HOMEWORK #1

SUBMISSION DUE DATE: 30/11/2020 23:00
SUBMISSION IN SINGLES OR PAIRS

INTRODUCTION

The K-means algorithm is a popular clustering method for finding a partition of N unlabeled observations into K distinct clusters, where K is a parameter of the method.

In this assignment you will implement this algorithm in both Python and C.

The goals of the assignment are:

- Get familiar with the working environment and the IDEs.
- Practice the material taught in class both for C and Python.
- Transform a known algorithm into working executable code.
- Read input stream and process it.
- Create an interface for programs.
- Experience the difference in the programming effort and the running time of both languages.

Throughout this document, the sections are in blue and their subsections are in lighter blue.

DEVELOPMENT ENVIRONMENT AND SETUP

This assignment requires connecting to the faculty server, **Nova**. Experience it as soon as possible to ensure everything works. If you have issues connecting to Nova, please visit https://www.cs.tau.ac.il/system/ for support.

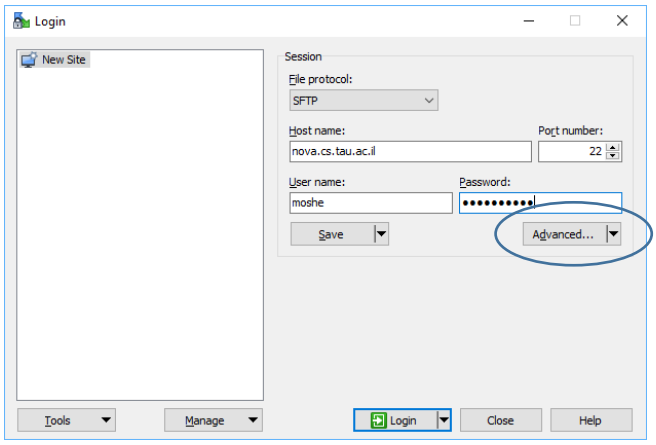
Students may work in any development environment they like. However, we encourage you to use eclipse CDT and PyCharm as your development tool (those are environments we are more confident with giving appropriate support). Your submission will be automatically checked, so your code should run properly on Nova. You can remotely connect to the server as described below.

USEFUL LINKS

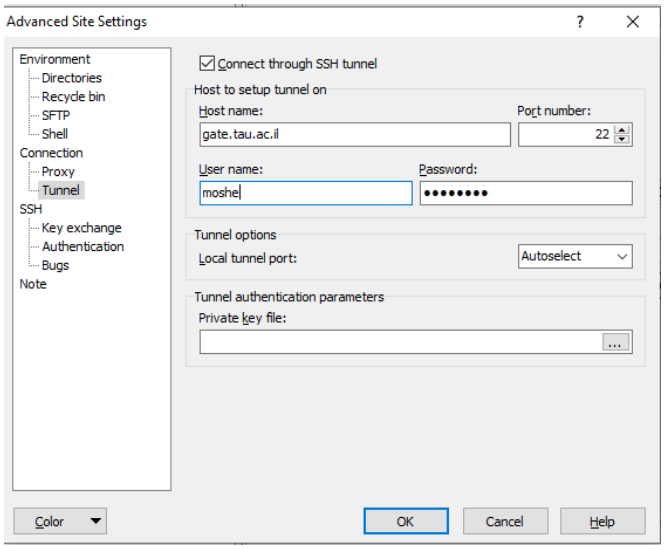
You can download PuTTY and WinSCP from here: http://www.cs.tau.ac.il/system/downloads . Recall that we recommend working with eclipse; please follow the installation guidelines for eclipse in Moodle. For PyCharm Community version, see: https://www.jetbrains.com/pycharm/download/
(note the installation instruction section there as well).

