Exam

Large-Scale Data Analysis

Fabian Gieseke, Ninh Pham, Christian Igel

8 June 2017, 8:00

Submission Deadline:
15 June 2017, 16:00

You have to submit your individual solution of the exam electronically via the **Digital Exam** system. The submission deadline is **June 15, 2017, 16:00**. The exam must be solved **individually**, i.e., you are **not allowed** to work in teams or to discuss the exam questions with other students. A solution consists of (1) a pdf file `answers.pdf` containing your answers and (2) a `code.zip` file containing the associated code (you do *not* have to include the original datasets). The correction will be done in an anonymous fashion; hence you should not mention your name somewhere. Instead, please **add your exam number** at the beginning of your `answers.pdf`.

WARNING: The goal of this exam is to evaluate your individual skills. We have to report any suspicion of cheating, in particular collaboration with other students, to the head of studies. Note that, if proven guilty, you may be expelled from the university. Please do not put yourself and your fellow students at this risk.

Feel free to ask questions via Absalon, but make sure that you do not reveal any significant parts of the solution. In doubt, just approach us directly via e-mail! Any additional hint given by us will be made available to all of you via Absalon. Some further comments:

1. *You **are allowed** to reuse the code that was made available to you via Absalon as well as the code you have developed within your team. Naturally, this will lead to common parts in the source code between you and your previous Absalon teammates. In case you reuse code snippets you have found on the internet, please make sure that you provide a reference to this external source.*

2. *We will generally assume that you make use of the Ubuntu virtual machine, i.e., all instructions provided are tailored to this system. You can use your own system to implement/test your code, but you have to make sure that the code you deliver runs on the virtual machine!*

3. *Please structure your code according to the different subtasks by, e.g., using one directory per subtask (e.g., `1_deeplearning`, `2_neighbors`, `3_lsh`, `4_trees`, and `5_hadoop`).*

4. *In case you notice any inaccuracies in the problem descriptions below, please let us know. If needed, we will provide updates and additional comments via Absalon. Thus, make sure that you check Absalon regularly!*

5. *Good luck! :-)*

1 Deep Learning: In a galaxy far, far away (25 pts)

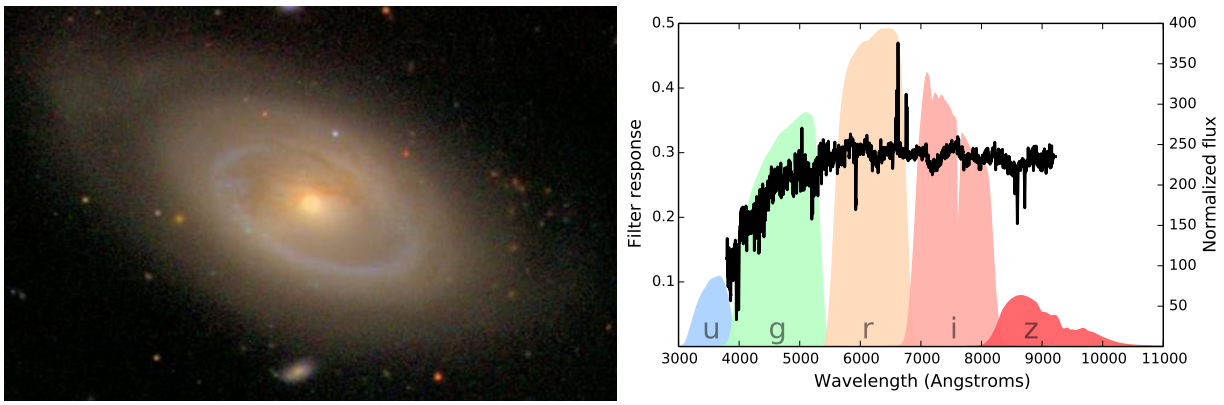
In astronomy, estimating distances to galaxies is of great significance. It allows us to get a 3D view of our Universe, which we can use to understand the distribution of matter. Also, since

light travels at finite speed, a larger distance means that we are looking further back in time, thus seeing how the Universe looked like billions of years ago. An accurate distance, therefore, translates to knowing the age of the Universe at the time the light was emitted.

Unfortunately, estimating distances is nontrivial. Distances are so great that light has been travelling millions or even billions of years before it hits our detectors. We cannot interact with the galaxies in any way, so we are left with no option but to infer what we can from the light. The distance of a galaxy can be estimated based on its *redshift*. Redshift is caused by the Doppler effect, which shifts the spectrum of an object towards longer wavelengths when it moves away from the observer. As the universe is expanding uniformly, we can infer the velocity of a galaxy by its redshift and, thus, its distance to Earth.

