Machine Learning

Exercises

Contents

Software configuration

This tutorial uses Python 3.5 with the Spyder IDE and and libraries such as Matplotlib, Numpy and Scikit-Learn. The easiest way to set up the software for this tutorial is to use the free Anaconda package distributed by Continuum Analytics™ which you can download from:

https://www.anaconda.com/products/individual

To open the Spyder IDE[1] in Windows click the link in the Anaconda folder on your Start menu (probably in All Programs). If you are running Spyder for the first time, it may ask you to select a folder (the Spyder workspace) for your Python projects. Choose a convenient folder.

To test your instalation, copy the following code and paste it on the iPython console, on the bottom-right of the Spyder interface:

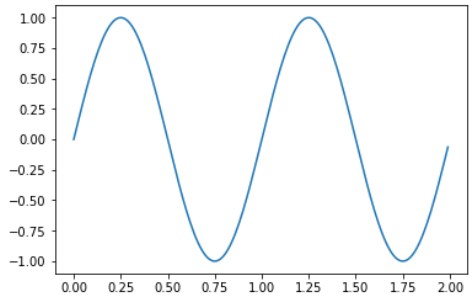
```
%matplotlib inline
from pylab import *
t = arange(0.0, 2.0, 0.01)
s = sin(2*pi*t)
plot(t, s)
show()
```

Press enter to execute the code. This should be the result:

```
Python 3.6.5 |Anaconda, Inc.| (default, Apr 29 2018, 16:14:56)
Type "copyright", "credits" or "license" for more information.

IPython 6.4.0 -- An enhanced Interactive Python.

In [1]: %matplotlib inline
   ...: from pylab import *
   ...: t = arange(0.0, 2.0, 0.01)
   ...: s = sin(2*pi*t)
   ...: plot(t, s)
   ...: show()
   ...:
   ...:
```



[1] Integrated Development Environment, consisting of a source-code editor, console and other tools to make development easier.

Chapter 1

Python basics

Variables: data types and scope. Flow control and functions. Importing modules. Files and plotting.

1.1 Using the console; basic data types

Note: when copying text from the pdf file, some characters may not be copied correctly. Check the code carefully after pasting or type the code yourself.

Open the Spyder IDE. On the lower right corner of the Spyder interface you can find the iPython console. This is an interactive Python console that you can use to immediately execute commands. We will start with some simple commands just to illustrate some aspects of Python. We will start by creating two string variables. Strings in Python can be delimited with `"` or `'`, but must begin and end with the same delimiter. You can also use three delimiters to create multi-line strings. Type this in the console

```
first_string = "single line string"
second_string = '''this string
has two lines'''
```

This created two string variables. Note that the first string is delimited with a single double-quote character (`"`) while the second string is delimited by a sequence of three single-quote characters (`'`, repeated three times) in order to allow line breaks within the string. Also note that in Python the most common convention for compound names is to use all lower-case and separate the words with the underscore character.

Now type the following directly in the console:

```
first_string
second_string
print(second_string)
```

You should see this output as you type and enter the commands:

```
first_string
Out [3] : 'single line string'
```

```
second_string
Out [4] : 'this string\nhas two lines'

print(second_string)
this string
has two lines
```

Console commands that return values echo these values if they are not attributed to any variable. Note, however, that the echo may not display exactly as the `print` command does. In this case, the line break is represented with the escape sequence
`n`. You can use these escape sequences when creating strings too.

Strings, like everything else in Python, are objects. You can use the `dir` command to list the methods belonging to an object:

```
dir(first_string)
```

Typing this, you will see a long list of methods available in the string object. By convention, methods whose name begins with an underscore are private and not meant to be called from outside the object[1].

Also note that even constants are objects in Python. Try these examples:

```
first_string.upper()
Out[12]: 'SINGLE LINE STRING'

first_string.startswith('single')
Out[13]: True

first_string.startswith('abc')
Out[14]: False

'abc'.startswith('a')
Out[15]: True

'ABC'.lower()
Out[16]: 'abc'

'ABC'.lower
Out[17]: <function lower>
```

Note that the fourth and fifth examples use methods from constant objects. Also note that, when invoking a function (or method) it is always necessary to write the parenthesis even when there are no arguments. Otherwise the command is interpreted as merely referencing the function object (functions are also objects in Python) instead of executing it.

To index elements or slices of ordered arrays, like a string, we use the square brackets followed by an index value (zero-based) or a slice with the syntax `[beginning:end:step]`, where the step is optional. Here are some examples. Try them out (one at a time) to understand how slicing works.

```
first_string[0]
first_string[1]
first_string[2:4]
```

[1]There is a difference between a single and a double underscore, but we will not worry about that in this course.

```
first_string[-4:]
first_string[4:]
first_string[:4]
```

Note that indexes are zero-based and negative indexes count from the end, as illustrated here with the string `'Python'`:

```
+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
  0   1   2   3   4   5
 -6  -5  -4  -3  -2  -1
```

Slices are set at the intervals between elements in the array:

```
+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

Thus, a slice `[1:3]` will span the second and third elements.

Apart from strings, other common data types are integers and floating point numbers, lists, tuples and dictionaries. Here are some examples:

```
nothing = None                          # no value
minimum_wage = 505                      # integer
pi_squared = 9.8696                     # float
first_name = 'Ludwig'                   # string
years = [1999,2000,2001]                # list
coordinates = (23.4, 12.6, 13.5)        # tuple
animal_classes = {'fox':'mammal',       # dictionary
                  'snake':'reptile',
                  'fly':'insect'}
```

Dictionaries are defined by a comma-separated list of `key:value` pairs. The values can be any objects but the keys must be immutable objects. These are objects whose data cannot be changed, and include numbers, strings and tuples. Lists cannot be used as keys in dictionaries because list contents can be changed. To illustrate this, try these commands:

```
a_list = [1,2,3]
a_list[1] = 4
a_string = 'abc'
a_string[1] = 'x'
a_tuple = (1,2,3)
a_tuple[1] = 4
```

You will notice that trying to change the contents of a string or tuple will raise an exception

1.2 The first script

Instead of using the console directly, it is often better to write the code to a source file. Create a new file in Spyder and save it with a `.py` extension in the same folder to where you extracted this session's material. Let us start by reading the file planets.csv, containing the name, mean distance from the Sun (in Astronomical Units) and orbital period (in Earth years) for each planet in our solar system. To read a text file, we simply create a file object using the `open` function and then read all lines to a list of strings. Finally, we close the file to free it.

```
1 fil = open('planets.csv')
2 lines = fil.readlines()
3 fil.close()
```

If you do not care when the file is freed, you can read a file in a single line of code:

```
1 lines = open('planets.csv').readlines()
```

Python's garbage collection will eventually dispose of the file object and free up the file. This is often adequate for most purposes.

You can run your script by pressing `F5`. Since the `lines` variable has a global scope, you can access its contents in the console:

```
lines
Out[36]:
['Name,mean distance from Sun (AU),orbital period (Earth years)\n',
 'Mercury,0.39,0.24\n',
 'Venus,0.72,0.62\n',
 'Earth,1,1\n',
 'Mars,1.52,1.88\n',
 'Jupiter,5.2,11.86\n',
 'Saturn,9.54,29.46\n',
 'Uranus,19.18,84.01\n',
 'Neptune,30.06,164.8\n']
```

This list of strings contains all lines in the text file. Also note that the newline characters are included in the strings. String objects have a `strip()` method that returns the string without any whitespace in the beginning and end of the string. This is useful if you need to clean up the strings after reading.

As an exercise, let's write a function that receives a planet's name and returns its orbital period reading this file. This is the function, type it in the beginning of your script:

```
1 def planet_period(planet):
2     """Return orbital period
3     Uses planets.csv
4     """
5     lines = open('planets.csv').readlines()
6     for line in lines:
7         parts = line.split(',')
8         if parts[0] == planet:
9             return float(parts[2])
```

A function definition begins with the keyword `def` followed by the name of the function, the parameters in parentheses and then a colon (`:`). When you press enter after a colon the Spyder editor

should automatically indent your code. This indentation is part of Python's syntax and indicates a code block, so be careful of the proper alignment of your indented blocks.

Lines 2 and 3 are the docstring for the function. This is optional but may be useful if you use many functions because it is displayed by the `help` function using the command `help(function_name)`. You can also use the `help` function on any other function or method (in which case you must give the object too, for example, `help(line.split)`).

Then we read all the file contents and loop through the lines. The `for variable in iterable:` construct loops through all elements of an iterable object. In this case, the list with all strings. The block inside the loop is indented, and includes splitting the line on the commas to separate the fields and then checking if the first field is equal to the planet name. This if block is also indented, and the `return` command immediately exits the function returning the values indicated after the command. Note the `float` call to typecast the string into a floating number.

Now run your script (`F5`) to declare this function. Then you should get this result in the console if everything is working:

```
planet_period('Mars')
Out[38]: 1.88
```

Since the function declaration has a global scope in the script, you can access it in the console after running the script. However, you cannot access variables with a local scope inside the function.

1.3 Using libraries: Numpy

Now we will use the Numpy library to store the planet data in a two-dimensional matrix. To use a module or a library in Python we must import it into our namespace. Here is one way to do this:

```
1 import numpy
2
3 def load_planet_data(file_name):
4     """Return matrix with orbital radius and period"""
5     rows = []
6     lines = open(file_name).readlines()
7     for line in lines[1:]:
8         parts = line.split(',')
9         rows.append( (float(parts[1]),float(parts[2])) )
10    return numpy.array(rows)
11
12 data = load_planet_data('planets.csv')
```

In this example, first we import the Numpy library. Note that the convention in Python is to group all `import` statements at the beginning of the module (the `.py` file). Then we read the file contents as a list of strings and, starting from the second line to skip the column headers, we split the fields on the commas and append tuples to the `rows` list. Note that the tuple is formed using parentheses, and contains the distance from the Sun and the orbital period.

After the loop ends (note the indentation) we return an array object from the Numpy library created using the list of tuples. An array of arrays (lists or tuples), in this case, results in a two-dimensional matrix, where each row corresponds to each of the inner elements of the outer array.

Finally, in this script we call the function and store the resulting matrix in the `data` variable. Note the indentation, indicating that this call is outside the function definition. Run the script and type the following commands in the console, one at a time:

