Lab 2: Parametric Classifiers

Machine Learning 2019/2020
Ruben Wiersma and David Tax

WHAT This nonmandatory lab consists of several programming and insight exercises/questions on k-nn density estimation.

WHY The exercises are meant to familiarize yourself with the basic concepts of parametric classifiers.

HOW Follow the exercises in this notebook either on your own or with a friend. If you want to skip right to questions and exercises, find the → symbol. Use Mattermost (https://mattermost.ewi.tudelft.nl/ml/channels/town-square) to discuss questions with your peers. For additional questions and feedback please consult the TA's during the lab session.

The bayes classifier

In this assignment, you will implement your own bayes classifier. Because this is your first assignment, we will walk you through the steps from loading and inspecting the data, to running your classifier.

Specifically, this assignment consists of the following steps:

1. Classification using Gaussian distributions
2. Getting to know the data
3. Validation sets
4. Univariate model
5. Probability density function
6. Posterior probabilities
7. Bayes classifier

Work your way through these exercises at your own pace and be sure to ask questions to the TA's when you don't understand something. It's important that you get what is happening here, as it is a fundamental building block of machine learning.

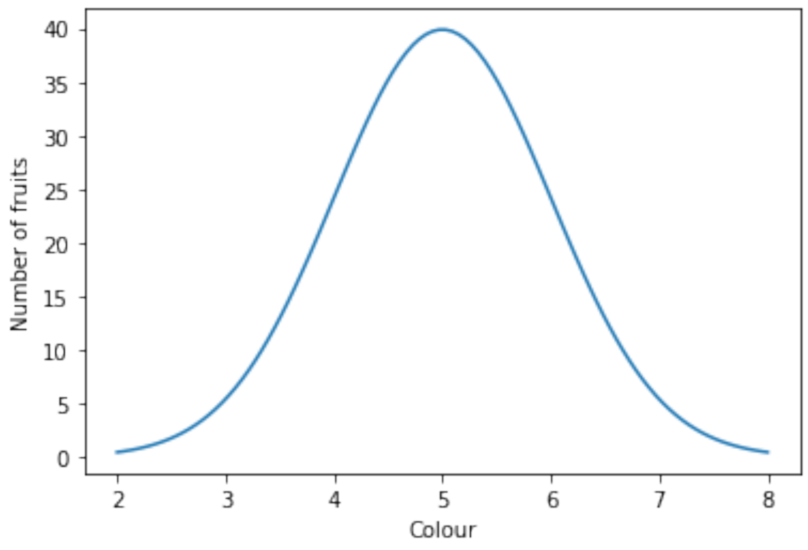
0. Classification using Gaussian distributions

We are starting with a very important notion in machine learning: probability distributions. Occurrences of data typically follow probability distributions that we know how to model.

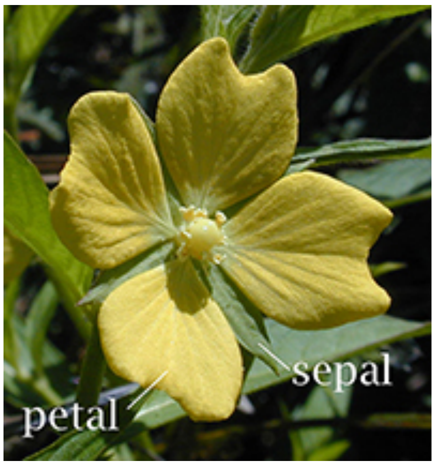
Say, you want to classify apples vs. oranges. A *feature* that you could use to classify them is their colour. We know, of course, that oranges are orange and apples (the golden delicious kind) are green, but each orange is a slightly different shade of orange. Likewise, the apples are all a different shade of green. If we would plot the colour values against the number of fruits with that colour, we would see, however, that there are probably more oranges with a certain type of shade than with other colours. They tend to follow known probability distributions.

In this assignment, we will assume data that has a normal distribution and try to estimate the parameters of the normal distribution to correctly fit our data, hence the name *parametric* classifiers. We will then use Bayes' rule to build a classifier based on the probability distribution.

Just to refresh your mind, this is what a normal distribution looks like:



Instead of apples and oranges, we will try to classify flowers from Fisher's Iris dataset. The dataset contains the measurements of *length* and *width* of the *sepals* and *petals* of 150 flowers.



Using these 4 attributes (*length* and *width* of both), the flowers should then be classified as one of 3 species of Iris flower:

- Iris setosa
- Iris versicolor
- Iris virginica

This dataset is such a classic example that is even included in machine learning libraries. The following code will load the dataset from `scikit-learn` (this was installed with conda) into the variable `iris`.

→ **Exercise 0.1** Run the code and inspect what data is contained in `iris`. Can you identify the 4 attributes? What other information is contained in `iris`?

