

VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

System for Support of Biomedical Data Analysis

**System na podporu analýzy
biomedicínských dat**

Diploma Thesis Assignment

Student: **Bc. Jan Vargovský**

Study Programme: N2647 Information and Communication Technology

Study Branch: 2612T025 Computer Science and Technology

Title: **System for Support of Biomedical Data Analysis**
Systém na podporu analýzy biomedicínských dat

The thesis language: English

Description:

The analysis of biomedical data is a current task, mainly thanks to the ever-evolving technologies for obtaining and preprocessing biological samples. The result is always real data suffering from many problems associated with mistakes, inaccuracies, incompleteness, etc.

This diploma thesis aims to prepare a software environment supporting experiments with real-world data using machine learning methods based on modern approaches (e.g. deep learning). This environment will be addressed to researchers and domain experts in biology and biochemistry specializing in the field of biomedical data analysis. The designed software environment will allow them to plan experiments and prepare data, configure and execute the experiment with results including a PDF report with tables and appropriate visualizations, save experiment configuration and results, and repeat the experiment.

Partial goals:

1. Review of systems, frameworks, and methods in the field of biomedical data analysis.
2. Design of system architecture, implementation of the user interface (the web is assumed), integration of the used frameworks (e.g. ML.NET, Accord.NET, Math.NET, ILNumerics, Tensorflow) and design of data storage.
3. Implementation and integration of selected methods, testing. Implementation of the methods with regards to thousands to tens of thousands of instances and tens to hundreds of attributes is expected. Experimental tasks in the area of classification, clustering, and feature selection are assumed.
4. Documentation of implemented methods and system components.

References:

- [1] Aggarwal, C. C. Neural Networks and Deep Learning: A Textbook. Springer.
[2] Chen, J. Y., Lonardi, S. (Eds.). (2009). Biological data mining. CRC Press.

Others according to the instructions of the supervisor.

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

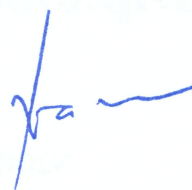
Supervisor: **doc. Mgr. Miloš Kudělka, Ph.D.**

Date of issue: 01.09.2018

Date of submission: 30.04.2019



doc. Ing. Jan Platoš, Ph.D.
Head of Department



prof. Ing. Pavel Brandštetter, CSc.
Dean



I hereby declare that this master's thesis was written by myself. I have quoted all the references I have drawn upon.

Ostrava, 30th April 2019

.....
Kuzosky

I would like to thank my supervisor Miloš Kudělka for the topic, advice and help throughout the thesis. Also, I would like to thank my family and friends for all their support.

Abstrakt

Analýza biomedicínských dat je aktuální úlohou zejména díky stále se vyvíjejícím technologiím pro získávání a předzpracování biologických vzorků. Tato diplomová práce prezentuje prostředí pro podporu experimentů s metodami z oblasti strojového učení založenými na neuronových sítích. První část práce popisuje základní pojmy strojového učení a neuronových sítí. Druhá část popisuje požadavky, architekturu, použité technologie a všechny možnosti aplikace.

Klíčová slova: strojové učení, neuronové sítě, analýza biomedicínských dat, vizualizace, učení s učitelem, klasifikace

Abstract

The analysis of biomedical data is a current task, mainly thanks to the ever-evolving technologies for obtaining and preprocessing biological samples. This diploma thesis presents a software environment supporting experiments with real-world data using machine learning methods based on neural networks. The first part of the work discusses core concepts of machine learning and neural networks. The second part describes requirements, architecture, used technologies and all the capabilities of the application.

Key Words: machine learning, neural networks, biomedical data analysis, visualization, supervised learning, classification

Contents

| | |
|---|-----------|
| List of symbols and abbreviations | 9 |
| List of Figures | 10 |
| List of Tables | 11 |
| Listings | 12 |
| 1 Introduction | 13 |
| 2 Available solutions | 14 |
| 2.1 Weka | 14 |
| 2.2 Microsoft Azure Machine Learning Studio | 14 |
| 2.3 Neural Designer | 14 |
| 2.4 RapidMiner Studio | 14 |
| 2.5 OpenML | 15 |
| 3 Machine learning | 16 |
| 3.1 Data | 16 |
| 3.2 Preprocessing | 21 |
| 3.3 Supervised learning | 23 |
| 3.4 Unsupervised learning | 26 |
| 4 Neural networks | 28 |
| 4.1 Perceptron | 29 |
| 4.2 Multilayer perceptron | 30 |
| 4.3 Types | 32 |
| 4.4 Hyperparameters | 34 |
| 5 Application | 36 |
| 5.1 Requirements | 36 |
| 5.2 Use cases | 36 |
| 5.3 Technology stack | 40 |
| 5.4 Modules | 43 |
| 5.5 Storage | 50 |
| 5.6 API | 51 |
| 5.7 Experiments | 53 |
| 6 Conclusions and future work | 58 |

| | |
|-----------------------------------|-----------|
| Appendix | 60 |
| A How to develop | 61 |
| B How to build and install | 63 |

List of symbols and abbreviations

| | |
|------|-------------------------------------|
| API | – Application Programming Interface |
| CLI | – Command Line Interface |
| CPU | – Central Processing Unit |
| CSS | – Cascading Style Sheets |
| CSV | – Comma-Separated Values |
| GPU | – Graphics Processing Unit |
| GUI | – Graphical User Interface |
| HDF | – Hierarchical Data Format |
| HTML | – Hypertext Markup Language |
| HTTP | – Hypertext Transfer Protocol |
| MLP | – Multilayer Perceptron |
| ML | – Machine Learning |
| REST | – Representational State Transfer |
| SPA | – Single-Page Application |
| TPU | – Tensor Processing Unit |
| URL | – Uniform Resource Locator |
| WSGI | – Web Server Gateway Interface |

List of Figures

| | | |
|----|---|----|
| 1 | Low entropy. | 19 |
| 2 | Medium entropy. | 19 |
| 3 | High entropy. | 19 |
| 4 | Regression model - optimal fit. | 24 |
| 5 | Regression model - underfitting. | 24 |
| 6 | Regression model - overfitting. | 24 |
| 7 | Model training - optimal fit, optimal training iterations. | 25 |
| 8 | Model training - underfitting, not enough training iterations. | 25 |
| 9 | Model training - underfitting, not enough memory capacity. | 25 |
| 10 | Model training - overfitting, too much training iterations. | 26 |
| 11 | Clustering examples. | 27 |
| 12 | Clustering examples 2. | 27 |
| 13 | Perceptron. | 29 |
| 14 | Binary classification using perceptron. | 30 |
| 15 | Multilayer perceptron. | 31 |
| 16 | Hopfield network. | 33 |
| 17 | Application workflow. | 37 |
| 18 | Application - datasets use cases. | 38 |
| 19 | Application - models use cases. | 39 |
| 20 | Application - component diagram. | 41 |
| 21 | Docker overview. | 43 |
| 22 | Application - workflow. | 44 |
| 23 | Application - dataset example. | 45 |
| 24 | Application - dataset upload. | 46 |
| 25 | Application - dataset import. | 46 |
| 26 | Application - list of datasets. | 47 |
| 27 | Application - modify preprocessing. | 49 |
| 28 | Application - Swagger UI. | 52 |
| 29 | Application - Iris experiment - settings. | 54 |
| 30 | Application - Iris experiment - test run results. | 55 |
| 31 | Application - Iris experiment - 10-fold cross-validation results. | 56 |

List of Tables

| | | |
|---|--|----|
| 1 | Blood type data before preprocessing using the label encoding. | 22 |
| 2 | Blood type data after preprocessing using the label encoding. | 22 |
| 3 | Blood type data before preprocessing using the one-hot encoding. | 23 |
| 4 | Blood type data after preprocessing using the one-hot encoding. | 23 |
| 5 | XOR problem. | 30 |
| 6 | Dataset format comparison. | 52 |
| 7 | Leukemia experiment results. | 57 |

Listings

| | | |
|---|---------------------|----|
| 1 | Dockerfile. | 63 |
|---|---------------------|----|

1 Introduction

The analysis of biomedical data is a difficult task. Mainly due to the fact that the biological data suffer from missing, invalid, inaccurate or incomplete values. The biological data is not the only case though. In fact, most of a real world data does have the aforementioned problems. In order to have the analyses meaningful, the data has to be meaningful too. The data has to be informative, independent and simple. Preprocessing is the part of analyses that deals with the data and transforms it into a good shape. For instance during the preprocessing you might remove a column or remove an entire row.

The analyses are difficult and require knowledge of at least two domains - the biomedical and technical. The biomedical domain is often supplied by a doctor that measures data using a machines. The doctor knows what the values mean and what the consequences are. On the other hand, the technical domain is supplied by a data scientist who is aware of techniques, tools and algorithms to do the analyses using soft computing. The soft computing is a discipline that does not focus on exact solutions but rather approximations and partial truths that are developed much faster. One component of the soft computing is a machine learning. The machine learning is a field of study that gives computers the ability to learn without being explicitly programmed.

Deep learning¹ was used to detect cancer metastases on gigapixel pathology images². It detected 92.4% of the tumors compared to human pathologist who did exhaustive search and achieved 73.2% sensitivity [1]. Deep neural networks were also used to classify skin cancer [2] and to identify tuberculosis in chest X-rays [3].

The main goal of this work is to give a non-technical people a powerful software environment to prepare the data, plan an experiment, configure and execute the experiment. The software environment should not require an in-depth knowledge of cutting edge technologies. It should be as simple as possible. It means the user experience using the software's user interface have to be straightforward and intuitive. Created models from the experiments might be passed to the technical person - the data scientist which will tune the model (increase its accuracy).

The rest of this work is organized as follows: In Chapter 2 a few software solutions for machine learning tasks are mentioned. Fundamental terminology of machine learning as well as its tasks are described in Chapter 3. Chapter 4 shall focus on history, applications and types of neural networks. The proof of concept application is described in Chapter 5. There are sections for requirements, use cases, used technologies, modules, storage, the API and experiments. The conclusion and future work are mentioned in the last Chapter 6.

¹Special branch of machine learning based on deep neural networks (more than 1 hidden layer)

²Submitted to one of challenges on <https://grand-challenge.org>

2 Available solutions

There are many machine learning software applications out there. This chapter will look at a few of them.

2.1 Weka

Weka³ is a collection of machine learning algorithms for data mining tasks. It contains tools for data preparation, classification, regression, clustering, association rules mining, and visualization. It is able to use machine learning to derive useful knowledge from databases that are far too large to be analyzed by hand. Weka's users are ML researchers and industrial scientists, but it is also widely used for teaching. It is not capable of multi-relational data mining. Weka offers both CLI and GUI variants.

2.2 Microsoft Azure Machine Learning Studio

Microsoft Azure Machine Learning Studio⁴ is a GUI-based integrated development environment for constructing and operationalizing machine learning workflow. Everything is done using drag-and-drop in an interactive canvas. You drag-and-drop datasets and analysis modules onto an interactive canvas, connecting them together to form an experiment. There is no programming required, just visually connecting datasets and modules to construct a predictive analysis model. It is capable of many machine learning tasks such as classification, regression, clustering and visualization but also anomaly detection, recommendation, statistical functions, text analytics and computer vision.

2.3 Neural Designer

Neural Designer⁵ is a desktop GUI application based on neural networks and offers regression and classification tasks only. It supports files or databases as data source. Preprocessing and visualization using tables, charts and pictures is also available. It also offers feature selection using genetic algorithms.

