533169533, test acc: 0.4951635191156149,
depth: 1
For 47-th forest train acc: 0.2769502457002457, test acc: 0.2802088131429449,
depth: 1
For 48-th forest train acc: 0.2336455773955774, test acc: 0.23952095808383234,
depth: 1
For 49-th forest train acc: 0.28044379606879605, test acc: 0.29064947029018884,
depth: 1
```

8.4.2 Exercise - AdaBoost with a different base estimator

Now you should try to change the properties of the decision tree that will be used in AdaBoost. Try to again plot the relationship between the depth of the Trees and the resulting test accuracy.

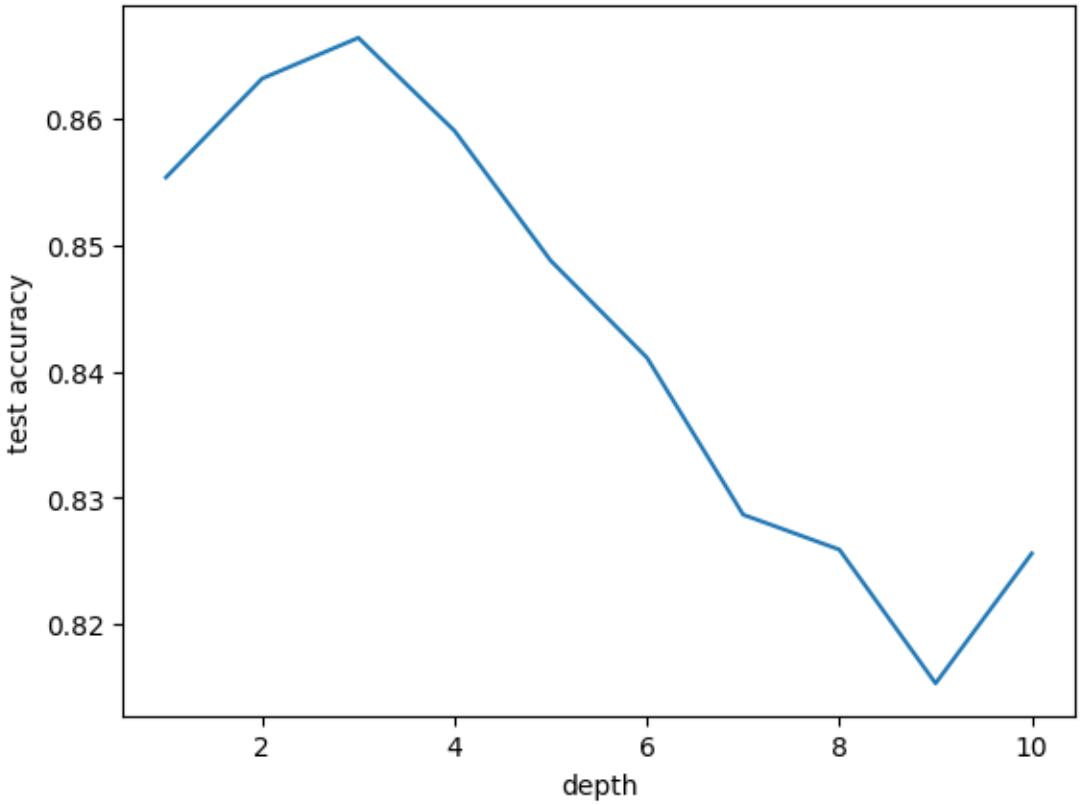
Note: This might take a while!

Solution 8.4.2

Click to Show/Hide Here or Navigate to Solution 8.4.2 on page 164

[ ]:

[107]:
```
Text(0, 0.5, 'test accuracy')
```



8.5 Exercise - Automated Hyperparamter search

Previously we have learned how to perform validation and cross validation. We also tested how changing various hyperparameters led to improved results. This time we will try to use sklearn to automate this process.

To speed things up we will use the Cancer dataset [14].

[108]:
```
from sklearn import datasets

data = datasets.load_breast_cancer()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=456789)
```

Now you should try to use `sklearn.model_selection.RandomizedSearchCV` to find the optimal parameters for a random forest. You should try to vary the maximum depth of the trees, the number of trees, the criterion. YOu calso try other parameters.

In order to see how the search is performed you should initialize the `RandomizedSearchCV` with parameter `verbose=2`. Changing the parameter `n_iter` will change how many iterations we want.

Solution 8.5

Click to Show/Hide Here or Navigate to Solution 8.5 on page 165

[ ]:

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
[CV] END criterion=gini, max_depth=1, min_samples_split=4, n_estimators=20;
total time=   0.1s
[CV] END criterion=gini, max_depth=1, min_samples_split=4, n_estimators=20;
total time=   0.1s

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:305:
UserWarning: The total space of parameters 72 is smaller than n_iter=100.
Running 72 iterations. For exhaustive searches, use GridSearchCV.
  warnings.warn(

[CV] END criterion=gini, max_depth=1, min_samples_split=4, n_estimators=20;
total time=   0.1s
[CV] END criterion=gini, max_depth=1, min_samples_split=4, n_estimators=20;
total time=   0.1s
[CV] END criterion=gini, max_depth=1, min_samples_split=4, n_estimators=20;
total time=   0.1s
[CV] END criterion=gini, max_depth=1, min_samples_split=4, n_estimators=50;
total time=   0.1s
[CV] END criterion=gini, max_depth=1, min_samples_split=4, n_estimators=50;
total time=   0.2s
... Some output omitted for clarity ...
[CV] END criterion=entropy, max_depth=16, min_samples_split=8, n_estimators=100;
total time=   0.2s
[CV] END criterion=entropy, max_depth=16, min_samples_split=8, n_estimators=100;
```

```
total time=   0.2s
[CV] END criterion=entropy, max_depth=16, min_samples_split=8, n_estimators=100;
total time=   0.2s
[CV] END criterion=entropy, max_depth=16, min_samples_split=8, n_estimators=100;
total time=   0.2s
[CV] END criterion=entropy, max_depth=16, min_samples_split=8, n_estimators=100;
total time=   0.2s
Best params:  {'n_estimators': 100, 'min_samples_split': 4, 'max_depth': 11,
'criterion': 'entropy'}
              precision    recall  f1-score   support

           0       0.95      0.93      0.94        45
           1       0.96      0.97      0.96        69

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

Chapter 9

Initial Data Analysis, Visualization and Clustering

In this chapter we will cover some basics of initial data analysis and visualization. Since both simple initial data analysis and data visualization usually mostly depends on calling the correct functions on the data there will be no exercises only a showcasing of existing functions.

Then we will move onto clustering where we will implement our own k-means clustering method.

9.1 Initial Data Analysis

We will load the Census dataset [2] we have already used before and do some basic data analysis.

[112]:
```
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data
```

```
--2024-01-11 14:59:17--  https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: ‘adult.data.3’

adult.data.3            [   <=>              ]   3.79M  7.25MB/s    in 0.5s

2024-01-11 14:59:18 (7.25 MB/s) - ‘adult.data.3’ saved [3974305]
```

[113]:
```
import pandas as pd
data = pd.read_csv('adult.data', names=['age', 'workclass', 'fnlwgt',
                                        'education', 'education-num',
                                        'marital-status', 'occupation',
                                        'relationship', 'race', 'sex',
```

```
                                        'capital-gain', 'capital-loss',
                                        'hours-per-week', 'native-country',
                                        'salary'], sep=",")
print(data.head())
```

```
   age          workclass  fnlwgt   education  education-num  \
0   39          State-gov   77516   Bachelors             13
1   50   Self-emp-not-inc   83311   Bachelors             13
2   38            Private  215646     HS-grad              9
3   53            Private  234721        11th              7
4   28            Private  338409   Bachelors             13

        marital-status          occupation    relationship    race      sex  \
0        Never-married        Adm-clerical   Not-in-family   White     Male
1   Married-civ-spouse     Exec-managerial         Husband   White     Male
2             Divorced   Handlers-cleaners   Not-in-family   White     Male
3   Married-civ-spouse   Handlers-cleaners         Husband   Black     Male
4   Married-civ-spouse      Prof-specialty            Wife   Black   Female

   capital-gain  capital-loss  hours-per-week  native-country  salary
0          2174             0              40   United-States   <=50K
1             0             0              13   United-States   <=50K
2             0             0              40   United-States   <=50K
3             0             0              40   United-States   <=50K
4             0             0              40            Cuba   <=50K
```

We can learn most important information by calling the `describe` method on the dataframe. We can also use the `info` method for more information. The `info` method is important as it shows whether the data contains empty fields. It also shows the type of data.

[114]:
```
print(data.describe(include='all'))
print(30 * '*')
print(data.info())
```

```
                age workclass        fnlwgt education  education-num  \
count   32561.000000     32561  3.256100e+04     32561   32561.000000
unique           NaN         9           NaN        16            NaN
top              NaN   Private           NaN   HS-grad            NaN
freq             NaN     22696           NaN     10501            NaN
mean       38.581647       NaN  1.897784e+05       NaN      10.080679
std        13.640433       NaN  1.055500e+05       NaN       2.572720
min        17.000000       NaN  1.228500e+04       NaN       1.000000
25%        28.000000       NaN  1.178270e+05       NaN       9.000000
50%        37.000000       NaN  1.783560e+05       NaN      10.000000
75%        48.000000       NaN  2.370510e+05       NaN      12.000000
max        90.000000       NaN  1.484705e+06       NaN      16.000000

            marital-status       occupation relationship    race    sex  \
count                32561            32561        32561   32561  32561
```