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [4.6, 3.6, 1. , 0.2],
        [5.1, 3.3, 1.7, 0.5],
        [4.8, 3.4, 1.9, 0.2],
        [5. , 3. , 1.6, 0.2],
        [5. , 3.4, 1.6, 0.4],
        [5.2, 3.5, 1.5, 0.2],
        [5.2, 3.4, 1.4, 0.2],
        [4.7, 3.2, 1.6, 0.2],
        [4.8, 3.1, 1.6, 0.2],
        [5.4, 3.4, 1.5, 0.4],
        [5.2, 4.1, 1.5, 0.1],
        [5.5, 4.2, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.2],
        [5. , 3.2, 1.2, 0.2],
        [5.5, 3.5, 1.3, 0.2],
        [4.9, 3.6, 1.4, 0.1],
        [4.4, 3. , 1.3, 0.2],
        [5.1, 3.4, 1.5, 0.2],
        [5. , 3.5, 1.3, 0.3],
        [4.5, 2.3, 1.3, 0.3],
        [4.4, 3.2, 1.3, 0.2],
        [5. , 3.5, 1.6, 0.6],
        [5.1, 3.8, 1.9, 0.4],
        [4.8, 3. , 1.4, 0.3],
        [5.1, 3.8, 1.6, 0.2],
        [4.6, 3.2, 1.4, 0.2],
        [5.3, 3.7, 1.5, 0.2],
        [5. , 3.3, 1.4, 0.2],
        [7. , 3.2, 4.7, 1.4],
        [6.4, 3.2, 4.5, 1.5],
        [6.9, 3.1, 4.9, 1.5],
        [5.5, 2.3, 4. , 1.3],
        [6.5, 2.8, 4.6, 1.5],
        [5.7, 2.8, 4.5, 1.3],
        [6.3, 3.3, 4.7, 1.6],
        [4.9, 2.4, 3.3, 1. ],
        [6.6, 2.9, 4.6, 1.3],
        [5.2, 2.7, 3.9, 1.4],
        [5. , 2. , 3.5, 1. ],
        [5.9, 3. , 4.2, 1.5],
        [6. , 2.2, 4. , 1. ],
        [6.1, 2.9, 4.7, 1.4],
        [5.6, 2.9, 3.6, 1.3],
        [6.7, 3.1, 4.4, 1.4],
        [5.6, 3. , 4.5, 1.5],
        [5.8, 2.7, 4.1, 1. ],
        [6.2, 2.2, 4.5, 1.5],
        [5.6, 2.5, 3.9, 1.1],
        [5.9, 3.2, 4.8, 1.8],
        [6.1, 2.8, 4. , 1.3],
        [6.3, 2.5, 4.9, 1.5],
        [6.1, 2.8, 4.7, 1.2],
        [6.4, 2.9, 4.3, 1.3],
        [6.6, 3. , 4.4, 1.4],
        [6.8, 2.8, 4.8, 1.4],
        [6.7, 3. , 5. , 1.7],
        [6. , 2.9, 4.5, 1.5],
        [5.7, 2.6, 3.5, 1. ],
        [5.5, 2.4, 3.8, 1.1],
        [5.5, 2.4, 3.7, 1. ],
        [5.8, 2.7, 3.9, 1.2],
        [6. , 2.7, 5.1, 1.6],
        [5.4, 3. , 4.5, 1.5],
        [6. , 3.4, 4.5, 1.6],
        [6.7, 3.1, 4.7, 1.5],
        [6.3, 2.3, 4.4, 1.3],
        [5.6, 3. , 4.1, 1.3],
        [5.5, 2.5, 4. , 1.3],
        [5.5, 2.6, 4.4, 1.2],
        [6.1, 3. , 4.6, 1.4],
        [5.8, 2.6, 4. , 1.2],
        [5. , 2.3, 3.3, 1. ],
        [5.6, 2.7, 4.2, 1.3],
        [5.7, 3. , 4.2, 1.2],
        [5.7, 2.9, 4.2, 1.3],
        [6.2, 2.9, 4.3, 1.3],
        [5.1, 2.5, 3. , 1.1],
        [5.7, 2.8, 4.1, 1.3],
        [6.3, 3.3, 6. , 2.5],
        [5.8, 2.7, 5.1, 1.9],
        [7.1, 3. , 5.9, 2.1],
        [6.3, 2.9, 5.6, 1.8],
        [6.5, 3. , 5.8, 2.2],
        [7.6, 3. , 6.6, 2.1],
        [4.9, 2.5, 4.5, 1.7],
        [7.3, 2.9, 6.3, 1.8],
        [6.7, 2.5, 5.8, 1.8],
        [7.2, 3.6, 6.1, 2.5],
        [6.5, 3.2, 5.1, 2. ],
        [6.4, 2.7, 5.3, 1.9],
        [6.8, 3. , 5.5, 2.1],
        [5.7, 2.5, 5. , 2. ],
        [5.8, 2.8, 5.1, 2.4],
        [6.4, 3.2, 5.3, 2.3],
        [6.5, 3. , 5.5, 1.8],
        [7.7, 3.8, 6.7, 2.2],
        [7.7, 2.6, 6.9, 2.3],
        [6. , 2.2, 5. , 1.5],
        [6.9, 3.2, 5.7, 2.3],
        [5.6, 2.8, 4.9, 2. ],
        [7.7, 2.8, 6.7, 2. ],
        [6.3, 2.7, 4.9, 1.8],
        [6.7, 3.3, 5.7, 2.1],
        [7.2, 3.2, 6. , 1.8],
        [6.2, 2.8, 4.8, 1.8],
        [6.1, 3. , 4.9, 1.8],
        [6.4, 2.8, 5.6, 2.1],
        [7.2, 3. , 5.8, 1.6],
        [7.4, 2.8, 6.1, 1.9],
        [7.9, 3.8, 6.4, 2. ],
        [6.4, 2.8, 5.6, 2.2],
        [6.3, 2.8, 5.1, 1.5],
        [6.1, 2.6, 5.6, 1.4],
        [7.7, 3. , 6.1, 2.3],
        [6.3, 3.4, 5.6, 2.4],
        [6.4, 3.1, 5.5, 1.8],
        [6. , 3. , 4.8, 1.8],
        [6.9, 3.1, 5.4, 2.1],
        [6.7, 3.1, 5.6, 2.4],
        [6.9, 3.1, 5.1, 2.3],
        [5.8, 2.7, 5.1, 1.9],
        [6.8, 3.2, 5.9, 2.3],
        [6.7, 3.3, 5.7, 2.5],
        [6.7, 3. , 5.2, 2.3],
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]), 'target': array([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, 2, 2, 2, 2, 2, 2, 2, 2,
 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]), 'ta
 rget_names': array(['setosa', 'versicolor', 'virginica'], dtype='<
 U10'), 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n-------
 -------------\n\n**Data Set Characteristics:**\n\n    :Number of I
 nstances: 150 (50 in each of three classes)\n    :Number of Attrib
 utes: 4 numeric, predictive attributes and the class\n    :Attribu
 te Information:\n        - sepal length in cm\n        - sepal wid
 th in cm\n        - petal length in cm\n        - petal width in c
 m\n        - class:\n                - Iris-Setosa\n
 - Iris-Versicolour\n                - Iris-Virginica\n
 \n    :Summary Statistics:\n\n    ============== ==== ==== =======
 ===== ====================\n                    Min  Max   Mean
 SD   Class Correlation\n    ============== ==== ==== ======= =====
 ====================\n    sepal length:   4.3  7.9   5.84   0.83
 0.7826\n    sepal width:    2.0  4.4   3.05   0.43   -0.4194\n
 petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)\n    pet
 al width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n    ======
 ======== ==== ==== ======= ===== ====================\n\n    :Miss
 ing Attribute Values: None\n    :Class Distribution: 33.3% for eac
 h of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael Ma
 rshall (MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nTh
 e famous Iris database, first used by Sir R.A. Fisher. The dataset
 is taken\nfrom Fisher\'s paper. Note that it\'s the same as in R,
 but not as in the UCI\nMachine Learning Repository, which has two
 wrong data points.\n\nThis is perhaps the best known database to b
 e found in the\npattern recognition literature.  Fisher\'s paper i
 s a classic in the field and\nis referenced frequently to this day
 .  (See Duda & Hart, for example.)  The\ndata set contains 3 class
 es of 50 instances each, where each class refers to a\ntype of iri
 s plant.  One class is linearly separable from the other 2; the\nl
 atter are NOT linearly separable from each other.\n\n.. topic:: Re
 ferences\n\n   - Fisher, R.A. "The use of multiple measurements in
 taxonomic problems"\n     Annual Eugenics, 7, Part II, 179-188 (19
 36); also in "Contributions to\n     Mathematical Statistics" (Joh
 n Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern
```

