Machine Learning 2019-2020

Home Assignment 1

Yevgeny Seldin **Christian Igel**
Department of Computer Science
University of Copenhagen

The deadline for this assignment is **26 November 2019 22:00**. You must submit your *individual* solution electronically via the Absalon home page.

A solution consists of:

- A PDF file with detailed answers to the questions, which may include graphs and tables if needed. Do *not* include your source code in the PDF file.
- A .zip file with all your solution source code (Matlab / R / Python scripts / C / C++ / Java / etc.) with comments about the major steps involved in each question (see below). Source code must be submitted in the original file format, not as PDF.
- IMPORTANT: Do NOT zip the PDF file, since zipped files cannot be opened in the speed grader. Zipped pdf submissions will not be graded.
- Your PDF report should be self-sufficient. I.e., it should be possible to grade it without opening the .zip file. We do not guarantee opening the .zip file when grading.
- Your code should be structured such that there is one main file (or one main file per question) that we can run to reproduce all the results presented in your report. This main file can, if you like, call other files with functions, classes, etc.
- Your code should include a README text file describing how to compile and run your program, as well as a list of all relevant libraries needed for compiling or using your code.
- Handwritten solutions will not be accepted, please use the provided latex template to write your report.

1 Make Your Own (5 points)

Imagine that you would like to write a learning algorithm that would predict the final grade of a student in the Machine Learning course based on his profile, for example, his grades in prior courses, his study program, etc. Such an algorithm would have been extremely useful: we could save significant time on grading and predict the final grade when the student just signs up for the course. We expect that the students would also appreciate such service and avoid all the worries about their grades. Anyhow,

1. What profile information would you collect and what would be the sample space $\mathcal{X}$?
2. What would be the label space $\mathcal{Y}$?
3. How would you define the loss function $\ell(Y', Y)$?
4. How would you evaluate the performance of your algorithm? (In terms of the loss function you have defined earlier.)
5. Assuming that you have achieved excellent performance and decided to deploy the algorithm, would you expect any issues coming up? How could you alleviate them?

There is no single right answer to the question. The main purpose is to help you digest the definitions we are working with. Your answer should be short, no more than 2-3 sentences for each bullet point. For example, it is sufficient to mention 2-3 items for the profile information, you should not make a page-long list.

2 Illustration of Markov's and Chebyshev's Inequalities (25 points)

1. Make 1,000,000 repetitions of the experiment of drawing 20 i.i.d. Bernoulli random variables $X_1, \ldots, X_{20}$ (20 coins) with bias $\frac{1}{2}$.
2. Plot the empirical frequency of observing $\frac{1}{20}\sum_{i=1}^{20} X_i \geq \alpha$ for $\alpha \in (0.5, 0.55, 0.6, \ldots, 0.95, 1)$.
3. Explain why the above granularity of α is sufficient. I.e., why, for example, taking $\alpha = 0.51$ will not provide any extra information about the experiment.

4. In the same figure plot the Markov's bound[1] on $\mathbb{P}\big(\frac{1}{20}\sum_{i=1}^{20} X_i \geq \alpha\big)$.

5. In the same figure plot the Chebyshev's bound[2] on $\mathbb{P}\big(\frac{1}{20}\sum_{i=1}^{20} X_i \geq \alpha\big)$. (You may have a problem calculating the bound for some values of α. In that case and whenever the bound exceeds 1, replace it with the trivial bound of 1, because we know that probabilities are always bounded by 1.)

6. Compare the three plots.

7. For $\alpha = 1$ and $\alpha = 0.95$ calculate the exact probability $\mathbb{P}\big(\frac{1}{20}\sum_{i=1}^{20} X_i \geq \alpha\big)$. (No need to add this one to the plot.)

Do not forget to put axis labels and a legend in your plot!

We will get back to this question in Home Assignment 2, so it will pay off if you can reuse your code easily.

3 Tightness of Markov's Inequality (5 points)

In the previous question you have seen that Markov's inequality may be quite loose. In this question we will show that in some situations it is actually tight. Let ε^* be fixed. Design an example of a random variable X for which

$$\mathbb{P}(X \geq \varepsilon^*) = \frac{\mathbb{E}[X]}{\varepsilon^*}.$$

Prove that the above equality holds for your random variable.