```
unique                   7               15            6       5      2
top      Married-civ-spouse   Prof-specialty      Husband   White   Male
freq                 14976             4140        13193   27816  21790
mean                   NaN              NaN          NaN     NaN    NaN
std                    NaN              NaN          NaN     NaN    NaN
min                    NaN              NaN          NaN     NaN    NaN
25%                    NaN              NaN          NaN     NaN    NaN
50%                    NaN              NaN          NaN     NaN    NaN
75%                    NaN              NaN          NaN     NaN    NaN
max                    NaN              NaN          NaN     NaN    NaN

        capital-gain  capital-loss  hours-per-week  native-country  salary
count   32561.000000  32561.000000    32561.000000           32561   32561
unique           NaN           NaN             NaN              42       2
top              NaN           NaN             NaN   United-States   <=50K
freq             NaN           NaN             NaN           29170   24720
mean     1077.648844     87.303830       40.437456             NaN     NaN
std      7385.292085    402.960219       12.347429             NaN     NaN
min         0.000000      0.000000        1.000000             NaN     NaN
25%         0.000000      0.000000       40.000000             NaN     NaN
50%         0.000000      0.000000       40.000000             NaN     NaN
75%         0.000000      0.000000       45.000000             NaN     NaN
max     99999.000000   4356.000000       99.000000             NaN     NaN
*****************************
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64
 3   education       32561 non-null  object
 4   education-num   32561 non-null  int64
 5   marital-status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital-gain    32561 non-null  int64
 11  capital-loss    32561 non-null  int64
 12  hours-per-week  32561 non-null  int64
 13  native-country  32561 non-null  object
 14  salary          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
None
```

We can also check for empty values directly. Empty values are `None` or various types of NaN values for floats.

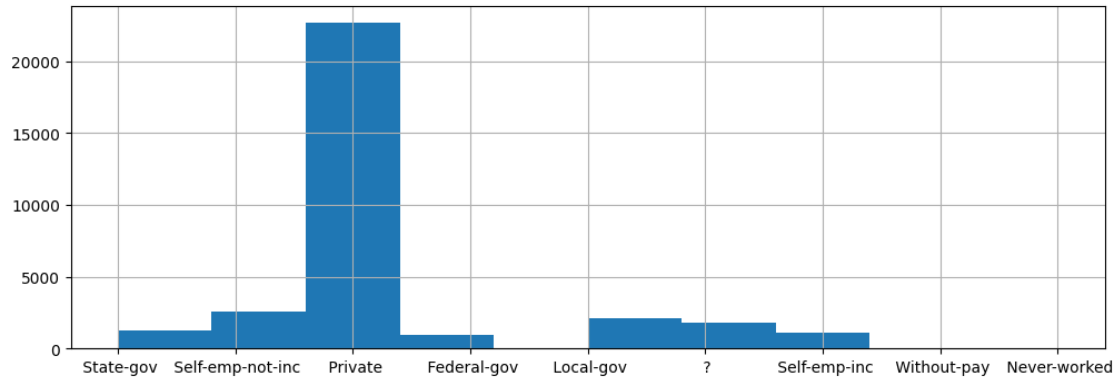
[115]:
```
print(data.isnull().sum())
```

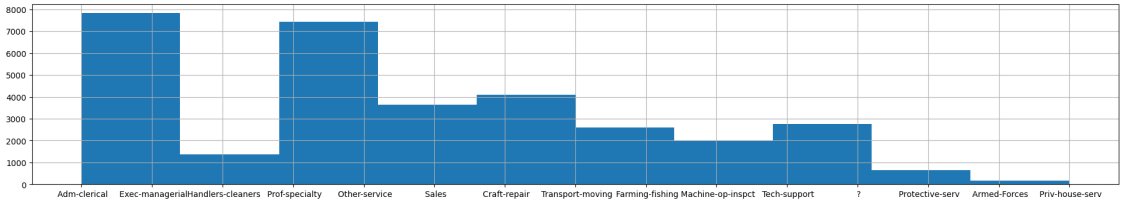
```
age               0
workclass         0
fnlwgt            0
education         0
education-num     0
marital-status    0
occupation        0
relationship      0
race              0
sex               0
capital-gain      0
capital-loss      0
hours-per-week    0
native-country    0
salary            0
dtype: int64
```

9.2 Visualization

We can also perform some basic visualization. For instance we may be interested in histograms for the various features. Pandas directly enables us to draw a histogram using the `DataFrame` `plot.hist` method.

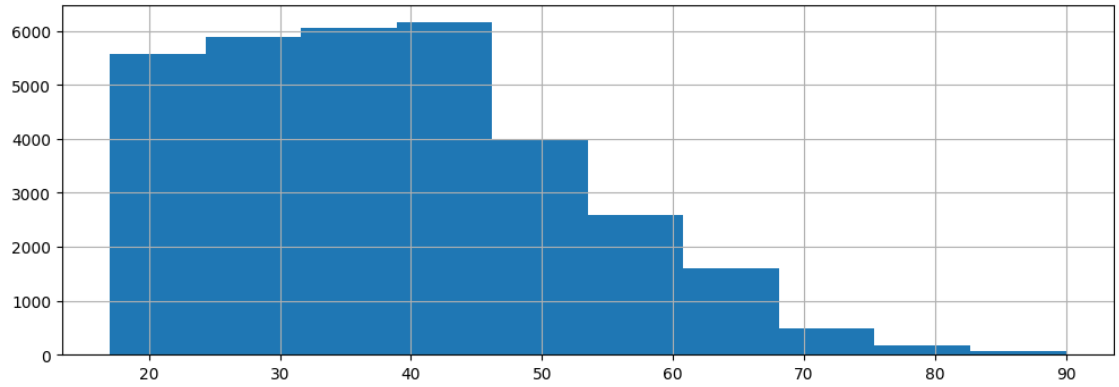
[116]:
```
from matplotlib import pyplot as plt

plt.figure(figsize=(12, 4))
data.workclass.hist()
plt.show()

plt.figure(figsize=(24, 4))
data.occupation.hist()
plt.show()
```

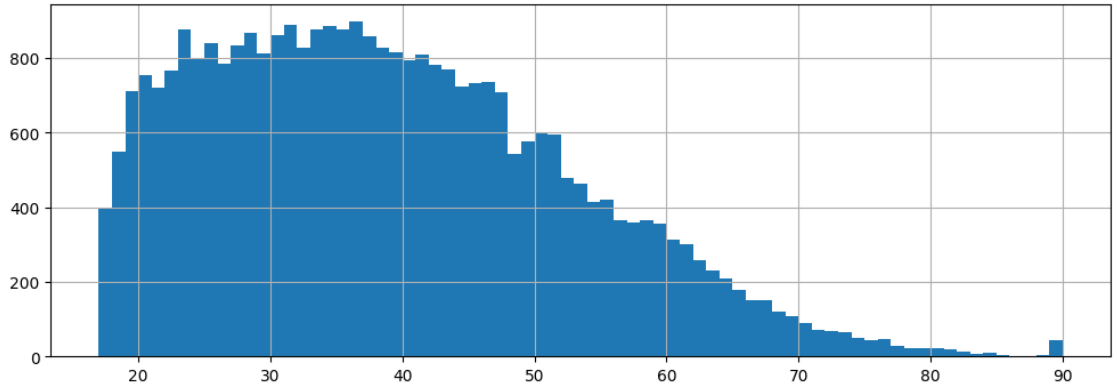




We can also show histograms of numerical values, but we may need to define how many bins we are interested in.

[117]:
```
plt.figure(figsize=(12, 4))
data.age.hist()
plt.show()