```
Classification and Scene Analysis.\n     (Q327.D83) John Wiley & S
ons.  ISBN 0-471-22361-1.  See page 218.\n   - Dasarathy, B.V. (19
80) "Nosing Around the Neighborhood: A New System\n     Structure 
and Classification Rule for Recognition in Partially Exposed\n
Environments".  IEEE Transactions on Pattern Analysis and Machine\
n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (
1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions\n
on Information Theory, May 1972, 431-433.\n   - See also: 1988 MLC
Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptu
al clustering system finds 3 classes in the data.\n   - Many, many
more ...', 'feature_names': ['sepal length (cm)', 'sepal width (cm
)', 'petal length (cm)', 'petal width (cm)'], 'filename': 'F:\\Pro
grams\\Anaconda3\\lib\\site-packages\\sklearn\\datasets\\data\\iri
s.csv'}
```

1. Getting to know the data

The dataset is stored as a dictionary, a data structure in Python that resembles a hashmap. We can access items in the dictionary with a dot `.`, so we access the data and their target labels with `iris.data` and `iris.target`. If we want to know what each digit means, we can access the names with `iris.target_names`.

→ **Exercise 1.1** Run the code fragment and confirm what it is doing. Try to understand the indexing and print the following data:

- The last five flowers. Expected result: a [5 x 4] array.
- Only the third feature of each flower. Expected result: a [150 x 1] array.
- The names of the first ten flowers. Expected result: a [10 x 1] array with strings.
- Three separate arrays (one for each class). Expected result: three [50 x 4] arrays. Try doing this without assuming anything about the indices for each class, i.e.: do not simply use `class1 = iris.data[:50, :]`

Hint Look at the indexing chapter in last week's lab for help.

Hint For the final exercise, you can use `np.where`. This function takes in a boolean statement and returns the indices for which the statement is true. Example use: `np.where(iris.target == 0)` returns all indices where the target label is 0.