```
data
data[:,0]
data[:,[0]]
data[:,0]>1
data[data[:,0]>1,:]
```

The first one outputs the matrix object, which is a two-dimensional array. You can check the values to see if you have all data correctly loaded. The second returns a one-dimensional array with the values of the first column, since we specify that we want all elements with all indexes in the first dimension (the rows) and index 0 in the second dimension (columns).

The third outputs all values in the first column but keeps the two dimensions of the array, returning a column. Note the difference between the two, as this is sometimes an important detail when manipulating data.

The fourth command returns a boolean array with the result of comparing all values in the first column with the value of 1. Finally, the fifth shows how you can use these boolean masks to pick out specific values from the array. Indexing with a boolean mask will return those indexes with values of `True`.

Exercises

- Select the rows corresponding to planets with orbital periods greater than 10 years.
- How many planets have orbital periods greater than 10 years? Hint: you can use the `len` function to find the length of an array or the `numpy.sum` function to find the total sum of array elements, which considers `True` to be 1 and `False` to be 0.
- Select the orbital periods of the planets whose orbital periods in years are greater than twice the orbital radius in AU. Note that you can use algebraical operators, such as sum or multiplication, with array objects.

1.4 Plotting

We'll start by plotting our data, using the Matplotlib library. Here is an example of the basics:

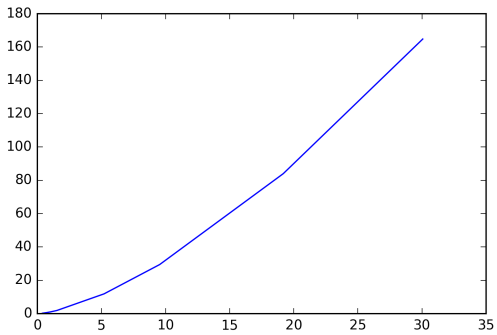
```
import matplotlib.pyplot as plt
plt.figure()
plt.plot(data[:,0],data[:,1])
plt.show()
plt.close()
```

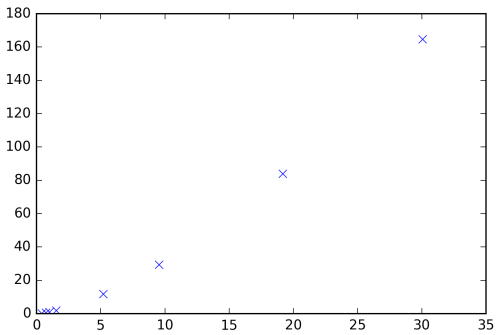
First we import the pyplot module in the matplotlib library but, to avoid having to write `matplotlib.pyplot` every time we use one of its functions, we use the `as` keyword to give it an alias, so now we can refer to it as `plt`.

Then we call the `figure` function to create a new figure for plotting, `plot` to plot the data, giving two arrays for the X and Y values, and then `show()` for displaying the graph. Finally, `close()` frees all the figure objects.

By default, this will display a line joining all the points. But we can specify the format of the plot using a format string as the third argument. Simply replace the `plot` command above with `plt.plot(data[:,0],data[:,1],'x')` [2]

You can also save the figures with the `savefig` function. For example, using `plt.savefig('planets.png',dpi=300)` after plotting and before closing the figure will save the image as a png file with a resolution of 300 dots per inch. This should be the result of your plots:





Using the iPython console

When running your scripts without the iPython console's inline plotting feature, you need to call the (show) method for each plot you wish to display. This will create a window showing the plot and suspend execution until the window is closed. You can use the `close` method to dispose of the plot object.

The iPython console has an inline plotting feature that not only shows the plots in the console when you use the (show) command but also displays any plots active in memory when the script ends if they are not visible. Thus, when using the iPython console, such as when running your scripts in Spyder, you do not need the (show) if you do not close your plots. But note that this is a feature of the iPython console and, without calling the (show) method, your plots will not be displayed if your script is run without the iPython console.

You can disable the iPython console's inline plotting feature with the magic command

