The Mini-Project

1DV501/1DT901: Introduction to programming

Jonas Lundberg, office B3024

Jonas.Lundberg@lnu.se

The slides are available in Moodle

October 8, 2020

Today ...

- Mini-project
  - Rules
  - Teams
  - Presentation
  - Gitlab
- Mini-project Problem (four parts)
- Measuring time
- Plots using `matplotlib`
- Recommended approach

Reading instructions: Only these slides!

Mini-project: Introduction

The Python Mini-Project is a small project exercise where you in a team handle a given task which will presented orally and in a report by the end of the course.

Deadline: Program dependent, October 28-30

Mini-project information

- *The Mini-project* Moodle section is the main source of project information
- Here we will publish:
  - Information about teams and team members (Done!)
  - Rules (Done!)
  - Project task (Done!)
  - Gitlab instruction videos (Soon!)
  - Templates for written report (Not ready yet!)
  - Templates for oral presentation (Not ready yet!)
  - Detailed schedule for presentations (Not ready yet!)

Mini-project: Rules

- You work in teams of two to three students. Your tutoring supervisor will assign you a team
- By the end of the course you will present your results in an oral presentation and by a written report.
- The written report should follow a given template.
- All team members should be prepared to answer questions regarding all parts of the project.
- Feel free to use information found on the Internet but:
  - Give a proper reference to the source
  - Be prepared to answer detailed questions, you must understand what you are doing
- Plagiarism is cheating! Any sign of plagiarism $\Rightarrow$ all involved students fail (also students giving away their code).

Teams and Team Members

Teams are put together (by your tutoring supervisor) following these criteria:

- Each team has three members. We rather have 2+2 than four.
- Campus students with campus students
- Distance students with distance students
- Kalmar students with Kalmar students
- Swedish speaking students with Swedish speaking students
- etc

Thus, we will try to put together students having a similar study situation.

Important: Contact your tutoring supervisor if ...

- You can not find your name in any list
- You end up in "wrong" team.
- You have questions about project teams

The teams are now available in the Mini-project section in Moodle.

Written Report, Oral presentation

- The team effort is presented as:
  1. An oral presentation
  2. A written report
  3. Entire team code available in Gitlab
- All of it should be available at the time of your oral presentation
- Exact schedule for oral presentations will be presented later on (Roughly October 28-30, we might need to use more than one day due to shortage of suitable lecture rooms.)
- Templates for written report and oral presentation will be presented in Moodle

Gitlab

- Git is a distributed version-control system for tracking changes in source code during software development.
- Git is designed for coordinating work among programmers, but it can be used to track changes in any set of files.
- We use Gitlab to provide you with a Git system
- Tobias Andersson-Gidlund has put together several instructions video to get you started with Gitlab.
- The Gitlab videos will soon be available in Moodle
- It is an excellent tool for projects with members working on a common code base $\Rightarrow$ **Start to use it!**

The Mini-project Task

The project is about working with and analyzing large texts. Another important part is to understand hashing and binary search trees.

The problem can be divided in four parts:

1. Divide the large texts into a sequence of words.
2. Implement two data structures suitable for working with words as data: a) A hash based set, and b) a binary search tree based table (dictionary).
3. Use your two data structures to solve certain word related tasks
4. Measure the time to perform certain operations on your map and set implementation.

Part 1: Divide text into words

The project is about working with and analyzing large texts. As examples of large texts you have the two large text files you used in Assignment 3.

Part 1

1. Divide the large texts into a sequence of words.
2. As a part of your presentation you should define what you meant by a word and motivate your decision.
3. However, as a rule, a word doesn't contain any digits, or symbols like ".", ",", "!", "?", etc.,
4. and we consider words like "can't" and "John's" as a single word.

Suggestion: Save all words in a file to simplify future project exercises.

Part 2: Implement data structures

Lecture 9 outlines the basic ideas of two implementation techniques:

1. Binary search trees
2. Hashing

Your task is to implement:

- a set `word_set.py` (suitable for words) based on hashing
- a table `table.py` (key-value pairs) based on binary search trees.

We want you to follow the simple (non-class based) idea for how to implement a data structure that was used in the linked list example in Lecture 9.

Part 2: Additional limitations

- The nodes in the binary search tree based table implementation are lists of size 4 (key,value,left-child,right-child).
- The hash-based set is built using a list to store the buckets where each bucket is another list. The initial bucket size is 10 and rehashing (double the bucket list size) takes place when the number of elements equals the number of buckets.