```
First five flowers:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
Their labels:  [0 0 0 0 0]
And the label names:  ['setosa' 'versicolor' 'virginica']
Last five flowers:
 [[6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
Only the third feature:  [1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5
```

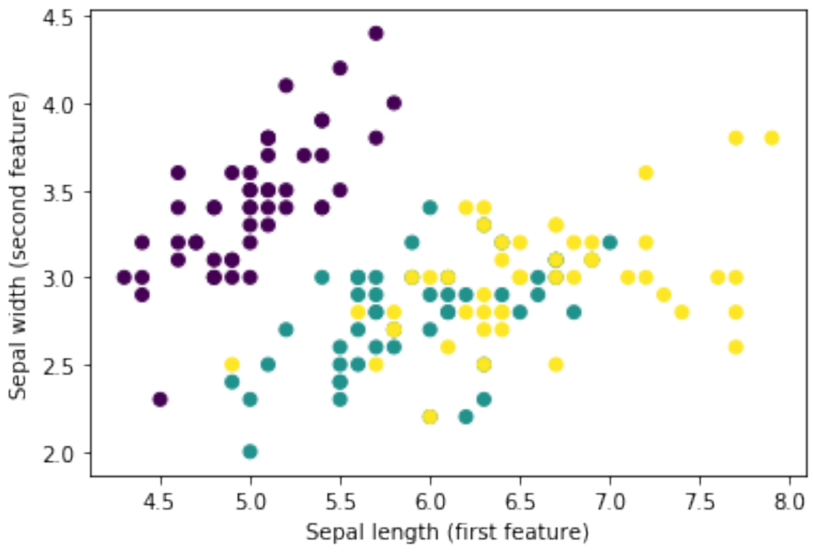
```
1.5 1.6 1.4 1.1 1.2 1.5 1.3 1.4
 1.7 1.5 1.7 1.5 1.  1.7 1.9 1.6 1.6 1.5 1.4 1.6 1.6 1.5 1.5 1.4 1
.5 1.2
 1.3 1.4 1.3 1.5 1.3 1.3 1.3 1.6 1.9 1.4 1.6 1.4 1.5 1.4 4.7 4.5 4
.9 4.
 4.6 4.5 4.7 3.3 4.6 3.9 3.5 4.2 4.  4.7 3.6 4.4 4.5 4.1 4.5 3.9 4
.8 4.
 4.9 4.7 4.3 4.4 4.8 5.  4.5 3.5 3.8 3.7 3.9 5.1 4.5 4.5 4.7 4.4 4
.1 4.
 4.4 4.6 4.  3.3 4.2 4.2 4.2 4.3 3.  4.1 6.  5.1 5.9 5.6 5.8 6.6 4
.5 6.3
 5.8 6.1 5.1 5.3 5.5 5.  5.1 5.3 5.5 6.7 6.9 5.  5.7 4.9 6.7 4.9 5
.7 6.
 4.8 4.9 5.6 5.8 6.1 6.4 5.6 5.1 5.6 6.1 5.6 5.5 4.8 5.4 5.6 5.1 5
.1 5.9
 5.7 5.2 5.  5.2 5.4 5.1]
All label names:  ['setosa' 'setosa' 'setosa' 'setosa' 'setosa' 's
etosa' 'setosa' 'setosa'
 'setosa' 'setosa']
Class:  setosa ; Items:
 [[[5.1 3.5 1.4 0.2]
  [4.9 3.  1.4 0.2]
  [4.7 3.2 1.3 0.2]
  [4.6 3.1 1.5 0.2]
  [5.  3.6 1.4 0.2]
  [5.4 3.9 1.7 0.4]
  [4.6 3.4 1.4 0.3]
  [5.  3.4 1.5 0.2]
  [4.4 2.9 1.4 0.2]
  [4.9 3.1 1.5 0.1]
  [5.4 3.7 1.5 0.2]
  [4.8 3.4 1.6 0.2]
  [4.8 3.  1.4 0.1]
  [4.3 3.  1.1 0.1]
  [5.8 4.  1.2 0.2]
  [5.7 4.4 1.5 0.4]
  [5.4 3.9 1.3 0.4]
  [5.1 3.5 1.4 0.3]
  [5.7 3.8 1.7 0.3]
  [5.1 3.8 1.5 0.3]
  [5.4 3.4 1.7 0.2]
  [5.1 3.7 1.5 0.4]
  [4.6 3.6 1.  0.2]