```
%matplotlib
```

or enable it again with

```
%matplotlib inline
```

Note that these commands starting with `%` are not Python instructions but special commands for the iPython console and only work with this console, and in Jupyter notebooks.

[2] You can find more information on plotting here: http://matplotlib.org/users/pyplot_tutorial.html

1.5 Exercise 1

The mass of the Sun can be computed from the orbital distance and velocity of any planet from the following equation:

$$M_{Sun} = \frac{v^2 r}{G}$$

Where v is the orbital velocity in m/s, r is the orbital radius in m and G is the gravitational constant $6.67 \times 10^{-11} Nm^2 kg^{-2}$.

The file *planets.csv* has data on the orbital radii and periods of the planets in our solar system. The periods are in Earth years (one Earth year is 3.16×10^7 seconds long) and the radii are in astronomical units (one AU is 1.496×10^{11} meters). To obtain the orbital velocity you need to divide the perimeter of the orbit by the period, in the appropriate units. The perimeter of a circle is $2\pi r$, and you can obtain the value of π from the `numpi.pi` constant.

$$v = \frac{2\pi r}{\tau}$$

Compute the mass of the sun using the data for each planet and then compute the average and the standard deviation of those estimates. You can use the `numpy.mean()` and `numpy.std()` functions for these statistics. Note that this is not the most accurate way of computing the mass of the Sun, but it provides an approximate estimate and the purpose of this exercise is to practice some simple computations with arrays.

Note that you can distribute algebraic operations over all elements of a Numpy array easily. Any operation involving two arrays with the same dimensions will be applied over all corresponding elements [3]. Also, Python accepts numbers written in scientific notation. For example, 6.67e-11 for the gravitational constant. The average result should be a mass of approximately 1.98×10^{30} grams for the Sun, with a standard deviation of 2.95×10^{28}

1.6 Exercise 2

The file polydata.csv has the data for a polynomial fitting example. Fit polynomial curves of degree 3 and 15 and plot each curve superimposed with the data.

[3] This is also possible in some cases where the arrays do not have the same dimensions. See more on broadcasting algebraic operations here, if necessary: `http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html`

Chapter 2

Introductory concepts

Mathematical Background: Linear Algebra, Calculus and Probability for Machine Learning.

2.1 Vectors and Matrices

1. *Pacients, symptoms, and treatments:* Assume you receive a dataset from a hospital with m patients, n symptoms, and p treatments. Your data is organized in two matrices $S \in \mathbb{R}^{m \times n}$, and $T \in \mathbb{R}^{m \times p}$ such that

$$S_{ij} = \begin{cases} 1 & \text{if patient } i \text{ has symptom } j \\ 0 & \text{otherwise} \end{cases}$$

$$T_{ik} = \begin{cases} 1 & \text{if patient } i \text{ was treated with } k \\ 0 & \text{otherwise.} \end{cases}$$

   a) What is the meaning of the second column of S? And what about row 500 of T?

   b) We define the vector of all ones as $\mathbf{1}$ and the transpose of a matrix A as A^T. Describe in plain English the following quantities, and, further, mention dimensions and entrywise expressions:

      i. $S\mathbf{1}$.
      ii. $S^T\mathbf{1}$.
      iii. S^TS.
      iv. SS^T.

   c) Consider matrix $P \in \mathbb{R}^{n \times p}$ where P_{jk} is the total number of patients with symptom j that received treatment k. Express P in matrix notation as a function of matrices S and T.

   d) How would your conclusions change if the encoding of the binary variables changed from $\{0, 1\}$ to $\{-1, 1\}$?

2.2 Calculus

(Exercises from the Dive into Deep Learning book.)

1. *Derivative computation:* Compute the derivative of

   a) $f(x) = 3x$;

   b) $f(x) = x^x$.

2. *Meaning of zero derivatives:* What is the meaning of $f'(x) = 0$ for some x? Give an example of a function f and a location x for which this might hold.

3. *Geometrical interpretation:* In Python, plot the function $y = f(x) = x^3 - \frac{1}{x}$ and plot its tangent line at $x = 1$ and at $x = 2$.

2.3 Probability

(Exercises from the Dive into Deep Learning book.)

1. *Gaming a coin flip in Python:* Suppose that we stumbled upon a real coin for which we did not know the true $P(\text{heads})$. To investigate this quantity with statistical methods, we need to (i) collect some data; and (ii) design an estimator. Data acquisition here is easy; we can toss the coin many times and record all of the outcomes.

   a) Write a function in Python that given a value of $p = P(\text{heads})$ and the number of tosses N returns a vector with the outcomes of the tosses. *Hint: try out* `random.random()`.

   b) As you might have guessed, one natural estimator is the fraction between the number of observed heads by the total number of tosses, e.g., $\hat{p} = \frac{\#n_H}{\#N}$, where $N = n_H + n_T$. Try a $p = 0.6$ and test estimates for $\hat{p}$ while varying N in $\{2, 5, 8, 20, 100, 1000, 10,000\}$.

   c) Each time you run this sampling process, you will receive a new random value that may differ from the previous outcome. Dividing by the number of tosses gives us the frequency of each outcome in our data. Note that these frequencies, like the probabilities that they are intended to estimate, sum to 1. How is $\hat{p}$ varying as you vary N? What is the relation with p?

   d) For what N are you sure you should bet on heads in this coin flip?

2. *Bounds:* Given two events A and B, and their probabilities $P(A)$ and $P(B)$, compute upper and lower bounds on $P(A \cup B)$ and $P(A \cap B)$. *Hint: Graph the situation with Venn diagrams.*

Chapter 3

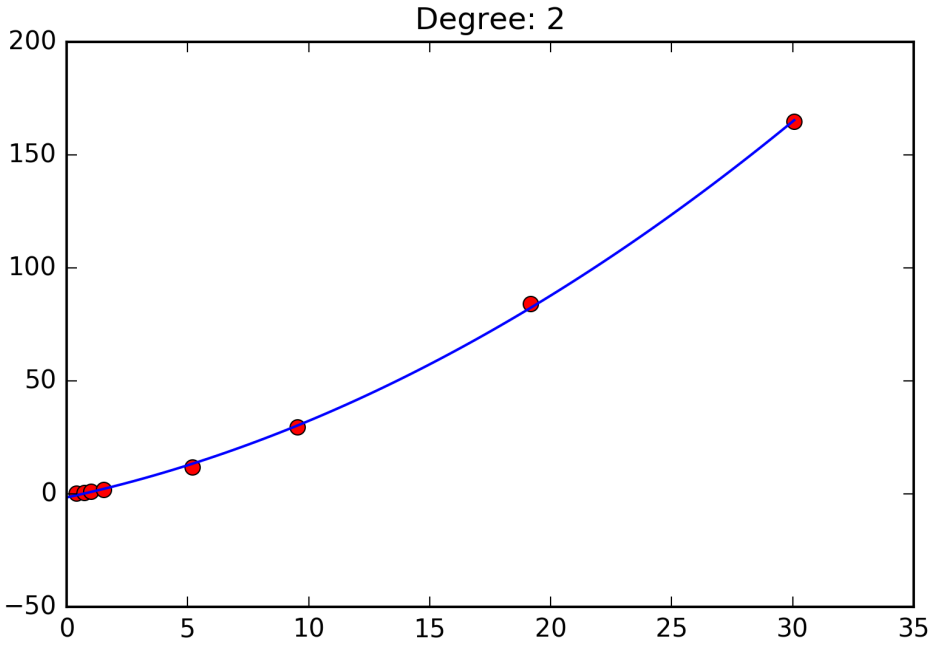
Data Preparation and Linear Regression

Linear regression using polynomials. Training, validation and testing. Selecting the best model by validation on the best hypothesis. Pipelines.

3.1 Fitting a Regression line that is nonlinear on the data but linear on the parameters

Go back to the `planets.csv` dataset from Chapter 1.

Fit a second degree polynomial to the original data, with the radius in AU and the period in years, and plot the data superimposed with the polynomial curve. The result should look like this:



Use the Numpy functions `polyfit` and `polyval`. To see the difference between using numpy and Scikit-Learn run this notebook.

3.2 Linear Regression

Download the bluegills.txt file to your working folder. This file contains data on the size of bluegill fish (in millimetres) of different ages (in years) [1]. You can easily load this data using the `loadtxt()` function from the Numpy library.

Data split

Split it into three sets: half the points for *training*, one quarter for *validation* and one quarter for *testing*. Do not forget to shuffle the data points. Firstly code the split using Numpy, then research the Scikit-Learn `train_test_split` documentation and do an alternate implementation using it.

The *training set* will be used to fit each polynomial curve. The *validation set* will be used to estimate the validation error of each polynomial curve and thus select the best one. The *test set* will be used to obtain an unbiased estimate of the true error of selected curve since selecting curves based on the error measure will make that measure biased. As we are working under the data sampling independence assumption, remember to shuffle the data randomly before splitting it because the order of the data in the file may not be random.

Data standardization

Data standardization is a process of transforming variables in a dataset to a common scale. It is necessary when variables are measured in different scales and have different units of measurement. The purpose of standardization is to eliminate the differences in magnitude among variables, allowing for fair comparisons, analysis, and model learning.

There are several methods to standardize variables. One common approach is standardizing by standard units, also known as z-scores. This method involves subtracting the mean of the variable from each observed value and then dividing by the standard deviation. This transformation results in values expressed in standard units, which indicate how far each value deviates from the mean in terms of standard deviations. Due to how common this transformation is, practitioners frequently refer to it simply as *standardization*.

Another method is range-based standardization, which involves dividing the variable by a range. This range can be the overall range (maximum minus minimum), or a range determined by chosen percentiles such as quartiles. Standardizing by chosen percentiles provides a more robust scale, less sensitive to outliers.

Additionally, variables can be standardized by dividing the values by their mean. This type of scaling results in transformed values with standard deviations equal to their coefficient of variation.

Data standardization helps ensure that variables with different scales and units contribute equally to the analysis. It allows for meaningful comparisons and calculations, particularly in statistical learning methods that consider the variance of variables. By putting variables on a common scale, the dominance of variables with larger magnitudes is mitigated, preventing biased results.

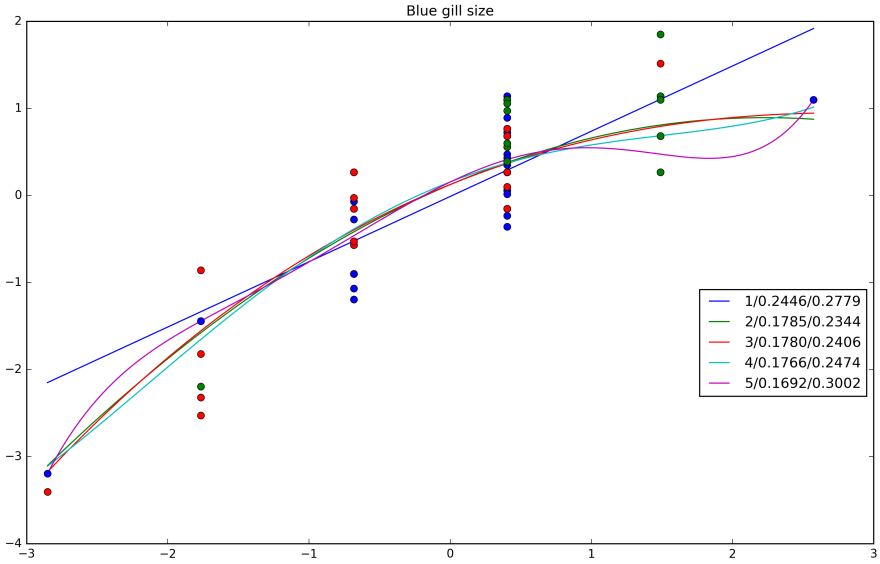
Overall, data standardization is an important step in data analysis to ensure fair comparisons and accurate interpretations of the data.

[1] Data from [1] and https://onlinecourses.science.psu.edu/stat501/node/325

You will begin by transforming features to z-scores. Check the online documentation for the Numpy functions you will use (`loadtxt()`, `std()`, `mean()`) and implement the transformation. Now check the Scikit-Learn library documentation for the `StandardScaler` class and use this class in your code.

Model training

Polynomial degree choice Now select the best polynomial curve from polynomials of degree 1 through 6. Your implementation should report the best degree for the polynomial curve and also the test error of the best model. Also, do the plot of the different polynomials. It should look something like this, with different colors for the training, validation and test sets. The labels show the degree and the training and validation errors.



To plot several lines in the same chart, you can create the figure, plot the data, and then plot each line before calling the `plt.show()` function. To add a legend, see the tutorial on `http://matplotlib.org/users/legend_guide.html`. For formatting a string, you can use the string `format()` method. There are many examples online. See, for example, `http://www.python-course.eu/python3_formatted_output.php`.

Industry best practices: using pipelines

Explore the documentation of the Scikit-Learn Pipeline class and understand the benefits of using it.

Why use pipelines in the machine learning workflow?

Pipelines are a powerful tool in the machine learning workflow for several reasons:

1. Code organization and reproducibility: Pipelines allow you to organize your code in a modular and reusable way. By defining a sequence of data preprocessing steps and model training steps, you can easily reproduce the entire workflow and apply it to new datasets.

2. Data leakage prevention: Data leakage occurs when information from the test set is inadvertently used during the model training process, leading to overly optimistic performance estimates.

Pipelines help prevent data leakage by ensuring that all preprocessing steps are applied only to the training data, and any transformations are learned from the training set and applied consistently to the test set.

3. Efficient hyperparameter tuning: Pipelines enable you to optimize the hyperparameters of your model in a more systematic and efficient manner. By encapsulating all preprocessing steps and model training within a pipeline, you can perform cross-validation and grid search on the entire pipeline, exploring different hyperparameter combinations and selecting the best model.

4. Scalability and deployment: Pipelines facilitate the deployment of machine learning models into production systems. Once a pipeline is defined and trained, it can be easily serialized and deployed as a single entity. This simplifies the process of integrating the model into a production environment, making it more scalable and maintainable.

How to use pipelines in the machine learning workflow?

To use pipelines in the machine learning workflow, follow these steps:

1. Data preprocessing: Define the sequence of preprocessing steps, such as feature scaling, one-hot encoding, or imputation of missing values. Each step should be encapsulated within a transformer object.

2. Model training: Define the machine learning model that will be used for training, such as a regression modele. This model should be encapsulated within an estimator object, like `LinearRegression`.

3. Pipeline construction: Build the pipeline by combining the preprocessing steps and the model training step. Specify the order in which the steps should be applied.

4. Hyperparameter tuning: Use techniques like cross-validation and grid search to optimize the hyperparameters of the pipeline. This involves specifying the range of hyperparameter values to explore and selecting the best combination based on performance metrics.

5. Model evaluation and deployment: Evaluate the performance of the pipeline on a holdout test set. If satisfied with the performance, serialize the pipeline and deploy it in a production environment for real-time predictions.

By using pipelines, you can streamline and automate the entire machine learning workflow, from data preprocessing to model training and deployment. This not only improves efficiency and reproducibility but also helps ensure the integrity and reliability of your machine learning models.

Refactor your code using a pipeline. You are now ready to implement your regressor using a pipeline.

3.3 Reflection questions

After these exercises, you should be able to answer the following questions:

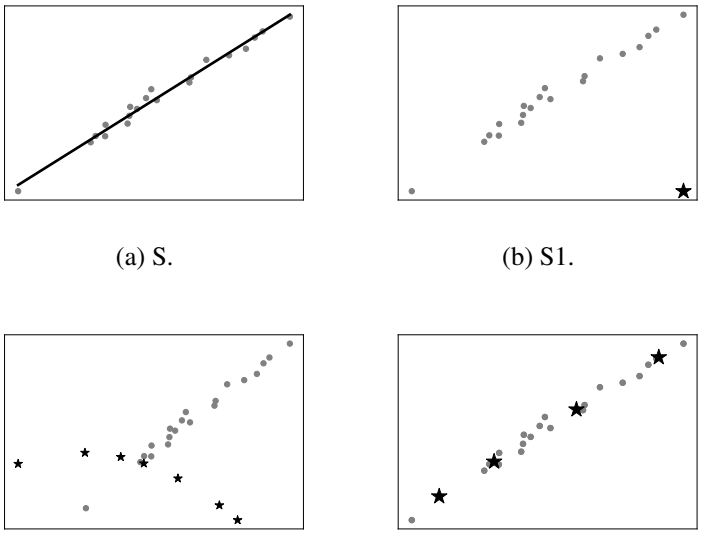
- What is the difference between the training error, the test error and the true error of a model?

- Why can we not use the training error to estimate the true error?
- Why do we need the test error? Why not use the validation error to estimate the true error?
- Explain what is the data leakage problem, and how can it harm your results.

3.4 Test questions

To be answered with pen and paper.

Regression: Linear regression allows approximating the variable to be predicted in a data set. Figure 3.1a shows the optimal line in black fitted to the dataset S represented with gray dots. Observe in Figure 3.1 three duplicates of dataset S still as gray dots, where new data points (stars) were added to S to form the three new S1, S2, and S3.



(a) S. (b) S1.

(c) S2. (d) S3.

Figure 3.1: Original dataset S and new the datasets S1, S2, and S3.

For each of the changed datasets, S1, S2, and S3 indicate whether the model would be different and, if so, how would this difference change the model parameters? Sketch the new regression line on the plots (b), (c), and (d) and justify your answer below.

1. Dataset S1: A new point is added to S in the lower right corner.
2. Dataset S2: New points (stars) added to S.
3. Dataset S3: New points exactly on the line of Fig. 3.1a.

Regression: Linear regression allows approximating the variable to be predicted in a data set. Consider the training dataset in the following Table, where x is the feature value, and y is the regression label.

x	y
1	0
2	1

Consider that the regressor has the form $y = \beta_0 + \beta_1 x$, where β_0 and β_1 are the parameters of the model.

1. Draw the points and the regression line.
2. Compute the optimal β_0 and β_1.
3. What will be the mean squared error of the model, if evaluated in the following test points?

x	y
0	0
-1	-1

4. Compute the prediction y for a point $x = 0.5$.

Chapter 4

Linear Regression, over and underfitting, regularization

4.1 Exercise: Linear Regression, Deterministic and Probabilistic perspectives

Let us assume that our data model is

$$y = f(x) + \epsilon \tag{4.1}$$

where $f(x)$ is the Mother Nature process we want to predict given feature x, and ϵ an independent identically distributed (i.i.d.) zero-mean Gaussian random variable with standard deviation σ, representing measurement noise or unmodelled effects on y. In our example, y can be the fish size and x can be the fish age.

1. Explain why is the fish size in (4.1) not deterministic, given the age.

