Homework 1

Large-Scale Data Analysis

Fabian Gieseke

25 April 2017

Submission Deadline:
2 May 2017, 10:00

Teams: Please form and work in teams of ideally (and at most) three students. Get in touch with the TA while attending the first practical session in case you have problems finding other teammates. Get in touch with me (`fabian.gieseke@di.ku.dk`) if you are unable to attend the first practical session. The teams will generally remain the same for all assignments.
Note: The final exam is a one-week take-home assignment as well, but has to be solved and submitted individually!

Assignments: There will be four or five regular homework assignments. They must be passed in order to be eligible for the final exam. Here, "passing an assignment" means that you have to make a serious attempt to solve the corresponding exercises, which can, e.g., be measured in terms of points for the individual subtasks.
For this assignment, you will generally need to get at least 50 out of the 100+10 points.

Virtual Machine: We will generally assume that you make use of the Ubuntu virtual machine, i.e., all instructions provided are tailored to this system. Of course you are free to use your own desktop environment, tools, and Python installation to implement and test your code. Keep in mind, however, that the output might differ depending on the Python version and the packages being installed.
Important: You have to make sure that the code you deliver runs on the virtual machine!

Deliverables: One submission per team via Absalon (you should be able to form your teams while submitting). For each submission, please upload *two separate files*: (1) a pdf file `answers.pdf` containing your answers and (2) a `code.zip` file containing the associated code as zip file (do *not* include the datasets). Note: It's perfectly fine if you adapt some existing code (e.g., a code snippet you have found on the internet)—just make sure that you provide a link/reference to this external source.

1 VirtualBox, Python, Jupyter (20 pts)

For most of the homework assignments, we will make use of a virtual machine. More precisely, we will resort to VirtualBox[1] and an adapted Ubuntu 16.04 (64bit) image. You can find the installation instructions on the slides for the first practical session (`T1_LSDA.pdf`), which are available via Absalon.

1. Follow the instructions provided on the slides to install VirtualBox and the virtual machine. Note: You do not necessarily have to follow all the slides; it's up to you to do the optional parts (which might be useful for future lectures and assignments).

[1] `https://www.virtualbox.org/wiki/Downloads`

2. Write a small Python program that outputs the names and the Absalon IDs of all team members. Execute the program in the terminal and take a screenshot.

3. Start a new Jupyter notebook that outputs these names and Absalon IDs as well. Execute all cells of the notebook and take another screenshot.

What to submit?

Include the following in your write-up (`answers.pdf`):

1. Screenshot of the virtual machine showing the output of the Python program

2. Screenshot of the virtual machine showing the output of the Jupyter notebook

2 Big Data Applications (20 pts)

We have seen a couple of big data domains and data analysis examples during the first lecture. One example was change decection based on terabytes of satellite images, which can become very challenging both due to the sheer amounts of data and due to the computionally demanding model fitting processes. Can you think of further big data domains and associated data analysis tasks?

1. Give an example for a domain/application that already is or will be faced with a lot of data.

2. Sketch an assoicated data analysis task.

3. Provide a very rough estimate for the data volumes that need to be stored/processed for this task (e.g., every day, month, or year).

4. Might the data analysis become computationally challenging for your example? If yes: What are the expected bottlenecks (e.g., disk space, compute time, ...)?

What to submit?

Describe your example in your write-up (`answers.pdf`) by providing answers to the questions raised above (about 1-3 sentences per question). You can describe a data science application that already exists today or think of applications that might arise in the near future. Feel free to add images to your write-up that illustrate your example (provide references to the original sources).

3 Processing Big Test Data (40 + 10* pts)

Depending on the data at hand, the application of a data science model can take a significant amount of time. This is, for instance, often the case in astronomy, where the "test sets" can be composed of millions or even billions of objects. In this exercise, you will process a subset of a real-world dataset that was used to find new, distant galaxies.

Download and the `HW1.zip` file from Absalon and extract it to the `LSDA` subdirectory of your home directory. The directory `HW1` should contain the following files and directories: `data`, `nn_chunks.py`, `nn_h5.py`, `nn_line.py`, `nn.py`. The `data` directory contains two csv files, `train.csv` and `test.csv`, which contain $n = 10,000$ and $m = 100,000$ lines and an additional comment line, respectively. Each line contains a pattern $\mathbf{x}$ with $d = 15$ features and a single real-valued label y (last value).

Have a look at the `nn.py` file (e.g., right-click on "Open With gedit"). The script (1) loads the training data, (2) instantiates a nearest neighbor regression model, (3) loads the test data, and (4) computes predictions. The nearest neighbor model is based on the *Scikit-Learn* package and resorts to the `brute` algorithm provided by the class to compute

nearest neighbors, which, in turn, computes a $n \times m$ distance matrix containing all pairwise distances between training and test points.[2]

1. Start the virtual machine and execute the Python file `nn.py` in the terminal. What's the output? Can you give a rough explanation for what's going on?

2. Adapt the `nn_line.py` file to process the `test.csv` file line by line (see comments in the code!). For each line, extract the relevant pattern, apply the model for this pattern, and append the predicted value to the `preds` list. What is the mean of all predictions (output of the last line)?