Inferring the redshift of a galaxy is relatively easy if we have a highly detailed spectrum of the light. However, if only images of the galaxy are available the task is much more challenging. By taking images of the galaxy in various band-pass filters and summing up all the light we receive in each filter, we can approximate a very coarse spectrum. Using this, we can estimate a "photometric redshift", that is, a redshift estimated from images.

In the data files you will find the integrated light of 10,000 galaxies in five different band-pass filters (termed u, g, r, i, and z, see Figure 1), measured in two different ways (the so-called model magnitudes and Petrosian magnitudes). You also get the derived colours, which are simply the subtraction of one magnitude from another. These can sometimes be informative.



(a) (b)

Figure 1: An example from the *Sloan Digital Sky Survey* (SDSS) (Aihara et al., 2011). (a) An image of the spiral galaxy NGC 5750. (b) Its associated spectrum overlapping the five photometric intensity band filters *u,g,r,i,z*.

You are provided the following data sets (via Digital Exam):

`LSDA2017GalaxiesTrain.csv`	4000 galaxies for training
`LSDA2017GalaxiesValidate.csv`	1000 galaxies for validation
`LSDA2017GalaxiesTest.csv`	5000 galaxies for testing

The last column in the files are the spectroscopically derived redshifts, which we regard as the "ground truth", that is, our target outputs or labels. The other columns are the input features. Use `LSDA2017GalaxiesTrain.csv` for training and `LSDA2017GalaxiesValidate.csv` for validation, that is, to select the solution network. `LSDA2017GalaxiesTest.csv` is used to evaluate the solution network, it must not be used in the training process.

1. *Variance (1 pt):* Compute the variance σ^2_{red} of the redshifts in the training data `LSDA2017GalaxiesTrain.csv` . The variance serves as a baseline for assessing the learning performance.

   Deliverables: Report variance of redshifts in the training data

2. *Deep network for regression (12 pts):* Modify your deep neural network from the "weed"-classification assignment for redshift prediction. Set the batch size to 128.

Use two hidden layers, with 64 and 32 neurons, respectively. Use the *mean squared-error* (based on `squared_difference`) as error function. Train using the data in `LSDA2017GalaxiesTrain.csv` and monitor the error on the validation data `LSDA2017GalaxiesValidate.csv`. Store the network with the lowest validation error (i.e., error on `LSDA2017GalaxiesValidate.csv`) you encounter in the training process. You need not check the validation error in every step. Evaluate the network with the lowest validation error on the test data in `LSDA2017GalaxiesTest.csv`.

Deliverables: Report the lines in the code you added and changed in the report; report and briefly discuss validation and testing error of the network with the lowest validation error; submit executable code.

3. *TensorBoard (12 pts):* Use TensorBoard to visualize the compute graph as well as the training and validation error.

   Deliverables: Include the plot of the compute graph in your report as well as the TensorBoard visualizations of the training and validation error.

Note: When you restart the program, you may need to delete the directory with the log data – and perhaps wait a short time – beforehand or to set a different directory.

2 Nearest Neighbors & Feature Selection (10 pts)

The first part of Homework 5 was about forward feature selection for nearest neighbor models. A related scheme is *backward feature selection*, which incrementally removes "bad" features. More precisely, in the first round, one selects the feature whose removal leads to the smallest error on a validation set. In the second round, after having removed the first feature, one identifies and removes a second feature. This process continues until a pre-defined number of features are left.

1. *Backward Selection (8 pts):* Consider the same scenario as in Homework 5 (datasets and parameters) and implement a corresponding backward selection scheme for the nearest neighbor regression model to select 5 features! Again, use the instances given in `validation.csv` to measure the validation errors of the intermediate models (via the mean squared error). Which features are left in the end? What is the test error of the induced final model (based on the instances given in `test.csv`)?

   Note: You can find a solution for the forward selection scheme on Absalon. You are allowed to adapt this code and/or reuse the code you have submitted as a team.

2. *Computation of Nearest Neighbors (2 pts):* Assume you are given another dataset with few training and validation instances in a high-dimensional space (e.g., $n = 1,000$ training, $m = 1,000$ validation instances, and $d = 100,000$ features). Furthermore, assume you would like to select 1000 features via backward selection and that you are interested in *exact* nearest neighbors during this process. Which of the following techniques would be, most likely, best suited for conducting all the involved nearest neighbor search: (a) Brute-force, (b) LSH, or (c) k-d trees? Justify your choice (1-3 sentences).

Deliverables: Provide all source code files in the `code.zip` file. Include answers to the questions raised above in your write-up (`answers.pdf`).

3 Locality-Sensitive Hashing (25 pts)

In class and homework assignment, we have seen how LSH works on approximately finding all pair similarity with Jaccard similarity. The time complexity of a brute force algorithm is $O(N^2)$ where N is the number of points. LSH tries to break the $O(N^2)$ barrier by only taking into account near pairs and pruning out most of far away pairs.