Furthermore, we will provide code skeletons outlining which functions we expect for each data structure. They also contains an example program showing how the various can be used.

Part 3: Count word related entities

Using your set and table implementation you should be able to handle a few word related questions:

- By using the set you should be able to count how many unique words that are used in the texts.
- By using the table you should be able count how many words of a given length a given text has, and of present a histogram (plotted using matplotlib).
- By using the table you should also be able to present a list of the top-10 most frequently used words.

You are not allowed to use Python's set and dictionary classes. However, it might be a good idea to compare your results with a similar program using them. The results should be the same.

Part 4: Time Measurements

Tasks

- Measure the time to look-up X thousand elements (keys) in maps of different sizes (larger than X) in your binary search tree based table implementation. Measure also the max tree depth at the same sizes. We expect two plots (using matplotlib) showing look-up time vs table size, and max depth vs table size.
- Design an experiment measuring the time it takes to add new elements to your hash set and show how it correlates to the rehashing. Measure also the max bucket size at various set sizes. We expect a few plots (using matplotlib) showing how the time to add (say) 1000 unique elements differs at various table sizes due to the rehashing mechanism, we also expect a plot showing how the max bucket size changes with the set sizes.

We strongly suggest that you use the words in the text file `eng_news_100K-sentences.txt` as your data in the experiments. The file contains about 2 million words, out of which 80.000 are unique.

How to get started?

For a team with three members we suggest the following start:

- Member 1: Handle Part 1: Read an arbitrary text file and divide it in a sequence or words. Suggestion: Store the words (one on each line) in a separate file for future use.
- Member 2: Implement `word_set.py` based on hashing.
- Member 3: Implement `table.py` (key-value pairs) based on binary search trees. (This is the hardest part!)
- Members 1: Try to solve exercises in Part 3 using Python's set and dictionary rather than your own implementation
- ...

Remember: Writing the report takes a few days!

Also, ask your tutoring supervisor if you don't understand a certain part of the "Project Task" formulation. Better to ask for the way than to walk in the wrong direction!

Measuring time

Measuring time in Python is easy using the `time` module

```
import time

start = time.time()                    # Start clock

  "Do something that takes time "

elapsed = time.time() - start      # Elapsed time
print("Elapsed time in seconds is:", elapsed)
```

- `time.time()` return the time in seconds since the epoch.
- On most OS systems, the epoch is January 1, 1970, 00:00:00 (UTC)
- Expect time measurement to be correct down to (at least) milliseconds

Advices

- Do not trust time measurements of less than (say) 30 ms
- Solution: Repeat experiment many times and compute average values

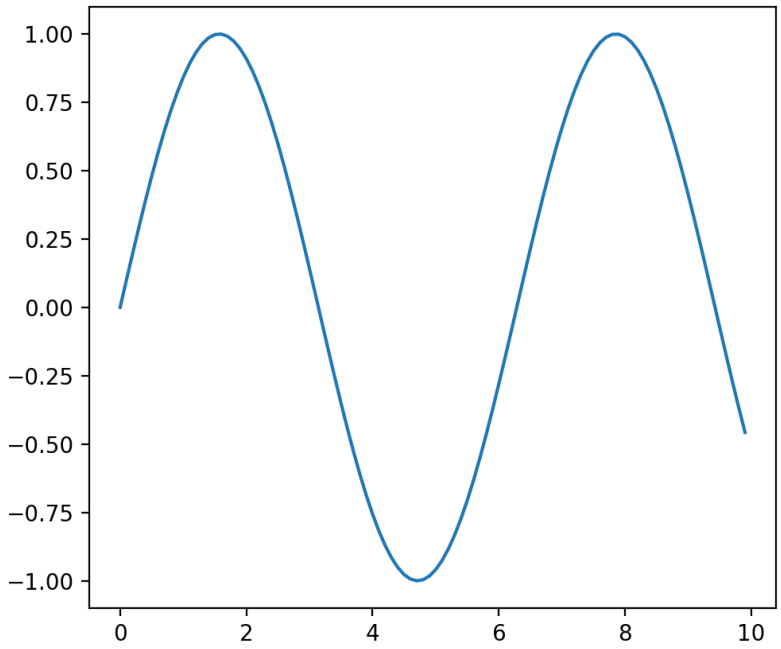
```
repeat = 10
before = time.time()
for i in range(repeat):
    ... do something
after = time.time()
estimated = (after-before)/10    # average time
```

- Measuring time for fast operations like binary search and hashing is very difficult.
- Requires very large input sizes to get trustworthy time measurements
- Large sizes ⇒ memory problems ⇒ garbage collection becomes a problem
- Hence, play around with various input sizes and repetitions to get trustworthy time measurements. Trustworthy ⇒ same results in repeated runs
- Also remember
  - Your experiment is not the only activity running on the computer
  - Try to shut down other applications
  - However, the operating system will still be working in the background

  Repeated runs give very different result ⇒ something is wrong!