2.4 RapidMiner Studio

RapidMiner Studio⁶ is a visual data science workflow designer accelerating the prototyping and validation of models. RapidMiner Studio provides powerful visual programming environment. It is composed of modules for data exploration using descriptive statistics and visualization, data preparation and cleansing, and over 1500 machine learning algorithms and functions.

³<https://www.cs.waikato.ac.nz/ml/weka>

⁴<https://studio.azureml.net>

⁵<https://www.neuraldesigner.com>

⁶<https://rapidminer.com>

2.5 OpenML

OpenML⁷ is a cross-platform programming environment for machine learning. OpenML operates on the following core concepts: datasets, tasks, flows and runs. Datasets are simply data in tabular form. A task is a tuple of a dataset, machine learning task such as classification or clustering, and evaluation method. A flow is a particular machine learning algorithm, such as multilayer perceptron. A run is a particular flow, that is algorithm, with a particular parameter setting, applied to a particular task.

People who are behind OpenML believe that machine learning should be as open and accessible as possible. OpenML is a place where researchers can share significant breakthroughs but also work more effectively, be more visible, and collaborate with others to tackle harder problems [4]. It is also a place where you can share interesting datasets with the people who love to analyze data, and build the best solutions together, saving you valuable time, increasing your visibility, and speeding up discovery.

⁷<https://www.openml.org>

3 Machine learning

This section gives you an overview of fundamentals of a machine learning (ML). The basic terminology that is widely used will be discussed. First, the thesis will focus on the data, what data types are distinguished and how it affects what we can do with it. Then, the method how we can measure how much the information the data contains. The following part is dedicated to preprocessing the data, so we will talk about how we can transform data into a better shape. Two main areas will be focused on, namely a supervised learning and an unsupervised learning. What they are about and what tasks we can perform. The problems we can face will be discussed, including the methods how to identify and fix them.

3.1 Data

Data in the machine learning comes in many forms. It might be a table (collection of rows and columns) or a mathematical graph (set of nodes and edges). Data is defined as a universe of objects. The universe of objects is normally very large and we have only a small subset of it. However, the small subset can be also very large and we can get rid of many objects because they can be redundant and usually we do not benefit from having multiple almost identical objects.

Each object is described by a tuple of variables that correspond to its properties or so called the features. In machine learning a feature is often called as attribute. Alternative name is also a column but it depends on the source of data. We will use all the terms because they are exchangeable.

A tuple of variable values of the object is called a record or an instance. A collection of a records is referred to as a dataset. For the sake of simplicity the dataset can be a simple table but it is important to be aware of the aforementioned terms for the future reading.

3.1.1 Types of features

In general, there are many types of features. Specifying the correct variable type might lead to reduce amount of required memory for a loaded dataset or to reduce size of a saved dataset on a disc but more importantly misinterpretation of feature type might cause a problems during the analyses, it can be completely wrong due to the fact that some types have to be treated differently.

There are at least six types of variables, namely: nominal, binary, ordinal, integer, interval-scaled and ratio-scaled. This segregation between the types of features can be useful in some cases but in practice we segregate them only between the categorical and continuous feature types only.

- Categorical feature
 - Nominal variable

- Binary variable
- Ordinal variable
- Continuous feature
 - Integer variable
 - Interval-scaled variable
 - Ratio-scaled variable

Categorical feature

Categorical features are usually a finite set of values. They can be easily interpreted using a string, an integer, a boolean or an enum type. Categorical features might be divided into nominal, binary and ordinal variables.

Nominal variable

A nominal variable is used to put objects into categories. For instance a blood type (A, B, AB and O) or a color (red, green, blue). The nominal variable is often referred to as a label. Although the nominal variable is often in a numerical form, there are no mathematical operations among the values. Consider blood types encoded into a numerical form, namely A=0, B=1, AB=2 and O=3. In the encoded form it is natural to add one value to another but if we know a nominal variable is concerned, it is suddenly meaningless.

Binary variable

A binary variable is just a special case of the nominal variable. It is restricted to two possible values only such as true and false, or 1 and 0.

Ordinal variable

An ordinal variable is basically a nominal variable with one extra property, it can be ordered. For instance a size (small, medium and large).

Continuous feature

Continuous features can take any values. Continuous features might be divided into integer, interval-scaled and ratio-scaled variables.

Integer variable

An integer variable is a variable that takes an integer value. For instance number of blood donations. Unlike nominal variable it is possible to do mathematical operations, with respect to scale and unit, of course.

Interval-scaled variable

An interval-scaled variable is a variable that takes a numerical value and distances between the ordered values are equal. Also, the distance between two ordered values is a constant. For example temperature measured in Celsius or Fahrenheit. The problem with interval-scaled value is that they do not have a true zero value. In our example with temperature, 0 degrees Celsius does not mean absence of temperature because there is no such thing as no temperature.

Ratio-scaled variable

A ratio-scaled variable is similar to interval-scaled but they do have a true zero value. For example weight (1kg, 5kg, 10kg) or length (0.5m, 1m, 2m).

3.1.2 Types of data

In general there are two main types of data, namely unlabeled and labeled data.

Labeled data consists of specially designated attribute, usually called a label or a class, that we are trying to predict. The prediction is based on the rest of its attributes. Typically, we have a set of labeled data to train our magic black box, formerly called a model. The label is excluded and is not going as an input into the model but rather the model is supposed to guess the label for us based on the inputs. Once we train a model we evaluate a test against an unseen data and compare whether the prediction was successful or not.

Unlabeled data does not consist of any special attribute, there is the data only, nothing else. One of the most common goals is to divide the data into groups. Once the groups are formed we have to identify the indirect hidden patterns, rules or relationships. Those groups should have something in common and we are supposed to find out what it is. This is called a knowledge discovery. Important parameter in this case is how many groups are we supposed to form. This is usually not known in advance and we often have to try and test multiple choices.

There are specific tasks based on the type of data that are discussed in Chapters 3.3 and 3.4.

3.1.3 Entropy

An entropy is a numerical value that tells us how much information we have in a given set of data. The entropy is not related to the machine learning only, it comes from the Information Theory in which it is widely used as the basis for calculating efficient ways of representing messages for transmission by telecommunication systems [5].

We can calculate entropy E using the following formula

$$E = - \sum_{i=1}^N P(x_i) \log_2(P(x_i)) \tag{1}$$

where

N is number of unique records,

$P(x_i)$ is the probability of value x_i .

The logarithm to base 2 is used because entropy usually calculates the number of required bits to encode a message.

Suppose a dataset with four binary values, 0 and 1. The first case is when all values are 0, see Figure 1. $P(0) = 1$ and $P(1) = 0$ and therefore the entropy

$$\begin{aligned} E &= -P(0)\log_2(P(0)) \\ &= -1 \cdot \log_2(1) \\ &= -1 \cdot 0 \\ &= 0 \end{aligned}$$

The values do not provide any information. We always know what the information is whether it was being said or not.

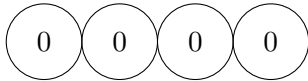


Figure 1: Low entropy.

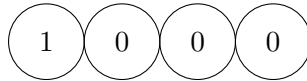


Figure 2: Medium entropy.

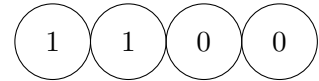


Figure 3: High entropy.

Suppose the same dataset but one of the values is 1, see Figure 2. So, $P(0) = 0.75$ and $P(1) = 0.25$ and therefore the entropy

$$\begin{aligned} E &= -P(0)\log_2(P(0)) - P(1)\log_2(P(1)) \\ &= -0.75 \cdot \log_2(0.75) - 0.25 \cdot \log_2(0.25) \\ &= 0.75 \cdot 0.415 + 0.25 \cdot 2 \\ &= 0.311 + 0.5 \\ &= 0.811 \end{aligned}$$

In the terms of the Information Theory it tells us that we need 0.811 bits in average to encode information in binary form. Since there is nothing like 0.811 bits the value is rounded up, so in this case we would need 1 bit per value.

Suppose third case with equal distribution, so two values of 0 and two values of 1, see Figure 3. $P(0) = 0.5$ and $P(1) = 0.5$ and therefore the entropy

$$\begin{aligned}
E &= -P(0)\log_2(P(0)) - P(1)\log_2(P(1)) \\
&= -0.5 \cdot \log_2(0.5) - 0.5 \cdot \log_2(0.5) \\
&= 0.5 + 0.5 \\
&= 1
\end{aligned}$$

This is, in fact, a special case of more generalized form where $P(x_i) = \frac{1}{N}$, so all probabilities of values x_i are the same and entropy formula might be simplified

$$\begin{aligned}
E &= -\sum_{i=1}^N P(x_i)\log_2(P(x_i)) \\
&= -\sum_{i=1}^N \frac{1}{N}\log_2\left(\frac{1}{N}\right) \\
&= -\log_2\left(\frac{1}{N}\right) \\
&= \log_2 N
\end{aligned} \tag{2}$$

3.1.4 Ideal features

We can see the entropy helps us identify a good feature, at least in terms of potential information. However, you might think of feature such as a customer id or any other id. Those features do have a high entropy nevertheless from the machine learning point of view they are completely meaningless. Ideal features should be

- informative
- independent
- simple

Prefer clear and easy to understand features rather than complex ones. Try to avoid rare values. Rare values are considered all discrete values that appeared, let's say, less than 5 times. Those values would be hard to be identified as potential patterns. The feature should not correlate with any other feature. The feature should not be redundant. It means that feature A should not be able to be computed, somehow, from feature B.

It is quite common to have a dataset with more features than instances. People tend to record all available information because it is easier to avoid the hard decision whether the information may be or may not be useful in the future.

Less is more, at least in terms of used features for our experiments. There is a thing called curse of dimensionality. The curse of dimensionality indicates that the number of samples needed

to estimate a function with a given level of accuracy grows exponentially with respect to the number of input variables of the function [6]. We may adopt the idea in the machine learning. In other words, the more features, the bigger dataset and the more computation power and time is required.

Dimensionality reduction is a process which reduces number of features. One approach is to select a smaller subset of (good) features and discard the rest. Another approach is to search for optimal mapping between the original dimension into defined number of dimensions. Each new dimension is a linear/non-linear combination of original features.

3.2 Preprocessing

The preprocessing is part of machine learning that transforms the data into a better shape. Many algorithms expect the data not to have missing values, they provide better values with rescaled values or they simply do not know how to interpret a string, so we have to convert it to e.g. a number. In this chapter we will show common techniques how to preprocess the data to have better and reliable results.

3.2.1 Missing values

Missing values is quite self-describing concept. I just mention that sometimes they are interpreted as Not Available (NA) or Not a Number (NaN). We will show three approaches how to deal with them.

The first approach discards all instances that consist of any missing value. In general this is useful because we do not introduce any manual noise to the data. However, this is not useful when a large proportion of data has a missing value. This is a recommended approach for all instances that a feature label is missing unless we are sure what the label is.

The second approach uses the most common or average value. In case of categorical attributes we may calculate the most common value and replace all missing ones with it. However, be careful and check what the proportions are. Check whether the data is balanced or not. The balanced data has similar proportions such as 51% for X and 49% for Y, it is hard to pick the most common value. On the other hand, unbalanced data has proportions such as 80% for X and 20% for Y. In this case it is more reasonable to pick the most common value. It is recommended to use this in case you have unbalanced data. In case of continuous attributes it is straightforward to use the average value.