plt.figure(figsize=(12, 4))
data.age.hist(bins=data.age.max() - data.age.min())
plt.show()
```





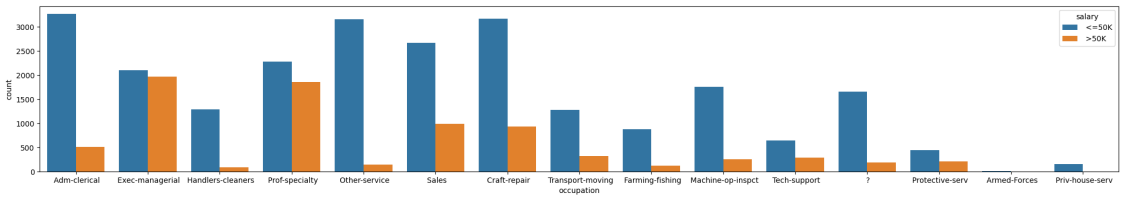
9.3 Nicer plots with Seaborn

For nicer visualizations we can use the Seaborn package. We will not cover all of its possibilities here as any exercises would just require you to call one of the package functions properly. If you are interested in more you can check out its many tutorials. For example this one on visualizing categorical data.

With seaborn we can for example visualise more complex data relationships. We can for instance visualize the ages together with sex and occupation of the people represented in the data. For this we can use the countplot from the Seaborn library.

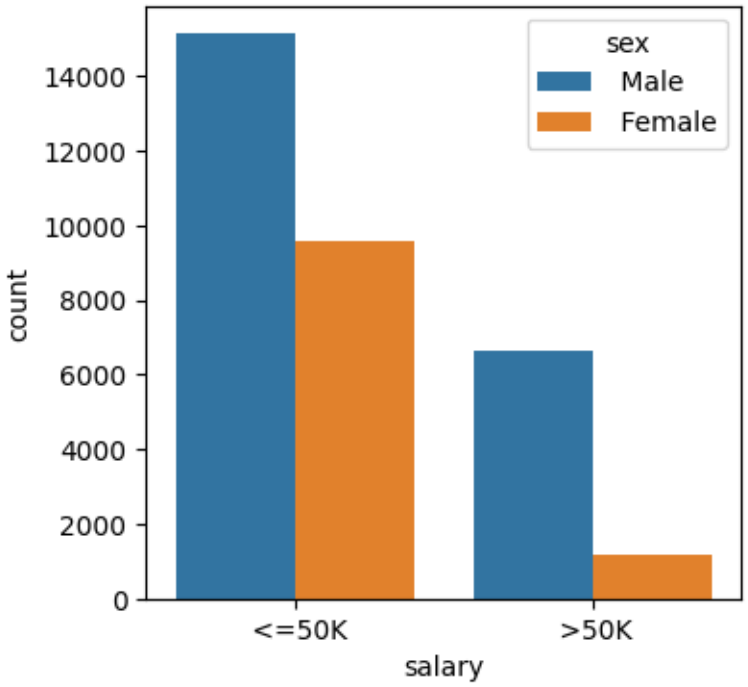
[121]:
```
import seaborn as sns
plt.figure(figsize=(26, 4))
sns.countplot(data, x="occupation", hue="salary")
```

[121]:
```
<Axes: xlabel='occupation', ylabel='count'>
```



[122]:
```
plt.figure(figsize=(4, 4))
sns.countplot(data, x="salary", hue="sex")
```

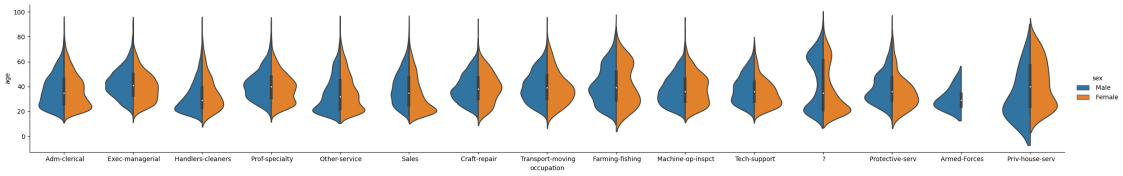
[122]:
```
<Axes: xlabel='salary', ylabel='count'>
```



The catplot method allows us to visualize 3 different data features at once! See some examples.

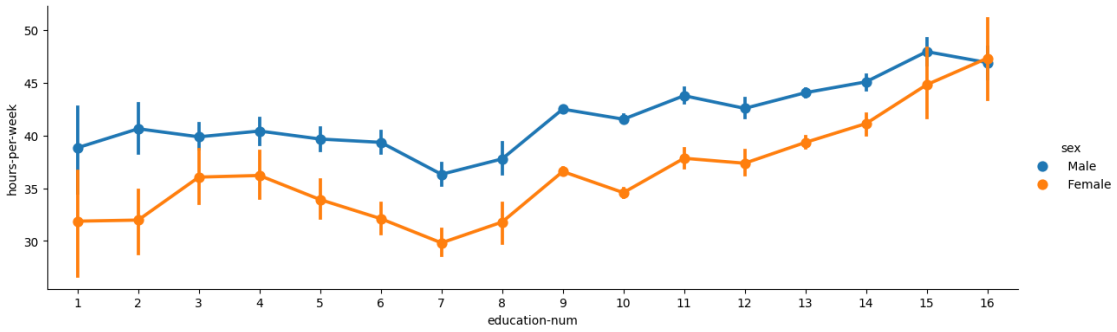
[123]:
```
sns.catplot(x="occupation", y="age", hue="sex", kind="violin", split=True,␣
↪data=data, height=4, aspect=24/4)
```

[123]: <seaborn.axisgrid.FacetGrid at 0x7943bcb43040>



[124]:
```
sns.catplot(x="education-num", y="hours-per-week", hue="sex", kind="point",␣
↪data=data, height=4, aspect=12/4)
```

[124]: <seaborn.axisgrid.FacetGrid at 0x7943bcc7ebc0>

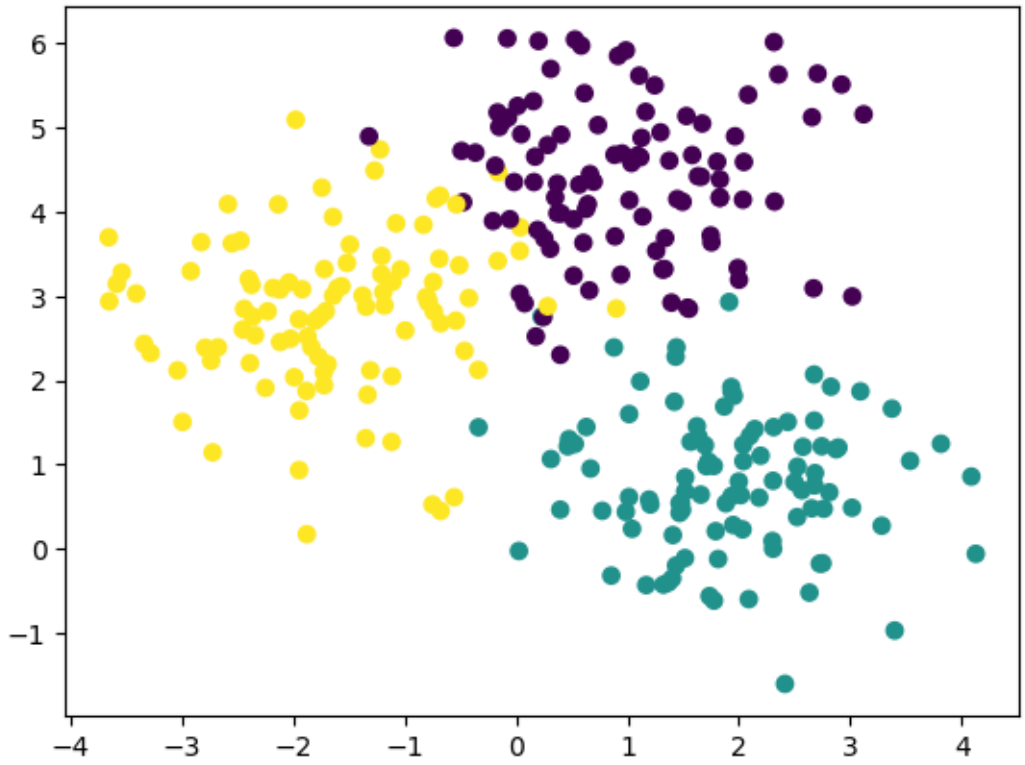


9.4 Clustering

In this subsection we will not cover standard sklearn clustering methods as you can easily find them in the documentation. Instead we will implement our own k-Means clustering algorithm.

First we will need to generate some random data for clustering.

[125]:
```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=300, centers=3, n_features=2, cluster_std=0.9,
                  random_state=0)

plt.scatter(X[:, 0], X[:, 1], c=y)
plt.show()
```



9.4.1 Exercise - k-means clustering algorithm

Implement your own version of k-means clustering algorithm.

The clustering algorithm works by iteratively moving k centers in two steps:

1. Every data point is assigned to the center closest to it.