  [5.1 3.3 1.7 0.5]
  [4.8 3.4 1.9 0.2]
  [5.  3.  1.6 0.2]
  [5.  3.4 1.6 0.4]
  [5.2 3.5 1.5 0.2]
  [5.2 3.4 1.4 0.2]
  [4.7 3.2 1.6 0.2]
  [4.8 3.1 1.6 0.2]
  [5.4 3.4 1.5 0.4]
  [5.2 4.1 1.5 0.1]
  [5.5 4.2 1.4 0.2]
  [4.9 3.1 1.5 0.2]
  [5.  3.2 1.2 0.2]
  [5.5 3.5 1.3 0.2]
```

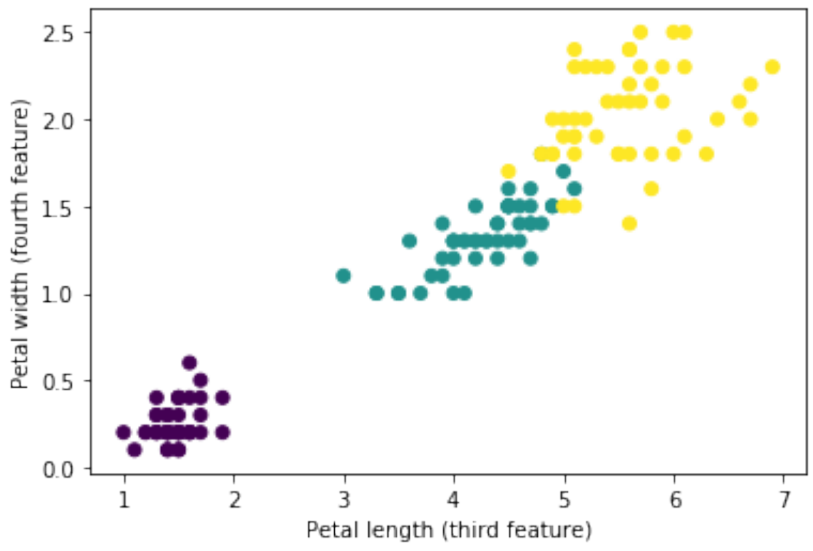
```
   [4.9 3.6 1.4 0.1]
   [4.4 3.  1.3 0.2]
   [5.1 3.4 1.5 0.2]
   [5.  3.5 1.3 0.3]
   [4.5 2.3 1.3 0.3]
   [4.4 3.2 1.3 0.2]
   [5.  3.5 1.6 0.6]
   [5.1 3.8 1.9 0.4]
   [4.8 3.  1.4 0.3]
   [5.1 3.8 1.6 0.2]
   [4.6 3.2 1.4 0.2]
   [5.3 3.7 1.5 0.2]
   [5.  3.3 1.4 0.2]]]
Class:  versicolor ; Items:
 [[[7.  3.2 4.7 1.4]
   [6.4 3.2 4.5 1.5]
   [6.9 3.1 4.9 1.5]
   [5.5 2.3 4.  1.3]
   [6.5 2.8 4.6 1.5]
   [5.7 2.8 4.5 1.3]
   [6.3 3.3 4.7 1.6]
   [4.9 2.4 3.3 1. ]
   [6.6 2.9 4.6 1.3]
   [5.2 2.7 3.9 1.4]
   [5.  2.  3.5 1. ]
   [5.9 3.  4.2 1.5]
   [6.  2.2 4.  1. ]
   [6.1 2.9 4.7 1.4]
   [5.6 2.9 3.6 1.3]
   [6.7 3.1 4.4 1.4]
   [5.6 3.  4.5 1.5]
   [5.8 2.7 4.1 1. ]
   [6.2 2.2 4.5 1.5]
   [5.6 2.5 3.9 1.1]
   [5.9 3.2 4.8 1.8]
   [6.1 2.8 4.  1.3]
   [6.3 2.5 4.9 1.5]
   [6.1 2.8 4.7 1.2]
   [6.4 2.9 4.3 1.3]
   [6.6 3.  4.4 1.4]
   [6.8 2.8 4.8 1.4]
   [6.7 3.  5.  1.7]
   [6.  2.9 4.5 1.5]
   [5.7 2.6 3.5 1. ]
   [5.5 2.4 3.8 1.1]
   [5.5 2.4 3.7 1. ]
   [5.8 2.7 3.9 1.2]
   [6.  2.7 5.1 1.6]
   [5.4 3.  4.5 1.5]
   [6.  3.4 4.5 1.6]
   [6.7 3.1 4.7 1.5]
   [6.3 2.3 4.4 1.3]
   [5.6 3.  4.1 1.3]
   [5.5 2.5 4.  1.3]
   [5.5 2.6 4.4 1.2]
   [6.1 3.  4.6 1.4]
   [5.8 2.6 4.  1.2]
```

