CS5011: A3 - Learning - Pump it Up: Data Mining the Water Table

Assignment: A3 - Assignment 3

Deadline: 7th of April 2021

Weighting: 25% of module mark

Please note that MMS is the definitive source for deadline and credit details. You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

1 Objective

This practical aims to construct and use artificial neural networks to solve a water-pump classification problem with real-world data. The problem and data themselves come with various challenges, which we will try to solve one by one during the practical.

2 Competencies

- Design, implement, and document an artificial neural network system for classification problems and relevant data preprocessing techniques.
- Understand different classification metrics and apply them in specific contexts.
- Understand, use and adapt a neural network within an AI system.

3 Practical Requirements

3.1 Introduction

Access to safe water is one of the most critical necessities and a very basic human right. However, according to a report by the World Bank[1] in 2018, "more than 23 million citizens in Tanzania retrieve drinking water from unimproved sources, and 41 million people use unimproved sanitation facilities". In many rural areas of the country, the water pump systems are poorly maintained, resulting in low-quality water or no access to water. This had negatively affected many people's health and quality of life. The Tanzanian Ministry of Water and other NGOs have been seeking for solutions to improve the maintenance operations at waterpoints to make sure that clean water is accessible to local communities. In this practical, we will look into a real-world dataset on operating conditions of water pumps collected at several waterpoints across Tanzania. The data was provided by Taarifa and the Tanzanian Ministry of Water, and

[1] `https://www.worldbank.org/en/news/press-release/2018/03/20/improving-water-supply-and-sanitation-can-help-tanzania-achieve-its-human-development-goals`

is downloadable from the DrivenData's competition "*Pump it Up: Data Mining the Water Table*" [2] [1]. Each example in the competition dataset represents various pieces of information about a water pump (input features) and the pump's status (output label). There are three possible statuses: `functional`, `functional needs repair`, and `non functional`. The task is to predict the status of a pump.

This dataset comes with a number of challenging characteristics of real-world data. The input features are of mixed types (numerical, categorical and datetime, see the Appendix at the end of this document for a full list of features). Some parts of the data is missing. The dataset is also imbalanced, i.e., the amounts of data among classes (pump's statuses) are uneven.

3.2 Core Tasks

Imagine we want to use our prediction model to make a maintenance plan for water pumps. Due to limited resources, the engineers will only visit and fix the pumps labelled as `functional needs repair`. It is important that our prediction is correct so that pumps that needs repair are not skipped by the engineers.

Since the main aim is to predict whether a pump needs repair, we will merge the two other classes (`functional` and `non functional`) into one. The resulting binary classification problem, in which we want to predict where a pump needs to be repaired, will be used for all the tasks of this part.

The dataset provided by the competition is imbalanced. However, we will first start with a very simple setting where we only consider numeric input features and with balanced data (Task 1). We will then move to Task 2 where all input features provided by the competition are considered (but still with balanced data). In Task 3 and Task 4, we will investigate the issue with imbalanced data (Task 3) and how to deal with it (Task 4). We will use artificial neural networks (NN) as the prediction models for all tasks in this part.

The datasets used for each task are available as `.csv` files on studres and are mentioned in the task specifications below. *For simplicity, you can assume that the column order is fixed as in the files provided, and that for numeric input features, there will be no missing data.*

Task 1: *Building a neural network prediction model using only numeric input features*

For this task, we will consider a very simple setting where we only use numeric input features for the prediction. The dataset for this task can be found on studres (`task1_train.csv`, `task1_test.csv`, and `task1_test_nolabels.csv`).

Your task is to design, build, and train a feed-forward neural network (NN) model to predict whether a pump needs repair using the given data. In addition, the trained network is to be used by the engineers to predict which, among a list of newly acquired waterpoints, need pump repairs according to their input conditions. The list of ids of those needing repairs is to be outputted in order for the engineers to carry out the work together with information about the quality of this prediction.