The third and last approach is the custom value. The most common custom value is 0 that usually indicates missing value in a mathematical form.

There are many other methods that deal with missing values but we should be fine using the mentioned ones only. Keep in mind that all altering the data may introduce a noise in the data that affects your experiments.

3.2.2 Normalization

Many algorithms converge faster when using smaller values, which means less computation time and faster results. When using distance based algorithms such as k-Nearest Neighbor, it gives us significantly better results. You might think of an object with two attributes, the first attribute uses values from 0 to 1 and the second attribute uses values from 0 to 100. Now if we calculate e.g. the Euclidean distance it happens that the second attribute has much more power just because of its larger scale. To overcome this side effect we normalize them to the same scale so they contribute equally.

Normalization is a technique to make the value within a range from 0 to 1, and therefore this approach is intended to be used for continuous attributes only. The calculation is following

$$x'_i = \frac{x_i - \min(x)}{\max(x) - \min(x)} \quad (3)$$

where

x is a series of values,

x_i is the value to be normalized,

x'_i is the normalized value.

3.2.3 Encoding

Encoding is a technique to convert categorical variables into a numerical data. The main reason why we do this is to make algorithms understand the data. There are two approaches how to achieve this.

The first approach is called a label encoding. The label encoding is a simple substitution method that assigns each unique value its numerical value. The encoding usually starts from 0 and goes up to $N - 1$ categories. For example consider the blood type attribute and how we preprocess it using the label encoding, see Tables 1 and 2.

| Id | Blood Type |
|----|------------|
| 1 | A |
| 2 | B |
| 3 | AB |
| 4 | O |
| 5 | A |

Table 1: Blood type data before preprocessing using the label encoding.

| Id | Encoded Blood Type |
|----|--------------------|
| 1 | 0 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |
| 5 | 0 |

Table 2: Blood type data after preprocessing using the label encoding.

The second approach is called a one-hot encoding. The one-hot encoding creates a new attribute for each category. The values are binary, namely 0 and 1. 1 indicates that the object

corresponds to the category whereas 0 indicates the opposite. We will use the same example with the blood type attribute and how we preprocess it using the one-hot encoding, see Tables 3 and 4.

| Id | Blood Type |
|----|------------|
| 1 | A |
| 2 | B |
| 3 | AB |
| 4 | O |
| 5 | A |

Table 3: Blood type data before preprocessing using the one-hot encoding.

| Id | A | B | AB | O |
|----|---|---|----|---|
| 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 |
| 5 | 1 | 0 | 0 | 0 |

Table 4: Blood type data after preprocessing using the one-hot encoding.

It should be mentioned that it is not necessary to have exactly one 1 for the encoded set of attributes. It is possible to have multiple 1s or even none of them but it is more like a tag rather than a label then.

Let's talk about the pros and cons among the aforementioned approaches. First, the label encoding does have one good property, it does not increase the dimensionality unlike the one-hot encoding. Think about an example with a few categories (e.g. 2) and about an example with a lot categories (e.g. 100). Second, the label encoding does create an order relationship among the values. Some algorithms can assume that the higher categorical value, the better category, which is not always true. You have to take into account those facts and balance between the dimension size and information accuracy.

3.3 Supervised learning

In supervised learning, we are given a labeled data and our goal is to make a model that can predict a label. If the label is categorical, we are talking about a classification task. If the label is numerical, we are talking about a regression task.

In both tasks we usually divide a given dataset into two or three sets. A training, a testing and optionally a validation set. A typical scenario in the supervised learning is following

1. Preprocessing phase
2. Training phase
3. Testing phase

The preprocessing phase involves all the techniques that were discussed in Chapter 3.2.

During the training phase we pass the training set into a model and try to learn hidden patterns. This is usually done iteratively, every iteration should improve the prediction results and thus its prediction accuracy or the so-called score. The score is often observed metric in each iteration. Next observed metric is a loss. The loss metric is used as a quality estimator.

For instance, the mean squared error (MSE) may be used as the loss function. The loss function is actually being optimized (minimized) while training the model.

During the testing phase we pass the testing set and evaluate the predictions, whether they are as expected or not. Based on the predictions we evaluate model's accuracy.

Optimal model has roughly the same prediction scores for a training set as well as for a testing set, see Figure 4. If the scores differ a lot, we are either underfitting or overfitting, see Figures 5 and 6. In order to observe this behavior we need a validation set. The purpose of the validation set is to identify whether the model is still learning or starts overfitting.

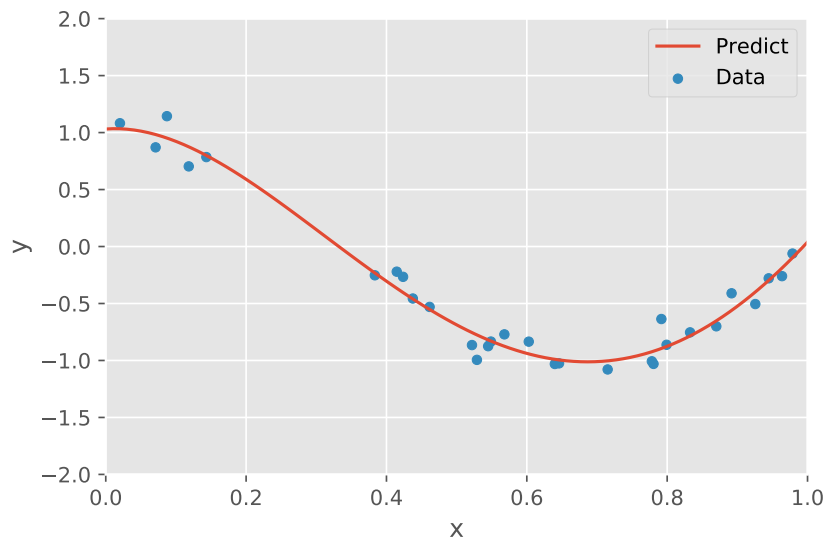


Figure 4: Regression model - optimal fit.

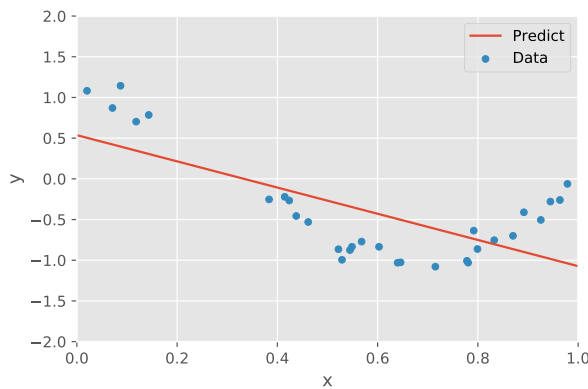


Figure 5: Regression model - underfitting.

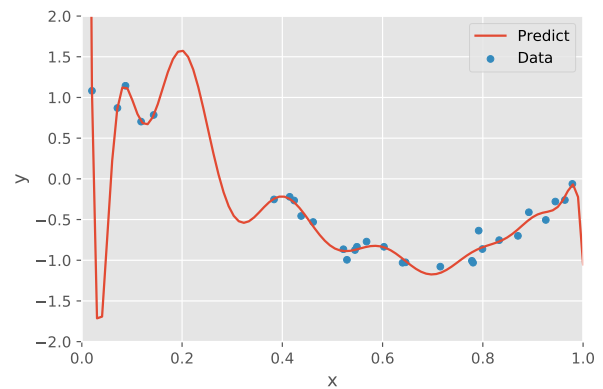


Figure 6: Regression model - overfitting.

The validation set is presented to the model during the training phase but only to predict the labels. The model must not learn from them. The predictions are then evaluated, so we calculate the accuracy and loss. It is expected that accuracy increases and loss decreases during the iterations. We have to make sure that accuracy and loss are similar for both training and validation sets. Those values must be similar also in the testing phase but this is checked later

on. Also, we have to check whether the trends of the accuracy are the same for both training and validation sets.

If both metrics are still getting better and better, we are fine. It means that accuracy is increasing and loss is decreasing for both sets, see Figure 7. Once the values are stabilized we should stop the training phase and move to the testing phase.

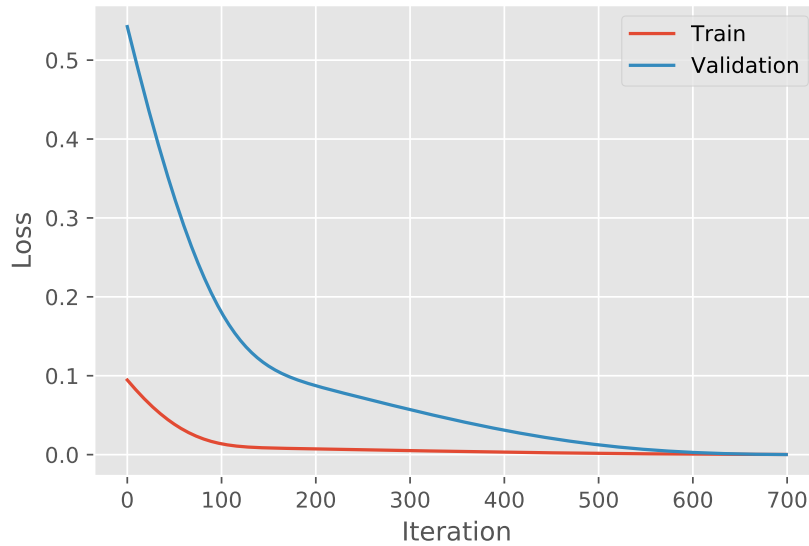


Figure 7: Model training - optimal fit, optimal training iterations.

If accuracy of the training set is good but accuracy of the validation set is bad, it indicates that we are underfitting. It may have two reasons. First, we did not perform enough train iterations, so the model did not have enough time to recognize all patterns, see Figure 8. We can improve the performance by performing more training iterations. Second, a model is not able to fit all patterns in its memory, see Figure 9. We can improve the performance by adding more memory capacity to the model (depends on the model type).

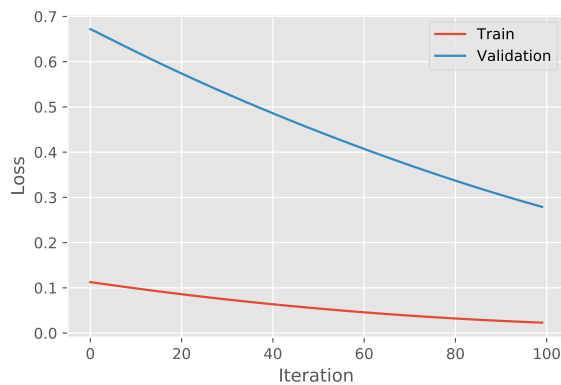


Figure 8: Model training - underfitting, not enough training iterations.

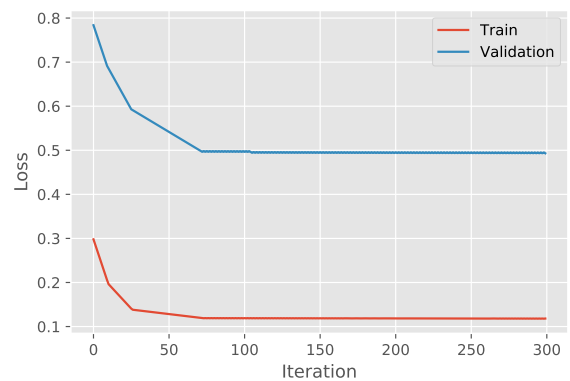


Figure 9: Model training - underfitting, not enough memory capacity.