CONNECTING TO NOVA AND FILE TRANSFER

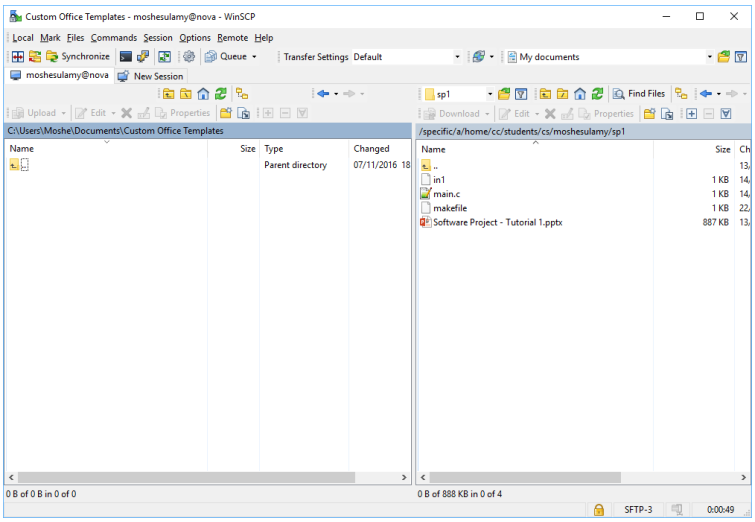
In order to transfer files to/from Nova, you can make use of WinSCP:



Before logging in, select "Advanced" (circled above) to define a tunnel to *gate.tau.ac.il*:

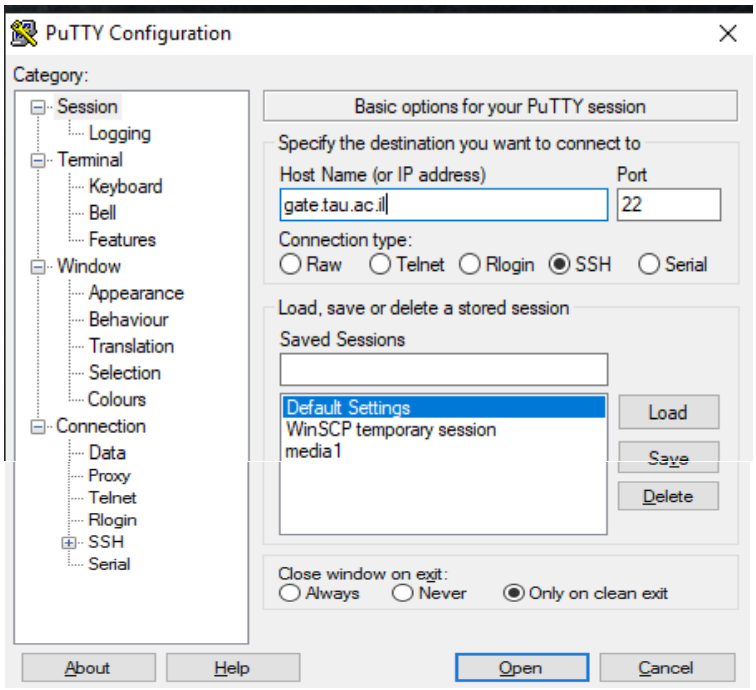


Once connected, drag files from the left part (your computer) to the right (Nova):



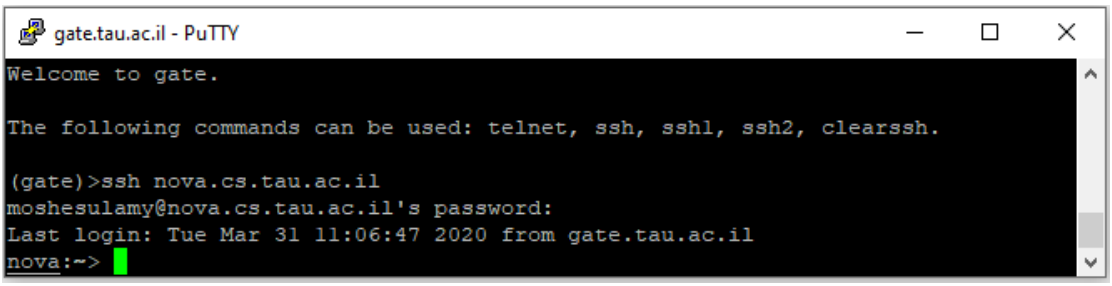
REMOTE CONNECTION TO NOVA

To connect to Nova, use SSH or PuTTY (*gate.tau.ac.il*):



After pressing the “Open” button, enter your username and password for authentication (the username and password are identical to your Moodle authentication). Once connected, you should run the command ***ssh nova.cs.tau.ac.il***

When successfully connected to Nova, you will see a terminal similar to the following:



Now you can execute shell commands with this tool. Read the section below to find out more about shell commands.