Simple x,y-plot using matplotlib

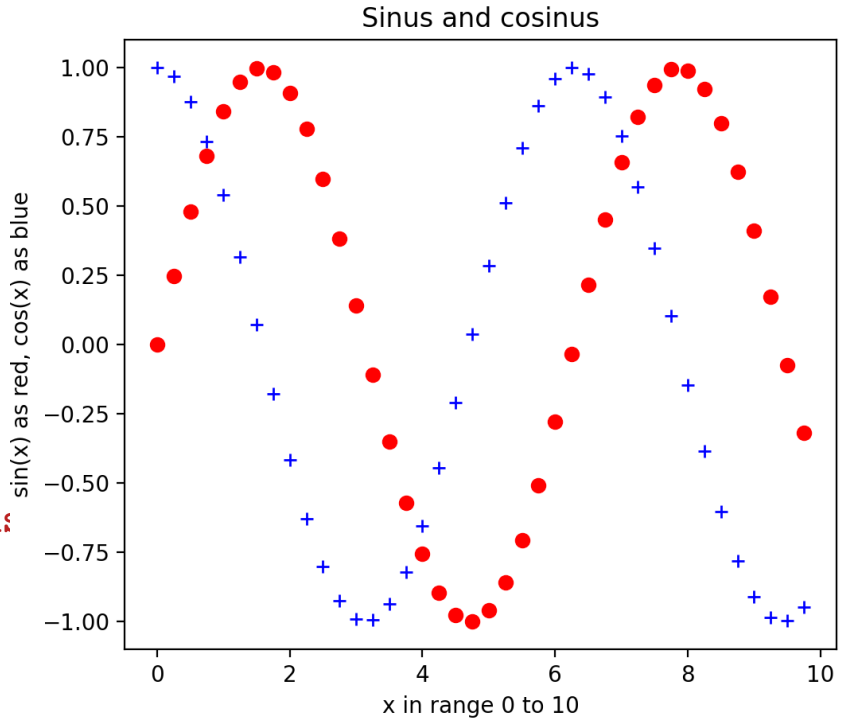
```
import matplotlib.pyplot as plt
import math

# x = 0.1, 0.2, 0.3, ..., 9.9, 1
x = [n/10 for n in range(0,100)]
y = [math.sin(i) for i in x]
plt.plot(x, y)
plt.show()      # Display plot
```



- `matplotlib` is a very powerful plotting library for Python
- However, producing a x,y-plot given two lists x and y is easy
-