2. If the noise has zero mean, $\mathbb{E}[\epsilon] = 0$ then what is the mean of y? *Hint: The expectation operator $\mathbb{E}$ is linear, and thus, if β is a constant and X, Y are two random variables, then $\mathbb{E}[\beta X + Y] = \beta\mathbb{E}[X] + \mathbb{E}[Y]$.*

3. What is the variance of y?

4. A Gaussian distribution with mean μ and variance σ^2 has as density

$$p(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\frac{(z-\mu)^2}{\sigma^2}}.$$

   Write the density for one data point, y_n considering your answers in the previous questions of this exercise.

5. Assume that you collected your fish dataset in a random, independent way. Then the full distribution of your data can be factorized. Write the full joint distribution for all your data $\{x_n, y_n\}_{n=1}^N$.

6. Now, we assume our linear model $M_\alpha(x) = \alpha^T \tilde{x}$, where $\tilde{x} = \begin{bmatrix} x \\ 1 \end{bmatrix}$, approximates well $\mathbb{E}[f(x)]$. Write the maximum likelihood problem using $M_\alpha(x)$ as an hypothesis class for our data.

7. If you compose any function with a monotonic one, the maximizer will not change (why?). So, you can compose the likelihood derived in the last exercise with the natural logarithm. Write the expression of the log-likelihood of your data considering the model $M_\alpha(x)$.

8. Take the problem of maximizing the log likelihood and transform it into

$$\underset{\alpha}{\text{minimize}} \frac{1}{2} \sum_{n=1}^{N} (y_n - M_\alpha(x_n))^2.$$

What is the relation with the deterministic data fitting linear regression?

4.2 Exercise: smaller data set

Repeat the exercise in Section 3.2, now with the yield.txt data set. This is a hypothetical result from an experiment giving different yields as a function of temperature. Note that this data set is much smaller (only 15 points). Run your script several times and note how the results vary significantly as a function of the random attribution of data points to training, validation or test sets. This is an important issue to bear in mind when dealing with real data.

4.3 Overfitting and underfitting

Underfitting occurs when a machine learning model is too simple or lacks the capacity to capture the underlying patterns in the data. It results in poor performance both on the training data and new, unseen data. An underfit model oversimplifies the relationships between the features and the target variable, leading to high bias and low variance.

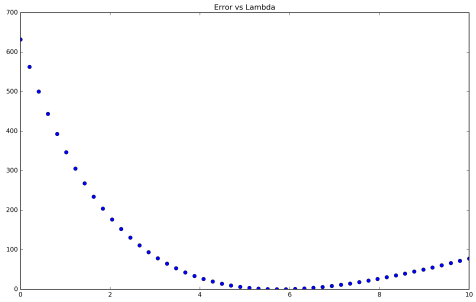
Overfitting, on the other hand, happens when a machine learning model is overly complex and captures noise or random fluctuations in the training data. The model becomes too tailored to the training data and fails to generalize well to new data. Overfitting is characterized by low bias but high variance.

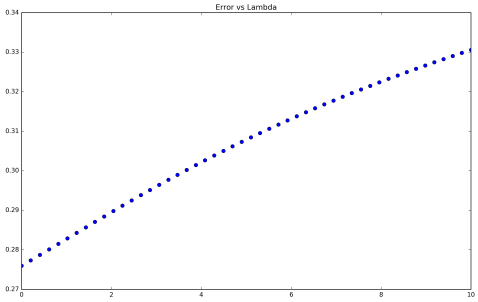
Both underfitting and overfitting are undesirable as they result in poor predictive performance. The goal is to find the right balance between model simplicity and complexity to achieve optimal performance on unseen data.

Follow the tutorial on over and undefitting here.

4.4 Ridge regression

Repeat the bluegill regression but using Ridge regression expanding the original data to degree 5. Explore λ values from 0.001 through 10, on a logarithmic scale. Note that the validation error will depend very much on the attribution of data points to training or validation sets. Run your code several times to see how validation error varies with λ. Here are two examples of possible results:





These examples illustrate how sampling and selection of data for training and validation can have a large effect on the results.

4.5 Interpretation of the linear model coefficients

Follow this tutorial.

Chapter 5

Probabilistic ML, Cross-validation, Optimization

Probabilistic Machine Learning, Model selection with cross-validation, Optimization with Gradient Descent.

5.1 Exercise: Probability Mass Function versus Probability Distribution

Assume you have a bag with 25 balls numbered 1 to 9. In Fig. 5.1 it is possible to see the number of balls in the bag with each of the number from 1 to 9.

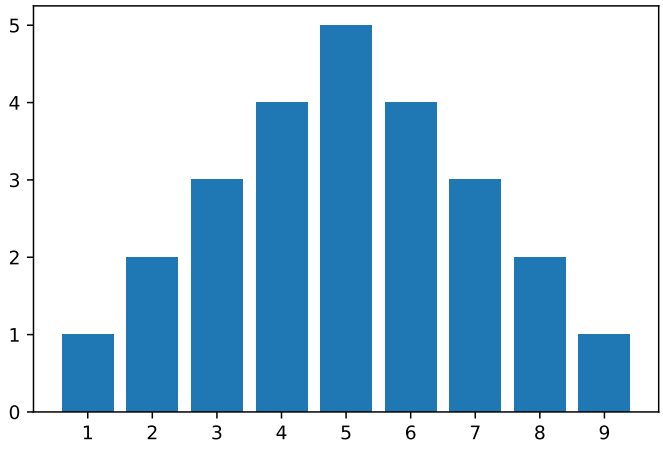


Figure 5.1: Frequency histogram with the number of balls in the bag with each number.

To compute, for example, the probability of drawing number 5 by dividing the number of balls with the number 5, n_5, by the total of balls in the bag, N. So, the probability of pulling the number 5 is $\mathbb{P}(x = 5) = \frac{n_5}{N} = \frac{5}{25} = 0.2$. Applying this to all numbers, we can get the probability of drawing each number, as we can see in Fig. 5.2

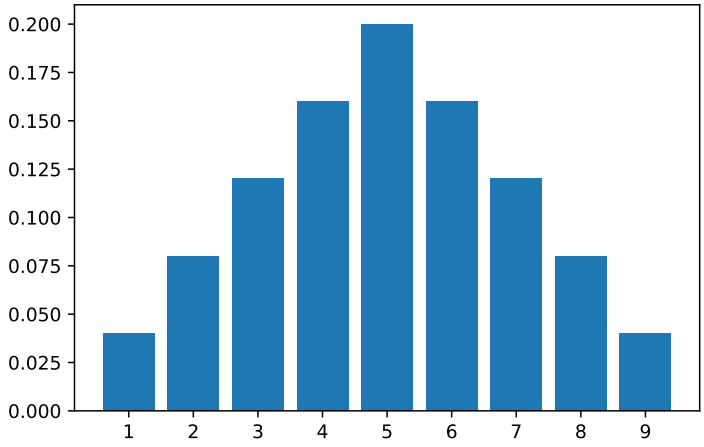


Figure 5.2: Discrete Probability Mass Function

1. What is the probability of drawing a number between 1 and 3? Compute the probability assuming you only have access to the probability of drawing each number. *Hint: Start by computing* $\mathbb{P}(x = 1)$, $\mathbb{P}(x = 2)$ *and* $\mathbb{P}(x = 3)$.

2. Assuming you only have access to the probability of drawing each number, present an expression to calculate the probability of pulling a number between any i and j such that $i \geq 1$, $j \leq 9$ and $i < j$.

3. Now, we go from the discrete case to the continuous case. Instead of being able to choose an integer between 1 and 9, assume that we can choose any *real* number between 1 and 9. For this case, we see that, following the previous idea, the probability of choosing any specific number is zero, given the infinity of number between 1 and 9:

$$\mathbb{P}(x = k) = \lim_{N \to +\infty} \frac{n_k}{N} = 0. \tag{5.1}$$

   However, given the probability density function, $p(x)$, it is possible to calculate the probability of pulling a number between i and j. Based on the same idea of exercise 2, present an expression to compute this probability value for the continuous case. *Hint: Remember that for summing over continuous variables we have to use integration.*

4. In Fig. 5.3, we can see a probability density function with a similar shape as in Fig. 5.2, but with real-valued support, such that the density function is that of a random variable following a normal distribution $p(x) = \mathcal{N}(x; \mu, \sigma^2)$ with mean value $\mu = 5$ and standard deviation $\sigma = 2$ (see (5.2)).

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\} \tag{5.2}$$

   With the expression obtained before, compute the probability of randomly sampling a number between 1 and 3, i.e., $\mathbb{P}(1 \leq x \leq 3)$. To do this, write the probability in terms of Comulative Density Functions, and then use the primitive `scipy.stats.norm.cdf`.

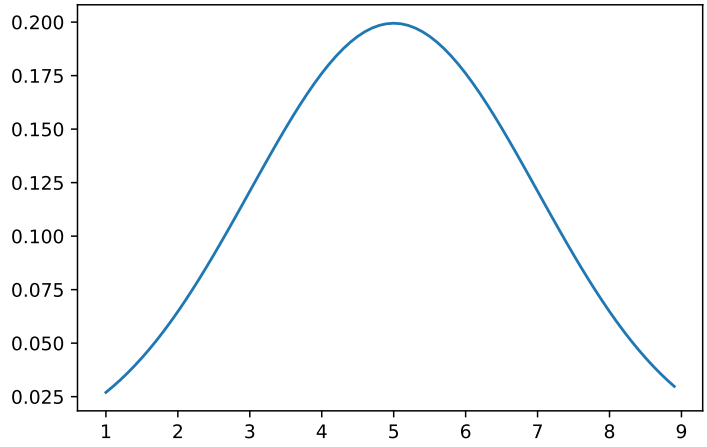


Figure 5.3: Continuous Probability Distribution

5.2 Cross-Validation

Cross-validation is a technique used in machine learning and statistical modeling to evaluate the performance of a predictive model. It is particularly useful for ML engineers and Data Scientists to understand this concept.

In cross-validation, the available dataset is divided into multiple subsets or "folds." The model is trained on a portion of the data and then tested on the remaining fold. This process is repeated multiple times, with each fold serving as the test set. The performance metrics, such as accuracy or error rate, are then averaged across all the folds to give an overall assessment of the model's performance.

By using cross-validation, we can assess how well the model generalizes to unseen data and avoid overfitting, when data is scarce.

Cross-validation and validation are both techniques used to evaluate the performance of predictive models, but they differ in how they utilize the available data.

Validation is typically performed by splitting the dataset into two parts: a training set and a validation set. The model is trained on the training set and then evaluated on the validation set. This allows us to assess how well the model performs on unseen data from the same distribution as the training data.

On the other hand, cross-validation goes a step further by partitioning the dataset into multiple folds or subsets. The model is trained on a combination of these folds and evaluated on the remaining fold. This process is repeated multiple times, with each fold serving as the test set once. The performance metrics are then averaged across all the folds to provide a more robust assessment of the model's performance.

Thus, while validation involves a single split of the dataset into training and validation sets, cross-validation performs multiple iterations of training and testing on different subsets of the data. Cross-validation provides a more comprehensive evaluation of the model's performance and helps to mitigate the potential bias or variance issues that can arise from a single validation split.

Cross validation for estimating the generalization error

To estimate the generalization error using cross-validation, the following steps can be followed:

1. Split the available dataset into k subsets or folds.
2. For each fold, designate it as the test set and combine the remaining k-1 folds to form the training set.
3. Train the model on the training set and evaluate its performance on the test set.
4. Repeat steps 2 and 3 for each fold, using a different fold as the test set each time.
5. Calculate the performance metric (e.g., accuracy, error rate) for each fold.
6. Average the performance metrics across all the folds to obtain an overall estimate of the model's generalization error.

By repeating this process k times, with each fold serving as the test set once, we can obtain a more reliable estimate of how well the model will perform on unseen data.

It is important to note that the choice of k, the number of folds, can vary depending on the available dataset and computational resources. Common choices include 5-fold, 10-fold, or even leave-one-out cross-validation (where k equals the number of samples in the dataset).

By estimating the generalization error through cross-validation, we can assess how well the model will perform on unseen data, helping us make more informed decisions about model selection and hyperparameter tuning.

Cross validation for model selection and hyperparamenter tuning

By using cross-validation for model selection, we can make more informed decisions about which model to choose based on its performance across different subsets of the data. This helps us avoid selecting a model that performs well on one specific validation split but may not generalize well to unseen data.

To perform cross-validation for model selection, the following steps can be followed:

1. Split the available dataset into k subsets or folds.
2. For each fold, designate it as the validation set and combine the remaining k-1 folds to form the training set.
3. Train a set of candidate models on the training set.
4. Evaluate the performance of each candidate model on the validation set.
5. Repeat steps 2-4 for each fold, using a different fold as the validation set each time.
6. Calculate the performance metric (e.g., accuracy, error rate) for each candidate model on each fold.
7. Average the performance metrics across all the folds for each candidate model.
8. Select the model with the best average performance as the final model.

By repeating this process k times, with each fold serving as the validation set once, we can obtain a more reliable estimate of how well each candidate model will perform on unseen data. This helps us select the model that is likely to generalize well to new data.

It is important to note that the choice of k, the number of folds, can vary depending on the available dataset and computational resources. Common choices include 5-fold, 10-fold, or even leave-one-out cross-validation (where k equals the number of samples in the dataset).

5.3 Exercise: Model selection with cross-validation

Read the excerpt of the Learning from Data book indicated on the course page.

Cross-validation is especially useful if you do not have too much data. This exercise will show how to deal with those situations.

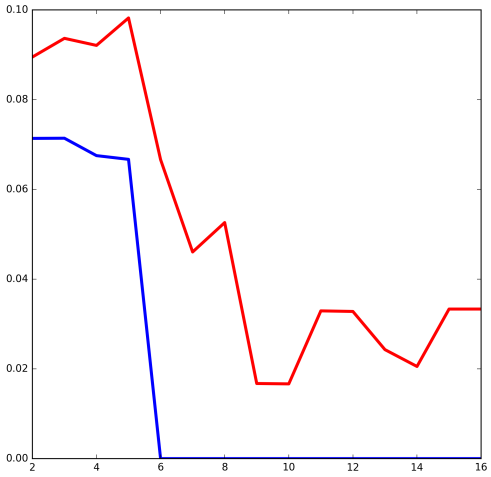
Download the data.txt file to your working folder. This file contains a synthetic data set, Optionally, download the t3_aux.py file with auxiliary functions for this exercise, which you can import into your own module (`poly_16features(X)` and `create_plot`).

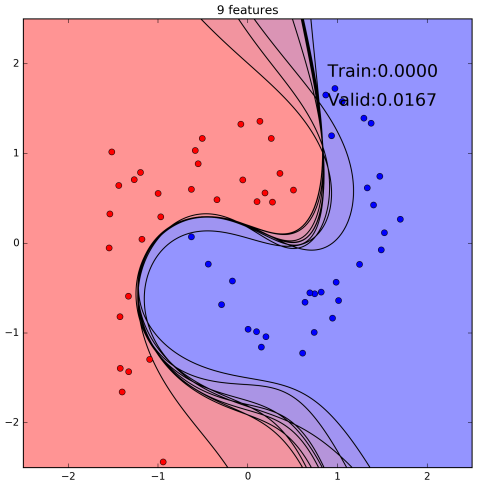
Load the data, split it into the training and test sets, and z-score it. Note that, in regression, as we did in the previous tutorial, we also standardize the values to predict so that the results are more reproducible and to avoid numerical instability. In classification, however, the values to predict are class labels and it makes no sense to standardize those. So apply standardization only to the feature vectors. Then expand the data to 16 features:

1. x_1
2. x_2
3. $x_1 \times x_2$
4. x_1^2
5. x_2^2
6. x_1^3
7. x_2^3
8. $x_1^2 \times x_2$
9. $x_1 \times x_2^2$
10. x_1^4
11. x_2^4
12. $x_1^3 \times x_2$
13. $x_1 \times x_2^3$
14. $x_1^2 \times x_2^2$
15. x_1^5
16. x_2^5

Use one third of the data for the test set. The training set will be used to select the number of features to use by cross-validation. Remember to shuffle the data randomly before splitting it because the order of the data in the file may not be random. Use stratified sampling for splitting the data and for the cross-validation folds. See the lecture materials for more details.

Now select the best logistic regression model by 10-fold cross-validation. You can use the code samples on the lecture materials but make sure you understand the different steps. Plot the training and cross-validation error for the different models to select the best one. If several models have very similar cross-validation errors, prefer a model with fewer features. This is an example of what the results may be, but note that your actual result may be different due to the random assignment of points to training and test sets. The figures below show the error plot and the plot for the nine features model, which appears to be the best one in this case.



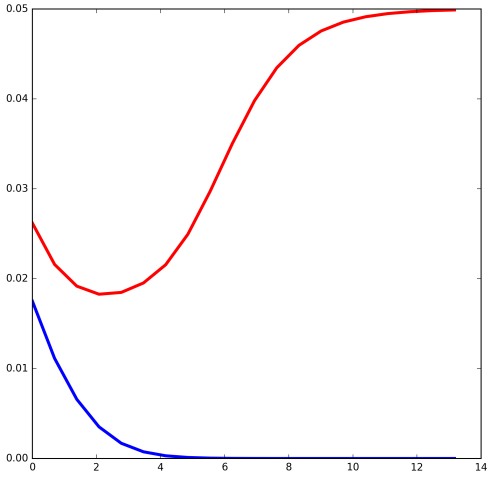


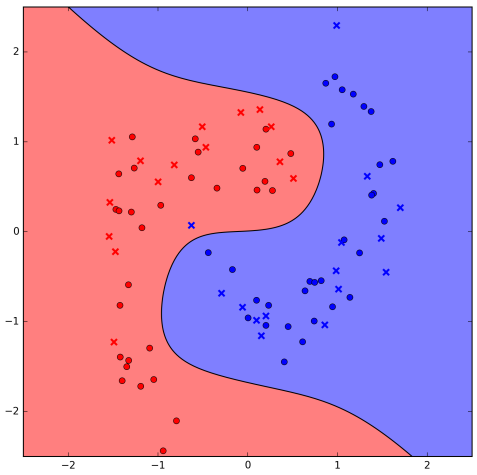
Try running your code a few times to get a feeling for how the results vary depending on the assignment of points to training and testing sets. Note that this is often the case with small data sets, although the effect is less pronounced the more data there is available.

Once you select the correct number of features, train a logistic regression with that number of features using the full training set and measure the test error.

5.4 Exercise: Regularization with Cross Validation

Use cross validation to find the best C value for regularization of the 16-features model. Start with $C = 1.0$ and double the value of C at each iteration for 20 iterations, reporting the C value and the cross-validation error for each iteration. The figure below shows the plot of the training and cross-validation error as a function of the logarithm of C and the hypothesis trained with the best C value.





5.5 Exercise: Optimization

Remember the Linear Regression problem

$$\underset{\alpha}{\text{minimize}} \|X\alpha - y\|^2.$$

1. Write the gradient of the loss.
2. Write the gradient descent iteration.
3. Implement the gradient descent algorithm in Python.
4. Using the `make_regression` method in Scikit-Learn run your learning algorithm and find one value for the learning rate where the algorithm converges to the optimum, and one value where it diverges.
5. Implement the Batch Gradient Descent and the Stochastic Gradient Descent algorithms and compare them and the vanilla Gradient Descent regarding the performance in time per gradient step, overall wall time, and iteration count, for the same accuracy in the solution. Systematize your findings in a table and emphasize the best performing algorithm for each paramenter. Plot the iteration progress of the loss for the three optimization algorithms.

5.6 Questions

After these exercises, answer the following questions:

- Why does cross-validation give us a more reliable estimate of the true error of the model?
- What is the purpose of stratified sampling for splitting the data into the folds, training and test sets?
- How does regularization work on logistic regression?

5.7 Extra material: Plotting contours

This is not part of the course subject matter, but if you want to know how to plot the contours for the classifier, here is a code example. Note that this is only useful for two-dimensional data so, in practice, you will seldom have use for this sort of plots. The plot is done using the contour plotting functions from the `pyplot` module, computing the classification for all points in a two-dimensional grid. The `poly_16features` function is the function that expands the original data into the 16 features.