The evaluation metric used for this part is classification accuracy. The following points should be considered in your implementation and your report:

- What data-preprocessing should be done?
- Assuming that we use two hidden layers for the NN, what are the number of input and output nodes, number of hidden nodes per hidden layer, choice of activation function and loss function?
- How is the training done (e.g., learning rate, mini-batch size, avoiding over-fitting)?

[2] `https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table`

Your submitted program must support three functionalities, namely `train`, `test` and `predict`, as described below:

- `train`: Given an input training dataset (`task1_train.csv`), your program should train an NN prediction model and save it (together with any necessary data pre-processing) to disk.
- `test`: Given your saved (and trained) NN prediction model (and data pre-processors), and a test dataset (`task1_test.csv`), your program should load the trained model, applying data-preprocessing on the test data, and output the corresponding prediction to a text file. It should also output to the screen the evaluation metric value(s) on the given test set.
- `predict`: Given your saved (and trained) NN prediction model (and data pre-processors), and a dataset of pumps without labels (`task1_train_nolabels.csv`), your program should load the trained model, applying data-preprocessing on the given dataset, and output to the screen the list of ids of pumps needing repair.

You must also include in your submission your final trained model (produced by the functionality `train`) on the dataset given for this task.

For more details about requirements on the submitted program and on how to organise your submission, see Section 4. Requirements for the report can be found in Section 5.

Task 2: *Building a neural network prediction model using all input features*

This task is similar to the previous one. The difference is that categorical input features are also used. For those extra features there are missing data. Some features also have a large number of categories. The data pre-processing will be more complicated. Your task is to identify the necessary pre-processing steps, implement them, build an NN prediction model for the given dataset, report the results and compare them with the results obtained from Task 1. The output requirements are the same as in Task 1. The dataset used for this task is also available on studres (`task2_train.csv`, `task2_test.csv`, and `task2_test_nolabels.csv`).

Task 3: *Identifying issues with imbalanced data*

For this task, we will use the whole dataset provided by the competition (but still with two output classes instead of three). This dataset is imbalanced, i.e., the number of examples per class is uneven. The dataset can be found on studres (`task3_train.csv`, `task3_test.csv`, and `task3_test_nolabels.csv`).

The first step is to use the program created in Task 2 to train a NN model on the new dataset and evaluate it. After that, analyse the results and discuss in your report the potential issues with using classification accuracy as the evaluation metric. Which alternative metrics should we look at instead in such situations?

The required submission for this task includes the report and the trained NN used in your analysis.

Task 4: *Dealing with imbalanced binary classification*

This task is a continuation of Task 3, where we will look into solutions to deal with imbalanced binary classification problems. We really do not want to miss the true `functional needs repair` pumps in our prediction as it means they will be skipped by the engineers. Of course incorrectly identify too many pumps as `functional needs repair` is also not great due to the limited resources we have. Propose your solutions for this task, implement them and discuss your results in the report.

The required outputs for this task are similar to the ones described in Task 1. You can add more arguments into the programs' command lines if needed (for example, if you are investigating more than one solution approaches, you can add an argument to your program to indicate which approach is being used), but remember to include precise instructions on how to run your programs with the modified argument list in your report. The output printed to the screen by the `test` program should also include all evaluation metrics used for this task.

3.3 Advanced-Level Tasks

For this part, you can choose at most two advanced-level tasks. Please note that there is no benefit for implementing more than two. It is strongly recommended that you ensure you have completed the core tasks in the previous section before attempting any of those requirements.

Task 5: ***Dealing with imbalanced multiclass classification***

This task extends the scenario considered in Task 4 by dealing directly with three output classes instead of merging the two majority classes into one. The dataset for this task is available on studres (`task5_train.csv`, `task5_test.csv`, and `task5_test_nolabels.csv`). Do your approaches in Task 4 still work as expected, or what changes do you make for this multi-class case? How do the results look like?