```
  [5.  2.3 3.3 1. ]
  [5.6 2.7 4.2 1.3]
  [5.7 3.  4.2 1.2]
  [5.7 2.9 4.2 1.3]
  [6.2 2.9 4.3 1.3]
  [5.1 2.5 3.  1.1]
  [5.7 2.8 4.1 1.3]]]
Class:  virginica ; Items:
 [[[6.3 3.3 6.  2.5]
  [5.8 2.7 5.1 1.9]
  [7.1 3.  5.9 2.1]
  [6.3 2.9 5.6 1.8]
  [6.5 3.  5.8 2.2]
  [7.6 3.  6.6 2.1]
  [4.9 2.5 4.5 1.7]
  [7.3 2.9 6.3 1.8]
  [6.7 2.5 5.8 1.8]
  [7.2 3.6 6.1 2.5]
  [6.5 3.2 5.1 2. ]
  [6.4 2.7 5.3 1.9]
  [6.8 3.  5.5 2.1]
  [5.7 2.5 5.  2. ]
  [5.8 2.8 5.1 2.4]
  [6.4 3.2 5.3 2.3]
  [6.5 3.  5.5 1.8]
  [7.7 3.8 6.7 2.2]
  [7.7 2.6 6.9 2.3]
  [6.  2.2 5.  1.5]
  [6.9 3.2 5.7 2.3]
  [5.6 2.8 4.9 2. ]
  [7.7 2.8 6.7 2. ]
  [6.3 2.7 4.9 1.8]
  [6.7 3.3 5.7 2.1]
  [7.2 3.2 6.  1.8]
  [6.2 2.8 4.8 1.8]
  [6.1 3.  4.9 1.8]
  [6.4 2.8 5.6 2.1]
  [7.2 3.  5.8 1.6]
  [7.4 2.8 6.1 1.9]
  [7.9 3.8 6.4 2. ]
  [6.4 2.8 5.6 2.2]
  [6.3 2.8 5.1 1.5]
  [6.1 2.6 5.6 1.4]
  [7.7 3.  6.1 2.3]
  [6.3 3.4 5.6 2.4]
  [6.4 3.1 5.5 1.8]
  [6.  3.  4.8 1.8]
  [6.9 3.1 5.4 2.1]
  [6.7 3.1 5.6 2.4]
  [6.9 3.1 5.1 2.3]
  [5.8 2.7 5.1 1.9]
  [6.8 3.2 5.9 2.3]
  [6.7 3.3 5.7 2.5]
  [6.7 3.  5.2 2.3]
  [6.3 2.5 5.  1.9]
  [6.5 3.  5.2 2. ]
  [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]]
```

Next, we can make plots of our data to see how it is distributed.

→ **Exercise 1.2** Run the following code to plot the petal length and width of each flower as a scatterplot. Inspect the code carefully, as you will need to write your own code for plotting later on.





→ **Question 1.1** How are the points distributed? Could you fit a probability distribution that you know on this data (e.g. uniform, normal, etc.)?

2. Validation sets

Now that we have an idea what our dataset looks like, our goal is to create a model that will predict the class of each flower based on its attributes. In order to evaluate how well the model fits, we will also need a validation set where we can test some of our predictions. For this, we will split the data randomly in a train and validation set.

→ **Exercise 2.1** Use the code below to split the dataset into a train and validation set.

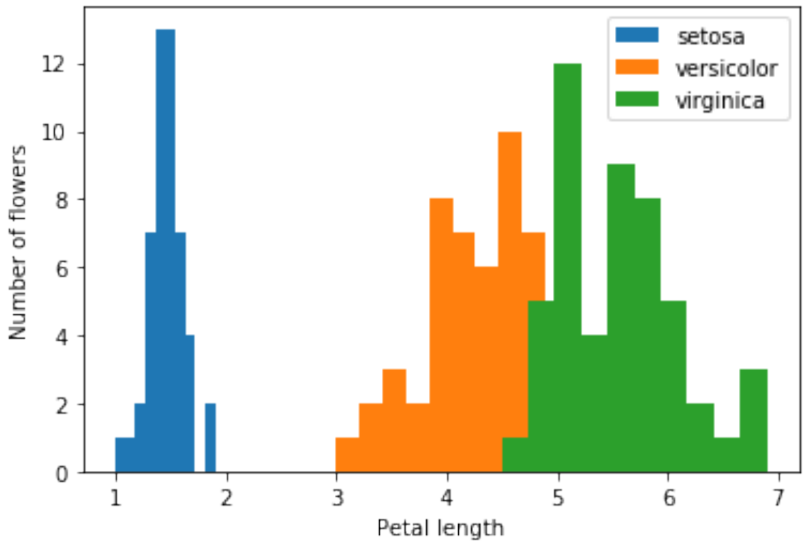
Tip we use the function `train_test_split`. It is easy to confuse the test set with the validation set, but it is important to separate the two: you can use your validation set when you are creating and adjusting your model. In the contrary, the test set is only to be used after you are done adjusting your model. If you change your model to better fit the test set, you risk overfitting on the data that you have. In other words: you perform well on the data you know, but might do really bad on data you don't know.

3. Univariate model

Looking at the plots of the data from the previous section, you might assume that separating the different classes would be a lot easier based on the petal data (3rd and 4th variable) than on the sepal data (1st and 2nd variable), as it is easier to distinguish the different clusters in that plot. In fact, for now we will only focus on one variable, the petal length (3rd variable), as it looks like it might be useful just on its own and this will simplify the model a lot.

Let's first take a look at the distribution of flowers along this variable to confirm that our assumption of a normal distribution is correct.