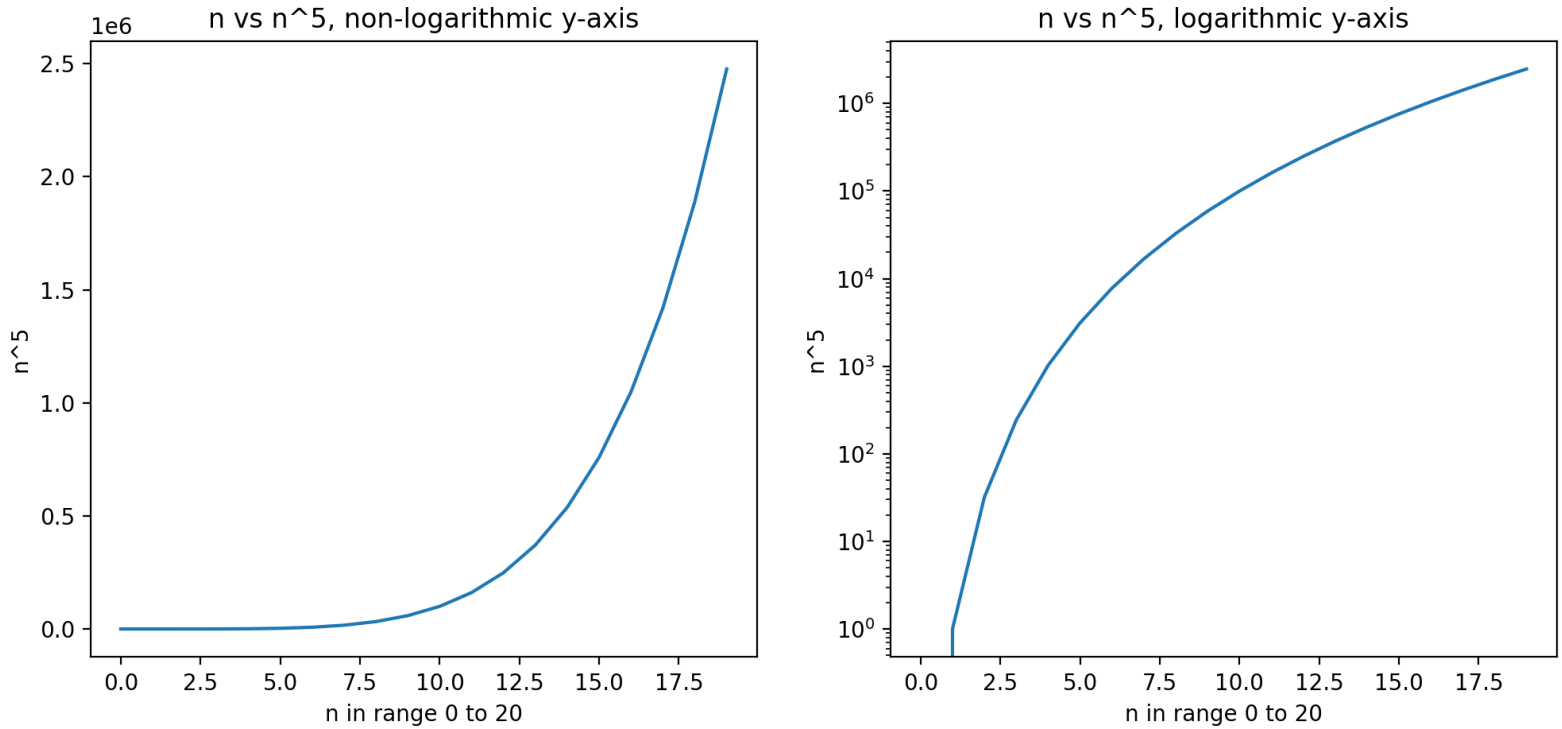
Adding titles and axis-labels

```
# x = 0.25, 0.5, 0.75, ... , 10.0
x = [n/4 for n in range(0,40)]
y1 = [math.sin(i) for i in x]
y2 = [math.cos(i) for i in x]
plt.plot(x, y1,'ro')
plt.plot(x, y2, 'b+')
plt.title('Sinus and cosinus')
plt.xlabel('x in range 0 to 10')
plt.ylabel('sin(x) as red, cos(x) as
plt.show()      # Display plot
```



- `'ro'` ⇒ red circles, `'b'+` ⇒ blue plus. Called markers, plenty to chose from
- `title`, `xlabel`, `ylabel` ⇒ figure title and x- and y-axis labels
- Also, two times `plt.plot(...)` ⇒ two plots in same figure