2. The centers are moved to a position defined as the center of all of the data points assigned to it in the previous step.

These two steps repeat until the centers do not change between iterations. The algorithm is initialized with the k centers determined randomly. It is also possible that in step 1 there will be a center which will not be assigned any data points. In that case the new center position will be determined randomly (same as during initialization).

Notes:

- We will initialize the random centers using the uniform distribution (e.g. `np.random.rand()`) spanning from the lowest to the highest value for each feature. You can also try a different initialization strategy such as selecting a random data point.
- We will also set a `max_iter` threshold as a keyword argument which will stop the algorithm after cetrain number of iterations. This is especially important during debugging as you may accidentaly create an endless loop.

- You will need to calculate distances from all points to all centers. You can implement this on your own either using for loops or broadcasting. However the best solution is to use `scipy.spatial.distance_matrix` which generates a matrix containing the distances between all points and centers. It is possible to do this in a more effective way, but that is out of the scope of this lab.
- There are multiple parts of the code where you could use a for cycle. Using numpy style code is better, but feel free to start with the for cycles and once your code works you can work on removing them.

The signature of the function will be `my_k_means(k, X, max_iter=100)` with `k` being the number of clusters, X being the data points in the shape `(num_points, num_features)`. The function will return two outputs. The first output will be an array of indices assigning each data point to a cluster via an index (e.g. `[1, 0, 2, 1, 1, 0 ...]`) and the second output will return an array of shape `(k, num_features)` cotaining the centers of clusters on each row.

Solution 9.4.1

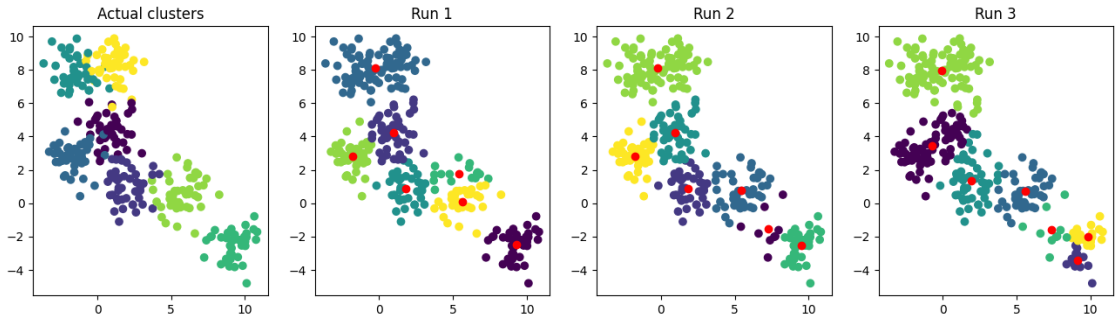
Click to Show/Hide Here or Navigate to Solution 9.4.1 on page 166

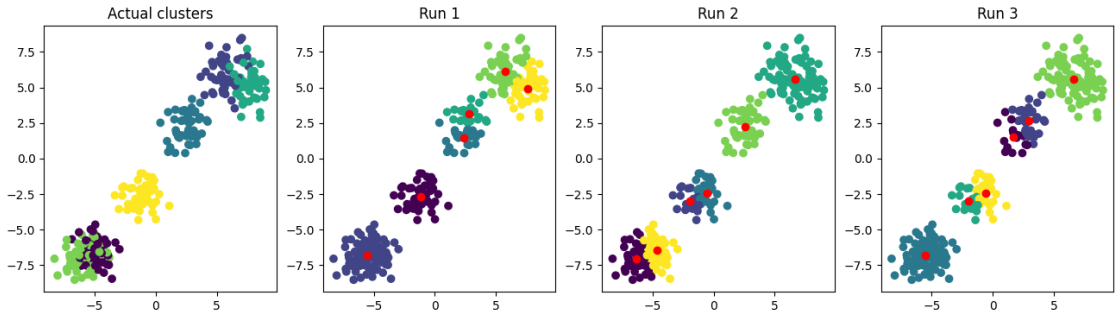
[ ]:
```
```

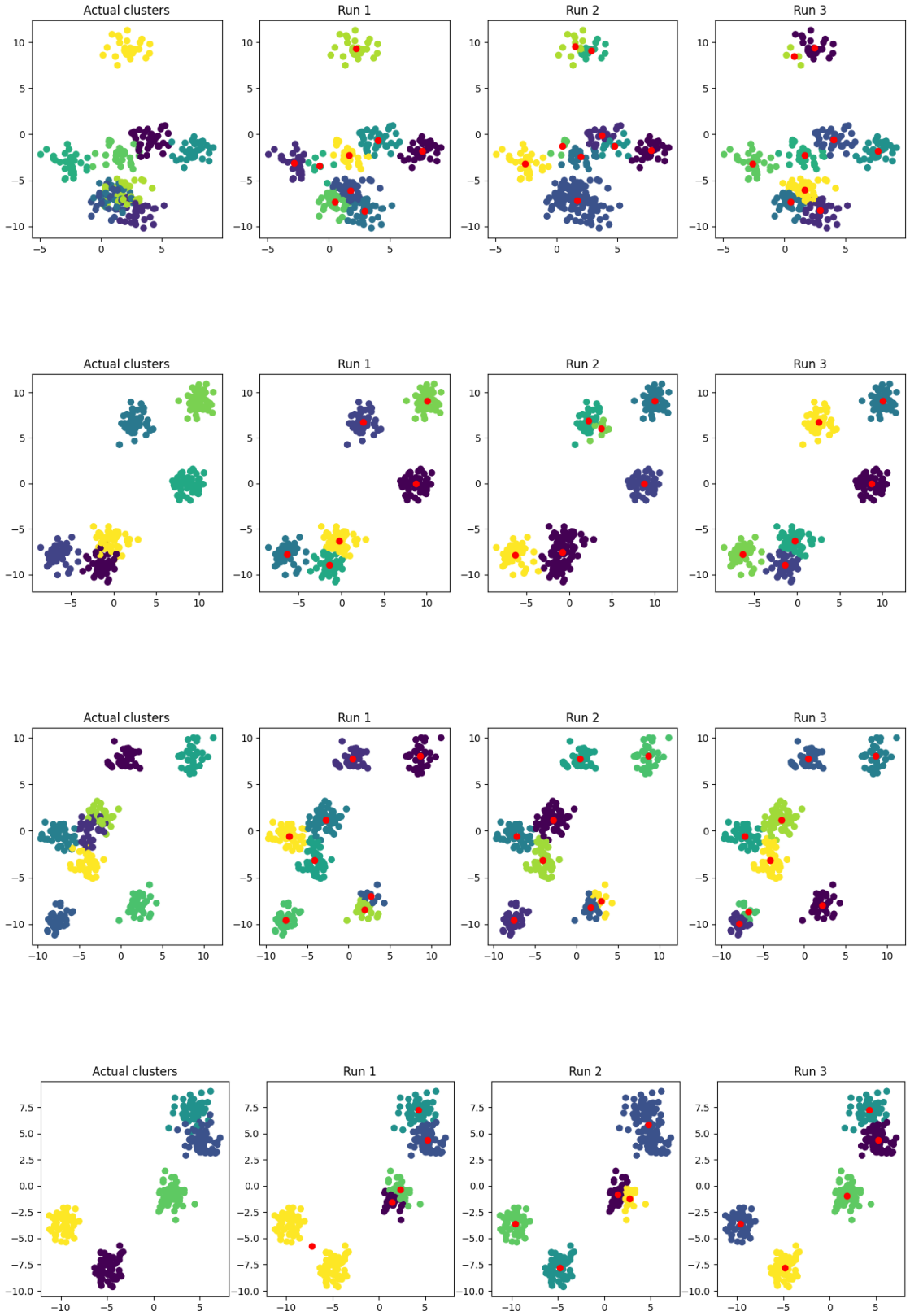
The following code will test your implementation on various randomly generated clusters.

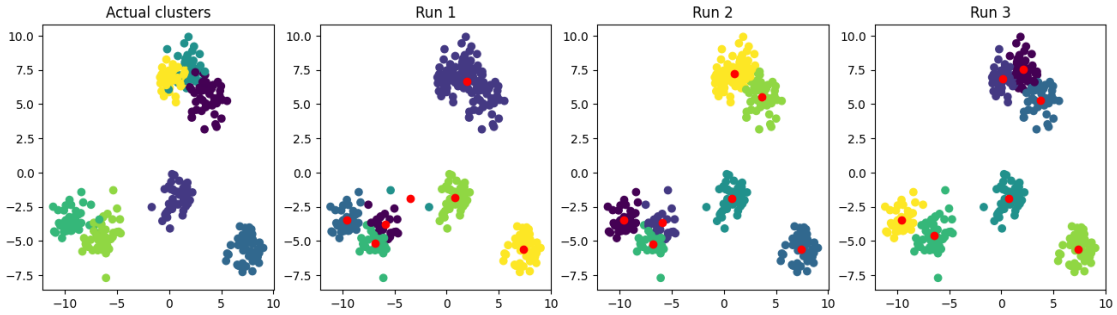
[127]:
```
for r in range(10):
  plt.figure(figsize=(16,4))
```

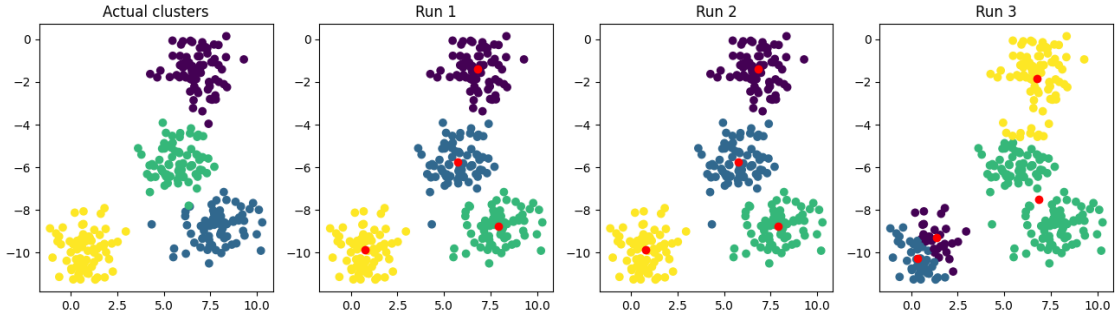
```
k = np.random.randint(2, 10)

