Machine Learning A

2021-2022

Final Exam

Christian Igel **Fabian Gieseke** **Yevgeny Seldin** **Sadegh Talebi**
Department of Computer Science
University of Copenhagen

You must submit your individual solution of the exam electronically via the **Digital Exam / Digital Eksamen** system. The deadline for submitting the exam is **Friday, 5 November 2021, at 16:00**. The exam must be solved **individually**. You are **not allowed** to work in groups or discuss the exam questions with other students. For fairness reasons any questions about the exam will be answered on Absalon. If your question may reveal the answer to other students, please, email it personally to the lecturers and we will either answer it on Absalon or tell you that we cannot answer your question.

WARNING: The goal of the exam is to evaluate your personal achievements in the course. We believe that take-home exams are most suitable for this evaluation, because they allow to test both theoretical and practical skills. However, our ability to give take-home exams strongly depends on your honesty. Therefore, any suspicion of cheating, in particular collaboration with other students, will be directly reported to the head of studies and prosecuted in the strictest possible way. It is also strictly prohibited to post the exam questions or parts thereof on the Internet or on discussion forums and to seek help on discussion forums. And you are not allowed to store your solutions in open access version control repositories or to post them on the Internet or on discussion forums. Be aware that if proven guilty you may be expelled from the university. Do not put yourself and your fellow students at risk.

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 this PDF file. Do *not* include the task description or parts thereof in your report.
- Please, use the provided LaTeX template for typing your report. Hand-

written solutions will not be accepted.

- Your solution source code 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.
- Your code should be structured such that there is one main file 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 also include a README text file describing how to run your program.

1 Alzheimer's disease diagnosis (20 points)

This section of the exam considers the task of automatic diagnosis of Alzheimer's disease. The data are a subset of the data analyzed by Sørensen et al. (2016, 2017), and we refer to these two studies for further background information.

Alzheimer's disease (AD) is a severe neurodegenerative disease. The vast majority of AD diagnoses are uncertain. Often, a magnetic resonance imaging (MRI) scan is part of the diagnosis, but only to rule out other causes of the symptoms. However, MRI scans seem to carry much more quantitative and diagnostic information. A system developed by DIKU and the DIKU spin-off company Biomediq can separate cognitively normal, dementia, and pre-dementia patients with high accuracy when incorporating MRI data, as demonstrated by winning the MICCAI CADDementia Grand Challenge in 2014 (Sørensen et al., 2016, 2017). Here we consider data preprocessed in a similar way as in that system. However, we consider features extracted from MRI only and restrict the analysis to two classes, healthy and AD.

1.1 Data understanding and preprocessing

Download and extract the data.

Consider the training data `trainInput.csv` and the corresponding labels `trainTarget.csv` as well as the test data `testInput.csv` and the corresponding labels `testTarget.csv`.

The ith row in `trainInput.csv` are the features of the ith training pattern. The class label of the ith pattern is given in the ith row of `trainTarget.csv`. Analogously for the test data.

Report the class frequencies, that is, for each of the 2 classes report the number of data points divided by the total number of data points) for both the training and test data.

Deliverables: description of software used; frequency of classes

1.2 Principal component analysis

Perform a principal component analysis of the training data `trainInput.csv`. Plot the eigenspectrum (eigenvalue vs. rank of corresponding eigenvector after sorting by eigenvalue; be careful not to mix up singular values and eigenvalues). How many components are necessary to "explain 90 % of the variance"? Visualize the data by a scatter plot of the data projected on the first two principal components. Use different colors for the different classes in the plot.

Deliverables: description of software used; plot of the eigenspectrum; number of components necessary to explain 90 % of variance; scatter plot of the data projected on the first two principal components with different colors indicating the 2 different classes

1.3 Clustering

Perform 2-means clustering of `trainInput.csv`. After that, project the cluster centers to the first two principal components of the training data. Then visualize the clusters by adding the cluster centers to the plot from the previous exercise. Briefly discuss the results: Did you get meaningful clusters? Initialize the cluster centers with training data points from different classes. Take the *first data point* from each class you can find in `trainInput.csv` (i.e., the first class center is the data point in the very first line in the file).[1]

Deliverables: description of software used; one plot with cluster centers and data points; short discussion of results

1.4 Classification

The task is to evaluate several classifiers on the data. Build the models using the training data only. The test data must only be used for final evaluation.

1. Apply multi-nominal logistic regression. If you use regularization, describe the type of regularization you used. Report training and test loss (in terms of 0-1 loss).

[1] This is done in order to simplify the grading. It is in general not necessarily a good idea to take the first occurence.

Figure 1: Examples from the traffic sign data set.

2. Apply random forests with 200 trees. In one setting, set the number of features considered when looking for the best split to the square root of the total number of features. In a second setting, set the number of features considered when looking for the best split to the total number of features. Report training and test loss (in terms of 0-1 loss) and the out-of-bag (OOB) *error*.

