Homework 2

Large-Scale Data Analysis

(individual submission)

Fabian Gieseke & Christian Igel
Department of Computer Science
University of Copenhagen

30 April 2019

Submission Deadline:
14 May 2019, 23:59

This assignment is an **individual assignment**, i.e., you are supposed to solve the assignment on your own. In particular, the report and the code must be written completely by yourself!

Virtual Machine: For the individiual assignments, we will generally assume that you make use of the LSDA 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 installed packages.
Important: You have to make sure that your code runs on the virtual machine!

Deliverables: Please submit *two separate files* via Absalon: (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 is 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 Fun with Runtimes (20 pts)

1. **(8 pts):** Rank the following functions (slowest to fastest) by their order of growth, i.e., by their asymptotic behavior. Use "$\leq$" to indicate the order between two functions.

   - $f_1(n) = \max(n^4, 10^{100})$
   - $f_2(n) = n^{10} 2^n$
   - $f_3(n) = 0.99 \cdot n^2$
   - $f_4(n) = n^{0.99}$
   - $f_5(n) = 2^n$
   - $f_6(n) = n^3 \log_2 n$
   - $f_7(n) = 10^{100} \cdot n + 1$
   - $f_8(n) = n(\log_2 n)^{10}$

   Example: Given the three functions $f(n) = 10000$, $g(n) = n^2$, and $h(n) = n$, the ranking would be $10000 \leq n \leq n^2$ since $f \in \mathcal{O}(h)$ and $h \in \mathcal{O}(g)$.

2. **(2 pts):** Prove or disprove: $2^{2n} \in \mathcal{O}(2 \cdot 4^n - n^4)$

3. **(5 pts):** Prove that $\sum_{i=1}^{n} (\log_2 i)^2 \in \mathcal{O}(n(\log_2 n)^2)$.

4. **(5 pts):** Let $T(n)$ denote the runtime of an algorithm (in the Real-RAM model of computation) and let $T(n)$ be given as

$$T(n) = \begin{cases} 4 \cdot T(\frac{n}{4}) & \text{for } n \geq 4 \\ 1 & \text{otherwise} \end{cases}$$

Prove that $T(n) \in \mathcal{O}(n)$. Without loss of generality, you can assume that n is a power of 4.

Deliverables

Include the following in your write-up (`answers.pdf`): For 1., an ordered list of functions (slowest to fastest). For 2.-4., short proofs.

2 Nearest Neighbor Search (30 pts)

We have discussed how to accelerate nearest neighbor search using k-d trees. In this exercise, you are supposed to extend a k-d tree implementation given in the `NearestNeighbors.ipynb` Jupyter notebook that is provided to you. For all experiments, keep a leaf size of 20 and search for $k = 3$ nearest neighbors.

1. **Nearest Neighbor Classifier (10 pts):** Implement a nearest neighbor classifier from scratch that makes use of the provided k-d tree implementation to find the corresponding nearest neighbors. In the notebook, you already find the structure of the Python class (`NearestNeighborClassifier`) you are supposed to extend. The k-d tree shall be constructed during the training phase. Apply your nearest neighbor classifier to the covertype dataset; make use of the first 10 features only (see notebook).

   (a) Make use of the `time` package to measure the runtime needed for the `query` function of the k-d tree implementation. What is the runtime needed for the nearest neighbor search conducted during the `predict` phase?

   (b) What is the classification accuracy on the test set?

2. **Early Stopping (10 pts):** A simple way to accelerate the nearest neighbor search is to stop the process as soon as a fixed number of leaves has been visited per query instance.

   (a) Implement this variant by extending the query function of the k-d tree, i.e., `def query(self, X, k=1, max_leaves=None)`. In case `max_leaves=None`, the "normal" query process without early stopping should take place. In case `max_leaves=L` is an integer, then at most L leaves should be visited per query. Make use of $L = 10$ and apply the nearest neighbor model again. What is the runtime needed for the nearest neighbor search? What is the accuracy on the test set?

   (b) Are the nearest neighbor answers still correct/exact? Argue why they are still correct/why they are not correct anymore.

   (c) What is the worst case asymptotic runtime ($\mathcal{O}$-notation) per query for this variant given a training set of n points for which the tree was built? Prove your statement.