If accuracy of the training set is still being improved but accuracy of the validation set starts stagnating or even decreases, it means that we are overfitting. We may observe similar behavior also on the loss metric. Training set is still decreasing its loss but validation starts stagnating or increases, see Figure 10. We can fix this by lowering the amount of iterations, exactly to the inflection point. Overfitting also means that a model simply learned the training data and is not able to generalize to unseen data.

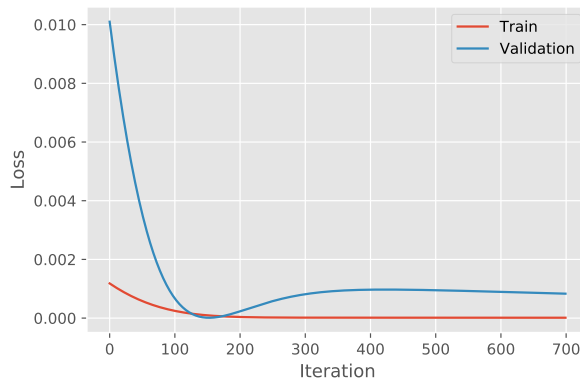


Figure 10: Model training - overfitting, too much training iterations.

3.4 Unsupervised learning

In unsupervised learning, we are given an unlabeled data. We will very briefly show one task that is usually done and it is a clustering. The clustering is about grouping objects together, so they form a cluster. Objects in same cluster should be similar to each other and dissimilar to the objects in other clusters.

There is a hierarchical clustering that use two approaches. First, bottom-up approach forms a cluster for each object and then merges them. Second, top-down approach forms one cluster and splits it recursively.

There is a centroid based clustering. Centroid is a virtual object that is the center of a cluster. Initially those centroids are chosen randomly from the training set. Then all the objects are assigned to a cluster based on the distance to the centroids. We have a set of objects in each cluster now and we shall recalculate the centroids again because they are no longer valid. As the centroids moved we reassign the objects to clusters again. We repeat this process couple of times till the centroids are moving. Once they converged we have a solution. We may repeat whole process and it is likely that we get a different solution since the initial state is random.

There are numerous clustering algorithms nowadays and since clustering does not have a single best solution, you usually use a few of them and compare the results. Look on a few data that is two dimensional and its calculated clusters, color represents a cluster, see examples in Figures 11 and 12. Which one is better and why?

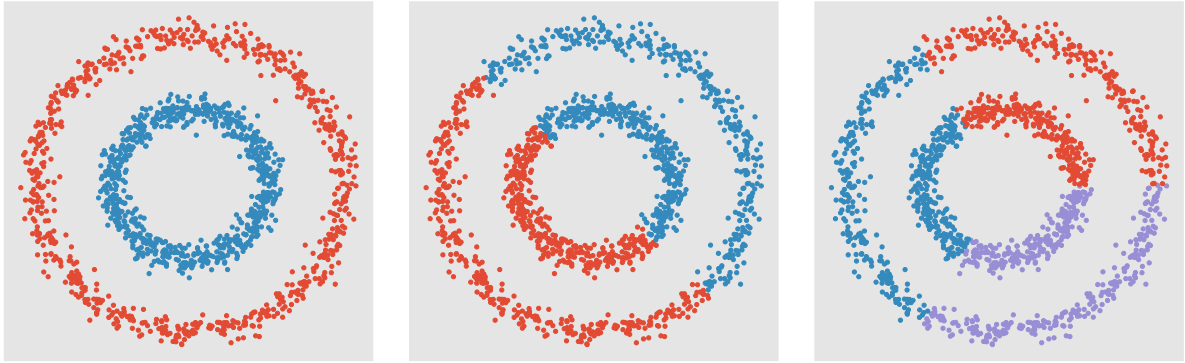


Figure 11: Clustering examples.

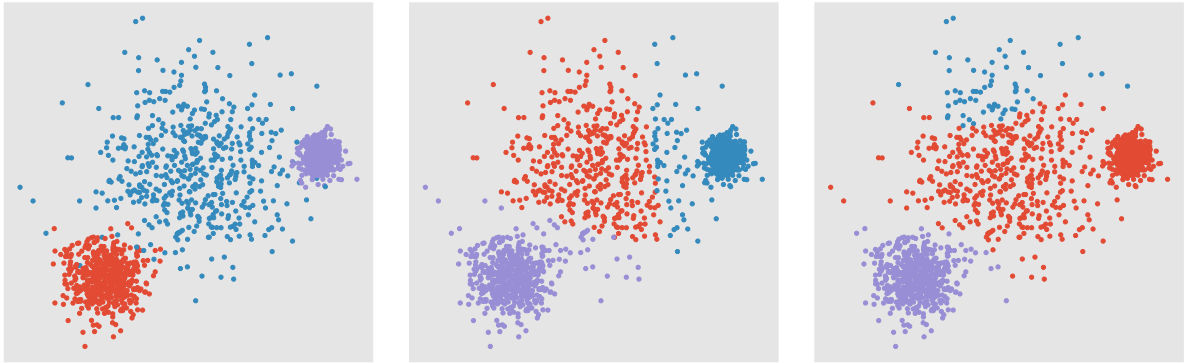


Figure 12: Clustering examples 2.

4 Neural networks

Neural networks are a popular machine learning technique. The following chapters give you an overview about neural networks. We will talk about how they were developed and its anatomy with a human brain. We will talk about the simplest neural network - a perceptron that is a fundamental unit in all neural networks. The history from the early beginning to the recent research will briefly be discussed. Also, we will mention a few of neural network types and what its applications are.

Neural networks are computational models that are inspired by a human nervous system. The human nervous system is made of cells named neurons. Neurons are connected to each other by synapses. The synapses are used to pass a transmission of signals from one neuron to another.

Neural networks are often explained as a simulation of a human brain but it is not true. Biological neurons can perform complex non-linear computations whereas neural networks are mostly working with layers. These layers are mostly forward only whereas biological neurons do not move forward only. Of course, there are special types of recurrent neural networks that do not have to move forward only and can even hold the information for next transmission but still, if we want to consider any neural network as a simulation of a human brain, it would be very naive simulation.

However, recent work on neural network shows that they are really valuable. Neural networks can be used for many machine learning tasks such as classification or clustering. Today, we use neural networks in many applications, a few of them are listed bellow

- Object detection or segmentation
- Text translation
- Speech recognition
- Document recognition
- Autonomous cars
- Generate or repair images
- Time series prediction
- Fraud detection

Neural networks are theoretically capable of memorizing anything, we just need a sufficient data and enough computation power.

4.1 Perceptron

Neural networks were inspired by a perceptron developed back in 1957 by Rosenblatt [7]. The perceptron is referred to as the simplest neural network [8]. The perceptron is a binary linear classifier. So, for instance the perceptron is able to distinguish a circle apart of a triangle. The perceptron takes a vector as an input and produces a single value output, either +1 or -1. We calculate the output value using following formula

$$y = \begin{cases} +1, & \text{if } \sum_{i=1}^n w_i x_i + b > 0 \\ -1, & \text{otherwise} \end{cases} \quad (4)$$

where

x is the input vector,

w is the weights vector,

n is the length of both vectors x and w ,

b is the bias,

y is the output.

Let's explain what is going on. The input vector actually forms variable values of a given object. The object has n features. The vector w consist of a weights, each weight w_i corresponds exactly to its input x_i . We might think of it as a weighted edge between the input and the perceptron. As we transmit the input through edges, we increase or decrease value using weights, this is done using a multiplication. Then, we sum up all the values, optionally add the bias and pass the result to an activation function. Our activation function is signum function. In the end, perceptron is just a linear function and an activation function. The bias is nothing else than the intercept of a function.

You can see the perceptron in Figure 13. The mentioned classification example with circles and triangles is in Figure 14.

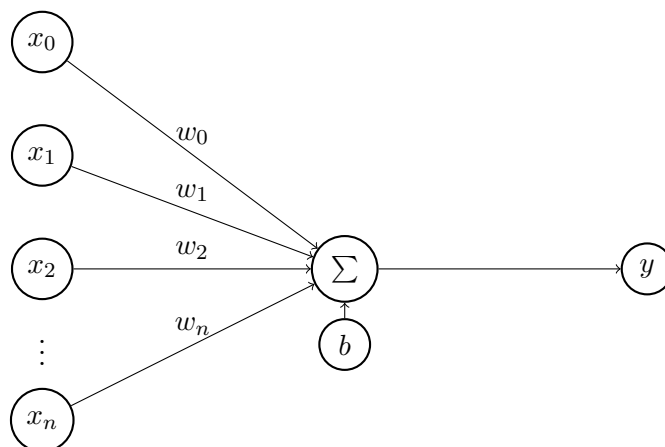


Figure 13: Perceptron.

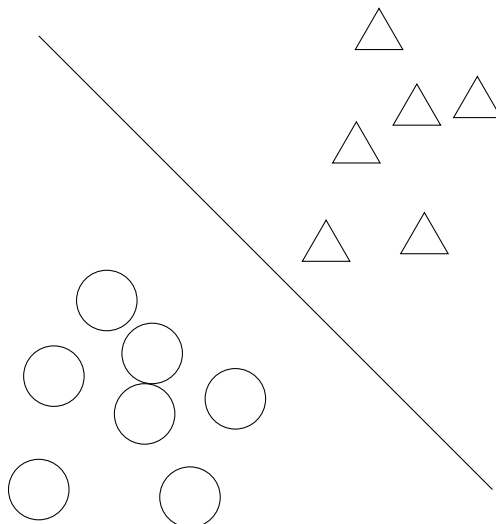


Figure 14: Binary classification using perceptron.

In the real world a single perceptron is not very useful. The perceptron has a limited expressiveness [9]. Suppose a simple XOR problem, see Table 5. It is not linearly separable and thus not solvable by the perceptron. By that fact research in neural networks slowed down that time.

| x_0 | x_1 | y |
|-------|-------|-----|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Table 5: XOR problem.

4.2 Multilayer perceptron

The first idea how to learn a multilayer perceptron (MLP) using a backpropagation algorithm was proposed in 1974 by Paul Werbos in his PhD thesis [10]. However, due to the skepticism of neural networks it did not become famous. Over 10 years passed and the backpropagation algorithm was proposed again by Rumelhart et al. in 1986 [11].

As its name refers, it is composed of more than one perceptron. In general, we usually split neural network into three pieces. An input layer, a hidden layer and an output layer. The input layer is there to transmit the inputs to the hidden layer only. The hidden layer is composed of multiple perceptrons. The last is the output layer that makes a prediction based on the last output of the previous hidden layer. The hidden layer usually does not have one layer only but

it can consist of multiple layers. Each layer is fully-connected with the previous/next layer. If you cut away any two adjacent layers and its edges, it forms a complete bipartite graph.

You can see multilayer perceptron in Figure 15. This neural network accepts 4 values as an input. There are 2 hidden layers, each consist of 8 neurons. Then, there are 3 output neurons. For the simplicity, the biases are hidden. In fact, when you are using large neural network they are not even needed.