X, y = make_blobs(n_samples=300, centers=k, n_features=2, cluster_std=0.9,␣
↪random_state=456*r)
plt.subplot(1, 4, 1)
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.title('Actual clusters')

for i in range(3):
  assignments, m = my_k_means(k, X)
  plt.subplot(1, 4, i+2)
  plt.scatter(X[:, 0], X[:, 1], c=assignments)
  plt.scatter(m[:, 0], m[:, 1], c='r')
  plt.title('Run {}'.format(i + 1))
plt.show()
```

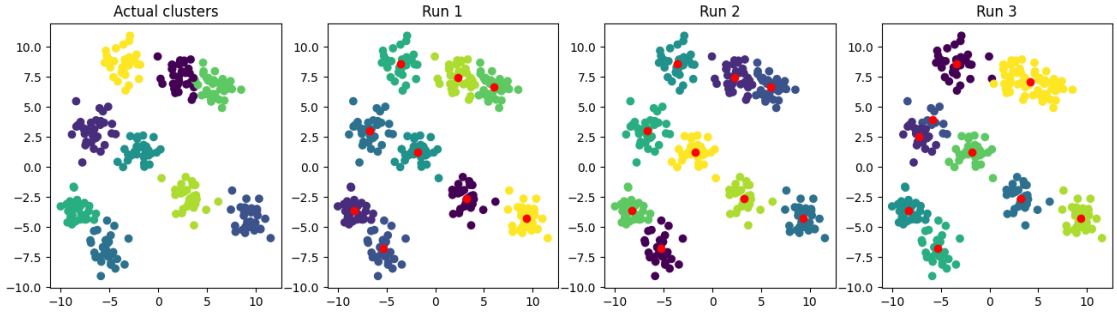


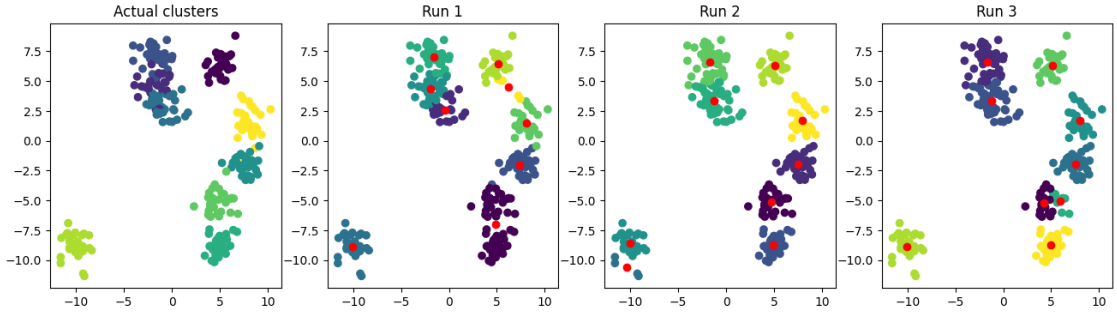



Actual clusters
Run 1
Run 2
Run 3
Actual clusters
Run 1
Run 2
Run 3
Actual clusters
Run 1
Run 2
Run 3
Actual clusters
Run 1
Run 2
Run 3


Actual clusters
Run 1
Run 2
Run 3


Actual clusters
Run 1
Run 2
Run 3


Actual clusters
Run 1
Run 2
Run 3


Actual clusters
Run 1
Run 2
Run 3

9.4.2 Exercise - Displaying the algorithm

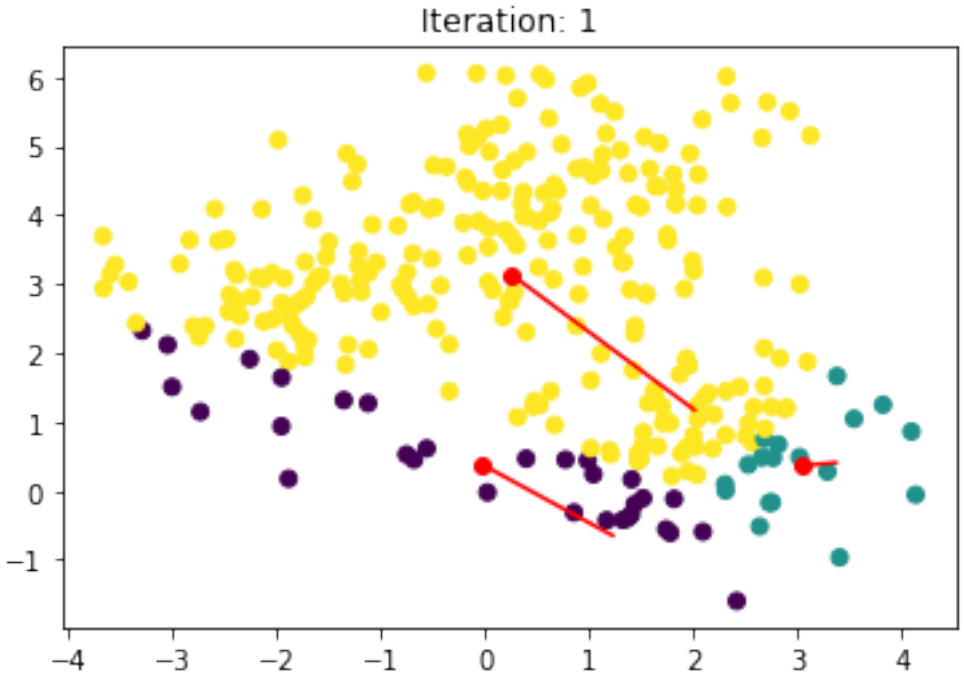
You can try to modify the function to diplay how the process works.

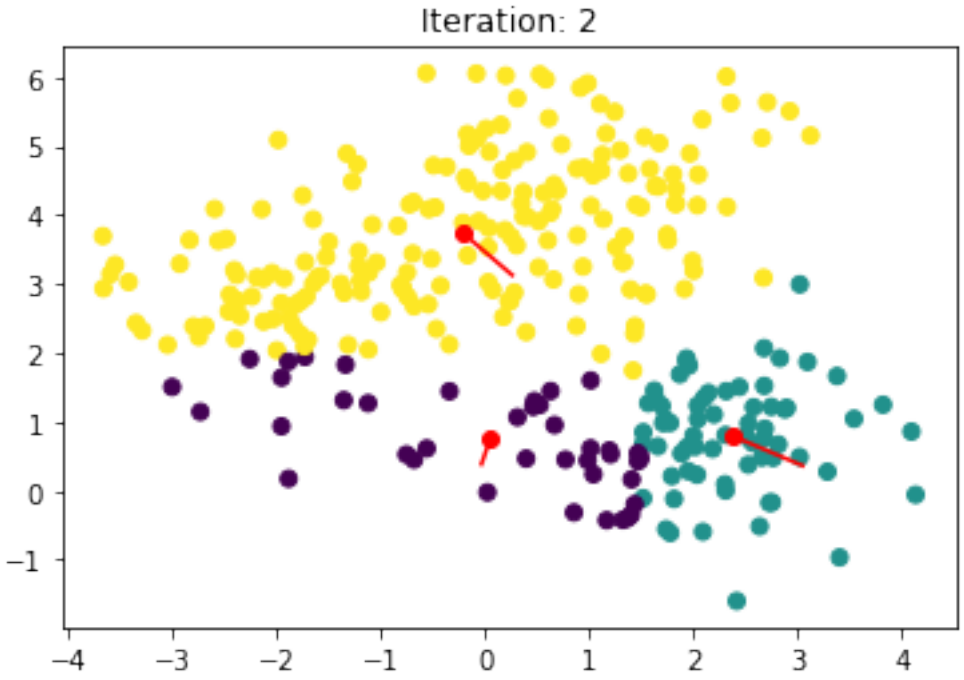
Solution 9.4.2

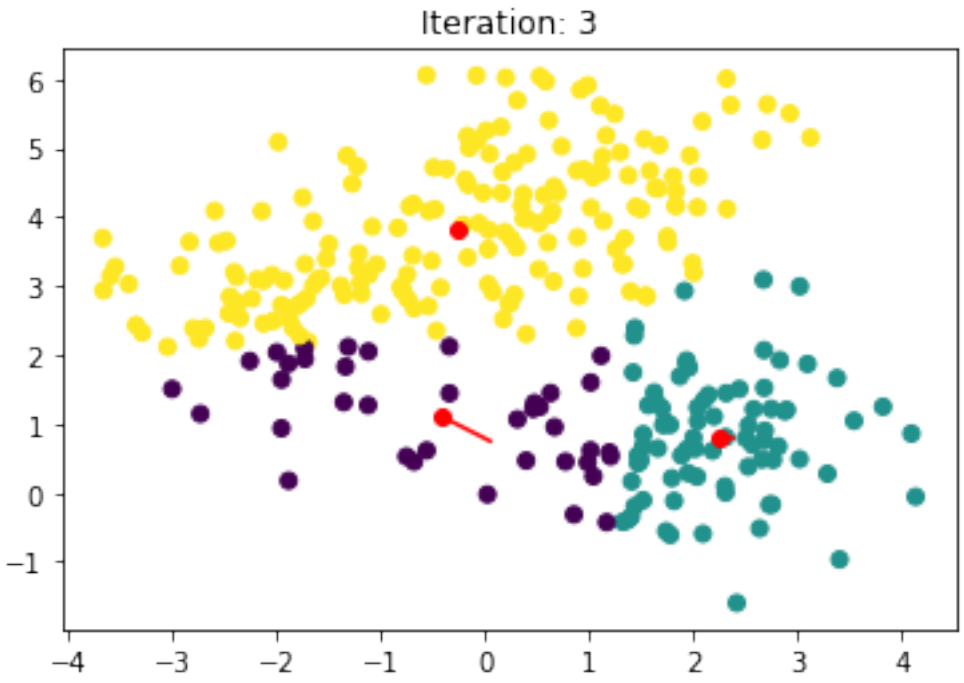
Click to Show/Hide Here or Navigate to Solution 9.4.2 on page 167

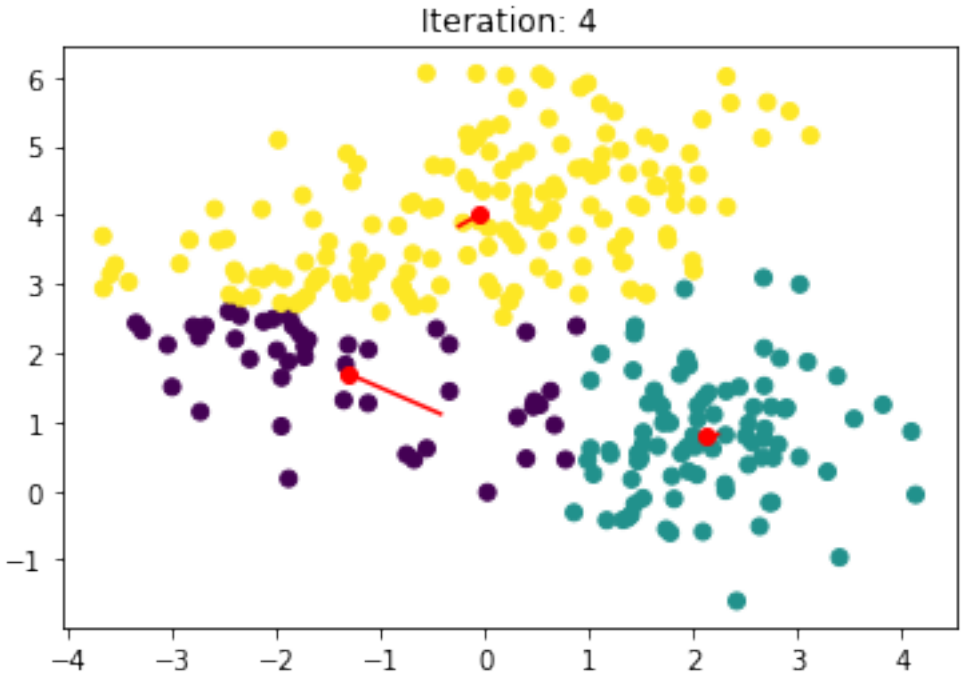
[ ]:

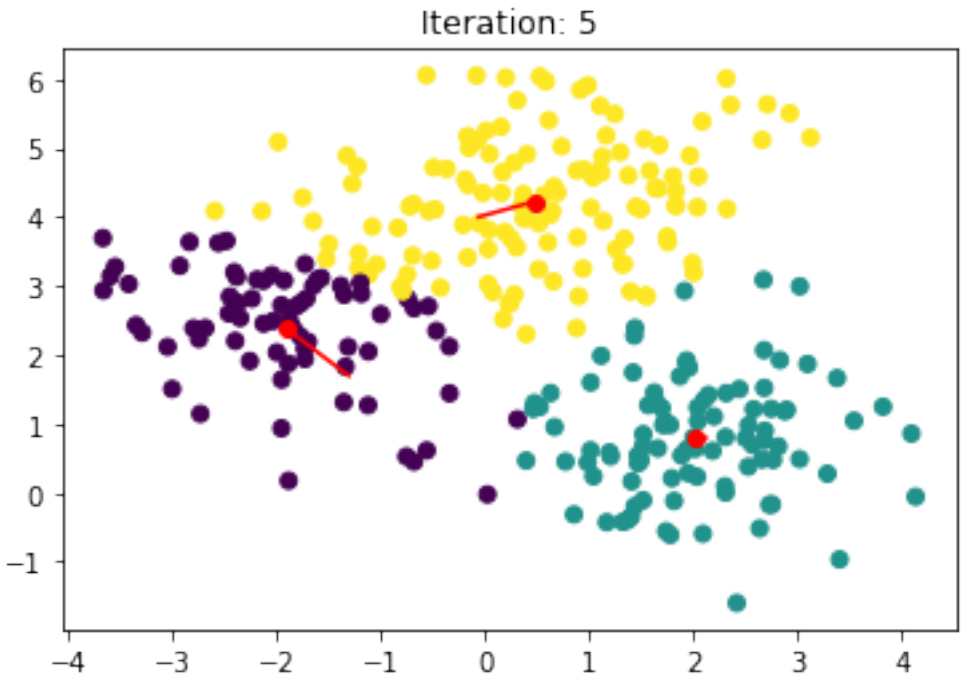
[ ]:
```
X, y = make_blobs(n_samples=300, centers=3, n_features=2, cluster_std=0.9,␣
↪random_state=0)