In the exam, you will investigate the LSH technique on the problem of **near neighbors reporting (NNR)** using *Jaccard* similarity. That is, *given a point set P of size N, a query point q and a threshold t, report **all** points $x \in P$ such that $J(x, q) \geq t$.* Due to the "curse of dimensionality", an algorithm for solving *exact* NNR in high dimensions needs $O(N)$ times per one query since it has to compute $O(N)$ similarity functions. In this work we will try to break the $O(N)$ barrier by allowing some approximation using LSH. That is, instead of returning the exact set of near neighbors, *approximate NNR* will return each similar point in P with high probability.

You need to write a Python program to solve both *exact* and *approximate* NNR on the "bag of words" data set from the UCI Repository[1] using the Jaccard similarity (same data as in Homework 2). We will make use the ***Enron Emails*** data set. We first partition the original data set into 2 sets: **the query set Q (the first 100 documents) and the point set P (all documents except the documents in the query set)**.

1. *Execute brute force algorithm (5 pts):* Given a threshold $t = 0.8$, for each query document $q \in Q$, report **all** near neighbors of q in P. That is, find all $x \in P$ such that the Jaccard similarity $J(x, q) \geq 0.8$. You need to report the running time of your bruteforce algorithm.

2. *Implement LSH framework to solve the approximate NNR (10 pts):* For each document $q \in Q$, we need to report its similar documents in P. You need choose suitable parameters, including number of bands and number of hash functions in one band such that each similar document, i.e. $J(x, q) \geq 0.8$, is reported with probability $\delta_1 \geq 0.9$ and dissimilar document, i.e. $J(x, q) \leq 0.4$, is reported with probability $\delta_2 \leq 0.01$.

3. *Verify the results returned by LSH (5 pts):* Compare the resulting accuracy and computational running time induced by brute force search and by LSH search. Explain the difference (3 - 4 sentences). What is the average number of examined documents for each method?

4. *Optimal choice of parameters (5 pts):* From theoretical aspect, what is the optimal choice of parameters to minimize the space usage given the scenario described above.

Deliverables: Provide all source code files in the `code.zip` file. Include answers to the questions raised in your write-up (`answers.pdf`).

4 Tree Ensembles for Huge Data (15 pts)

A popular strategy to build random forests for very large datasets in a distributed fashion is to consider relatively small random subsets of the data and to build one tree per such subset. This allows to generate tree ensembles in a distributed fashion (e.g., one worker node takes care of a single tree), while one makes, at the same time, use most of the available training data.

This part of the exam aims at constructing such tree ensembles. Instead of considering distributed computing scenarios, however, we focus on their construction using a single computer.

1. *Big Trees and Random Subsets (8 pts):* Implement a corresponding scheme for an execution on a single computer. More precisely, for each tree of the ensemble, consider a *new* random subset (uniform random subset without replacement) of size $M = 10,000$ that is extracted from all the available training instances. For each such subset, build a single classification tree (`DecisionTreeClassifier`) using the following parameters: `criterion='gini'`, `max_depth=None`, `min_samples_split=2`, `max_features=None`. Make use of the Landsat datasets that were provided for Homework 3 and build an ensemble consisting of 10 such trees using the training instances (`landsat_train.csv`). Afterwards, apply the model on the test instances (`landsat_test.csv`) and compute, for each class, the number of predictions made by the model for that class (the output varies depending on the random states used). What is the most dominant class?

   Note: You can assume that the number of training instances is known (i.e., n=25667779).

[1] https://archive.ics.uci.edu/ml/datasets/Bag+of+Words

2. *Runtime (3 pts):* Assume that you are given n training instances and that you build such an ensemble consisting of 100 trees. How much runtime is needed for the construction of a single tree, assuming that the corresponding random subset consists of $10,000$ instances and that this subset is already extracted/available in memory? Make use of the $\mathcal{O}$-notation and justify your answer (2-3 sentences).

3. *Detecting Rare Instances (4 pts):* An important problem in many domains is to detect "rare" classes. Assume that you are given a big training set with one billion training instances out of which 19 belong to class 1 (the "rare" class) and the remaining ones to class 0. Furthermore, assume that we build an ensemble of 100 trees based on random subsets of size $M = 10,000$ in the way described above. Would this ensemble be suited for detecting "rare" instances given new, incoming data? Justify your answer mathematically.

Deliverables: Provide all source code files in the `code.zip` file. Include answers to the questions raised above in your write-up (`answers.pdf`).