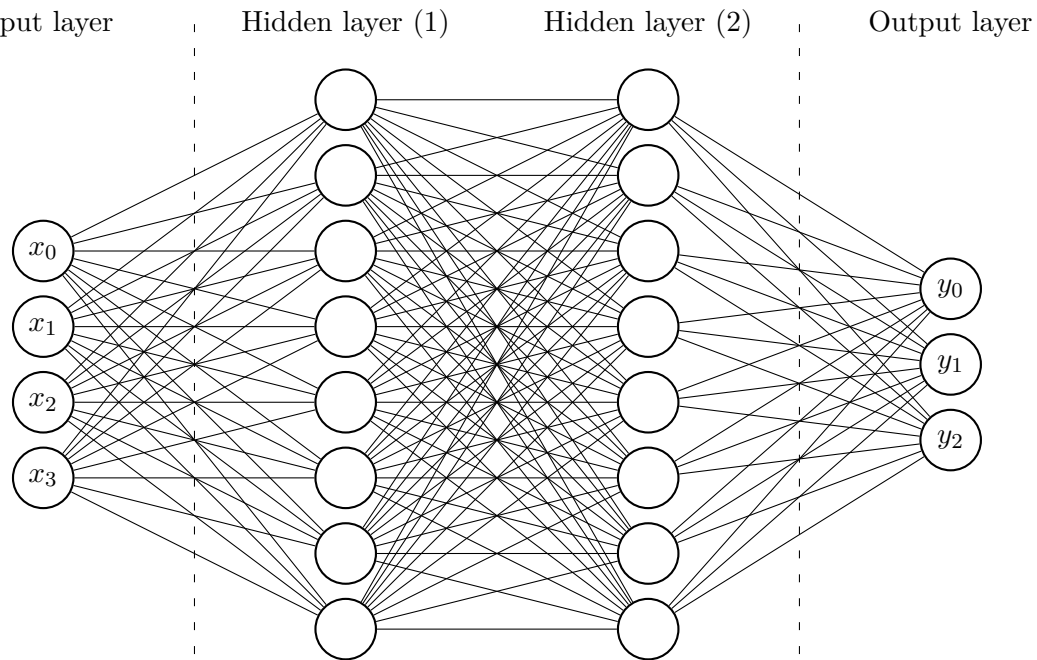


Figure 15: Multilayer perceptron.

The result is obtained using feed-forward mechanism. We pass the data into the input layer, it just distributes it to the first hidden layer. Each neuron in the first hidden layer computes its output. The outputs of the first hidden layer then acts as an input to the second hidden layer, it again computes its output and so on and so forth till we reach the output layer.

Activation function that we have shown in Equation 4 is not really practical for multilayer perceptron. Its successor was the sigmoid function that is defined as

$$y = \frac{1}{1 + e^{-x}} \quad (5)$$

However, in modern neural networks the sigmoid is considered as a historical and not commonly used. We rather use Rectified Linear Unit (ReLU) defined in Equation 6 or leaky ReLU defined in Equation 7. One of the reasons why to use ReLU over the sigmoid is faster convergence.

$$y = \max(0, x) \quad (6)$$

$$y = \begin{cases} x, & \text{if } x > 0 \\ 0.01x, & \text{otherwise} \end{cases} \quad (7)$$

Computations in the feed-forward mechanism can be done using massive parallelism. Each neuron in any layer is an individual unit that needs calculations from the previous layer only. Modern neural networks consist of hundreds to thousands of neurons in each layer. So, instead of using a Central Processing Unit (CPU) we rather use a Graphics Processing Unit (GPU) or rather a Tensor Processing Unit (TPU⁸). The speedup is enormous. A modern CPU can typically execute one or two arithmetic operations per instruction (without vector extensions), a GPU can execute thousands of operations per instruction and the TPU can multiply hundreds of thousands of operations (using a MatrixMultiply). Multiple neural network models were tested during the time in datacenters, the TPU is about 15-30 times faster than its contemporary GPU⁹ or CPU¹⁰ [12].

4.3 Types

This section will mention a few neural network types. However, we will not go too deeply as it is not the primary area of this work. We rather present the main idea of a specific neural network type and discuss what its applications are.

Hopfield networks

Hopfield networks were proposed as a model to store memory in 1982 [13]. Mathematically it forms a complete graph that is acyclic, weighted and undirected. If an object we would like to memorize has n binary features, we need also n neurons. Each neuron assumes binary values, usually 0 and 1 or -1 and $+1$.

You can see the Hopfield network with 6 neurons in Figure 16. Neurons are denoted from s_0 to s_5 with its x_i input, edges are represented as lines between the neurons, weight labels are not shown. Since the Hopfield network is undirected, weight w_{ij} from the neuron s_i to the neuron s_j is the same as w_{ji} .

The Hopfield networks are not typical machine learning technique that tries to find patterns within the data but rather to memorize such data. The Hopfield networks are designed to reconstruct the original data from just a portion of it. It may be useful for instance to reconstruct an image that we retrieved with invalid or missing pixels.

⁸Special type of Application Specific Integrated Circuit (ASIC) for the machine learning purpose, developed by Google in 2016

⁹Nvidia K80 GPU

¹⁰Server Intel Haswell CPU

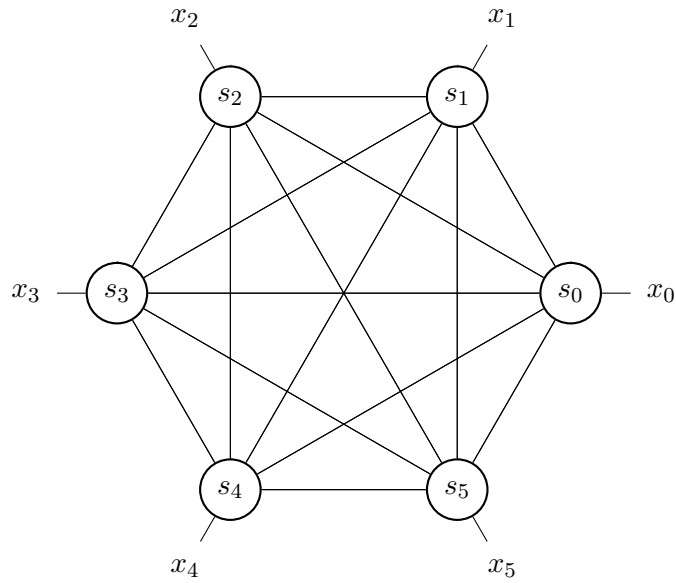


Figure 16: Hopfield network.

Recurrent neural networks

Traditional neural networks are producing output based on the given input only. Recurrent neural networks (RNN) are producing its output based on a given input and a previously given input. Once the data is fed in, it “holds” the information in the recurrent edges. This is handy for the data that is sequential such as time series, text or biological data. Text data can be formed as a sentence that is split into words. Each word is passed into the network one by one. Using this mechanism we are able to hold a semantics between the words.

The recurrent neural networks may be used to control a robot, drive a car, predict weather or a stock price, translate a text or recognize a speech.

Kohonen self-organizing map

A Kohonen Self-Organizing Map (SOM) is mostly used for visualization. It was introduced by Kohonen [14]. They are great to reduce a dimension of the data, we often reduce the dimension to 2 dimensions but we can use even 3 dimensions. For instance we use a 10x10 grid. So, we have 100 neurons in total. Each neuron is connected to all inputs and its neighbors to form a lattice. Two approaches are used, a rectangular with at most 4 neighbors or a hexagonal with at most 6 neighbors. Once the network is trained we might ask who the winner neuron for the given object is. The coordinates of the winner neuron are then joined with the given object. Once we have all the results for all our data, we can visualize it. It is basically a clustering algorithm but unlike traditional methods for clustering, the Kohonen self-organizing map gives you topological properties of the data. The Kohonen self-organizing map may be used to investigate similar text

documents, namely its topics. In general it gives us an easy way how to read and understand the data.

Convolutional neural networks

Convolutional neural networks (CNN) are designed to work with grid structured inputs. The vast majority of usages of convolutional neural networks focus on image data. The convolutional neural network is much like traditional feed-forward multilayer perceptron, except that the operations in its layers are spatially organized with sparse connections between layers [8]. Typically they are composed of three types of layers, namely convolution, pooling and the ReLU. Those layers are used multiple times. The last layer is usually the multilayer perceptron neural network.

The main idea in convolutional neural networks is a convolution operation. It is a dot product operation between the input and grid structured set of weights, formerly called a filter. The filter is smaller grid (e.g. 3x3 or larger) that is applied to all valid locations of the input. As we slide the filter over the locations, it forms a new grid which is called an activation map or a feature map. We may think of filters as feature identifiers. So, it can detect low level features such as an edge, a color or a curve. Those detected features are then fed into the multilayer perceptron which classifies the data based on features found by filters.

The convolutional neural networks are used for classification, object detection, segmentation and natural language processing.

4.4 Hyperparameters

Neural networks are definitely great choice for many machine learning tasks. They do have a really huge potential in many areas, however, they are extremely difficult to be understood. You may have trained a neural network model that fits all the requirements but then you ask, why it answers like this? What are the patterns that we were not able to distinguish ourselves? Well, there are convolutional neural networks that can be visualized quite easily but for all other types, it is hard and they often form what is called a black box model. Of course, neural networks are not the only machine learning model that is considered as a black box, there are plenty of others.

During the training phase, as we pass all the training data into the model, we have called it as one iteration so far. In the neural network terminology it is often called as an epoch. Unfortunately, we do not know how many epochs we need in advance. This is, fortunately, the easy part because we can stop the training anytime. The hard part is a neural network architecture, so how many neurons in each layer do we use and what are its activation functions? In general, smaller architecture is considered better because it would require less computations, so usually you train and predict faster. Both epochs and architecture are variables that we tune and we call them hyperparameters.

Finding the best hyperparameters is really hard because there are millions of possible combinations. Also, since we start with initial state that is randomized, each training is different and the final state may vary significantly. Imagine how time consuming it is when one training session requires several hours or even days. Experiences may help to find a suitable hyperparameters faster but there is not a straightforward recipe for it. However, there are recommendations. For instance when you preprocess features to be within range from 0 to 1, neural network converges (learns) faster [15].

Evolving hardware and thus increasing computing power significantly helped to develop automated machine learning solutions. These solutions try to search through the whole hyperparameters space to find the best hyperparameters for the given data on its own. Auto-Keras provides functions to automatically search for architecture and hyperparameters of deep learning models [16]. DARTS: Differentiable Architecture Search is able to efficiently design high-performance convolutional architectures for image classification and recurrent architectures for language modeling [17].

5 Application

5.1 Requirements

The application is addressed to the domain experts in biology and biochemistry. It is important to have intuitive user interface and the best user experience as possible because users are not expected to have deep knowledge of machine learning. The application has to be able to handle workload of dataset that consist of thousands to tens of thousands of instances with tens to hundreds of attributes. The application should help them to explore and visualize the data, preprocess the data, create a model for a classification task and use model for prediction.

From the technical perspective, the application should have simple and modular architecture, so it is easy to integrate new features in the future.

5.2 Use cases

Typical workflow is shown in Figure 17. Data collection is external thing but we want to copy the data into the application. In other words, we have to have a dataset stored in the application, so we can work with it. Dashed rectangle shows application boundary what it is capable of.

As we have the dataset in the application, we can see what objects and what features it has in its raw form, so we can show it as a table.