_, _ = my_k_means_vis(3, X, max_iter=100)
```

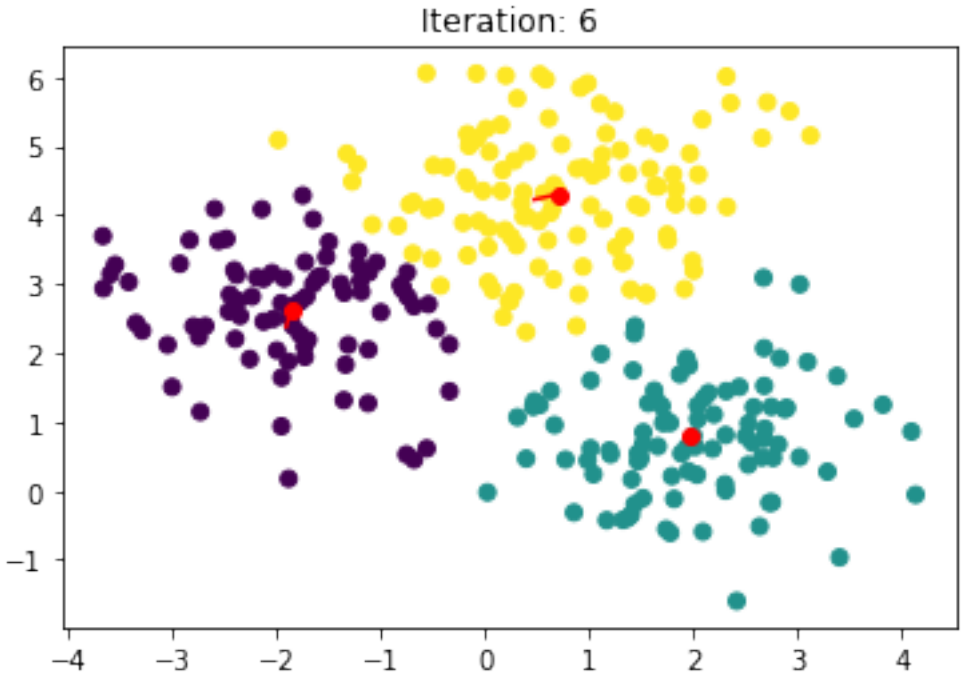

Iteration: 1
6
5
4
3
2
1
0
−1
−4
−3
−2
−1
0
1
2
3
4

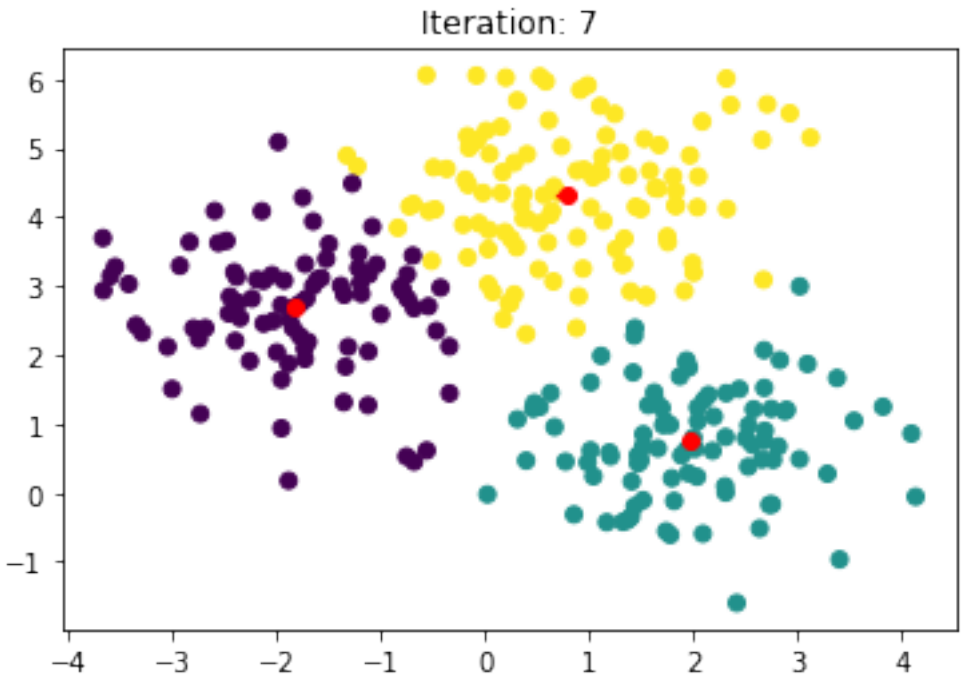

Iteration: 2
6
5
4
3
2
1
0
−1
−4
−3
−2
−1
0
1
2
3
4

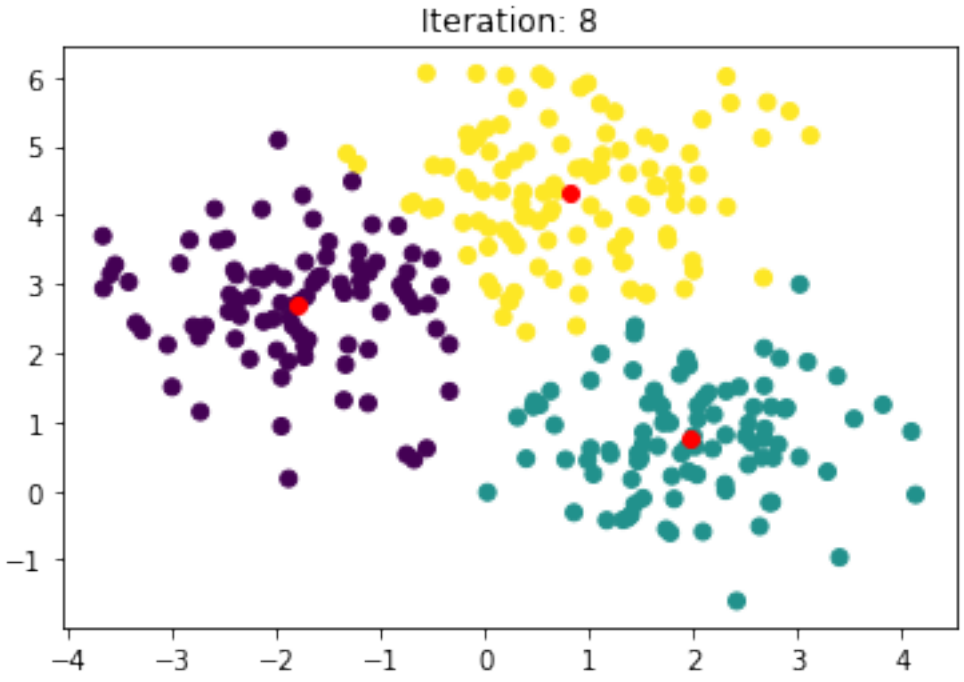

Iteration: 3
6
5
4
3
2
1
0
−1
−4
−3
−2
−1
0
1
2
3
4

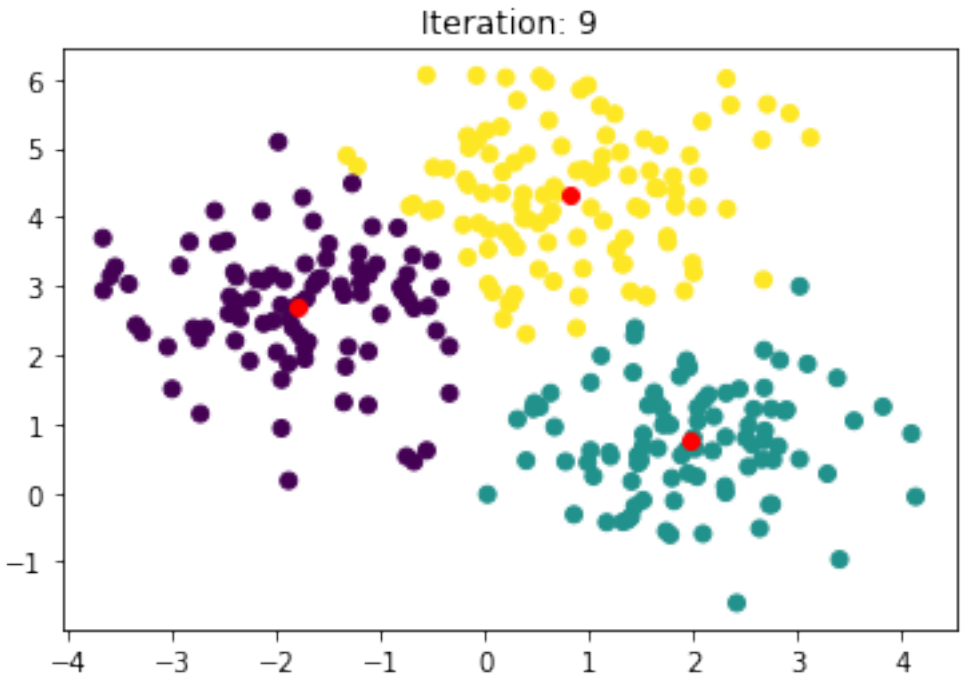

Iteration: 4
6
5
4
3
2
1
0
−1
−4
−3
−2
−1
0
1
2
3
4


Iteration: 5
6
5
4
3
2
1
0
−1
−4
−3
−2
−1
0
1
2
3
4


Iteration: 6
6
5
4
3
2
1
0
−1
−4
−3
−2
−1
0
1
2
3
4


Iteration: 7
6
5
4
3
2
1
0
−1
−4
−3
−2
−1
0
1
2
3
4


Iteration: 8
6
5
4
3
2
1
0
−1
−4
−3
−2
−1
0
1
2
3
4


Iteration: 9
6
5
4
3
2
1
0
−1
−4
−3
−2
−1
0
1
2
3
4

Solutions

Solution 1.1.1

Navigate to Exercise 1.1.1 on page 14

[ ]:
```
def chessboard(rows, cols):
  c = np.zeros([rows, cols])
  c[::2, 1::2] = 1
  c[1::2, 0::2] = 1
  return c
```

Solution 1.2.2

Navigate to Exercise 1.2.2 on page 16

[ ]:
```
def euclidian_distance_single(x, y):
  return np.sqrt(np.sum((x - y) ** 2))
```

Solution 3.0.1

Navigate to Exercise 3.0.1 on page 28

[ ]:
```
def entropy(s):
  v = s.value_counts(normalize=True).to_numpy()
  entropy = np.sum(-v * np.log2(v))
  return entropy
```

Solution 3.0.2

Navigate to Exercise 3.0.2 on page 29

[ ]:
```
def conditional_entropy(y, x):
  ce = 0
  x = x.astype('category')
  cats = x.cat.categories
  p = x.value_counts(normalize=True)

  for cat in cats:
    ce += p[cat] * entropy(y[x == cat])
  return ce
```

Solution 4.1.1

Navigate to Exercise 4.1.1 on page 32

[ ]:
```
X = data - np.mean(data, axis=0)
cov = np.cov(X.T)
v, w = np.linalg.eig(cov)

XX = X @ w

plt.scatter(XX[:, 0], XX[:, 1])
plt.xlim([-2, 2])
plt.ylim([-2, 2])
plt.show()

print("Portion of variance explained: ", v[0]/np.sum(v))
```

Solution 4.1.3

Navigate to Exercise 4.1.3 on page 35

[ ]:
```
pca = PCA(0.95)
pca.fit(cancer_data)
new_data = pca.transform(cancer_data)
plt.title("95 percent of variance explained 5th vs 6th feature")
plt.scatter(new_data[:, 5], new_data[:, 6])
plt.show()
print("We got {} features".format(new_data.shape[1]))
print("Percent of variance explained for each feature: ",
      pca.explained_variance_ratio_)

pca = PCA(2)
pca.fit(cancer_data)
new_data = pca.transform(cancer_data)
plt.title("Best 2 feautres only")
plt.scatter(new_data[:, 0], new_data[:, 1])
plt.show()

print("Percent of variance explained for each feature: ",
      pca.explained_variance_ratio_)
```

Solution 4.1.4

Navigate to Exercise 4.1.4 on page 38

[ ]:
```
# Note that the principal components should be mutually OG. Even if the are OG
# they may not be displayed as such due to aspect ratio. To enforce the ratio so
# that the vectors also look OG you need the following code which enforces the
# aspect ratio:

plt.figure(figsize=(12,6))
plt.xlim([2,8])
plt.ylim([-2,1])