3. **Distance-Based Pruning (10 pts):** Another way to accelerate the search is to stop the "backtracking" earlier: Let d denote the distance of the query point to the splitting hyperplane and let r denote the distance of the current best k-th nearest neighbor candidate to the query point (`neighbours.get_max_dist()` in the code). Usually, the second subtree is not visited in case $r < d$. Instead, we will now ignore the second subtree in case $\frac{r}{\alpha} < d$ for some user-defined $\alpha > 1$.

   (a) Implement this variant by extending the `query` function of the k-d tree implementation to `def query(self, X, k=1, max_leaves=None, alpha=1.0)`. Apply your nearest neighbor model again using $\alpha = 2.0$. What is the induced query runtime? What is the induced classification accuracy?

(b) Are the nearest neighbor answers still correct/exact? Can you make any statements about the quality of the points returned by this variant?

Deliverables

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

1. Report (a) runtime and (b) classification accuracy.
2. Report (a) runtime and classification accuracy, (b) answer to the question with short explanation (3-5 lines), and (c) asymptotic runtime along with short proof (3-5 lines).
3. Report (a) runtime and classification accuracy and (b) answers to both questions (3-5 lines each).

3 Sparse Least-Squares Support Vector Machines (25 pts)

In this exercise, you will work on a large text data set that was extracted from a collection of UseNet articles. The `aut-avn` dataset contains about $n = 71175$ instances in a $d = 20707$-dimensional feature space and the task is to discriminate instances from two classes, "simulated auto racing" vs. "simulated aviation" [1]. The dataset is sparse, i.e., most of the feature values are zero. Such datasets can be stored efficiently by only keeping track of the non-zero features.

We have discussed regularized least-squares models, which are typically used in the context of regression scenarios. However, these models can also used for classification problems. In particular, for labels $y_i \in \{-1, +1\}$, the corresponding models are also called *least-squares support vector machines* and are of the form

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

with hypothesis space $\mathcal{H}$. It is worth mentioning that the square loss is, in general, not an ideal choice for classification scenarios (it has been used extensively for classification tasks though).

In this exercise, you will implement such a least-squares support vector machine for a linear kernel function. More precisely, you will resort to a gradient-based optimizer, which makes use of function and gradient calls to find a good solution for the model parameter $\mathbf{c} \in \mathbb{R}^n$.

1. **Dense Data Matrix (5 pts):** Assume you would simply store the data set in a dense data matrix of the form $\mathbf{X} \in \mathbb{R}^{n \times d}$. Would this data matrix fit into the main memory (2 GB) of your virtual machine assuming that each value is stored as a float (double precision)?
2. **Getting Started (5 pts):** Have a look at the `Sparse Least-Squares SVMs.ipynb` notebook. The data matrix $\mathbf{X}$ object is of type `scipy.sparse.csc.csc_matrix`. Read about this data format and determine the number of nonzero entries stored in $\mathbf{X}$.
3. **Gradient-Based Optimization (15 pts):** Computing the optimal coefficient vector $\mathbf{c}^* \in \mathbb{R}^n$ by resorting to a linear system of equations is computationally challenging in this case. Instead, you will resort to gradient-based optimization. Have a look at the notebook and the instructions provided. The code resorts to the function `fmin_l_bfgs_b` of the `scipy.optimize` package. The details of this optimizer are not of importance for this task. In a nutshell, all you have to do is to implement two functions, one that computes the function values and another one computing the gradient for a given vector $\mathbf{c} \in \mathbb{R}^n$.
   - Your task is to implement both functions such that each function/gradient evaluation takes $\mathcal{O}(s + n)$ runtime and $\mathcal{O}(s + n)$ additional main memory.
   - Note that for the linear kernel, the kernel matrix $\mathbf{K}$ can be computed via $\mathbf{K} = \mathbf{X}\mathbf{X}^T$.

- Also implement the `predict` function efficiently. Note that you need to convert the real-valued predictions to $\{-1, +1\}$: For $f(\mathbf{x}) >= 0$, return $+1$. Otherwise, -1.