3. Apply k-nearest-neighbor classification. Use cross-validation to determine the number of neighbors. Report training and test loss (in terms of 0-1 loss). Describe how you determined the number of neighbors.

Deliverables: description of software used; training and test errors; OOB errors for the random forests; description of regularization and model selection process

2 Data augmentation (20 points)

We have a second look at the traffic sign recognition task (Stallkamp et al., 2012) considered in Assignment 6, see Figure 1.

Convolutional neural networks are highly complex models. To reduce the risk of overfitting and in order to be able to learn difficult tasks with a high input variability, many training examples are needed. *Data augmentation* is a way to enlarge the training data set by artificial training points generated from the available data. Data augmentation refers to applying transformations to the input data – the images – that do not change their labels. If we know, for example, that the classifications of images do not change when the images are rotated, then showing rotated versions of the images during the training process helps to learn this invariance property. In the context of neural networks, this type of data augmentation can be traced back to Baird (1992). It has been used successfully for CNNs, for instance already in the influential work by Krizhevsky et al. (2012) The learning goal of this part of the assignment is to get more experience in using data augmentation.

Inspect the notebook `Torch_Traffic_Signs_Basic_Template.ipynb`.

Deliverables: Answer the following questions briefly in your report: Which transformations are applied to the input images? Why is a transformation conditioned on the label?

Please add at least one additional (not completely nonsensical) transformation. *Deliverables:* Show the corresponding code in your report and briefly explain why you think that this may be a reasonable augmentation given the task.

3 Efficient use of data (20 points)

So far, given a data set S, in all our theoretical results we used part of the data set for training prediction models and another part for validating them. This way some data are only used for training and some data are only used for validation. In this question you will analyse a more efficient approach for using the data, where all data are used both for training and for validation. It is inspired by cross-validation, but unlike cross-validation, which is a heuristic providing no guarantees against overfitting, the approach that you will develop will come with theoretical justification.

So, we have a data set S of size n. We split the data set into two equal halves, $S = S_0 \cup S_1$. We train M models $\{h_{0,1}, \dots, h_{0,M}\}$ on the first half of the data and validate them on the remaining half. Let $\hat{L}(h_{0,i}, S_1)$ for $i \in \{1, \dots, M\}$ be the corresponding validation losses. Then we train another M models $\{h_{1,1}, \dots, h_{1,M}\}$ on the second half of the data and validate them on the first half. Let $\hat{L}(h_{1,i}, S_0)$ for $i \in \{1, \dots, M\}$ be the corresponding validation losses. Finally, we select the model $h_{j^*,i^*} = \arg \min_{j \in \{0,1\}, i \in \{1,\dots,M\}} \hat{L}(h_{j,i}, S_{1-j})$ with the smallest validation loss.

Derive a high-probability generalization bound for the loss of h_{j^*,i^*}.

Comment: no theorem in the lecture note can directly solve the question, because they all assume that $\hat{L}(h, S)$ is computed on the same S for all h. You have to make a custom derivation.

4 Learning by discretization (20 points)

You want to learn an arbitrary binary function on a unit square by discretizing the square into a uniform grid with d^2 cells. The hypothesis space is the space of all possible uniform grids with d^2 cells for $d \in \{1, 2, 3, \dots\}$, where each cell gets a binary label.

You have a sample S of size n to learn the function. Let $\mathcal{H}$ be the hypothesis set of uniform grids and let $d(h)$ denote the number of cells in the hypothesis h.

1. Derive a generalization bound for learning with $\mathcal{H}$.

2. Explain how to use the bound to select a prediction rule $h \in \mathcal{H}$.

3. What is the maximal number of cells as a function of n, for which your bound is non-vacuous? (It is sufficient to give an order of magnitude, you do not need to make the precise calculation.)

4. Explain how the density of the grid affects the bound. Which terms in the bound increase as the density of the grid increases and which terms in the bound decrease as the density of the grid increases?

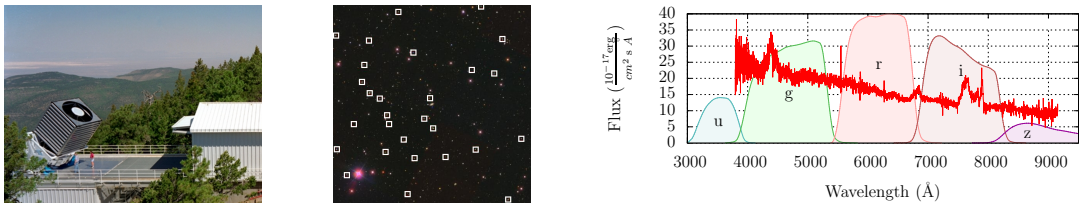
5 Regression (20 points)

Note that you do not need to understand the physical details given below to address this exercise (they are just provided for the sake of completeness). An important problem in astrophysics is to estimate the distance of objects to our Earth. Since one cannot directly measure these distances, one resorts to the light emitted by the objects and that reaches our Earth. In particular, one considers the so-called *redshift* z of an object: The larger the redshift, the larger is the distance an object to Earth. This redshift can be measured very accurately in case one has access to the detailed spectrum of the light for an object at hand. Unfortuntely, obtaining such spectra is very time-consuming and are, hence, only available for a small subset of the detected objects, see Figure 2. Instead, one has access to image data, which is used to estimate the redshift of the detected objects.