```
<matplotlib.legend.Legend at 0x15f33f8df28>
```



That looks about correct! Now, let's find the parameters of the normal distribution that describe our data best. The parameters that we need to find are the mean and standard deviation.

→ **Exercise 3.1** Using the training data from each of 3 classes, compute the mean (μ) and standard deviation (σ) for the *pedal length* attribute. The Maximum Likelihood Estimators for these are given by

(3.1)

$$\mu = \frac{\sum_{t=1}^{N} x^t}{N}$$

(3.2)

$$\sigma = \sqrt{\frac{\sum_{t=1}^{N} (x^t - m)^2}{N}}$$

Hint Try to use numpy's functions to perform operations on your input (e.g. `np.sum`, `np.sqrt`)

```
1.4486486486486485 4.297142857142857 5.663636363636364
0.1780094464872985 0.4650565030406608 0.5693261204309519
```

→ **Question 3.1** Do these mean values and standard deviations correspond to the histograms that we plotted? If not, try to fix your code.

4. Probability density function

The probability density function for a Gaussian distribution is defined as

(4.1)

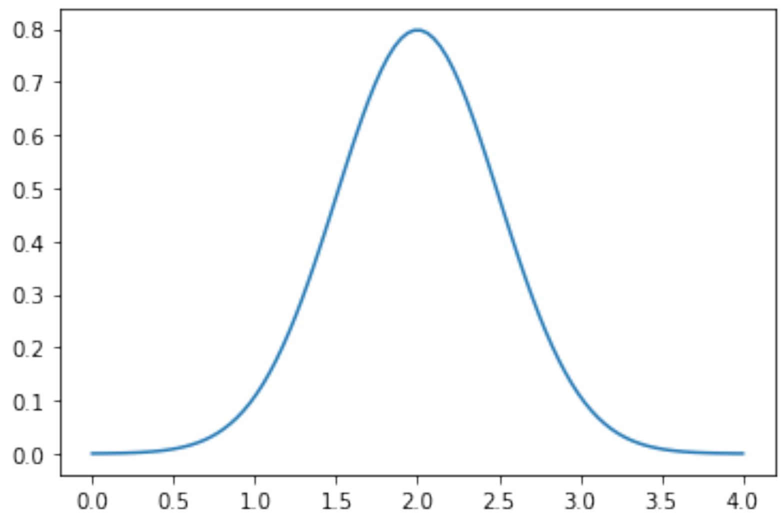
$$p(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where X is Gaussian (normal) distributed with mean μ and variance σ^2, denoted as $\mathcal{N}(\mu, \sigma^2)$.

That means that if we have estimates for μ and σ, we can compute the probability density for a specific value x.

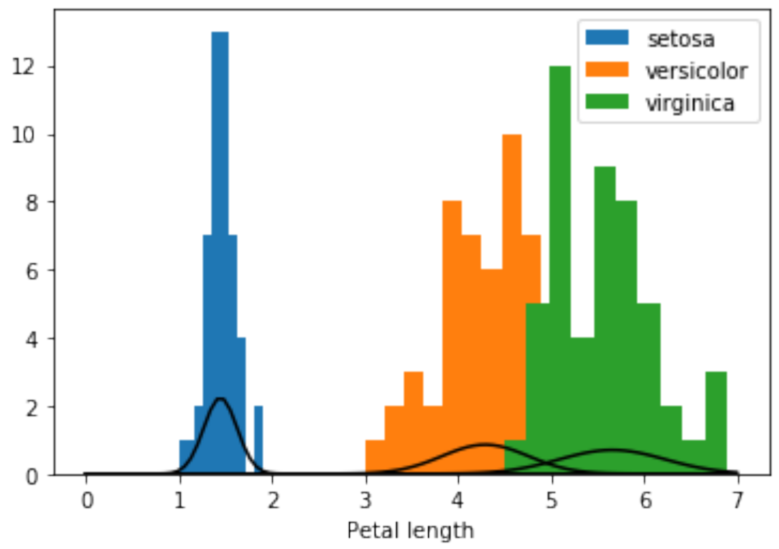
→ **Exercise 4.1** Implement the `normal_PDF` function below. Given `x`, `mean`, and `sd`, we want to return the result of $p(x|\mu, \sigma)$. Your PDF is plotted. Play around with different configurations of `mean` and `sd` to see how these parameters influence your normal distribution.

```
Your pdf function outcome:  0.008863696823876015  Scipy's function
outcome:  0.008863696823876015
```



We already made estimates for μ and σ for the *petal length* for each of the 3 classes, so we can now define PDFs for each separate class.

→ **Exercise 4.2** Plot the 3 functions using linspace (https://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.linspace.html) for a range of x-values aside the histograms of the classes.



→ **Question 4.1** Do your distributions and the histogram overlap? In what ways are the histogram and the probability distributions different?

Hint The histogram shows the number of flowers that have a petal length within a certain window (bin). That means the values shown in the histogram are absolute counts.

5. Posterior probabilities

The plot above shows the probability densities for a value x. For a normal distribution of a class, you only need to know the mean and the standard deviation to be able to determine the probability that a data point x is from a class provided C_i, i.e. $p(x|\mu_i, \sigma_i)$; e.g. the probability that an unknown fruit is an apple given the apples' class distribution. This is equivalent to the probability density given that specific class, i.e. $p(x|C_i)$.

→ **Question 5.1** Stop for a moment to try and understand what this probability means: $p(x|C_i)$.

Hint The | sign in $p(x|C_i)$ means: given that.

However, what would be useful for classification, is the posterior probabilities of the classes given the data, i.e. $P(C_i|x)$.

→ **Question 5.2** Can you explain this mathematical formulation in your own words?

→ **Question 5.3** Why is it helpful to know the posterior probability?

Hint What information do we have for test points coming in?

To get the posterior probability, we can use Bayes' rule:

(5.1)

$$P(C_i|x) = \frac{p(x|C_i)P(C_i)}{p(x)} = \frac{p(x|C_i)P(C_i)}{\sum_{k=1}^{K} p(x|C_k)P(C_k)}$$

Because here we have no prior knowledge of the distribution of the different classes, we can just assume all prior class probabilities $P(C_i)$ to be equal. For our 3 class problem, that would mean a probability of $\frac{1}{3}$ for each class, but we can also just factor the common prior out of the equation and simplify to

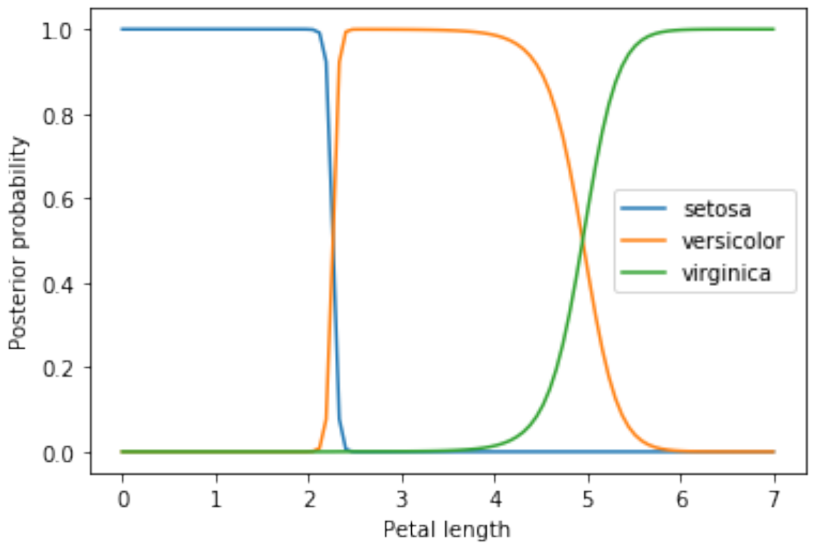
$$P(C_i|x) = \frac{p(x|C_i)}{\sum_{k=1}^{K} p(x|C_k)}$$

→ **Exercise 5.1** Finish the code to compute the posterior probability of a point x, given the mean, standard deviation, and class index.

Tip The mean and standard deviation are given as arrays. You can access the mean for class `i` with `mean[i]`.