The dataset can be preprocessed. Two options are available, namely split and modify. Split simply divides the given dataset into two new datasets based on split ratio, for instance for training and test sets. Modify gives us ability to remove, normalize, encode or fix missing values. Result can be stored as a new dataset and/or a configuration. The configuration can be used in exploration analyses and model creation, so we can test whether one applied configuration gives us better results than others. This configuration principle is important because the same preprocessing that we apply during training has to be applied also for new unseen objects otherwise we would get invalid results.

The exploration analysis is also an important part. Based on the feature type, the application provides basic statistical values (count, mean, standard deviation, min, Q1, median, Q3 and max) and plots such as cumulative histogram, bar plot, box plot or violin plot. The configuration can be also applied.

The dataset can be used as a training set and we can create a classification model using neural network. Labeled dataset and optionally a configuration are expected. Then, we can use either simple or advanced variant. Simple variant is really easy to use, all you have to do is click on a few buttons that you want to either train less/more or have less/more capacity. The advanced variant is more sophisticated and you basically design the whole architecture on your own. You can tune the hyperparameters using a test run and if results are promising, you fill model's name and save it. The model can be used to predict unseen data. Hierarchy of use cases are presented in Figures 18 and 19.

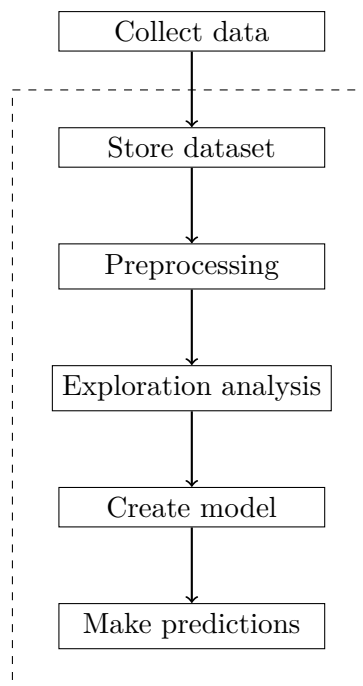


Figure 17: Application workflow.

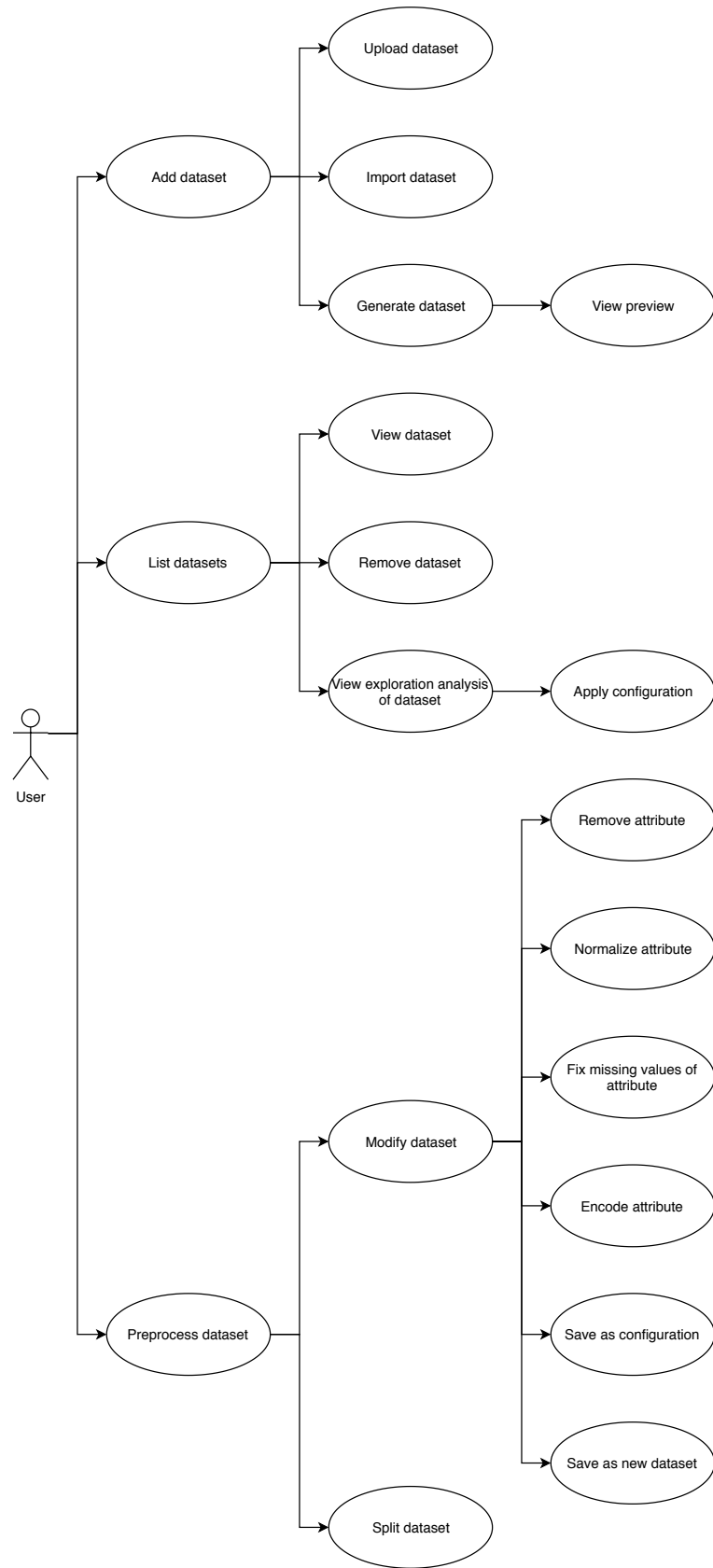


Figure 18: Application - datasets use cases.

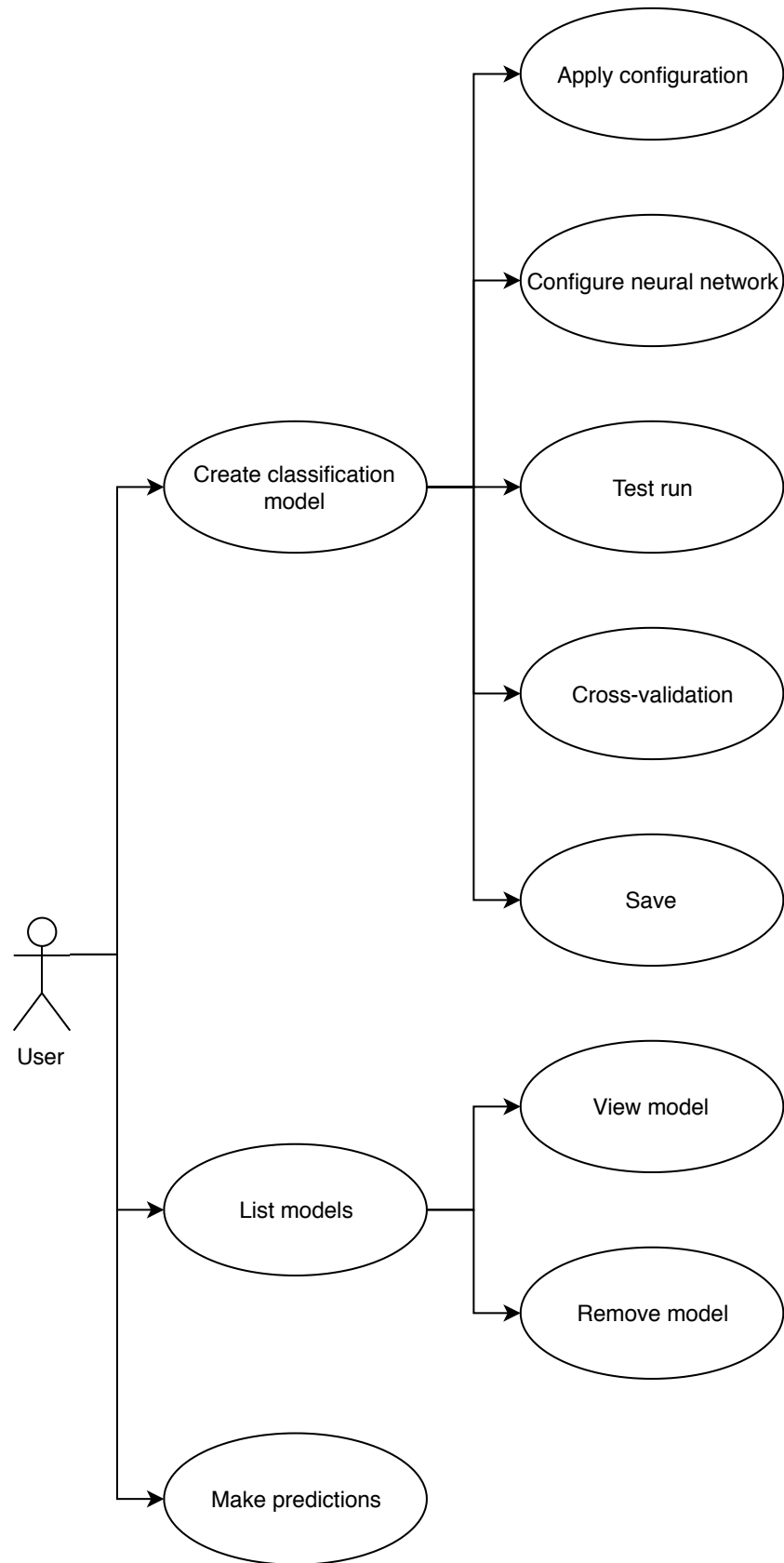


Figure 19: Application - models use cases.

5.3 Technology stack

The application uses standard client-server approach. In our case client is called as frontend and server is called as backend. Backend is written in the Python and frontend is written JavaScript. All used libraries are open source under a permissive software license. Here we list important libraries with a summary. Component diagram is presented in Figure 20.

5.3.1 Backend

Flask

Flask¹¹ is a micro web framework. It is used to create an API. Internally it is based on the Werkzeug¹² that is a comprehensive WSGI web application library. It is often used to serve the web application but since we want only some form of the API we use Flask-RESTPlus. Flask-RESTPlus is an extension for Flask that adds out-of-the-box support for REST API. It also has a documentation endpoint in an OpenAPI specification format, formerly known as Swagger¹³. Swagger is not only a specification but also a set of tools that can generate a code (client software development kits) and documentation.

Pandas

Pandas¹⁴ is a high-performance library for data manipulation and data analysis. It is designed to make working with data easy and intuitive. Internally it uses NumPy¹⁵ which is a fundamental package for scientific computing in the Python because it can represent multi-dimensional arrays in an efficient way.

Matplotlib

Matplotlib¹⁶ is a plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms [18]. It is used to create plots such as histograms, bar charts, boxplots or violin plots. Plots are usually in 2D but there is also a support in 3D.

Scikit-learn

Scikit-learn¹⁷ is a set of simple and efficient tools for machine learning [19, 20]. It has a lot of implemented state-of-the-art algorithms under the hood for all machine learning tasks such as classification, regression, clustering and dimensionality reduction. It supports dataset preprocessing but it also has tools to generate datasets with a given properties.