Two figures side-by-side



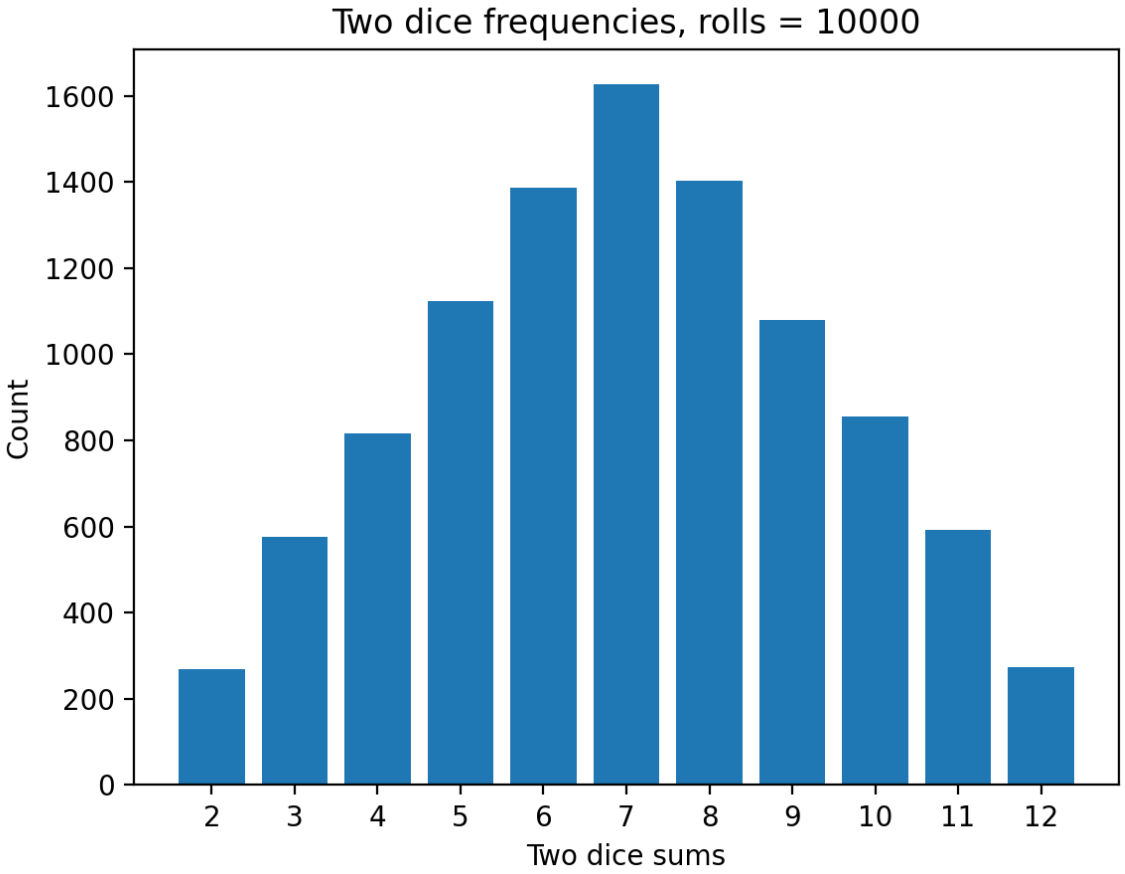
- Two plots in one figure. Python code on next slide.
- Notice also the logaritmic y-scale on the 2nd figure
- A logaritmic scale is useful when your data varies in magnitude $\Rightarrow$ it allows variations in small values not to be completely obscured by values several magnitudes larger.

Side-by-side continued

```
fig, (ax1, ax2) = plt.subplots(1, 2) # Two plots called ax1 and ax2
x = [n for n in range(0,20)]
y = [i**5 for i in x]

ax1.plot(x, y)
ax1.set_xlabel('n in range 0 to 20')
ax1.set_ylabel('n^5')
ax1.set_title('n vs n^5, non-logarithmic y-axis')
ax2.plot(x, y)
ax2.set_xlabel('n in range 0 to 20')
ax2.set_ylabel('n^5')
ax2.set_title('n vs n^5, logarithmic y-axis')
ax2.set_yscale('log')
plt.show()
```

- `fig, (ax1, ax2) = plt.subplots(1, 2)` $\Rightarrow$ two plots in a 2×1 format.
- `ax1, ax2`, the two subplots, are then configured separately
- Notice that the commands for setting x- and y-labels, and title, differs from the previous one-figure plots.
- `ax2.set_yscale('log')` $\Rightarrow$ use logaritmic scale for y-axis in 2nd plot