Now imagine a more realistic scenario as follows. If we misclassify a `functional needs repair` pump as `functional`, this would mean that the pump will still be activated but its issues will not be checked and repaired by the engineers. This can pose health risk to the community if the water is low-quality or contaminated, or causing other serious issues if the water pipes happen to burst. On the other hand, if we misclassify a `functional needs repair` pump as `non functional`, the pump will be de-activated and will be checked later (after all the functional ones are repaired), which will cause certain resource wasting. The former misclassification case is likely more serious than the latter one. This problem is called *cost-sensitive classification*, where we have different costs for different misclassification cases. At the same time we also have the issue of imbalanced data. Discuss your solution approaches for this scenario, implement them and analyse the results.

The required outputs for this task are similar to the ones described in Task 1, but feel free to add more arguments into the programs' command lines if needed. The output printed to the screen by the test program should also include all evaluation metrics used for this task.

Task 6: ***Improving performance using hyper-parameter tuning***

Performance of a machine learning method can be improved by tuning its hyper-parameters. For this task, you are asked to identify the possible hyper-parameter choices for the neural network binary classifier built in Task 4, setup the tuning experiment properly, tune those hyper-parameters using a method of your choice, and report the results.

The required outputs for this task include the final trained NN model produced at the end of this task, and a test program to use the saved model as described in previous tasks.

Task 7: ***Solving the imbalanced multiclass classification problem using other types of models***

The scenario considered in this task is the same as the one described in Task 5. However, you are allowed to use other types of prediction models rather than neural networks. Discuss your solution approaches, implement them and report the results.

The required outputs for this task include the final trained model produced, and a test program to use the saved model as described in previous tasks.

Task 8: ***Making the NN prediction system more flexible***

In the core tasks we provide the system with the ability of predicting the status of the pumps of newly seen waterpoints. In this task you are asked to build a new system that merges the train and prediction systems simulating a more dynamic system in which the engineers can input the status of additional waterpoints and their input conditions on the fly in addition to receiving predictions. The dataset used for this task is the same as Task 4's. The system should be provided with a simple text-based interface to interact with the engineers. The newly added data should be used to retrain the network. After training the system should be ready to predict the status of newer waterpoints, or to accept newer data points in a loop. In your report, please motivate your design choices and describe how your system deals with newly received training data.

4 Code Specification

4.1 Programming Languages and Libraries

For this practical, you can choose either `Java` or `Python` as the programming language. **You may only use Python if you are already familiar with this programming language**. Your implementation must run (and compile) without the use of an IDE.

Your system should be compatible with the version of `Java`/`Python` available on the School Lab Machines (`Java` version: Amazon Corretto 11 – JDK11, `Python` version: $\geq$ 3.7).

The `DL4J` [3] library and the `scikit-learn`[4] library are used for `Java` and `Python` implementation, respectively.

For `Java` implementation: The `DL4J`'s website recommends using `maven` to manage all dependencies. In case you are not familiar with `maven`, we provide a `.jar` library file which includes all the basic dependencies required by `DL4J`. This can be found in the example `DL4J` code on studres (`A3ExampleWithoutMaven.zip`)). We also provide an example `DL4J` code with `maven` (`A3ExampleWithMaven.zip`). You can use either of them as a starter code for your `Java` implementation.

Important note: the provided `DL4J` library `.jar` file is quite heavy (almost 1GB), therefore, DO NOT include that file in your submission even if you choose to use it. We will copy our local version of that file to your submission folder during marking. Also if you choose to use `maven`, remember not to include the `.jar` files downloaded by `maven` in your submission, the `maven` file `pom.xml` file is sufficient.

For `Python` implementation: if you use external libraries not included in `scikit-learn`, you must provide a `requirements.txt` file [5] listing those extra `Python` packages.