pca = PCA()
pca.fit(data)
plt.scatter(data[:, 0], data[:, 1])
plt.plot([pca.mean_[0], pca.mean_[0] + pca.components_[0, 0]],
 [pca.mean_[1], pca.mean_[1] + pca.components_[0, 1]], color='red')
plt.plot([pca.mean_[0], pca.mean_[0] + pca.components_[1, 0]],
 [pca.mean_[1], pca.mean_[1] + pca.components_[1, 1]], color='green')
plt.title("PCA")
plt.show()
```

Solution 4.1.5

Navigate to Exercise 4.1.5 on page 42

[ ]:
```
def reduce_color(img):
  img_r = np.array(img).reshape(img.shape[0] * img.shape[1], 3)
  pca = PCA(2)
  pca.fit(img_r)
  img_rt = pca.transform(img_r)
  img_ro = pca.inverse_transform(img_rt)
  img_r = img_ro.reshape(*img.shape)
  plt.imshow(img)
  plt.axis("off")
  plt.title("Original image")
  plt.show()

  plt.imshow(img_r)
  plt.axis("off")
  plt.title("Reduced colorspace image")
  plt.show()

reduce_color(img_liz)
reduce_color(img_building)
reduce_color(img_plant)
```

Solution 4.2.1

Navigate to Exercise 4.2.1 on page 47

[ ]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA()
lda.fit(X, y)
X_lda = lda.transform(X)

plt.scatter(X_lda[y == 0, 0], X_lda[y == 0, 1], color='red')
plt.scatter(X_lda[y == 1, 0], X_lda[y == 1, 1], color='green')
plt.scatter(X_lda[y == 2, 0], X_lda[y == 2, 1], color='blue')
plt.title("LDA")
plt.show()


pca = PCA()
pca.fit(X)
X_pca = pca.transform(X)
plt.scatter(X_pca[y == 0, 0], X_pca[y == 0, 1], color='red')
plt.scatter(X_pca[y == 1, 0], X_pca[y == 1, 1], color='green')
plt.scatter(X_pca[y == 2, 0], X_pca[y == 2, 1], color='blue')
plt.title("PCA")
plt.show()
```

Solution 5.2.1

Navigate to Exercise 5.2.1 on page 55

[ ]:
```
y_pred = linear.predict(x)
y_test_pred = linear.predict(x_test)

print("Accuracy on train data: ", np.mean(y_pred == y))
print("Accuracy on test data: ", np.mean(y_test_pred == y_test))
```

Solution 5.2.2

Navigate to Exercise 5.2.2 on page 55

[ ]:
```
from sklearn.svm import SVC

linear = SVC(kernel='linear')
linear.fit(x, y)
plot_clf_boundary(linear, x, y)


rbf = SVC(kernel='rbf')
rbf.fit(x, y)
plot_clf_boundary(rbf, x, y)


rbf_auto = SVC(kernel='rbf', gamma='auto')
rbf_auto.fit(x, y)
plot_clf_boundary(rbf_auto, x, y)


poly_3 = SVC(kernel='poly')
poly_3.fit(x, y)
plot_clf_boundary(poly_3, x, y)


poly_5 = SVC(kernel='poly', degree=5)
poly_5.fit(x, y)
plot_clf_boundary(poly_5, x, y)


poly_10 = SVC(kernel='poly', degree=8)
poly_10.fit(x, y)
plot_clf_boundary(poly_10, x, y)


svm_sigmoid = SVC(kernel='sigmoid')
svm_sigmoid.fit(x, y)
plot_clf_boundary(svm_sigmoid, x, y)
```

Solution 5.2.3

Navigate to Exercise 5.2.3 on page 61

[ ]:
```
def plot_clf_boundary_with_support_vectors(clf, x, y):
  x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
  y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
  h_x = (x_max - x_min) / 150
  h_y = (y_max - y_min) / 150
  xx, yy = np.meshgrid(np.arange(x_min, x_max, h_x), np.arange(y_min, y_max, h_y))
  z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

  # Put the result into a color plot
  z = z.reshape(xx.shape)
  plt.contourf(xx, yy, z, cmap=plt.cm.coolwarm, alpha=0.8)

  plt.scatter(x[:, 0], x[:, 1], c=y, cmap=plt.cm.coolwarm)
  plt.xlim(xx.min(), xx.max())
  plt.ylim(yy.min(), yy.max())
  plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],
              facecolors='none', edgecolors='y')
  # plt.xticks(())
  # plt.yticks(())
  plt.show()


linear = SVC(kernel='linear')
linear.fit(x, y)
plot_clf_boundary_with_support_vectors(linear, x, y)

poly_7 = SVC(kernel='poly', degree=7)
poly_7.fit(x, y)
plot_clf_boundary_with_support_vectors(poly_7, x, y)
```

Solution 5.2.4

Navigate to Exercise 5.2.4 on page 64

[ ]:
```
rbf = SVC(kernel='rbf')
rbf.fit(x_spiral, y_spiral)
plot_clf_boundary(rbf, x_spiral, y_spiral)


rbf_auto = SVC(kernel='rbf', gamma='auto')
rbf_auto.fit(x_spiral, y_spiral)
plot_clf_boundary(rbf_auto, x_spiral, y_spiral)
```

Solution 5.3.1

Navigate to Exercise 5.3.1 on page 67

[ ]:
```
def predict_linear(x, w, b):
  y = np.where(np.dot(x, w) + b > 0, 1, -1)
  return y
```

Solution 5.3.2

Navigate to Exercise 5.3.2 on page 71

[ ]:
```
def train_binary_cls(x, y, w, b, eps, n):
  m = len(y)
  for i in range(n):
    if i % 1000 == 0:
      loss = np.mean(np.clip(1 - (np.sum(w[np.newaxis, :] * x, axis=-1) + b) * y, 0.0, np.inf))
      print("At step {} loss: {}".format(i, loss))

    z = np.where((np.dot(x, w) + b) * y < 1, 1, 0)
    b -= eps/m * np.sum(-z * y)
    w[0] -= eps/m * np.sum(-z * y * x[:, 0])
    w[1] -= eps/m * np.sum(-z * y * x[:, 1])

  return w, b
```

Solution 6.1.1

Navigate to Exercise 6.1.1 on page 75

[ ]:
```
X, y = load_wine(return_X_y=True)
# we the dataset in a different way to have access to feature and target names
bunch = load_wine()
print("Loaded wine dataset with features: ", bunch.feature_names,
      " and classes: ", bunch.target_names)
print(X.shape)
print("First vector")
print(X[0, :])

for stat in (chi2, f_classif, mutual_info_classif):
  print("Using stat: ", stat)
  selector = SelectKBest(stat, k=8)
  X_new = selector.fit_transform(X, y)
  print("Features kept: ", selector.get_feature_names_out(bunch.feature_names))
  print(X_new.shape)
  print(X_new[0, :])
```

Solution 6.1.2

Navigate to Exercise 6.1.2 on page 76

[ ]:
```
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.svm import SVC

svc = SVC()

sfs_forward = SequentialFeatureSelector(svc, n_features_to_select=10,
                                        direction="forward").fit(X, y)
print("Features kept going forward:",
      sfs_forward.get_feature_names_out(bunch.feature_names))

sfs_backward = SequentialFeatureSelector(svc, n_features_to_select=10,
                                        direction="backward").fit(X, y)
print("Features kept going backward:",
      sfs_backward.get_feature_names_out(bunch.feature_names))
```

Solution 6.2.1

Navigate to Exercise 6.2.1 on page 77

[ ]:
```
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score

pca = PCA(2)
gnb = GaussianNB()

X_train_transformed = pca.fit_transform(X_train)
gnb.fit(X_train_transformed, y_train)
y_test_pred = gnb.predict(pca.transform(X_test))

print('Accuracy of PCA without normalization: ',
      np.sum(y_test_pred == y_test)/ len(y_test))


plt.scatter(X_train_transformed[:, 0], X_train_transformed[:, 1], c=y_train)
plt.title("PCA without normalization")
plt.show()


scaler = StandardScaler()
pca = PCA(2)
gnb = GaussianNB()

X_train_scaled = scaler.fit_transform(X_train)
X_train_scaled_transformed = pca.fit_transform(X_train_scaled)
gnb.fit(X_train_scaled_transformed, y_train)
y_test_pred = gnb.predict(pca.transform(scaler.transform(X_test)))

print('Accuracy of PCA with normalization: ',
      np.sum(y_test_pred == y_test)/ len(y_test))

plt.scatter(X_train_scaled_transformed[:, 0],
            X_train_scaled_transformed[:, 1], c=y_train)
plt.title("PCA with normalization")
plt.show()
```

Solution 6.2.3

Navigate to Exercise 6.2.3 on page 81

[ ]:
```
selector = SelectKBest(f_classif, k=8)
scaler = StandardScaler()
pca = PCA(6)
gnb = GaussianNB()

X_train_selected = selector.fit_transform(X_train, y_train)
X_train_scaled = scaler.fit_transform(X_train_selected)
X_train_transformed = pca.fit_transform(X_train_scaled)
gnb.fit(X_train_transformed, y_train)

y_pred = gnb.predict(pca.transform(scaler.transform(selector.transform(X_test))))
print("Accuracy with feature selection, normalization and PCA: ",
      np.sum(y_pred == y_test) / len(y_test))
```

Solution 6.3.1

Navigate to Exercise 6.3.1 on page 84

[ ]:
```
from sklearn.naive_bayes import CategoricalNB

# there is a weird behavior if we select random_state = 0 one of the categories
# will not be present in the training set, but only in the test set therefore
# I just found a different random state where this does not happen, otherwise
# you might get an error you can read more at
# https://github.com/scikit-learn/scikit-learn/issues/16028
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=10)

cnb = CategoricalNB()
cnb.fit(X_train, y_train)

y_pred = cnb.predict(X_test)

print('Categorical NB accuracy: ', np.sum(y_pred == y_test) / len(y_test))
```

Solution 6.4.1

Navigate to Exercise 6.4.1 on page 85

[ ]:
```
cat_feats = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety']
# we first remove the classification class
one_hot_data = car_data.drop('class', axis='columns').copy()

for cat in cat_feats:
  one_hot = pd.get_dummies(car_data[cat], prefix=cat)
  one_hot_data = one_hot_data.drop(cat, axis='columns')
  one_hot_data = one_hot_data.join(one_hot)

print(one_hot_data.shape)
print(one_hot_data)
```