BASIC SHELL COMMANDS

After the connection to Nova is established, you can execute shell commands. The results will be displayed on the same terminal (If there's any).

A few useful shell commands:

>> pwd
Prints the full pathname of the current working directory.
(The pathname is relative to the root directory which is the first directory in Linux).

>> time [command]
Displays information about the execution time of "*command*".
Use it to compare your programs (C vs Python) runtimes.

>> ls [dir]
Lists the content of the directory *dir* (Both files and directories). If no parameters are given, the result is the content of the current working directory.

>> cd [dir]
Changes the current directory to be *dir. If dir is empty, the control moves to the home directory*
Use the following shortcuts:

. – This is a shortcut for the current directory.
.. – This is a shortcut for the parent directory in the hierarchy.
~ – this is a shortcut for your home directory.

Examples:

cd ~/projects/git/k_means

cd ../../../k_means

>> mkdir [dir]
Creates a new directory with the name *dir*.

>> cp [file1] [file2] … [fileK] [dir]
Copies *file1,file2,…,fileK* to the directory *"dir".*
This following common example:
"*cp mytext.txt mytext2.txt*"
will copy the file "mytext.txt" into a file named "mytext2.txt".

>> rm [file1] [file2] … [fileK]
Removes the files *file1,file2,…,fileK.*

>>man [command]
The man command is used to display the manual page of the command "*command*".
Note: you can navigate through the manual page using the arrows in the keyboard. To exit the manual page press "*q*".

>> diff [file1] [file2]
Prints the difference between the two files (file1 and file2)
Note: Use the *man* command to see the manual page of the command *"diff".*

>> chmod [options] [file]

Changes the permissions of [file] according to [options].
Note: When transferring executable files to Nova, you may need to *"chmod 777 <file>"* to allow their execution.

THE K-MEANS CLUSTERING ALGORITHM

Given a set of N observations $\{x_1, x_2, \dots, x_N\}$, where each observation is a d - dimensional real vector, $x_i \in R^d$, we aim at finding K clusters to group the observations into.

Each observation is assigned to exactly one cluster and the number of clusters K is such that $K < N$.

We will denote the group of clusters by $\{S_1, S_2, \dots, S_K\}$.

Each cluster S_j is represented by its centroid μ_j which is a d - dimensional point.

The centroid represents the mean of the members in its cluster, hence the name K-means.

We denote the group of cluster centroids as $\{\mu_j \in R^d : 1 \leq j \leq K\}$.

The K-means algorithm is as follows:

1. Initialize cluster centroids $\{\mu_1, \mu_2, \dots, \mu_K\}$ from the first K observations.
2. Repeat until [(cluster centroids do not change from previous iteration) OR (we have completed MAX_ITER iterations)]:
    a. Assign each observation x_i to cluster S_j that minimizes:
    $$\|x_i - \mu_j\|^2 \quad \forall j,\ 1 \leq j \leq K$$
    b. Update each centroid μ_j as follows:
    $$\mu_j = \frac{\sum_{x_p \in S_j} x_p}{|S_j|}$$

The input file containing the observations and the MAX_ITER parameter used in step 2 are addressed in the section “PROGRAMS INTERFACE”.

In step 2.a. of the algorithm, the distance used to find the closest cluster’s centroid is the squared Euclidean distance and it is defined as (for two n - dimensional points p, q):

$$\|p - q\|^2 = \sum_{i=1}^{n} (p_i - q_i)^2$$

In step 2.b. of the algorithm, $|S_j|$ is the size of S_j.

INTERFACE

INPUT

In this assignment, both of the implementations (C and Python) will contain a command-line argument interface.

Those arguments are *K, N, d, MAX_ITER* (in this order) which stand for:

- *K* – the number of clusters required.
- *N* – the number of observations in the file
- *d* – the dimension of each observation and initial centroids
- *MAX_ITER* – the maximum number of iterations of the K-means algorithm

In addition, an input file which we'll denote as *filename*, will be directed to the standard input. It will contain the *d*-dimensional observations (see below for a detailed explanation on the file's format and the standard input used to process it).

INPUT FILE FORMAT

Consider the *filename*. The first *K* lines contain the initial *d*-dimensional centroids separated by commas. These lines should be used in step 1 of the algorithm.

The following *N – K* lines represent the remaining observations.

Each data point contains 2 decimal places (for example 1.00 or -298.77).