Some useful links:

- Access to the School lab PCs:
  `https://systems.wiki.cs.st-andrews.ac.uk/index.php/Lab_PCs`
  `https://systems.wiki.cs.st-andrews.ac.uk/index.php/Working_remotely`
  `https://systems.wiki.cs.st-andrews.ac.uk/index.php/Video_tutorials`
- Access to `Python` on the School lab PCs:
  `https://systems.wiki.cs.st-andrews.ac.uk/index.php/Python_on_Windows_lab_clients`
  `https://systems.wiki.cs.st-andrews.ac.uk/index.php/Python_on_Linux`

[3]`https://deeplearning4j.konduit.ai/`
[4]`https://scikit-learn.org/`
[5]`https://pip.pypa.io/en/stable/reference/pip_freeze/`

- DL4J:
  The official website of dl4j: https://deeplearning4j.konduit.ai/
  dl4j blog: https://blog.konduit.ai/
  dl4j-examples repository: https://github.com/eclipse/deeplearning4j-examples
  dl4j community forum, where you can post questions and get support: https://community.konduit.ai/
- scikit-learn: the official website of scikit-learn should have everything you need: https://scikit-learn.org/

4.2 Code Submission and Running

4.2.1 Code Running

Your code should run the following command:

```
java A3main <task_id> <train|test|predict> <other_arguments>
```

or

```
python A3main.py <task_id> <train|test|predict> <other_arguments>
```

More specifically, the three Java command lines for Task 1 should be:

```
java A3main task1 train <input_csv> <output_NN_name>
java A3main task1 test <input_csv> <input_NN_name> <output_txt>
java A3main task1 predict <input_csv> <input_NN_name>
```

Examples:

```
java A3main task1 train task1_train.csv task1_NN
java A3main task1 test task1_test.csv task1_NN task1_test_predictions.txt
java A3main task1 predict task1_test_nolabels.csv task1_NN
```

Note that the argument <output_NN_name> in the commands above is the prefix of files where the trained NN (and pre-processing) are saved. For example, when user runs the command line:

```
java A3main task1 train task1_train.csv task1_NN
```

The trained NN and its data-preprocessors will be saved in a file (or multiple files, depending on your implementation) with name(s) starting as task1_NN.

The <output_txt> file contains prediction of the input dataset. Each i^{th} line of this output text file must have one single predicted label (e.g., functional needs repair) corresponding to the prediction for the i^{th} input sample.

4.2.2 Code submission

Your source code should be placed in a directory called **A3src/** and should include all *non-standard external libraries* besides DL4J and scikit-learn, as described in Section 4.1. Your submissions must include all the required components described for each task.

Please do not put the datasets provided on studres into your submission!

Submit the whole folder as a single .zip file to MMS. Your source code should be well-structured and well-commented whenever possible. Where any code from other sources is used, you must provide clear description of which code is yours.

Please note that code and submission that do not adhere to these instructions may not be accepted.

5 Report

You are required to submit a report describing your submission **in PDF** with the structure and requirements presented in the additional document *CS5011_A_Reports*. The report should also include clear instructions on how to run your code. The core sections about design and implementation have an advisory limit of 1000 words for all tasks in total. The evaluation and analysis sections have an advisory limit of an additional 300 words per task.

Consider the following points in your report:

1. Describe all components of your solution approaches, including data-preprocessing methods, the network's structure and its hyper-parameters (e.g., number of hidden layers, number of hidden nodes, learning rate), the evaluation metrics used, and experiment setup (e.g., how was the data split? how were the evaluations done?), and insights into why they are chosen.
2. Report results and analyse them thoroughly, you are encouraged to use summarised tables and graphs where possible to visualise your results and analysis, and to support your findings.
3. DO NOT put in your report large tables with lots of numbers in it, or lots of similar graphs that might take up dozen of pages, as they will make it difficult for the readers to go through your report and understand the key points. Use summarised statistics and merging graphs together if possible. The detailed results of all experiments should be provided in separate `.csv` files instead.
4. Provide a summary of all your key findings for each task (either at the beginning or at the end of the report's section for the relevant task).