Solution 7.1.1

Navigate to Exercise 7.1.1 on page 89

[ ]:
```
# you can try this on various datsets
data = datasets.load_breast_cancer()
# data = datasets.load_iris()
# data = datasets.load_wine()

X = data.data
y = data.target

# you can see how the results may change a bit if we change the random state
X_train_all, X_test, y_train_all, y_test = train_test_split(X, y, test_size=0.2,
                                                            random_state=0)
X_train, X_val, y_train, y_val = train_test_split(X_train_all, y_train_all,
                                                  test_size=0.2, random_state=11)

ks = [1, 3, 5, 7, 9, 11, 15, 19, 23, 29]
metrics = ['chebyshev', 'euclidean', 'manhattan']


best_acc = 0
best_k = None
best_metric = None

for metric in metrics:
  for k in ks:
    knn = make_pipeline(StandardScaler(),
                        KNeighborsClassifier(n_neighbors=k, metric=metric))
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_val)
    acc = np.sum(y_pred == y_val)/ len(y_val)
    if acc > best_acc:
      best_acc = acc
      best_k = k
      best_metric = metric

    print('Accuracy for k = {} and {} metric: {}'.format(k, metric, acc))

print('Best hyperparams: k = {} with {} metric'.format(best_k, best_metric))

knn = make_pipeline(StandardScaler(),
                    KNeighborsClassifier(n_neighbors=best_k, metric=best_metric))
knn.fit(X_train_all, y_train_all)

y_pred = knn.predict(X_test)
print("Accuracy on test data: ", np.sum(y_pred == y_test)/ len(y_pred))
```

Solution 7.1.2

Navigate to Exercise 7.1.2 on page 91

[ ]:
```
from sklearn.model_selection import KFold

data = datasets.load_breast_cancer()
# data = datasets.load_iris()
# data = datasets.load_wine()

X = data.data
y = data.target

# you can see how the results may change a bit if we change the random state
X_train_all, X_test, y_train_all, y_test = train_test_split(X, y, test_size=0.2,
                                                            random_state=10)


k_fold = KFold(5)

for metric in metrics:
  for k in ks:
    knn = make_pipeline(StandardScaler(),
                        KNeighborsClassifier(n_neighbors=k, metric=metric))
    accs = []
    for fold, (train, val) in enumerate(k_fold.split(X_train_all, y_train_all)):
      knn.fit(X_train_all[train], y_train_all[train])
      y_pred = knn.predict(X_train_all[val])
      acc = np.sum(y_pred == y_train_all[val])/ len(y_train_all[val])
      accs.append(acc)
      print('Accuracy for k = {} and {} metric: {} for fold {}'
            .format(k, metric, acc, fold))

    all_acc = np.mean(accs)

    if all_acc > best_acc:
      best_acc = acc
      best_k = k
      best_metric = metric

    print('Accuracy for k = {} and {} metric: {} for all folds'
          .format(k, metric, all_acc))

print('Best hyperparams: k = {} with {}metric'.format(best_k, best_metric))

knn = make_pipeline(StandardScaler(),
                    KNeighborsClassifier(n_neighbors=best_k, metric=best_metric))
knn.fit(X_train_all, y_train_all)
```

```
y_pred = knn.predict(X_test)
print("Accuracy on test data: ", np.sum(y_pred == y_test)/ len(y_pred))
```

Solution 7.1.3

Navigate to Exercise 7.1.3 on page 93

[ ]:
```
def knn(x, k, x_train, y_train):
  # note that square root is not necessary since
  # we only need to know the order of closest neighbors
  dist = np.sum((x[np.newaxis, :] - x_train) ** 2, axis=-1)
  idx = np.argpartition(dist, k)[:k]
  cls, counts = np.unique(y_train[idx], return_counts=True)
  return cls[np.argmax(counts)]
```

Solution 7.2.1

Navigate to Exercise 7.2.1 on page 94

[ ]:
```
from matplotlib import pyplot as plt
from sklearn.metrics import RocCurveDisplay, precision_recall_curve

p, r, t = precision_recall_curve(y_test, y_proba)
plt.plot(r, p)
plt.title('Precision Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.show()

RocCurveDisplay.from_estimator(linear, X_test, y_test)
plt.title('ROC')
plt.show()
```

Solution 7.3.1

Navigate to Exercise 7.3.1 on page 97

[ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```

Solution 8.1.1

Navigate to Exercise 8.1.1 on page 105

[ ]:
```
from matplotlib import pyplot as plt

depths = np.arange(1, 53)
accs = []

for d in depths:
  dtc = DecisionTreeClassifier(max_depth=d)
  dtc.fit(X_train, y_train)
  y_pred = dtc.predict(X_test)
  accs.append(np.sum(y_pred == y_test)/len(y_test))

plt.plot(depths, accs)
plt.xlabel('depth')
plt.ylabel('accuracy')
plt.show()


min_leaf_list = np.arange(10, 301, 10)
accs = []

for l in min_leaf_list:
  dtc = DecisionTreeClassifier(min_samples_leaf=l)
  dtc.fit(X_train, y_train)
  y_pred = dtc.predict(X_test)
  accs.append(np.sum(y_pred == y_test)/len(y_test))

plt.plot(min_leaf_list, accs)
plt.xlabel('min leaf samples')
plt.ylabel('accuracy')
plt.show()
```

Solution 8.4.1

Navigate to Exercise 8.4.1 on page 110

[ ]:
```
forest = RandomForestClassifier(max_depth=20)
forest.fit(X_train.values, y_train)

print(20 * '*')
print('Random forest')
print(20 * '*')

print(classification_report(y_test, forest.predict(X_test)))

for i, dtc in enumerate(forest.estimators_):
  print("For {}-th forest train acc: {}, test acc: {}, depth: {}"
  .format(i, np.mean(dtc.predict(X_train.values) == y_train),
          np.mean(dtc.predict(X_test.values) == y_test), dtc.get_depth()))



adaboost = AdaBoostClassifier()
adaboost.fit(X_train.values, y_train)

print(20 * '*')
print('Random forest')
print(20 * '*')

print(classification_report(y_test, adaboost.predict(X_test)))

for i, dtc in enumerate(adaboost.estimators_):
  print("For {}-th forest train acc: {}, test acc: {}, depth: {}"
  .format(i, np.mean(dtc.predict(X_train.values) == y_train),
          np.mean(dtc.predict(X_test.values) == y_test), dtc.get_depth()))
```

Solution 8.4.2

Navigate to Exercise 8.4.2 on page 112

[ ]:
```
depths = np.arange(1, 11)
accs = []
for d in depths:
  dtc = DecisionTreeClassifier(max_depth=d)

  adaboost = AdaBoostClassifier(base_estimator=dtc)
  adaboost.fit(X_train, y_train)

  accs.append(np.mean(y_test == adaboost.predict(X_test)))

plt.plot(depths, accs)
plt.xlabel('depth')
plt.ylabel('test accuracy')
```

Solution 8.5

Navigate to Exercise 8.5 on page 114

[ ]:
```
from sklearn.model_selection import RandomizedSearchCV

forest = RandomForestClassifier()

params = {'max_depth': np.arange(1,21, 5), 'criterion': ['gini', 'entropy'],
          'min_samples_split': [4, 6, 8],
          'n_estimators': [20, 50, 100]}

rand_search = RandomizedSearchCV(forest, params, verbose=2, n_iter=100)

search = rand_search.fit(X_train, y_train)
print("Best params: ", search.best_params_)

best_forest = RandomForestClassifier(**search.best_params_)
best_forest.fit(X_train, y_train)

print(classification_report(y_test, best_forest.predict(X_test)))
```

Solution 9.4.1

Navigate to Exercise 9.4.1 on page 125

[ ]:
```
from scipy.spatial import distance_matrix

def my_k_means(k, X, max_iter=100):
  m = np.random.rand(k, X.shape[1])
  m *= np.max(X, axis=0) - np.min(X, axis=0)
  m += np.min(X, axis=0)

  old_m = np.zeros_like(m)
  iter = 0
  while np.any(m != old_m) and iter < max_iter:
    iter += 1
    old_m = np.copy(m)
    dist_matrix = distance_matrix(X, m)
    cluster_assignments = np.argmin(dist_matrix, axis=-1)
    for i in range(k):
      if np.sum(cluster_assignments == i) == 0:
        m[i] = np.random.rand()
        m[i] *= np.max(X, axis=0) - np.min(X, axis=0)
        m[i] += np.min(X, axis=0)
      else:
        m[i] = np.mean(X[cluster_assignments==i, :], axis=0)

  return cluster_assignments, m
```

Solution 9.4.2

Navigate to Exercise 9.4.2 on page 129

[ ]:
```
def my_k_means_vis(k, X, max_iter=100):
  m = np.random.rand(k, X.shape[1])
  m *= np.max(X, axis=0) - np.min(X, axis=0)
  m += np.min(X, axis=0)

  old_m = np.zeros_like(m)
  iter = 0
  while np.any(m != old_m) and iter < max_iter:
    iter += 1
    old_m = np.copy(m)
    dist_matrix = distance_matrix(X, m)
    cluster_assignments = np.argmin(dist_matrix, axis=-1)
    for i in range(k):
      if np.sum(cluster_assignments == i) == 0:
        m[i] = np.random.rand()
        m[i] *= np.max(X, axis=0) - np.min(X, axis=0)
        m[i] += np.min(X, axis=0)
      else:
        m[i] = np.mean(X[cluster_assignments==i, :], axis=0)

    plt.scatter(X[:, 0], X[:, 1], c=cluster_assignments)
    plt.scatter(m[:, 0], m[:, 1], c='r')
    plt.plot([old_m[:, 0], m[:, 0]], [old_m[:, 1], m[:, 1]], c='r')
    plt.title('Iteration: {}'.format(iter))
    plt.show()

  return cluster_assignments, m
```

Bibliography

[1] Stefan Aeberhard, Danny Coomans, and Olivier De Vel. Comparative analysis of statistical pattern recognition methods in high dimensional settings. *Pattern Recognition*, 27(8):1065–1077, 1994.

[2] Barry Becker and Ronny Kohavi. Adult Census Dataset. UCI Machine Learning Repository, 1996. DOI: 10.24432/C5XW20.

[3] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[4] Marko Bohanec and Vladislav Rajkovic. Knowledge acquisition and explanation for multi-attribute decision making. In *8th intl workshop on expert systems and their applications*, pages 59–78. Avignon France, 1988.

[5] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

[7] R. A. Fisher. Iris Dataset. UCI Machine Learning Repository, 1988. DOI: 10.24432/C56C76.

[8] Evelyn Fix and Joseph Lawson Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247, 1989.

[9] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[10] Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Arcene Dataset. UCI Machine Learning Repository, 2008. DOI: 10.24432/C58P55.

[11] David J Hand and Keming Yu. Idiot's bayes—not so stupid after all? *International statistical review*, 69(3):385–398, 2001.

[12] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[13] Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

[14] W Nick Street, William H Wolberg, and Olvi L Mangasarian. Nuclear feature extraction for breast tumor diagnosis. In *Biomedical image processing and biomedical visualization*, volume 1905, pages 861–870. SPIE, 1993.