A *filename* can be generated by using "gen_file.so" as described in the "SUPPLEMENTARY MATERIAL" section.

OUTPUT

At the end of the program, both implementations will display (*printf(…)* in C and *print(…)* in Python) the computed final *K*-centroids, each in a separate line. Each centroid will have the data points separated by a comma and each value will have a 2 decimal places format (similar format as the one of the initial centroids in *filename*).

You should keep the order of the final centroids identical to the order you read the initial centroids. I.e. make sure that the first line (which is your first calculated centroid) resulted from the initial first line of *filename* (the first initial centroid) and so on.

An example output (assuming K = 3, d = 2):

8.99,5.00
10.06,8.55
23.55,9.00

THE C IMPLEMENTATION

THE MAIN FUNCTION AND COMMAND-LINE ARGUMENTS

The *main* function is the entry point of any C program. Its signature should be:

```
int main(int argc, char* argv[])
```

argc will contain the number of arguments in **argv[]**.
You can access the 1st argument with **argv[1]** (the number of clusters K), and the 2nd argument with **argv[2]** (the number of observations N) and so on.
In this context, in order to transform strings to integers you can use the function *atoi* which has the following signature:

```
int atoi(const char* nptr)
```

atoi is available at **stdlib.h**

PROCESSING STANDARD INPUT

An input to the program could be given in several ways, such as user input (like keyboard) and files.
A file could be read directly or fed into the program's input using the '<' operator.
The **stdio.h** header provides a rich API to support input/output capabilities.

In it, we will focus on the following function:

```
int scanf(const char* format, ... )
```

This function reads data from the standard input stream.
For an in-depth reference please see:
https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_71/rtref/scanf.htm

For example, consider the following program for reading a stream of lines of data points separated by commas:

```
#include <stdio.h>

int main(void)
{
    double n1;
    char c;
    while (scanf("%lf%c", &n1, &c) == 2) printf("%f\n", n1);
    return 0;
}
```

COMPILE AND EXECUTE

As we noted, students may work in any development environment they choose. However, the code must compile with no errors and run on Nova successfully.

To see that this is indeed the case, please upload your source code and other related files to Nova (all the files in the same directory).

After the connection is established, set that same directory as the source code's directory. Then type in the following command and press enter to build your code (your source code will be called - kmeans.c):

```
>>gcc -ansi -Wall -Wextra -Werror -pedantic-errors kmeans.c -lm -o hw1
```

The above command will compile your program and create a binary file called **hw1**.
Note that the compilation flags used in the command above will be our default flags, so you should configure your eclipse installation with these flags (more information on Moodle).

Now you can run your program by executing the following command on Nova
(assuming K = 3, N = 5, d = 2, MAX_ITER = 100, filename = "input.txt"):

```
>> ./hw1 3 5 2 100 < input.txt
```

Note the use of the operator '<' to direct the content of "input.txt" into the standard input.

Before submitting your code, make sure your code compiles with no warnings and errors on **Nova**. If you encounter ones, it is your responsibility to fix them prior final submission.

THE PYTHON IMPLEMENTATION

COMMAND-LINE ARGUMENTS

The Python standard library has a recommended module for command-line parsing.
It is the one taught in class, **`argparse`** and it has a rich documentation:
https://docs.python.org/3.8/howto/argparse.html

All the command-line arguments defined should be positional ones and have the appropriate typing.

PROCESSING STANDARD INPUT

There are several ways to access the program's input in Python. We focus on the built-in function (https://docs.python.org/3/library/functions.html#input):
`input``([prompt])`

If the optional `prompt` argument is present, it is written to standard output without a trailing newline.
The function then reads a line from input, converts it to a string (stripping a trailing newline), and returns that. When EOF is read, EOFError is raised.
For example, consider the following program for reading a stream of lines of data points separated by commas:

```
while True:
    try:
        for num in input().split(','):
            print(float(num))
    except EOFError:
        break
exit(0)
```

EXECUTE

To execute your Python source code (will be called kmeans.py) on Nova use the following command from the same directory the source code is at
(assuming K = 3, N = 5, d = 2, MAX_ITER = 100, filename = "input.txt"):

```
>>python3.8.5 kmeans.py 3 5 2 100 < input.txt
```

This command uses the Python 3.8.5 version to run your code, allowing you to use all the features of that version. Ensure it runs on **Nova** with no warning/error messages.