6 Deliverables

A single ZIP file must be submitted electronically via MMS by the deadline. Submissions in any other format will be rejected.

Your ZIP file should contain:

1. A PDF report as discussed in Section 5
2. Your code and the trained NN models as described in Section 4 and in the specification of each task.

7 Assessment Criteria

Marking will follow the guidelines given in the school student handbook. The following factors will be considered:

- Achieved requirements and quality of the implementations provided.
- Quality of the report, including analysis and insights into the proposed solutions and results.

Some guideline descriptors for this assignment are given below:

- For a mark of 7 to 11: the submission implements Task 1. At the higher end of this range, the implementation, experiments, evaluations and report must be adequately done.
- For a mark of 11 to 13: the submission provides outputs expected for the higher range of the previous band and for Task 2, the implementation, experiments, evaluations should be adequate. At the higher end of this range, the report and comparisons should be of good quality.
- For a mark of 13 to 15: the submission provides outputs expected for the higher range of the previous band and for Task 3. At the higher end of this range, a good analysis on results of Task 3, with precise and insightful discussions on the evaluation metrics should be provided.
- For a mark of 15 to 17: the submission provides outputs expected for the higher range of the previous band and for Task 4. At the higher end of this range, very good implementation with clear, insightful and well-written report should be provided.
- For a mark of 17 and above: the submission provides outputs expected for the higher range of the previous band, and with either good attempts on two advance-level tasks (including implementation, experiments and report), or with excellent implementation, experiments and report on one advanced-level task.

8 Policies and Guidelines

Marking: See the standard mark descriptors in the School Student Handbook

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_-Descriptors`

Lateness Penalty: The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#latenesspenalties`

Good Academic Practice: The University policy on Good Academic Practice applies:

`https://www.st-andrews.ac.uk/students/rules/academicpractice/`

Nguyen Dang, Alice Toniolo

cs5011.lec@cs.st-andrews.ac.uk

March 10, 2021

References

[1] Peter Bull, Isaac Slavitt, and Greg Lipstein. Harnessing the power of the crowd to increase capacity for data science in the social sector. *arXiv preprint arXiv:1606.07781*, 2016.

Appendix 1: List of all features

1. `amount_tsh`: Total static head (amount water available to waterpoint)
2. `date_recorded`: The date the row was entered
3. `funder`: Who funded the well
4. `gps_height`: Altitude of the well
5. `installer`: Organization that installed the well
6. `longitude`: GPS coordinate
7. `latitude`: GPS coordinate
8. `wpt_name`: Name of the waterpoint if there is one
9. `num_private`
10. `basin`: Geographic water basin
11. `subvillage`: Geographic location
12. `region`: Geographic location
13. `region_code`: Geographic location (coded)
14. `district_code`: Geographic location (coded)
15. `lga`: Geographic location
16. `ward`: Geographic location
17. `population`: Population around the well
18. `public_meeting`: True/False
19. `recorded_by`: Group entering this row of data
20. `scheme_management`: Who operates the waterpoint
21. `scheme_name`: Who operates the waterpoint
22. `permit`: If the waterpoint is permitted
23. `construction_year`: Year the waterpoint was constructed
24. `extraction_type`: The kind of extraction the waterpoint uses
25. `extraction_type_group`: The kind of extraction the waterpoint uses
26. `extraction_type_class`: The kind of extraction the waterpoint uses
27. `management`: How the waterpoint is managed
28. `management_group`: How the waterpoint is managed
29. `payment`: What the water costs
30. `payment_type`: What the water costs

31. `water_quality`: The quality of the water
32. `quality_group`: The quality of the water
33. `quantity`: The quantity of water
34. `quantity_group`: The quantity of water
35. `source`: The source of the water
36. `source_type`: The source of the water
37. `source_class`: The source of the water
38. `waterpoint_type`: The kind of waterpoint
39. `waterpoint_type_group`: The kind of waterpoint