¹¹<https://www.palletsprojects.com/p/flask>

¹²<https://palletsprojects.com/p/werkzeug>

¹³<https://swagger.io>

¹⁴<https://pandas.pydata.org>

¹⁵<https://www.numpy.org>

¹⁶<https://matplotlib.org>

¹⁷<https://scikit-learn.org>

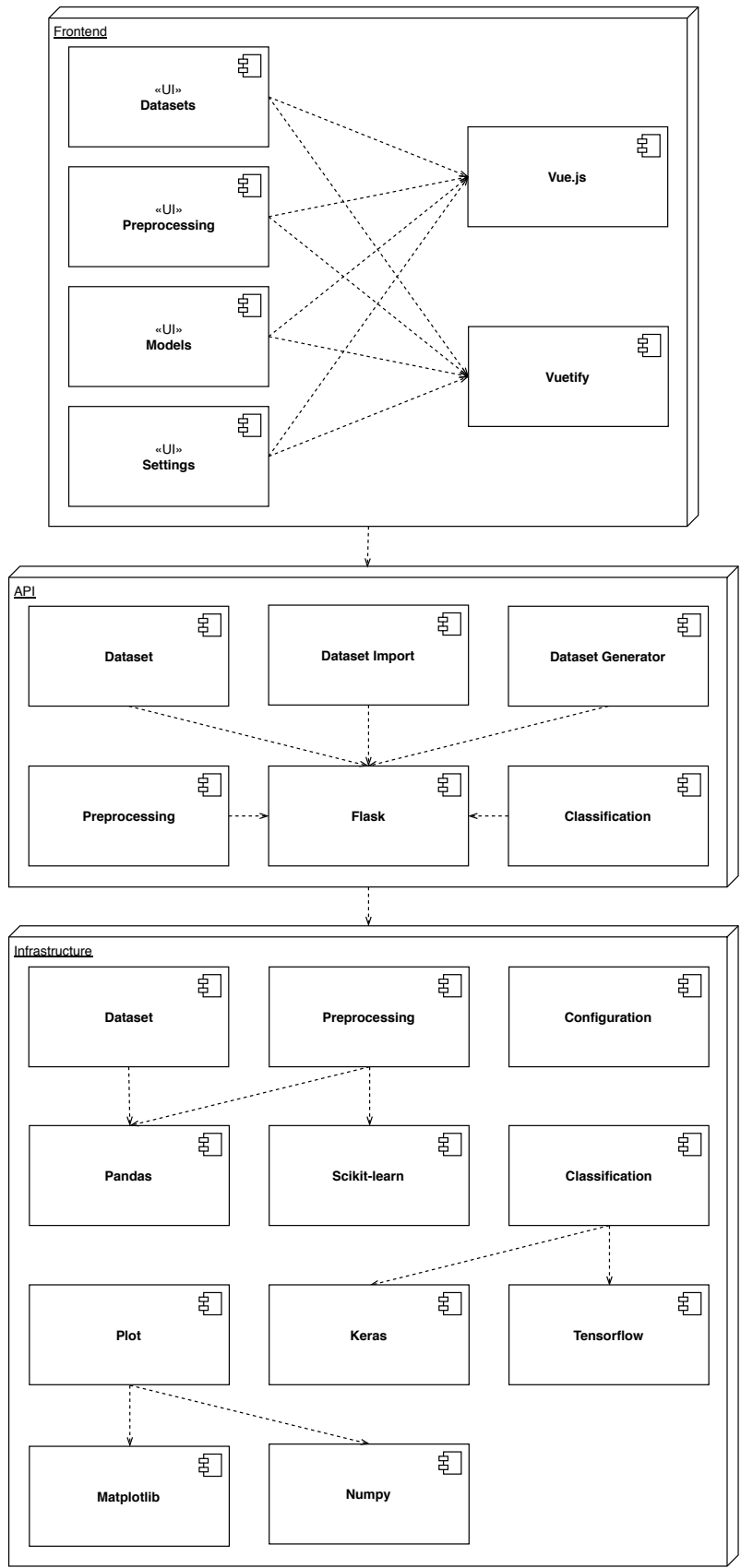


Figure 20: Application - component diagram.

TensorFlow

TensorFlow¹⁸ is a library for numerical computation using data flow graphs [21]. The graph nodes represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them.

Keras

Keras¹⁹ is a high-level neural networks API [22]. Keras offers consistent and simple APIs. It is capable of running on top of TensorFlow and other backend engines.

5.3.2 Frontend

Vue.js

Vue.js²⁰ is a progressive framework for building user interfaces and single-page applications (SPA). It uses an HTML-based template syntax for views or reusable components. Both views and components are combination of HTML, JavaScript and CSS.

Vuetify

Vuetify²¹ is a material design component framework for Vue.js. It aims to provide clean, semantic and reusable components. Vuetify is developed according to the Material Design²² specification. It supports all modern browsers on devices with vary resolutions from mobile to desktop.

5.3.3 Deployment

Frontend is built into a few static files that are served by Flask. Flask's web server is not intended to be used in the production and does not scale well. For that reason we use Gunicorn. Gunicorn²³ is a Python WSGI HTTP Server for UNIX. It's a pre-fork worker model. Once the Gunicorn starts, it creates a few *nix forks which handle requests in completely separate process.

Deployment often involves tons of problems. Famous quote is "it works on my machine" and therefore we have decided to use containers, namely Docker containers, in order to avoid such situations. Containers are an abstraction layer that packages code and dependencies together. We can run multiple containers on the same machine and share OS kernel with other containers, each in its isolated process in user space, it is much more effective and less error prone compared to virtual machines. This guarantees that applications will always run the same because everything the applications needs is shipped with it.

¹⁸<https://www.tensorflow.org>

¹⁹<https://keras.io>

²⁰<https://vuejs.org>

²¹<https://vuetifyjs.com>

²²<https://material.io>

²³<https://gunicorn.org>

Docker

Docker²⁴ is a platform for developers and sysadmins to develop, ship, and run applications. It is also a daemon process running on the host which manages images and containers, see Figure 21.

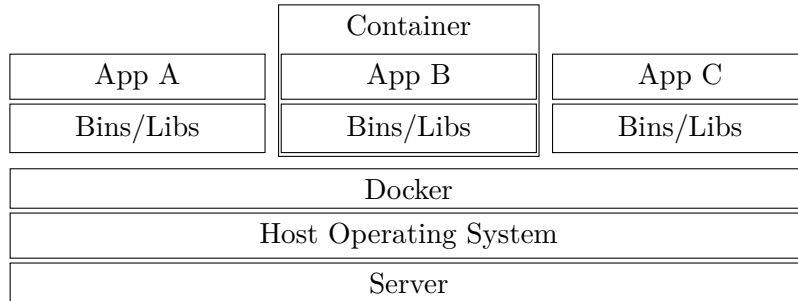


Figure 21: Docker overview.

Docker image is a lightweight, standalone, executable package of software that includes everything needed to run an application. Each image is a read-only template used to create and run a container. Docker images are typically created as union of layered file systems stacked on top of each other. Each layer adds one or more things and creates a new image. Images can be both Windows and Linux. In our application we use the Linux image with preinstalled Python and TensorFlow as our base image and add new custom layer that adds frontend and backend, as a result it forms a new image.

Docker image is built using a Dockerfile. Dockerfile is a script composed of instructions how the image should be created.

Once we have an image, we can materialize the image into a container. Container is a runtime instance of a Docker image. Container is intended to be stateless, so there should be no writing to the union file system because once the container is stopped and removed, all changes are lost. If the application need to bypass the union file system you shall use volumes. Volume is a link of paths between host's file system and container's file system. It adds us the ability to persist data outside of a container, so even if container is removed, data remains. It can be useful for instance to pass a configuration file to the container. Settings are usually set through environment variables though.

Necessary Docker commands to build, install and deploy the application are listed in Chapter B in appendix.

5.4 Modules

This section shows how the application looks like and how it is intended to be used. The first view that you will face when you open the application is shown in Figure 22. Its purpose is to provide some sort of quick navigation of usual workflow.

²⁴<https://www.docker.com>

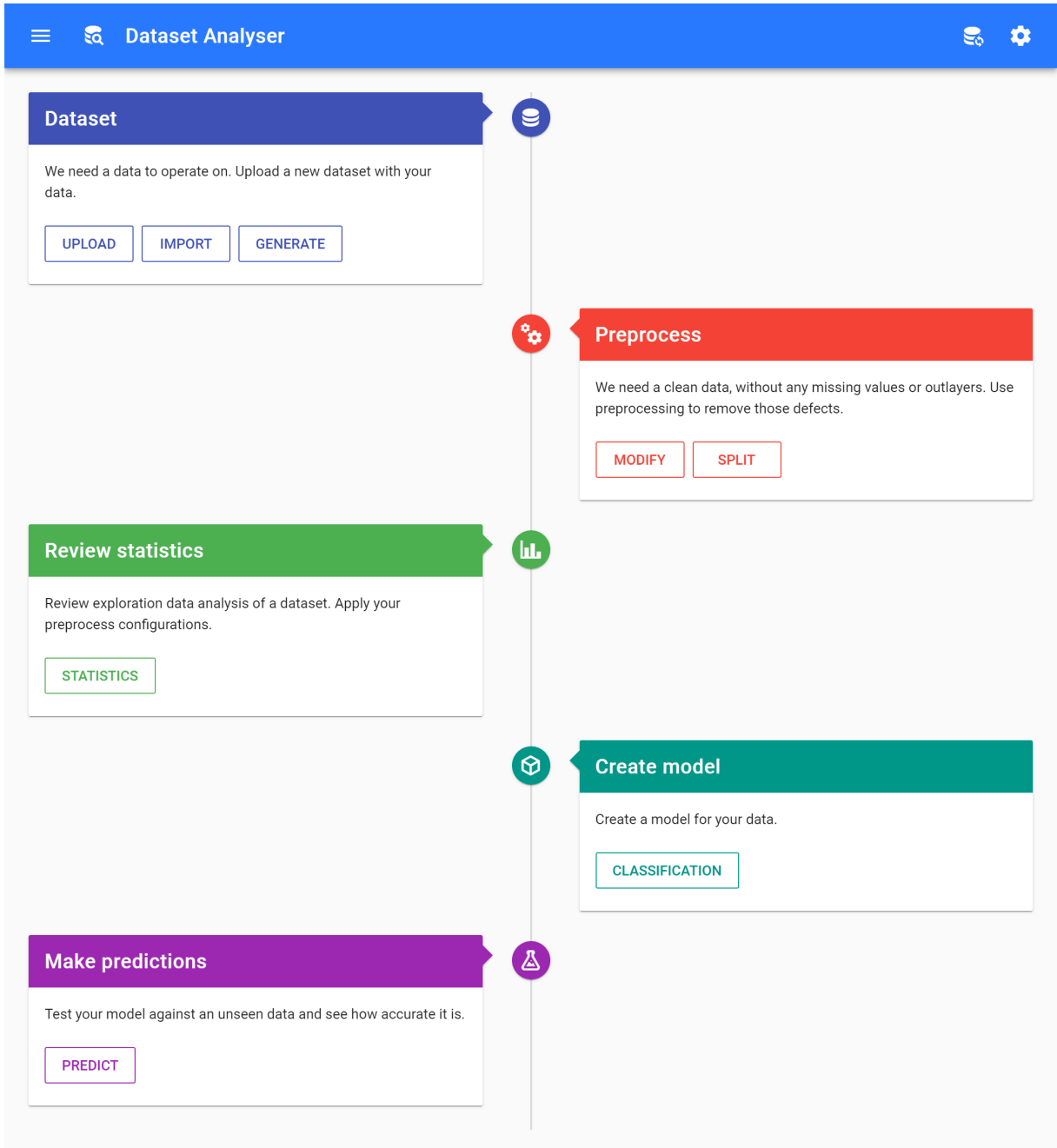


Figure 22: Application - workflow.