```
Posterior probability for class virginica :  0.9932437309327692
Flower belongs to class virginica
```

→ **Exercise 5.2** Plot the posterior probabilities for all 3 classes. Does the plot of these 3 posteriors make sense based on the data?



→ **Question 5.4** Where would you put the decision boundary for each class? In other words: where would you draw the line, separating each class. Could you formulate this mathematically?

6. Bayes Classifier

Now that we can compute the posteriors for every class, constructing a classifier is easy. The Bayes classifier is defined as

- Classify as C_i for which: $i = argmax_i \; P(C_i|x)$

→ **Exercise 6.1** Write the code for the `classify` function. It should classify a single data point x as one of the 3 classes, returning 0, 1 or 2 based on the class the flower is most likely to belong to. The other arguments of the function should therefore be the vector of mean estimates `means` and the vector of standard deviation estimates `sds`, where index `i` corresponds to class C_i .

```
Predicted class virginica
Flower belongs to class virginica
```

→ **Exercise 6.2** Finally, complete the `validate` function below. It should take a validation set, the expected class (ground truth; correct classifications of each element in the validation set) for all data points in that set and the vectors `means` and `sds` with which to calculate the classifications (decides which class each point belongs to) based on the distributions learnt from the training set. The function should return the percentage of elements in the validation set that were classified correctly.

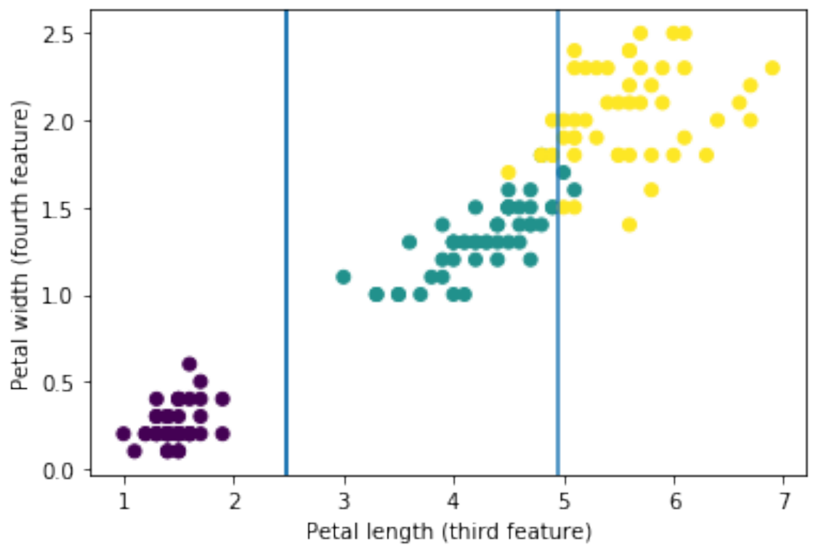
Hint You will only need to use the *petal length* variable from each data point to attempt to classify it (since that is how we trained our model).

```
0.9111111111111111
```

Let's return to our scatterplots and see how your classifier makes decisions. For this, we also plot the decision boundary. The function to create the decision boundaries does this in a very simple way:

- For each class, compute the posterior for 1000 points between 1 and 7
- If for any two classes the posteriors are equal at a point, add that point to the list of decision boundaries
- Plot vertical lines at these points

$\rightarrow$ **Question 6.1** Can you find a way to do improve this? Could you analytically solve the equations?


Petal width (fourth feature)
Petal length (third feature)