TIME MEASUREMENTS

Your code is graded for performance as well.
You should keep that in mind and try to make your code (C and Python) as efficient as possible.
That being said, you'll notice that the Python's runtime becomes increasingly slow as the number of observations, clusters and dimensions grow.
The following configuration (if it takes too long reduce it as necessary):
K = 15, N = 50,000, d = 15, MAX_ITER = 5 and the appropriate input file ('filename'), would result in a runtime difference in the orders of magnitude, favoring the C over the Python one.
To measure the runtime (not by coding), see the `time` command in the "BASIC SHELL COMMANDS" subsection.
There is no requirement to output your runtimes and the expected output should remain as defined in the "OUTPUT" subsection.

SUPPLEMENTARY MATERIAL

TESTER FILE

Attached to this assignment is an executable file **"km.so"**. Copy it to Nova and give it execution permissions (see the command `chmod` above).
Its execution API is:

```
>>python -m km.so K N d MAX_ITER filename file_to_compare
```

Where 'file_to_compare' would be your program's output in a file.
Note that unlike your API, the input file ('filename' and 'file_to_compare') are supplied as a command line argument and not directed to the standard input.

You can use the tester to check your output using the following procedure (assuming K = 3, N = 5, d = 2, MAX_ITER = 100, filename = "input.txt"):

1. Execute your Python code and output the result into a file (in this case named "y") using this command:
   ```
   >>python3.8.5 kmeans.py 3 5 2 100 < input.txt > y
   ```
2. Execute your C code and output the result into a file (in this case named "z") using this command:
   ```
   >>./hw1 3 5 2 100 < input.txt > z
   ```
3. Use the following command to check your Python's output (we allow certain Frobenius Norm difference between your centroid and the testers):
   ```
   >>python -m km.so 3 5 2 100 input.txt y
   ```
   The same can be done with your C output by:
   ```
   >> python -m km.so 3 5 2 100 input.txt z
   ```

Important note: compare your results to the tester ones with converging sets, meaning those that do not reach the MAX_ITER number of iterations.

INPUT FILE GENERATOR

In order to get the initial clusters and the observations for the K-means algorithm in this assignment, an input file is required. It is the file referred to the program's standard input and was denoted as 'filename'.

The supplied executable file **"gen_file.so"** serves as a generator for this file. Copy it to Nova and give it the permissions in the same manner that was explained in the previous subsection.

This file takes 2 command-line arguments N, d (in this order and with the same meaning as explained before).
In order to generate an input file named "input_5_2.txt" use the following command (assuming N = 5, d = 2):

```
>>python -m gen_file.so 5 2 > input_5_2.txt
```

ASSUMPTIONS AND REQUIREMENTS

Note that the following list applies to both programs in this assignment:

- You may assume the input file is in the correct format and that it is supplied.
- You may assume that all the command line arguments that are supplied are in the correct syntax (integer for example), but you need to make sure that all of them are supplied and that they make sense.
- In the C implementation, you may use **`assert`** to exit on any error in allocating memory.
- You may assume that any cluster will have at least one associated observation.
- You may not import external includes (in C) or modules (in Python) that weren't mentioned in this document.
- Hold the requirements as presented throughout this document (such as the output format of your programs, the K-means logic, the programs interface, Nova execution etc.).
- Any further questions should be answered by using the supplied tester and observing its behavior.

SUBMISSION

You may submit this assignment alone or in pairs. Please submit a zip file named **id1_id2_hw1.zip** replacing *id1* and *id1* with the actual **9-digit** ID of both partners. Only a **single** partner should submit the assignment! If you submit alone, name it **id_hw1.zip**.

The zipped file must contain the following files (at the root, **no folders**, the names are all lowercase):

- kmeans.c – Your source code for the C program.
- kmeans.py – Your source code for the Python program.

Submit **only these files!**

MAC users: create (or check) your zip files under Linux, as otherwise hidden auxiliary files are automatically added and will reduce your grade. To zip directly from Nova, you can consult the command `>> man zip` for more information.

REMARKS

- Please post questions you have about this assignment in the assignment forum (it will be opened later on).
- Late submissions are not acceptable unless approved by the TA.
- Borrowing from others' work is unacceptable and bears severe consequences.

GOOD LUCK