5 Hadoop (25 pts)

Make use of your virtual machine and the Hadoop cluster that is pre-installed. After having started your virtual machine, you need to start the Hadoop cluster by executing `./.start_hadoop` from your home directory (see the lecture slides). Afterwards, you will be able to submit jobs to the Hadoop cluster. If problems with the Hadoop cluster occur, restarting the virtual machine might help. Feel free to **get in touch with us via e-mail** if you have technical problems related to Hadoop (e.g., in case you do not manage to launch a Hadoop job using your virtual machine).

Note: For the following tasks, write efficient wrappers/combiners/reducers in the sense that only a constant amount of additional main memory is used by each of the workers. For instance, you can define a couple of "counters" (like for the word count example) or lists of constant size. An example for an inefficient reducer would be a reducer that simply collects all the flight distances per airline ID in a list to subsequently compute the minimal distance (this list could become very large depending on the amount of incoming data!). In doubt, feel free to get in touch with us!

1. *Airline Statistics (20 pts):* For this part, you will make use again of the airline dataset that you have worked with in the context of Homework 5 (`2016_1.csv`, ..., `2016_6.csv`). If not done already, copy these six files to a directory `airline_data` on the HDFS running on your virtual machine.

   (a) *Shortest Flight Distance (5 pts):* Write a mapper and reducer that compute the smallest flight distance per airline ID. The output generated by the reducer should contain, per line, an airline ID along with the associated minimal flight distance. What is minimal distance for airline ID 20436?

   (b) *Late Arrival Counts (5 pts):* Write a mapper and reducer that compute the total number of delayed flights per airline ID. Here, a delayed flight is one with a positive arrival delay (treat missing values as 0). The reducer should output, for each airline ID, the number of total flights, the number of delayed flights, and the percentage of delayed flights. What is the percentage of delayed flights for airline ID 20436?

   (c) *Mean and Standard Deviation for Arrival Delay (5 pts):* Write a mapper and reducer that compute the mean $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and the standard deviation $s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$ for the arrival delays $x_1, \ldots, x_n$ for each airline ID. Consider all delay values including the negative ones (treat missing values as 0). Use a combiner to reduce the work that has to be done by the reducer. A reducer should output, for each airline ID, the associated mean and standard deviation. What is the mean and standard deviation for airline ID 20436?

   (d) *Top-10 of Arrival Delays (5 pts):* Write a mapper and reducer that compute the 10 most delayed flights for each airline ID. Make use of a combiner to lessen the data

that has to be processed by the reducer. The reducer should output, for each airline ID, a list containing the delays for the 10 most delayed flights. What are the 10 most delayed flights for airline ID 20436?

Deliverables: Provide, for each subtask, the mapper.py/combiner.py/reducer.py files in a subdirectory (e.g., `1a_shortest_flight_distance`) of your overall source code archive. Include answers to the questions raised above as well as the content of the Hadoop output files (e.g., `part-00000`) in your write-up (`answers.pdf`).

2. *Landsat Statistics (5 pts):* Write a MapReduce job that computes the class counts w.r.t. the predictions made by the random forest model that you have used for Homework 3. This task is similar to task 2.3 of Homework 5, but this time, we are interested in the class counts (i.e., "How many water or grassland predictions are made by the model?"). Each mapper should compute predictions for all incoming data by loading the model from the file `model.save`, which is stored on the local file system of the worker node.[2] The reducer should output the class counts, i.e., for each class, the number of predictions belonging to that class.

   Compute the class counts for the predictions made by the model on all test instances (`landsat_test.csv`). Using a mapper that is similar to the one provided in the reference solution for Homework 5 (see Absalon) will take a very long time! Your task is to write an *efficient wrapper/combiner/reducer*. Execute the job on the Hadoop cluster! The overall execution should take at most a couple of minutes (this naturally depends on your particular hardware; it should not take more than, say, 30 minutes).

   Hint: In case the Hadoop execution causes errors, restarting the virtual machine might help.

   Deliverables: Provide the mapper.py/combiner.py/reducer.py files in a subdirectory of your overall source code archive. Add the output generated by the Hadoop job (both the log messages as well as the `part-00000` file) as well as a brief explanation for your implementation choices ("Why is your code faster now?") to your write-up (`answers.pdf`).

References

H. Aihara, C. A. Prieto, D. An, S. F. Anderson, É. Aubourg, E. Balbinot, T. C. Beers, A. A. Berlind, S. J. Bickerton, D. Bizyaev, et al. The eighth data release of the Sloan digital sky survey: first data from SDSS-III. *The Astrophysical Journal Supplement Series*, 193(2):29, 2011.

[2] A reference solution for Homework 3 is available on Absalon. Executing the first two files should yield the `model.save` file. Since the Hadoop cluster is a pseudo-distributed cluster that runs on a single machine, you can simply specify the absolute path to this file in the mapper (e.g., `/home/lsda/exam/hadoop/forests/model.save`).