```
def poly_mat(reg,X_data,feats,ax_lims):
    """create score matrix for contour
    """
    Z = np.zeros((200,200))
    xs = np.linspace(ax_lims[0],ax_lims[1],200)
    ys = np.linspace(ax_lims[2],ax_lims[3],200)
    X,Y = np.meshgrid(xs,ys)
    points = np.zeros((200,2))
    points[:,0] = xs
```

```
    for ix in range(len(ys)):
        points[:,1] = ys[ix]
        x_points=poly_16features(points)[:,:feats]
        Z[ix,:] = reg.decision_function(x_points)
    return (X,Y,Z)

def create_plot(X_r, Y_r, X_t, Y_t, feats, best_c):
    """create imege with plot for best classifier"""
    ax_lims=(-3,3,-3,3)
    plt.figure(figsize=(8,8), frameon=False)
    plt.axis(ax_lims)
    reg = LogisticRegression(C=best_c, tol=1e-10)
    reg.fit(X_r,Y_r)
    plotX,plotY,Z = poly_mat(reg,X_r,16,ax_lims)
    plt.contourf(plotX,plotY,Z,[-1e16,0,1e16], colors = ('b', 'r'),alpha=0.5)
    plt.contour(plotX,plotY,Z,[0], colors = ('k'))
    plt.plot(X_r[Y_r>0,0],X_r[Y_r>0,1],'or')
    plt.plot(X_r[Y_r<=0,0],X_r[Y_r<=0,1],'ob')
    plt.plot(X_t[Y_t>0,0],X_t[Y_t>0,1],'xr',mew=2)
    plt.plot(X_t[Y_t<=0,0],X_t[Y_t<=0,1],'xb',mew=2)
    plt.savefig('final_plot.png', dpi=300)
    plt.close()
```

Chapter 6

Classification

6.1 Exercise: Maximum Likelihood binary classification

Logistic regression in a discriminative classification model $p(y|x;\theta)$, where $x \in \mathbb{R}^d$ is the feature vector, $y \in \{0,1\}$ is the binary class label, and $\theta = (w, b)$ are the model parameters.

In binary logistic regression, the model is defined as

$$p(y|x;\theta) = \text{Bernoulli}(y|\sigma(w^T x + b)),$$

where the sigmoid function is $\sigma(z) = 1/(1+\exp(-z))$. The value of $z = w^T x + b = \log\left(\frac{p}{1-p}\right)$ is called the log-odds, logit or pre-activation. Note that p is the probability $p(y = 1|x;\theta)$.

1. Write the expressions for the probability of the example belonging to the positive class ($y = 1$) and the negative class ($y = 0$), as functions of the model paramenters and the feature vector. Write overall $p(y|x;\theta)$. *Hint: lookup the Bernoulli distribution on Wikipedia.*

2. If the misclassification loss is equal in both classes, then the optimal decision rule is to predict $y = 1$ if and only if the positive class is more likely than the negative class. The prediction function is the indicator function[1] of this statement: $f_\theta(x) = \mathbb{I}(p(y = 1|x;\theta) > p(y = 0|x;\theta))$. Write $f_\theta(x)$ as a function of the logits z and then of w, x, b, y.

3. The prediction function induces an optimal decision boundary. For $w^T = [1, 1]$ and $b = 0.5$, draw by hand the decision boundary in the feature space (x_1, x_2).

4. If we can perfectly separate the examples with this boundary, then the data is linearly separable. In the general case this is not so, and we can use the label probability to assess the confidence of the model on the predictions. Plot in 3D using Python $p(y = 1|x; w, b)$ for diferent values of $w = (w_1, w_2)$. How does w influence the confidence in the predictions?

[1] The indicator function is defined as $\mathbb{I}(c) = \begin{cases} 1 & \text{if } c \\ 0 & \text{otherwise} \end{cases}$

5. *Kernelization and feature engineering:* As in the regression case we can consider a nonlinear transformation of the feature vector to accomodate nonlinearity. Consider the transformed features $\phi(x) = (1, x_1^2, x_2^2)$ for $x \in \mathbb{R}^2$. The bias term is absorbed in ϕ. Draw by hand the decision boundary when $w = (-R^2, 1, 1)$, for some $R > 0$ of your choice.

6. Assume that the bias term is absorbed in w. Write the likelihood function for the full training dataset $p(y_1, \cdots, y_N | x_1, \cdots, x_N; w)$, considering that the data is independent and identically distributed (i.i.d.).

7. Write the non-negative loglikelihood loss for the training dataset.

8. Binary cross entropy is defined as

$$H(p, q) = -\sum_{x in \mathcal{X}} p(x) \log q(x)$$

   and it measures how two probabilities p and q differ from each other, in the same support $\mathcal{X}$ (what happens if $q(x) = p(x)$ for all x? and if they are very different?) Write the non-negative log likelihood as a function of the binary cross entropy.

9. Imagine you now choose a different encoding of the labels, $\tilde{y} = \{-1, 1\}$. Compute the probabilities $p(\tilde{y} = 1|x; w)$ and $p(\tilde{y} = 0|x; w)$.

10. Write the non-negative loglikelihood for $\tilde{y}$.

6.2 Exercise: multiclass classification

Consider now that the class label is defined as $y \in \{1, \cdots, C\}$ where C is the number of classes, and assume that one example has only one true label. In this case, the discriminative classification model is

$$p(y|x; \theta) = \text{Categorical}(y|\text{softmax}(Wx + b)),$$

where W is a $C \times d$ weight matrix, $b \in \mathbb{R}^C$, and $\theta = (W, b)$ are the parameters. The softmax function is defined as

$$[\text{softmax}(z)]_i = \frac{e^{z_i}}{\sum_{i=1}^{d} e^{z_i}}.$$

The Categorical probability mass function applied to a one-hot encoded label y is defined as $p(y|\pi) = \prod_{i=1}^{C} \pi_i^{y_i}$. Again we will assume b is absorbed into W.

1. Let the logits be $z = Wx$. Write the expression for $p(y = c|x; W)$, considering c the one-hot encoded label of class c.

2. Write the non-negative loglikelihood with respect to W.

3. Consider the task of image tagging, where a single image can be classified with more than one label. The general problem is called multi-label classification. How does this change the encoding of the labels, and the model formulated for the multi-class scenario above?

6.3 Multiclass classification: One-vs-All (OVA)

From the T5_plot.py import the `plot_logregs` function. This is an auxiliary function for plotting the combination of three logistic regression classifiers you will train for a one-versus-all (aka one-vs-rest) classification of the Iris data set. To load the Iris data set, import the datasets module from the ScikitLearn library and create your data matrices with only the sepal length and width attributes:

```
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data[:,[0,1]]
Y = iris.target
```

This way, the `X` variable will be a matrix with the two selected attributes and the `Y` variable will contain the class labels for the three variants of flowers (*Setosa*, *Versicolor* and *Virginica*).

Create a list of three `LogisticRegression` classifiers with low regularization (use a large value for C) and train each classifier so that class 0 corresponds to one of the three flower variants and class 1 corresponds to the other two. This is easy to do with a loop over the three class labels (0, 1, and 2), and in each iteration creating a class vector with values of 0 for all examples with the class corresponding to the iteration and values of 1 for examples with a different class. This should give you 3 logistic regression classifiers trained to distinguish each class from the remainder.

Plot the Iris data and the classifications of your 3 classifier using the one-vs-all algorithm. This is implemented on the `plot_logregs`, which simply requires the list of trained classifiers as argument and creates a `iris-logistic.png` file with the result. Look at the `plot_logregs` function to see how this is done. Note in particular this part:

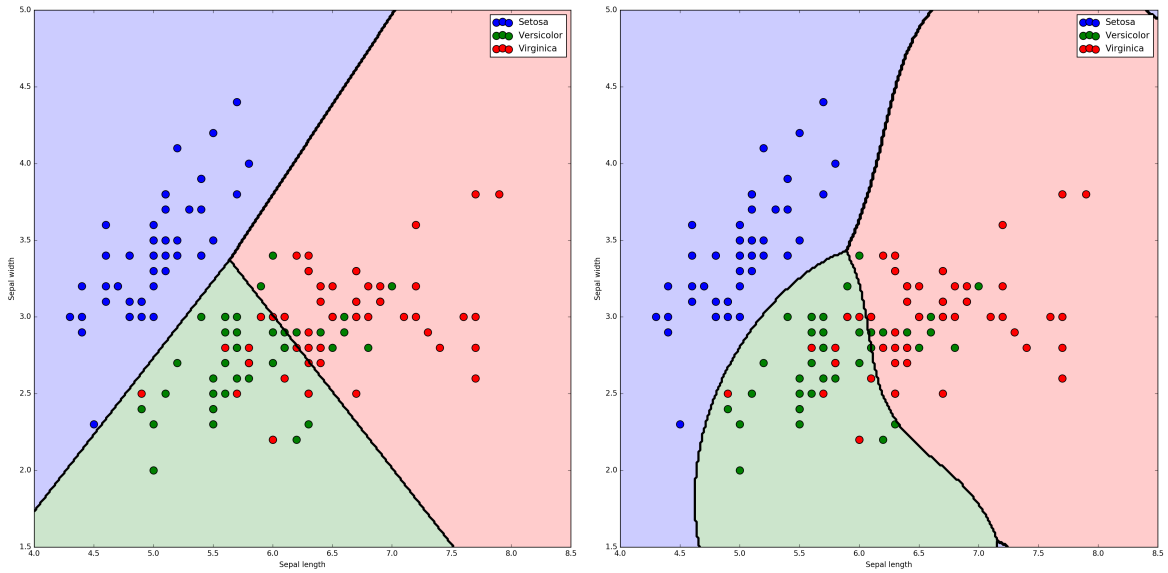
```
def plot_logregs(logregs,file_name='iris-logistic.png'):
    ...
    max_z = np.max(Zs,axis=0)
    Z = np.zeros(Zs[0,:,:].shape)
    for ix in range(3):
        tmp = Zs[ix,:,:]
        Z[tmp==max_z] = ix
```

Noting that the Z values are the probabilities of the points being in class 0, try to understand how the one-vs-all classification was implemented here.

Repeat this exercise using the `OneVsRestClassifier` class provided in the `sklearn.multiclass` module. You can create a one-vs-all classifier object by providing one classifier object to use for each classification. For example, to use support vector machines with a Gaussian kernel with a γ of 0.7 and a C value of 10 for regularization:

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
ovr = OneVsRestClassifier(SVC(kernel='rbf', gamma=0.7, C=10))
```

Import the `plot_ovr` function from the `T5_plot` module and use it to plot your classification using this one-vs-rest classifier, after training it with the Iris data set, to output the result to the `iris-ovr.png` file. The two plots should look like this:



6.4 Questions

After these exercises, you should be able to answer the following question:

- What is the one-vs-rest multiclass classification algorithm?

Chapter 7

Support Vector Machines

Introduction to Support Vector Machines. Identifying support vectors. Soft margins and regularization. Optimizing kernel parameters with cross-validation. Comparing classifiers.

Download the T4data.txt and the T4aux.py file to your working folder. The first file contains a synthetic data set for the exercises. The second file contains an auxiliary plotting function. This exercise covers lectures on SVM (Lecture 6), and on comparing classifiers (Lecture 5).

7.1 Exercise: Support Vector Machine and Support Vectors

In this exercise you will train a Support Vector Machine to classify the data set and examine the support vectors. Load the data and standardize it. Since this exercise will not require any selection or error estimation, do not split the data into a training and test sets. Use all data for training the `SVC` classifier. Note that, before standardizing, you will need to separate the features from the class column. The class is the last column on the data file. Fit a linear SVM to your data. To do this, use the `sklearn.svm.SVC` class with a linear kernel (with the `kernel='linear'` argument). Check the documentation for the details on the parameters and how to use the classifier[1]. After fitting, plot the data, discriminant and margins using the `plot_svm` function provided in the `T4aux.py` module. The arguments for this function are the data matrix (with the 1 and -1 classes in the last column; do not separate the features from the classes), the pre-trained SVM classifier object, the file name for the image and the value of C. Compare the results using C values of 0.1, 1 and 10. This should be the result: The `SVC` object, after fitting the classifier, includes the following useful attributers:

- `support_`, with the indexes of the support vectors in the data used for fitting and
- `dual_coef_`, with the product of the alpha values by class labels (-1 and 1) for the support vectors (the coefficients for the dual problem).

Using these attributes you can obtain the alpha values from the absolute value of the `dual_coef_` values. With this information from the support vectors and the plots, answer the following:

1. How many support vectors are there for each class (-1 and 1), for each value of C?

[1] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

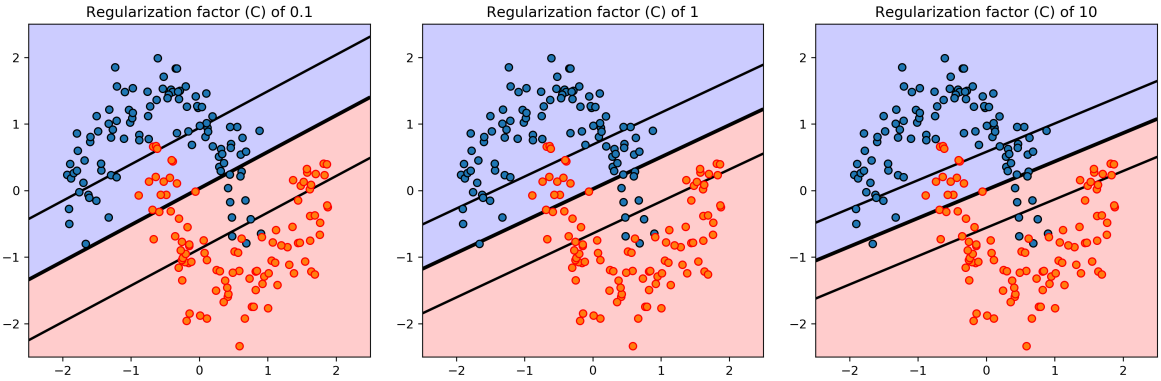


Figure 7.1: In these images, the thick black line is the discriminant and the thin lines are the margins.

2. How many of those support vectors correspond to examples at the margin or examples that violate the margin constraint?

3. Explain the variation of the margins with the value of C.

4. Explain the variation in the number of support vectors that violate the margin constraint as you vary the C value.

7.2 Exercise: Kernel trick

Load the data (`T4data.txt`), standardize it and split it one half for training and one half for test. Keep the test set for the final exercise and use the training set train SVM classifiers with C=1 and using the following kernels:

1. Polynomial, $K(x, y) = (\gamma x^T y + r)^d$, with degree 3, gamma of 0.5 and r of 0 (the r value is set in the `coef0` parameter and is 0 by default).

2. Sigmoid, $K(x, y) = tanh(\gamma x^T y + r)$, with gamma of 0.5 and r of -2.

3. Gaussian RBF, $K(x, y) = exp(-\gamma||x - y||^2)$, with gamma of 0.5.

Modify the plot function provided to plot a black cross (`'xk'`) on every support vector that violates the margin constraint and a black dot (`'.k'`) for every support vector in the margin. The images should look like Figure 7.2. Explain why the number of support vectors on the margin differ between the different kernels.

Exercise 2, optional

Using a value of 0 for the r parameter in the polynomial kernel is not a good idea in this case. Repeat the exercise above for the polynomial kernel but set $r = 1$ using the `coef0` argumment of the `SVC` class.

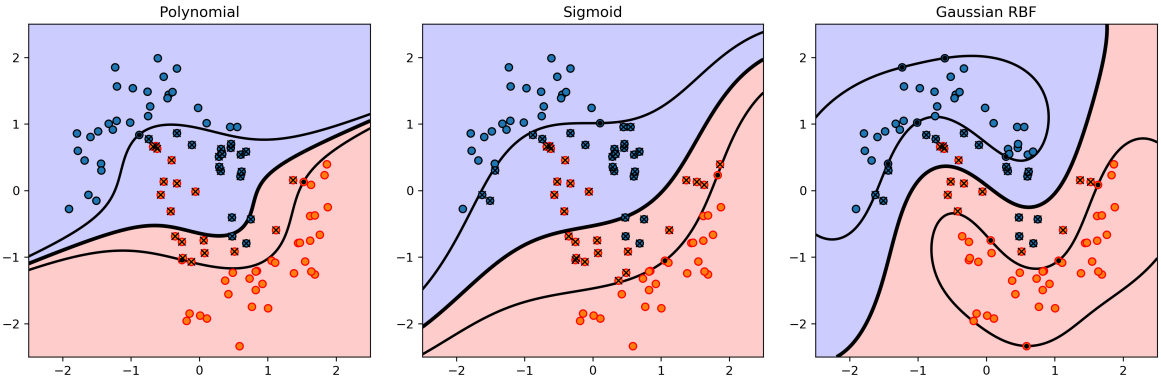


Figure 7.2: Comparing three different kernels. The thick black line is the discriminant, the thin lines are the margins; support vectors are marked with crosses (inside margins) or dots (on the margins)

7.3 Exercise 3: Optimize kernel parameters and compare classifiers

Use the training set from the previous exercise to optimize, with cross validation, the regularization parameter C for each of the three kernels above. Keep the remaining parameters constant, changing only the value of C, starting from 0.1 and doubling at each generation until greater than 10000. Plot the training and validation errors against the logarithm of C for each kernel and then train the SVM, for each kernel, with the best C. The result should look something like Figure 7.3. After optimizing the regularization for each kernel, use the test set to determine which classifiers perform significantly differently. Use the approximate normal and the McNemar tests as explained in Lecture 5 (and chapter 7 of the lecture notes). You can use the `predict` method of the SVM to obtain the predicted labels for the test set.