Argue why your implementation is efficient (i.e., provide, for each step of your function/gradient evaluations, a quick runtime analysis). Execute the whole notebook to train a sparse least-squares support vector machine for $\lambda = 0.001$. What is the classification on the test set?

Hints: You are supposed to make use of sparse matrix operations provided by the `scipy` *package. For the runtime analyses, you do not have to check the underlying implementations provided by this package. Instead, you can simply use the corresponding operations and sketch how you would implement them to obtain an asymptotically efficient approach (2-3 lines).*

Deliverables

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

1. Argument why the data will fit/will not fit into the main memory of the virtual machine.
2. Number of non-zero elements stored in the data matrix.
3. Short explaination why your function and gradient calls need, per call, at most $\mathcal{O}(s + n)$ runtime and $\mathcal{O}(s + n)$ additional main memory. Provide a similar argumentation for the predict function. Provide the classification accuracy on the test set.

4 Tree Ensembles (25 pts)

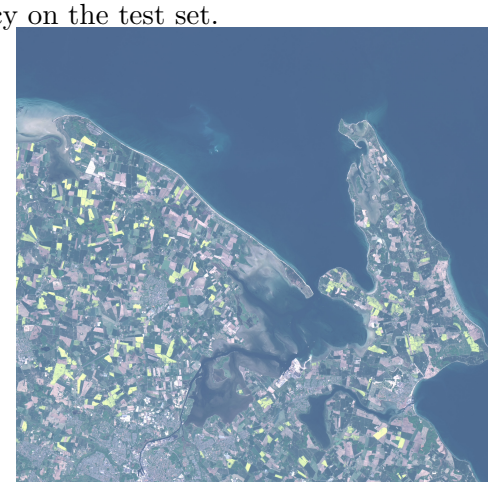

Figure 1: Landsat 8 image (true color)

In this exercise, you will deal with data from the *Landsat 8* satellite. The satellite takes images using *multiple bands*, which yields so-called *multispectral images* with multiple values given for each pixel. Using such multispectral images, one can generate "normal" true color images, see Figure 1.[1]

We will make use of three preprocessed Landsat 8 datasets, which are available via Absalon: `landsat_train.csv`, `landsat_validation.csv`, and `landsat_test.csv`. The training file contains $n = 5,000,000$ lines. Each line contains a label (first value) and $d = 9$ features separated via commas (each row corresponds to a pixel in an associated multispectral image). The validation file is of the same form and contains $1,335,558$ instances. The test file contains $9,000,000$ lines each 9 features (i.e., no label).

1. **Random Forests (10 pts):** Make use of the Scikit-Learn package and train a standard random forest on the training data (use `RandomForestClassifier(n_estimators=10, max_features=None)` to initialize the model). Afterwards, compute the validation accuracy using the instances given in `landsat_validation.csv`. Finally, apply the model to all instances given in `landsat_test.csv` and visualize the predictions (e.g., one color per class). Here, each line of `landsat_test.csv` corresponds to a pixel in a 3000×3000 image, where the first 3000 lines correspond to the first row, the following 3000 ones to the second row, and so on. Add the resulting image to your write-up.

2. **Random Forests with Restricted Trees (15 pts):** Consider the following variant of building a random forest: Instead of randomly sampling features at each node, one samples one subset of features for each tree, and builds the tree on these features.

[1] If you are interested, you can check out the details related to this type of data, but it's not needed for doing the assignment: `https://landsat.gsfc.nasa.gov/landsat-data-continuity-mission/`

(a) Implement this variant and build a tree ensemble consisting of 10 such trees. Each tree should be built using a *new* subset of 2 random features (uniform at random, without replacement). Use the `DecisionTreeClassifier` class provided by Skikit-Learn to build the individual trees (stick to the default parameters).

(b) What is the validation error for this variant? Why is the performance better/worse compared to the normal random forest?

Deliverables

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

1. Classification accuracy of the random forest on the validation set. Visualization of the predictions made by the random forest on the test set (3000×3000 pixels image).

2. Classification accuracy of the tree ensemble variant on the validation set. Argumentation why the accuracy is better/worse compared to the previous random forest (4-5 lines).

References

[1] V. Sindhwani and S. S. Keerthi. Large scale semi-supervised linear svms. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 477–484, New York, NY, USA, 2006. ACM.