3. Next, process the test data in a similar fashion, but this time, consider chunks of size `chunk_size=1000`. Do not make use of any additional Python packages (i.e., no additional imports). What's the runtime for the testing phase? Measure the corresponding runtime for the line-by-line approach as well and compare the results. *Hint: Use a so-called Python generator function (see, e.g., `https://wiki.python.org/moin/Generators`).*

4. *Optional (10* pts):* Finally, let's have a look at a Python package called `h5py`, which depicts an interface to the HDF5 data format.[3] You can install the package on your virtual machine by executing the command `sudo apt-get install python-h5py` in a terminal (the password is the same as for logging in).

   (a) Execute the file `nn_h5.py`. This should convert the file `data/test.csv` to `data/test.csv.h5`. Compare the sizes of both files and explain the differences.

   (b) What are potential benefits of using the HDF5 format to store your data?

   (c) Learn more about `h5py` on your own and extend the file `nn_h5.py`: Reopen the store, retrieve the two datasets stored in the store, and process the data in chunks using the two datasets `X` and `y`. Process the data in chunks of 5,000 instances and use the tool `htop` to check the memory consumption during the execution (open second terminal during execution). What is the maximum amount of main memory used during the execution?

What to submit?

Provide all adapted source code files in the `code.zip` file. Include the following in your write-up (`answers.pdf`):

1. Output of the script `nn.py` when executing the script in the virtual machine (e.g., screenshot of terminal). Short explanation for what's going on.

2. The mean of all predictions outputted at the end.

3. Runtime of adapted `nn_chunks.py` in seconds (testing phase). Corresponding runtime for the line-by-line approach. Short comparison: Do you have an idea why one of the approaches needs less time?

4. *Optional (10* pts)*: (a) Comparison of both file sizes and brief explanation for differences (1-2 sentences), (b) key ideas of HDF5 format (3-4 sentences), and (c) maximum amount of memory used during execution (roughly, might vary a bit if executed multiple times).

[2] The documentation for the model can be found here: `http://scikit-learn.org/0.17/modules/generated/sklearn.neighbors.KNeighborsRegressor.html#sklearn.neighbors.KNeighborsRegressor`

[3] See, e.g., `http://www.h5py.org/` and `https://www.hdfgroup.org/why-hdf/`.

4 Regularized Least-Squares (20 pts)

We have had a quick look at the concept of *regularized least squares* during the lecture: Let $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$ be the set of training examples, $\mathbf{y} \in \mathbb{R}^n$ the labels y_i as column vector, and $\lambda > 0$. Then, the goal is to find a model $f \in \mathcal{H}$ in a hypothesis space $\mathcal{H}$ that minimizes

$$\underset{f \in \mathcal{H}}{\text{minimize}} \underbrace{\sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2}_{\text{Small Loss}} + \underbrace{\lambda \|f\|^2}_{\text{Small Complexity}} \tag{1}$$

We have considered models of the form $f(\mathbf{x}) = \sum_{j=1}^{n} c_j K(\mathbf{x}_j, \mathbf{x})$ with real-valued coefficients $\mathbf{c} \in \mathbb{R}^n$ and *kernel function* $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. In this case, optimal coefficients $\mathbf{c}^* \in \mathbb{R}^n$ can be obtained by computing $\mathbf{c}^* = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y}$, where $\mathbf{K} \in \mathbb{R}^{n \times n}$ is the associated positive-semidefinite kernel matrix and $\mathbf{I} \in \mathbb{R}^{n \times n}$ the identity matrix.

Typically, a good value for the model parameter $\lambda > 0$ is not known beforehand and has to be estimated given the training data (e.g., via cross-validation). Unfortunately, computing an optimal solution again and again for each value for λ can become very time-consuming. Can you speed up this process? Prove the following:

> *Assume that one is given the eigendecomposition $\mathbf{K} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$ of the symmetric and real-valued kernel matrix $\mathbf{K}$, where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{\Lambda}$ a diagonal matrix containing the eigenvectors (precomputed). Then, one can compute an optimal solution $\mathbf{c}^*$ for each new value for λ in $\mathcal{O}(n^2)$ time.*

A matrix $\mathbf{Q} \in \mathbb{R}^{n \times n}$ is *orthogonal* if $\mathbf{Q}^T\mathbf{Q} = \mathbf{Q}\mathbf{Q}^T = \mathbf{I}$ or, equivalently, $\mathbf{Q}^T = \mathbf{Q}^{-1}$. Such a decomposition always exists for $\mathbf{K}$ (see, e.g., page 393 in: Golub and Van Loan. *Matrix Computations*, 3rd edition, 1996).

Hints: Start with replacing the kernel matrix in $\mathbf{c}^* = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y}$ and make use of the properties of the involved matrices. You do not need any special details related to regularized least squares—it's all about matrix calculus!

What to submit?

Include the following in your write-up (`answers.pdf`):

1. Description your steps/algorithm and short analysis of the runtime needed.