Hint: It is possible to design an example satisfying the above requirement with a random variable X that accepts just two possible values. What should be the values and the probabilities that X accepts these values?

4 Digits Classification with Nearest Neighbors (30 points)

In this question you will implement and apply Nearest Neighbors learning algorithm to classify handwritten digits.

[1] Markov's bound is the right hand side of Markov's inequality.

[2] Chebyshev's bound is the right hand side of Chebyshev's inequality.

Preparation

- Download `MNIST-cropped-txt.zip` file from Absalon. The file contains `MNIST-Train-cropped.txt`, `MNIST-Test-cropped.txt`, `MNIST-Train-Labels-cropped.txt`, and `MNIST-Test-Labels-cropped.txt` files.
- `MNIST-Train-cropped.txt` is a space-separated file of real numbers. It contains a 784×10000 matrix, written column-by-column (the first 784 numbers in the file correspond to the first column; the next 784 numbers are the second column, and so on).
- Each column in the matrix above is a 28×28 grayscale image of a digit, stored column-by-column (the first 28 out of 784 values correspond to the first column of the 28×28 image, the next 28 values correspond to the second column, and so on). Test yourself: reshape the first column into a 28×28 matrix and display it as an image - did you get an image of digit "5"?
- `MNIST-Train-Labels-cropped.txt` is a space-separated file of 10000 integers. The numbers label the images in `MNIST-Train-cropped.txt` file: the first number ("5") is the number drawn in the image corresponding to the first column; the second number corresponds to the second column, and so on.
- `MNIST-Test-cropped.txt` is a space-separated file of real numbers containing 784×2000 matrix of test images.
- `MNIST-Test-Labels-cropped.txt` is a space-separated file of 2000 integers labeling the images in `MNIST-Test-cropped.txt` file.

Tasks

1. Compare K-NN algorithms for $K = 1, 3, 5, 7, \ldots, 33$ in their ability to distinguish between digits "0" and "1".
2. Repeat the same for "0" and "8".
3. Repeat the same for "5" and "6".

Detailed Instructions

- For each of the tasks above pick the images of the corresponding digit from `MNIST-Train-cropped.txt` file for training.

- Pick the images of the corresponding digit from `MNIST-Test-cropped.txt` for the test set.
- **Validation**: use the first 80% of the training images for training and the last 20% of the images for calculating the validation error. Plot the validation error as a function of K and compare it with the test error. Test error should be calculated on the test set (as described above).
  - How well does the validation error match the test error?
  - How closely does the best value of K according to the validation error match the best value of K according to the test error?
  - How the validation and test errors behave as a function of K?
  - Does the best value of K depend on the difficulty of the task and how? (It is easier to tell apart "0" and "1" than "5" and "6"; the difficulty of separating "0" and "8" should be somewhere in between.)

Practical Details and Some Practical Advice

- Use square Euclidean distance to calculate the distance between the images. If $\mathbf{x}_1$ and $\mathbf{x}_2$ are two 784-long vectors representing two images, then the distance is $\|\mathbf{x}_1 - \mathbf{x}_2\|^2 = (\mathbf{x}_1 - \mathbf{x}_2)^T(\mathbf{x}_1 - \mathbf{x}_2)$.
- If you work with an interpreted programming language, such as Python or MATLAB, do your best to use vector operations and avoid for-loops as much as you can. This will make your code orders of magnitude faster.
- Assume that $\mathbf{X} = \begin{pmatrix} \begin{pmatrix} | \\ \mathbf{x}_1 \\ | \end{pmatrix}, \cdots, \begin{pmatrix} | \\ \mathbf{x}_n \\ | \end{pmatrix} \end{pmatrix}$ is a matrix holding data vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and you want to calculate distances between all these points and a test point $\mathbf{x}$. *Do your best to avoid a for-loop!* One way of doing so is to create another matrix $\mathbf{X}' = \begin{pmatrix} \begin{pmatrix} | \\ \mathbf{x} \\ | \end{pmatrix}, \cdots, \begin{pmatrix} | \\ \mathbf{x} \\ | \end{pmatrix} \end{pmatrix}$ and calculate all n distances in one shot using matrix and vector operations.
- Note that you can compute the output of K-NN for all K in one shot using vector operations.
- It is possible to write the whole exercise with a single for-loop over the test/validation set.
- You may find the following functions useful:

  - Built-in sorting functions for sorting the distances.
  - Built-in functions for computing a cumulative sum of elements of a vector `v` (for computing the predictions of K-NN for all K at once).
- It may be a good idea to debug your code with a small subset of the data.