Simple bar charts



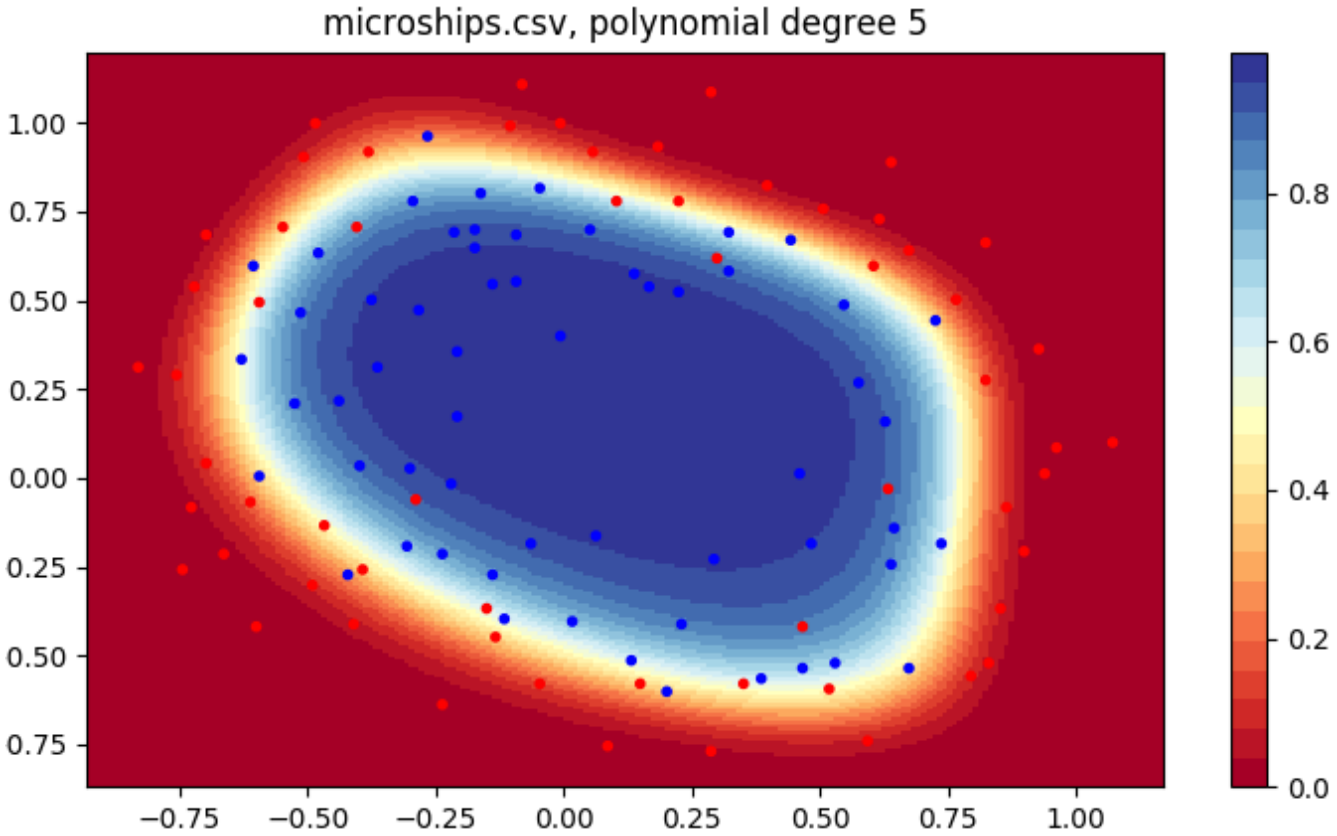
A bar chart for the two-dice problem in Assignment 2

Bar charts continued

```
freq = 11*[0]              # Frequencies for 10000 two dice rolls
for i in range(10000):
    r = rdm.randint(1,6)+rdm.randint(1,6)
    freq[r-2] += 1

# Show bar chart
x_pos = list(range(1,12))              # [1,2,3,..., 10,11]
labels = [str(n+1) for n in x_pos]     # ['2','3', ... '11','12']
plt.bar(x_pos, freq)
plt.xticks(x_pos, labels)     # Give each position a label
plt.title("Two dice frequencies, rolls = 10000")
plt.xlabel("Two dice sums")
plt.ylabel("Count")
plt.show()
```

- Each bar has a position (1 to 11), and a label ('2' to '12')
- `plt.bar(x_pos, freq)` $\Rightarrow$ assign each bar a position and a y-value (height)
- `plt.xticks(x_pos, labels)` $\Rightarrow$ assign each position a label

`matplotlib` - **Summary**



- Using `matplotlib` you can produce high-quality plots and visualizations
- From simple plots to multi-figure plots with all types of fancy features
- A lot of information on the Internet, but often very advanced stuff.
- Try to find tutorials to get started
- Above figure taken from our course in machine learning (2DV516)

This is it!

- This is the last lecture!
- You are now Python programmers!
- Regarding the project
  - Locate your team members and set up a meeting
    (Contact tutoring supervisor if you can't get in contact with any member.)
  - Start working
  - Problems? Visit tutoring sessions or post a question in Slack
  - Awkward team situation? 1) Try to sort it out, you are grown ups, 2) Contact your tutoring supervisor.
- Good luck with the project!
- Good luck with the Python test!
- Good luck with your future programming!