As mentioned above, the particular details are not important to approach this exercise. All you have to know is that this task can be formalized as regression problem and that you have access to a training set $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\} \subset \mathbb{R}^{10} \times \mathbb{R}$ and a corresponding test set, which are stored in `galaxies_train.csv` and `galaxies_test.csv`, respectively. Each file contains, for each row (object), a target value $y_i \in \mathbb{R}$ (first column) and a vector $\boldsymbol{x}_i \in \mathbb{R}^{10}$ of ten attributes (second to eleventh column); the first row in each file corresponds to the header (which contains column names). Your task is to build regression models that can be used to predict the target (redshift) for new, unseen objects!

Comment: For all parts of this exercise, you are allowed to make use of all the functions/classes provided by the Scikit-Learn package.

1. *Decision Tree Regression:* To start, build a single regression tree to address this task. More precisely, resort to the mean squared error $Q(S) = \frac{1}{|S|} \sum_{(\boldsymbol{x},y)\in S} (y - \bar{y})^2$ with $\bar{y} = \frac{1}{|S|} \sum_{(\boldsymbol{x},y)\in S} y$ as impurity measure and only



(a) Observatory (b) Image Data (c) Spectra

Figure 2: The Apache Point Observatory shown in Figure (a) gathers both images and spectra. The image data are given in terms of grayscale images that are based on five filters covering different wavelength ranges, called the `u`, `g`, `r`, `i`, and `z` bands. In Figure (b) an RGB image is shown that is based on such images of a particular region. For a small subset of detected objects (white squares), detailed follow-up observations in terms of spectra are available, see Figure (c). Image sources for Figures (a) and (b): `https://www.sdss.org`

stop the recursive contruction if the set is pure (thus, if you make use of Scikit-Learn, you can resort to the default parameter assignments of the `DecisionTreeRegressor` class).

Fully grown trees tend to overfit to the training data. One way to avoid overfitting to the training data is to restrict the decision tree's "freedom" by, e.g., restricting its maximum depth. Make use of 5-fold cross validation on the training data to find a suitable assignment for the maximum depth of the tree. Consider the range from 1 to 15 as possible assignments for the maximum depth. Fit the model again on all the training data using the optimal assignment for the maximum depth. Finally, compute the mean squared error (MSE) between the predictions made by the model for the test instances and the corresponding true labels. Visualize the model quality via a scatter plot ('true redshift' vs. 'predicted redshift'). You should see some "horizontal lines" in the plot (i.e., points located on horizontal lines)? Explain why.

2. *Extremely Randomized Tree Ensemble:* Fit an ensemble of $B = 500$ extremely randomized trees (fully grown); resort again to the mean squared error as impurity measure, do not draw bootstrap samples, and build fully grown trees (in case you make use of Scikit-Learn, this would correspond to resorting to the default values for the `ExtraTreesRegressor` class except for the number of trees). What is the induced MSE on the test set. Create a corresponding scatter plot to visualize the model quality. Do you get better results? Explain why there are no "horizontal lines" anymore in the plot.

3. *Impurity:* Consider a single tree of the ensemble of extremely randomized

trees generated above. Can it happen that a child of a node exhibits a worse impurity Q than the node itself? That is, $Q(L_{i,\theta}) > Q(S)$ or $Q(R_{i,\theta}) > Q(S)$, where S is the set corresponding to the node and $L_{i,\theta}$ and $R_{i,\theta}$ to the set of the left and right child, respectively, and where i is the chosen splitting dimension and θ the chosen threshold. If you think that this cannot happen, provide a formal proof. If you think that this can happen, provide a corresponding example (e.g., a simple toy example based on a single feature).

Deliverables: All your source code used to solve this exercise (as a Jupyter notebook/Python file(s)). Add to your report: (a) optimal assignment for the maximum depth, MSE for test data, scatter plot, (b) induced MSE for test data; scatter plot, (c) proof or example.

References

H. S. Baird. Document image defect models. In *Structured Document Image Analysis*, pages 546–556. Springer, 1992.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

L. Sørensen, C. Igel, N. Liv Hansen, M. Osler, M. Lauritzen, E. Rostrup, and M. Nielsen. Early detection of Alzheimer's disease using MRI hippocampal texture. *Human Brain Mapping*, 37(3):1148–1161, 2016.

L. Sørensen, C. Igel, A. Pai, I. Balas, C. Anker, M. Lillholm, and M. Nielsen. Differential diagnosis of mild cognitive impairment and Alzheimer's disease using structural MRI cortical thickness, hippocampal shape, hippocampal texture, and volumetry. *NeuroImage: Clinical*, 13:470–482, 2017.

J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012.