Deliverables For each of the three tasks you should include in your `.pdf` report:

- A plot of validation and test error.
- Do not forget to add titles, axis labels, and legends to your plots! We will deduct points if they are missing!

Your `.pdf` report should also include:

- Discussion of the three plots according to the guidelines provided in the exercise.

Include all your code in the accompanying `.zip` file.

As a general rule, you must not use the test data in the model building process at all (neither for training, data normalization, nor hyperparameter selection), because otherwise you may get a biased estimate of the generalization performance of the model (see Abu-Mostafa et al. (2012) Example 5.3).

5 Nearest Neighbors for Multiclass Classification (5 points)

In the previous question we did some cheating and in fact we did not classify the digits, but instead solved a bunch of binary classification problems, where we distinguished between pairs of digits (because binary classification is what we have learned in class). The complete solution is known as *multiclass classification* and in this particular case the label space is $\mathcal{Y} = \{0, 1, \ldots, 9\}$. Describe how you would modify the K-Nearest Neighbors algorithm from binary classification to multiclass classification? You do not have to implement the algorithm, a high-level pseudo-code, like Algorithm 1 in the lecture notes, is sufficient.

6 Linear regression (30 points)

Consider the data in the file `DanWood.dt`, which contains measurements from an experiment originally described by Keeping (1962) and used in several textbooks

(Daniel and Wood, 1980, Shiflet and Shiflet, 2006). The data is one of the statistical reference data sets of the US American *National Institute of Standards and Technology*.

The experiment studies a carbon filament lamp. For a given absolute temperature of the carbon filament the energy radiated from the filament was recorded. Temperature was measured in units of 1000 kelvin and energy radiation per cm^2 per second.

Each line in `DanWood.dt` is one training pattern. The first number is the input (x), the absolute temperature, and the second is the corresponding output / target / label (y), the radiated energy.

Tasks

1. Implement the linear regression algorithm as described in the lecture (although using the pseudo-inverse is not the best way, e.g., via **QR**-decomposition would be preferable). For vector and matrix operations, such as computing the inverse of a matrix, you can use high-level (library) functions.

2. Build an *affine* linear model $h : \mathbb{R} \to \mathbb{R}$ of the data described above using linear regression. Report the two parameters of the model as well as the mean-squared-error of the model computed over the complete data set.

3. Plot the data and the regression line. The plot must have proper axis labels and a legend.

4. Compute and report the variance[3] of the labels in the training data set (i.e., the variance of the y values). Now compare the variance to the mean-squared-error of your model. Discuss the quotient of the two quantities, the mean-squared-error over the variance. What does it mean if the quotient is larger or smaller than one?

5. Let us build a non-linear model

$$h(x) = ax^3 + b$$

   with $a, b \in \mathbb{R}$ minimizing the mean-squared-error. This is the same as applying the non-linear input transformation $x \mapsto x^3$ to the data and then performing linear regression with the transformed inputs (one can say that the inputs x are mapped to a feature space $\mathcal{Z}$ by the *feature map* $\phi(x) = x^3$). Do so, report the mean-squared-error, and plot the target and the regression line over the transformed inputs.

[3]Consider the uncorrected sample variance.

Deliverables: source code, plot of the data and the regression line, mean-squared error, parameters of the regression model, variance of y values, discussion of mean-squared-error over the variance; mean-squared error of the non-linear model; plot of the transformed data and the regression line

References

Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin. *Learning from Data*. AMLbook, 2012.

C. Daniel and F. S. Wood. *Fitting Equations to Data*, pages 428–431. John Wiley and Sons, 1980.

E. S. Keeping. *Introduction to Statistical Inference*, page 354. Van Nostrand Company, Princeton, NJ, 1962.

A. B. Shiflet and G. W. Shiflet. *Introduction to Computational Science: Modeling and Simulation for the Sciences*. Princeton University, 2006.