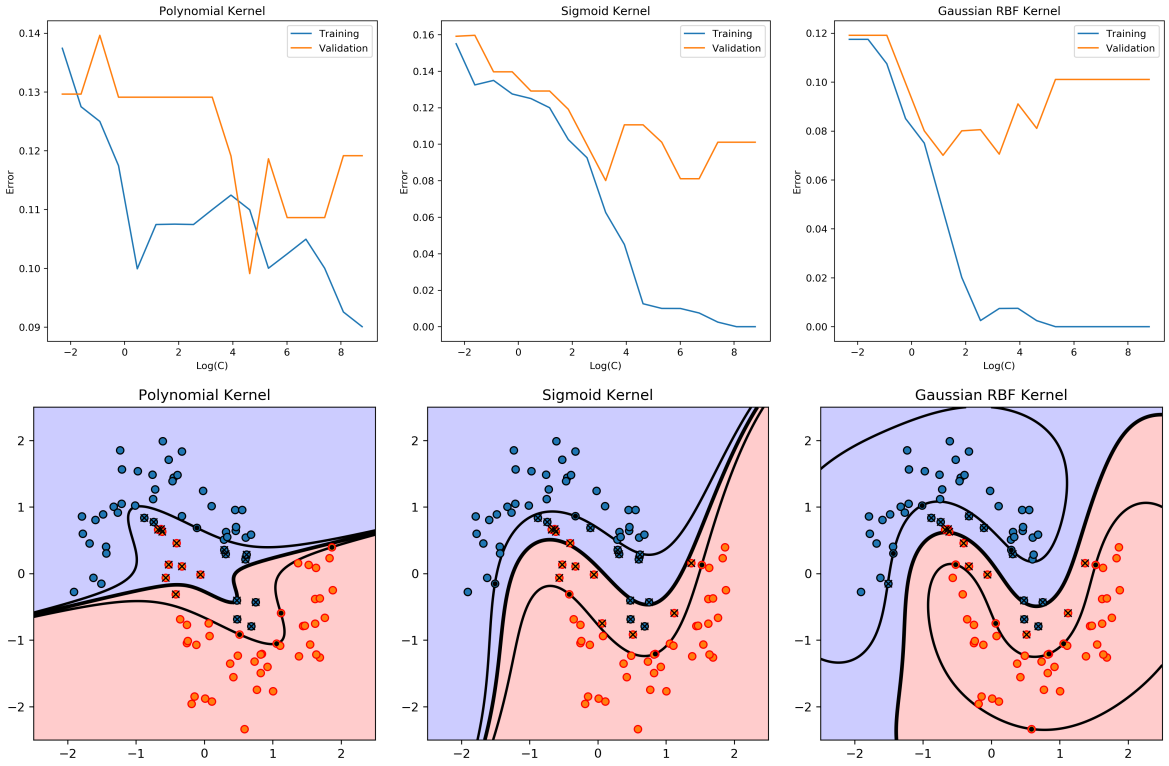


Figure 7.3: Optimizing the regularization parameter C for each kernel

Chapter 8

Decision trees and Ensembles

Decision trees and ensembles.

8.1 Decision Trees

Consider a decision tree model used for classification tasks. The decision tree recursively partitions the input space into regions based on feature values, and a class label is assigned to each region. The goal is to minimize the misclassification error.

Given a dataset $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where x_i represents the feature vector and y_i is the class label for the i-th instance, answer the following:

1. **Entropy Calculation:** Define entropy for a subset $S \subseteq \mathcal{D}$ with classes $C_1, C_2, \ldots, C_k$. Write the mathematical formula for entropy and explain its significance in the context of decision trees.

2. **Information Gain:** Describe the concept of information gain and provide its mathematical formula. Explain how decision trees use information gain to select the best feature for splitting the data.

3. **Gini Impurity:** Write down the formula for calculating Gini impurity for a set S. Compare and contrast Gini impurity with entropy in terms of their usage in decision tree algorithms.

4. **Decision Rule Formulation:** Explain how a decision rule is formulated at each node in a decision tree. Include a discussion on binary splits based on a threshold value for continuous features.

5. **Pruning Techniques:** Discuss the concept of pruning in decision trees. Describe one method for pruning a decision tree and explain its mathematical basis.

6. **Real-World Example:** Consider the following dataset related to weather conditions and the decision to play a sport:

Outlook	Temperature	Humidity	Wind	Play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rainy	Mild	High	Weak	Yes
Rainy	Mild	High	Strong	No

Using this dataset, formulate a small decision tree using entropy as the criterion for splitting. Calculate the entropy, Gini impurity, and information gain at each split.

8.2 Random Forests

1. Decision Tree Construction in Random Forests:

   Explain the process of constructing a decision tree within a random forest. What mathematical criteria are used to split nodes in a decision tree?

2. Ensemble Methods and Voting Mechanism:

   How does the ensemble method in a random forest contribute to its predictive accuracy? Describe the voting mechanism used in random forests for classification tasks.

3. Bootstrap Aggregating (Bagging) and Random Forests:

   Define bootstrap aggregating (bagging) and discuss its role in random forest models. How does it mathematically improve model robustness?

4. Feature Randomness in Split Selection:

   Discuss the mathematical rationale behind using a random subset of features for split selection in the construction of decision trees within a random forest. How does this contribute to the overall effectiveness of the model?

5. Overfitting in Random Forests:

   Describe how random forests mitigate the risk of overfitting compared to individual decision trees. Include a discussion on the mathematical basis of this advantage.

6. Variable Importance and Random Forests:

   Explain the mathematical methods used in random forest models to evaluate the importance of different variables (features) in making predictions.

7. Error Estimation in Random Forests:

   Discuss the Out-of-Bag (OOB) error estimate in random forests. How is it calculated, and what does it signify mathematically?

8. Random Forests in Regression Tasks:

   How are random forests adapted for regression tasks? Describe the mathematical approach used to aggregate outputs from different decision trees.

9. Hyperparameters in Random Forests:

   Discuss the key hyperparameters in random forest models and their mathematical implications on the model's performance and complexity.

10. Given a dataset with three classes (A, B, C) having 40, 30, and 30 instances respectively, calculate the Gini impurity of the dataset.

11. Consider a split in a decision tree that divides a dataset into two groups: one with 20 instances of class A and 10 of class B, and another with 20 instances of class B and 30 of class C. Calculate the information gain of this split.

12. Assume a decision tree in a random forest uses a random subset of 4 features out of 10 for split decisions at each node. Calculate the probability that a specific feature will be considered at a given node.

13. A random forest model is tested with two different settings: 50 trees and 100 trees. Explain how to quantitatively assess the impact of the number of trees on the model's performance.

14. A random forest model is used for classification with 5 features. After training, the model reports the following feature importances: [0.2, 0.3, 0.15, 0.25, 0.1]. Discuss how to interpret these values in the context of model prediction.

15. In a random forest model, two features A and B have a mean decrease in Gini impurity of 0.05 and 0.07, respectively. Interpret these values in terms of feature importance.

16. For a random forest model, calculate the average depth of trees if the maximum allowed depth is set to 10 and the minimum samples split is 5.

Chapter 9

Feature Selection, Extraction

Feature selection in supervised learning with f-scores. Feature extraction with Principal Component Analysis. Vector Quantization with K-Means clustering.

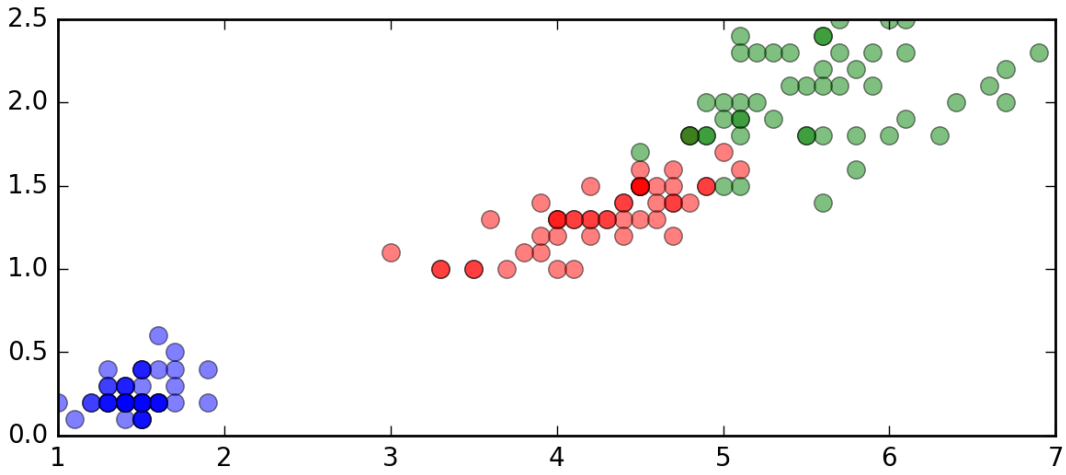
9.1 Exercise 1: Feature Selection

Load the Iris dataset and separate the features and class labels in two different variables. This data set is available from the `datasets` module:

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
```

Now use the `f_classif` function from the `sklearn.feature_selection` the ANOVA F-value for each feature, considering the class labels. Consult Lecture 15. Do the exercise by examining the result (F-values and probabilities) of the `f_classif` function and manually selecting the best features. Optionally, you can repeat this exercise using the `SelectKBest` class, but do not skip the previous steps to make sure you understand how the features are selected.

Plot the data projected on the two selected features. The result should look like this. If you need help reproducing this plot, check the last section on this chapter for the plotting function used in this plot.



9.2 Exercise 2: Principal Component Analysis

Dowload the dataset data1.csv and visualize it, using the relevant tools from Lecture 7. What can you say about the dataset? Look at the number of features, the maximum and minimum values, if there are any missing values, the data distribution of each feature, and across features. This process of data discovery in included in Exploratory Data Analysis and is covered in the very practical but precise Exploratory Data Analysis in Python by Prof. Allen Downey.

Discuss why do you think you could benefit from running a dimensionality reduction algorithm on this data.

Instead of standardizing, you only need to subtract the mean of each variable. Explain why.

Run the PCA implementation from Scikit-learn, for different values of the representation dimension $k \in \{1, 2, 3\}$, and plot your data points in these new representations. The new axes are called Principal Components. From the matrices in Lecture 7, where can we find these principal component vectors?

Plot the represented variance ratio by plotting in a bar plot the vector `pca.explained_variance_ratio_`, where `pca` is your trained model. Relate this plot with your previous answers in this Exercise.

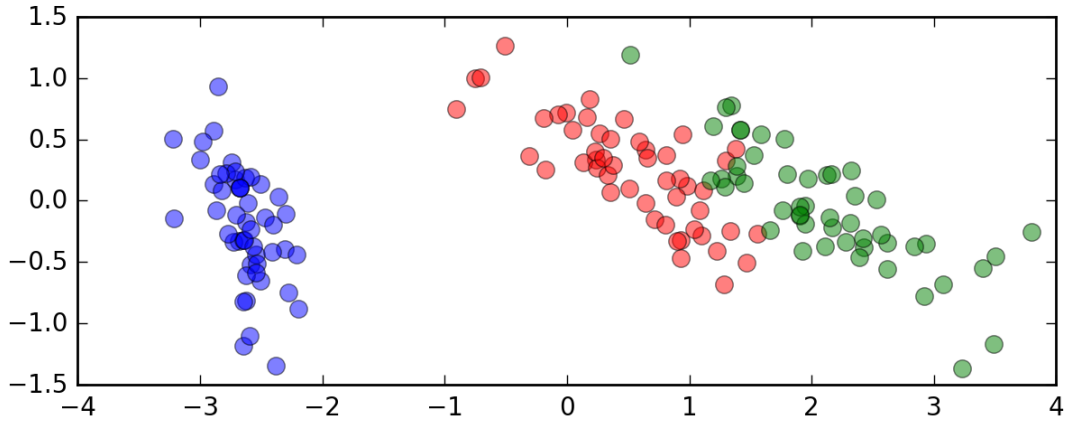
9.3 Exercise 3: Feature Extraction with PCA

Feature extraction is the process of computing features from the initial data. This can be a complex problem and is often very dependent on the data. It is used in image processing, sound processing, stream data such as event logs and so on. In this exercise we will look only at Principal Component Analysis, a generic method of dimensionality reduction which computes new features from linear combinations of existing features while maximizing the preserved variance in the data set. This can be used in unsupervised learning because it relies solely on the feature values.

Import the Iris dataset as before. Note that the Iris values are all lengths in millimetres and we want to find the axes of largest variance. Thus, in this case, we should not standardize or normalize the data because doing so will distort its shape. In unsupervised learning, we often need to be careful about how we transform the data because the shape of its distribution and the distances between the points may be important.

Use the `PCA` class from the `sklearn.decomposition` module to transform the Iris data into a new set of two features, corresponding to the two largest principal components. The result should look

like this.



9.4 Appendix: plotting code

This is the code for the plotting function used in the Iris exercises.

```
def plot_iris(X,y,file_name):
    plt.figure(figsize=(7,7))
    plt.plot(X[y==0,0], X[y==0,1],'o', markersize=7, color='blue', alpha=0.5)
    plt.plot(X[y==1,0], X[y==1,1],'o', markersize=7, color='red', alpha=0.5)
    plt.plot(X[y==2,0], X[y==2,1],'o', markersize=7, color='green', alpha=0.5)
    plt.gca().set_aspect('equal', adjustable='box')
    plt.savefig(file_name,dpi=200,bbox_inches='tight')
    plt.close()
```

This is the code for plotting the 3D representation of the colour space in the vector quantization exercise. Note that this is a very large plot because each plot in the chart corresponds to a pixel in the image. It can take a few minutes to complete and requires up to 500MB of RAM during execution. The `cols` variable contains the colour matrix for the pixels.

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(cols[:,0], cols[:,1], cols[:,2], c=cols,s=10)
ax.set_xlabel('Red')
ax.set_ylabel('Green')
ax.set_zlabel('Blue')
ax.set_xlim3d(0,1)
ax.set_ylim3d(0,1)
ax.set_zlim3d(0,1)
plt.savefig('T7-veg_rgb.png',dpi=200,bbox_inches='tight')
```

Chapter 10

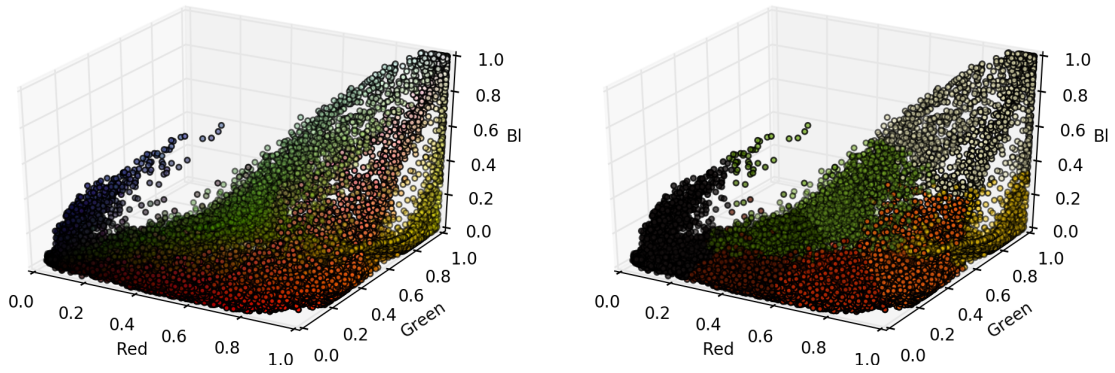
Vector Quantization and Clustering

Vector Quantization with K-Means clustering.

10.1 Exercise 1: Vector Quantization

Download the vegetables.png image file to your working folder[1]. Note that the image used in the example of Lecture 9 only has the red and green components, so all colour values could be represented easily in a 2D-plot with red and green as the axes. This exercise will use a more realistic image with all colour components, so the corresponding 3D plot is a little more complex. In this exercise we are going to compute k centroids to the 3D colour space with the K-Means clustering algorithm, and then convert each pixel into the closest centroid to compress the colour space reducing the number of different colours in the image. This is called vector quantization because we are converting vectors in a continuous space into a finite set of values (quantities).

The figures below show the original colour space and the compressed colour space with $k = 8$, with the point coordinates corresponding to the original colour.



In the Appendix section you can see the code for the 3D plot, but it is not necessary for you to do it. In this exercise we will just load the images, convert the colours and save the images.

We start by loading the image using the `imread` function from the `skimage.io` module. We will also need the `imsave` function to save the images computed at the end. These functions simplify the conversion between image files and Numpy matrices. Note that the image is coded with one byte per

[1] CC Jane Fresco, source https://en.wikipedia.org/wiki/File:Colorful_Photo_of_Vegetables.png

colour, with an integer range of 0-255. We will convert this into floating point values between 0-1 by dividing the matrix by 255.0.

```
import numpy as np
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import axes3d
from skimage.io import imsave,imread


img = imread("vegetables.png")
w,h,d = img.shape
cols = np.reshape(img/255.0, (w * h, d))
```

The reshape at the end places all the points into a matrix of colours with 3 columns. This is the matrix we will use for the K-Means quantization.

Now use the `KMeans` class from the `sklearn.cluster` module to compute 64 centroids and then convert all the colours to the nearest centroid. You can get the centroid positions from the `cluster_centers_` attribute of your K-Means object and the labels from the `predict` method. Look up the example on Lecture 17 and the documentation on the K-Means class. To convert the colours you just need to assign the nearest centroid values to the pixel colour values. For example:

```
c_cols = np.zeros(cols.shape)
for ix in range(cols.shape[0]):
    c_cols[ix,:]=centroids[labels[ix]]
```

Finally, reshape the colours matrix into an image matrix and save it. For example:

```
final_img = np.reshape(c_cols,(w,h,3))
imsave('image.png',final_img)
```

Where the `w` and `h` variables store the width and the height of the original image (see the first code snippet in this section).

The figures below show the comparison between the original image (left), the 64 colours compressed image (centre) and the 8 colours compressed image (right)

10.2 Questions

After these exercises, you should be able to answer the following questions:

- How is the result of the K-Means algorithm?
- How can we use K-Means for vector quantization?

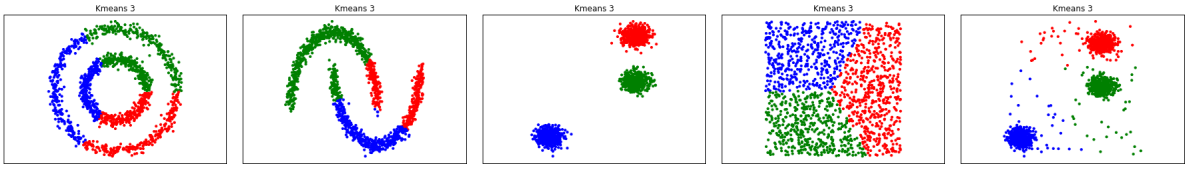
10.3 Comparing clustering algorithms

10.4 Exercise

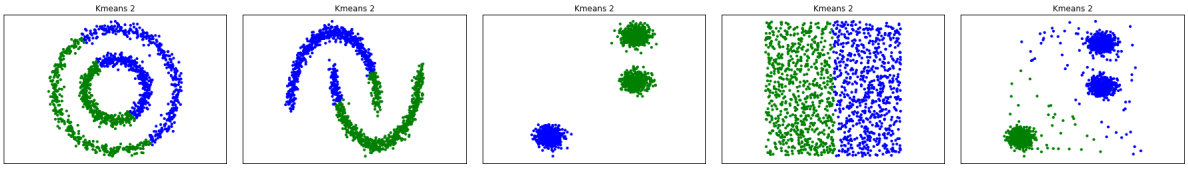
Start from the script provided (t8_clusters.py). The initial part of this script has the code to generate the different data sets you will be using to experiment with different clustering algorithms. The datasets are in the list `datasets`, and each dataset is a matrix with two features and 1500 examples. Note that the random seed is set. Experiment with it on and off.

Use the `plot_clusters` function to plot your labelled clusters for each algorithm. Try the following, using the default parameter values except where otherwise indicated. All clustering algorithms are in the `cluster` module.

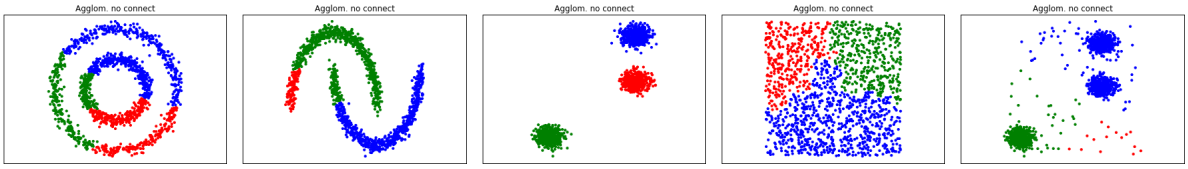
- `KMeans` with `n_clusters=3`



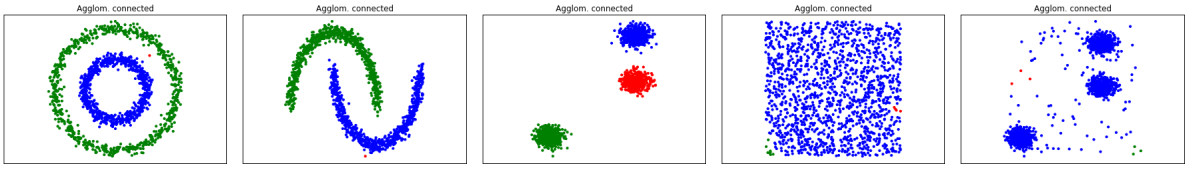
- `KMeans` with `n_clusters=2`



- `AgglomerativeClustering` with `linkage='average'` and `n_clusters=3`



- `AgglomerativeClustering` with `linkage='average'` and `n_clusters=3`, plus `connectivity` set to a connectivity matrix connecting the 10 nearest neighbours. Use the `kneighbors_graph` to obtain the connectivity matrix.



- Repeat all the experiments, changing the linkage argument to single and complete linkage. Comment on the results.

10.5 Questions

After these exercises, you should be able to answer the following questions:

- Can the K-Means algorithm adapt the number of clusters to the data?

- What is the function of the connectivity matrix in the Agglomerative Clustering algorithm?
- How does the linkage methods impact your clustering results?

Bibliography

[1] Sanford Weisberg. *Applied linear regression*, volume 528. John Wiley & Sons, 2005.