5.4.1 Datasets

All datasets are expected to have header on first line. Floating point numbers are expected to use dot as a decimal point. In case of CSV, separator is not strict and underlying engine should detect it automatically. It is recommended to use standard separators such as ',' or ';' though. In case of Excel, first sheet is loaded only. Missing values can be interpreted by any of the following values: "", "#N/A", "#N/A N/A", "#NA", "-1.#IND", "-1.#QNAN", "-NaN", "-nan", "1.#IND", "1.#QNAN", "N/A", "NA", "NULL", "NaN", "n/a", "nan", "null". Example of dataset in CSV format can be seen in Figure 23.

| <code>id,timestamp,temperature,weather</code> | \Rightarrow | <table border="1"><thead><tr><th>id</th><th>timestamp</th><th>temperature</th><th>weather</th></tr></thead><tbody><tr><td>0</td><td>10</td><td>30.5</td><td>hot</td></tr><tr><td>1</td><td>11</td><td>-15.5</td><td>cold</td></tr><tr><td>2</td><td>NA</td><td>NA</td><td>NA</td></tr></tbody></table> | id | timestamp | temperature | weather | 0 | 10 | 30.5 | hot | 1 | 11 | -15.5 | cold | 2 | NA | NA | NA |
|---|---------------|--|---------|-----------|-------------|---------|---|----|------|-----|---|----|-------|------|---|----|----|----|
| id | timestamp | temperature | weather | | | | | | | | | | | | | | | |
| 0 | 10 | 30.5 | hot | | | | | | | | | | | | | | | |
| 1 | 11 | -15.5 | cold | | | | | | | | | | | | | | | |
| 2 | NA | NA | NA | | | | | | | | | | | | | | | |
| <code>0,10,30.5,hot</code> | | | | | | | | | | | | | | | | | | |
| <code>1,11,-15.5,cold</code> | | | | | | | | | | | | | | | | | | |
| <code>2,,NaN,NA</code> | | | | | | | | | | | | | | | | | | |

Figure 23: Application - dataset example.

Upload

Dataset upload is quite straightforward, you select dataset file from your local file system and click on the upload button. You can even select multiple datasets, see Figure 24. Theoretically, if the application runs on your machine you can move/copy datasets directly on the file system and not even interact with the application.

Import

Dataset import provides an integration with the OpenML platform. You can explore available datasets²⁵ there and if you would like to do analysis on the dataset you just found, you can copy the url and paste it into the import form. The application automatically fetches basic information such as name, version and licence. If everything looks fine, you can start the import using the import button, see Figure 25. Keep in mind that import might take a while since everything is going through the network.

Generate

Dataset generate is a feature mainly intended for learning purpose. You can choose from up to 7 dataset types. There are types for all machine learning tasks, each type has different set of parameters. All types can choose how many instances will be generated. Then, depend on the task type, there are parameters such as number of features, number of informative features, number of classes, standard deviation of each cluster and noise. There are two buttons available, namely preview and save. Preview generates dataset based on the given parameters and then plots its first 3 features, if available. It can plot 1D, 2D but also 3D. If there are more than

²⁵<https://www.openml.org/search?type=data>

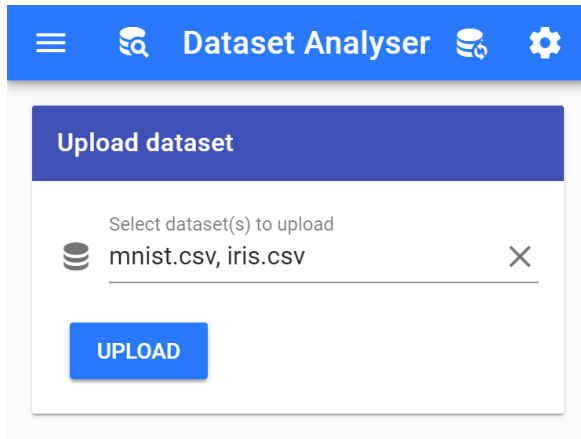


Figure 24: Application - dataset upload.

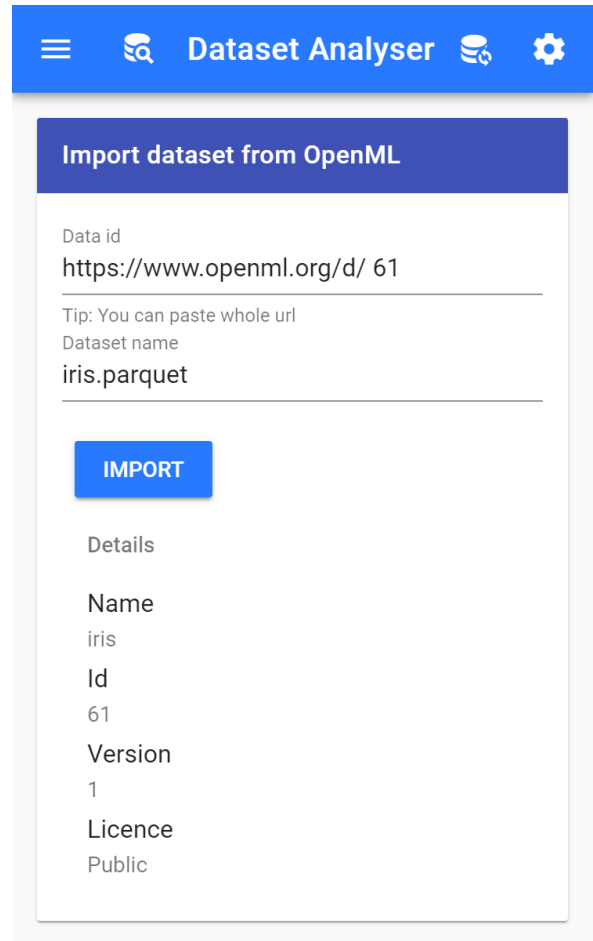


Figure 25: Application - dataset import.

3 features, only first 3 are used for a preview. Each preview is unique because it uses always different random seed. Usually you click preview a few times and when preview looks promising you fill dataset's name and click save.

List

Dataset list provides tabular view for all available datasets, see Figure 26. Table is composed of following columns: dataset type, name, size, date and time of create, date and time of last modification, and actions. All columns are sortable by clicking on its header name. Actions column offers two actions, namely detail and delete. Search is available above the table. The number of shown rows can be changed using pagination located under the table.

Detail

Dataset detail shows dataset as a table. The first column is row index and the rest are features. Search is available above the table. In case there are some missing values, their corresponding

The screenshot shows the 'Dataset Analyser' application interface. At the top, there is a blue header with a menu icon, a search icon, the text 'Dataset Analyser', and a settings icon. Below the header is a dark blue bar with the word 'Datasets'. Underneath is a search bar with a magnifying glass icon and a 'REFRESH' button with a refresh icon. The main content is a table with the following columns: Type, Name ↑, Size, Created at, Last modified at, and Actions. The table contains 8 rows of dataset information. At the bottom of the table, there is a pagination control showing 'Rows per page: 10' and '1-8 of 8' with navigation arrows.

| Type | Name ↑ | Size | Created at | Last modified at | Actions |
|------|----------------|--------|-----------------------|------------------------|---------|
| | iris-test.csv | 1 kB | 4/7/2019, 10:15:55 PM | 4/7/2019, 10:15:55 PM | |
| | iris-train.csv | 3 kB | 4/7/2019, 10:15:55 PM | 4/7/2019, 10:15:55 PM | |
| | iris.csv | 5 kB | 4/7/2019, 10:15:04 PM | 3/15/2019, 12:45:47 PM | |
| | mnist.csv | 238 MB | 4/7/2019, 10:13:52 PM | 3/12/2019, 10:09:56 AM | |
| | mnist.feather | 439 MB | 4/7/2019, 10:13:54 PM | 3/12/2019, 10:10:52 AM | |
| H5 | mnist.h5 | 440 MB | 4/7/2019, 10:13:57 PM | 3/12/2019, 10:12:13 AM | |
| | mnist.parquet | 21 MB | 4/7/2019, 10:14:04 PM | 3/13/2019, 8:16:06 PM | |
| | mnist.pickle | 439 MB | 4/7/2019, 10:13:52 PM | 3/12/2019, 10:11:37 AM | |

Figure 26: Application - list of datasets.

cell is highlighted with red color and also entire row is highlighted too. Missing values are replaced with NA value and thus they can be easily searched.

Settings

Settings is located at the top right corner. It opens modal window where you can select datasets you are working with. It can be handy when you have a lot of datasets stored and do not want to filter through all of them every time. Button next to it refreshes all datasets because they are loaded only once when you open the application.

5.4.2 Preprocessing

Modify

Modify is what we have called a preprocessing so far. It has a different name because there are other modules for similar purpose. However, preprocessing offers 4 modifications, namely remove, normalize, encode and fix missing values. Of course, those options are available with

respect to the feature type. For instance, remove is available for each attribute whereas normalize is available on numeric attributes only. There are up to 4 buttons available on the top that applies the modification on all features because you almost always have to fix missing values and encode categorical features.

Each option has its own parameters. For instance normalization interval is within 0 and 1 but you can choose any interval. Encoding offers both label encoding and one-hot encoding. Fix missing values has 5 options, namely: discard, 0, average, the most common and custom value.

Eventually, we can choose whether we modify the dataset and save it as a new one, save as a configuration, or both.

An example can be seen in Figure 27. Here we remove id attribute. Timestamp attribute is normalized. Temperature attribute is normalized and missing values are fixed using average value. Missing values of weather attribute are discarded and then label encoding is applied. Results are saved as a new dataset and also a configuration.

Split

Split is used to divide dataset into two disjoint sets. The size of each set is set using split ratio. Typically, you use this when you need to split a dataset into train and test sets.

5.4.3 Exploration analysis

Exploration analysis provides descriptive statistics for each attribute. A configuration is optional, so it can be applied or not. Each attribute has its own card with information and corresponding plots. NA values are excluded. Categorical attribute shows count, number of classes and its distribution in bar plot. Continuous attribute shows count, mean, standard deviation, min, Q1, median, Q3 and max. Cumulative histogram and combination of box and violin plots are displayed too. All the calculations and plots are cached, so first load usually takes a while but all following should be instant. There is no cache invalidation.

5.4.4 Models

Create classification

Classification model creation requires a labeled dataset. A configuration is optional, as usual. There are two variants, namely simple and advanced. The simple variant is addressed to anybody who does not know anything about neural networks. You tune hyperparameters using six buttons. Two for epochs, two for number of layers and last two for number of neurons. The advanced variant is addressed to people with a fundamental knowledge of neural networks. You can specify exactly number of epochs, validation split and each layer in neural network.

Internally it builds multilayer perceptron with the ReLU activation function and output layer using the softmax function. Adam is used as default optimizer [23]. Loss and accuracy metrics are observed for training set and also for validation set, if provided.

☰
Dataset Analyser
🔍 ⚙️

Modify dataset - remove, fix missing, normalize or encode values

Dataset

weather.csv

REMOVE 1/4

NORMALIZE 2/2

FIX MISSING VALUES 2/2

ENCODE 1/1

id (int64)

Remove

timestamp (int64)

Remove

Normalize Min Max

temperature (float64)

Remove

Normalize Min Max

Fix missing values Discard Using 0 Using average Using the most common Custom

weather (object)

Remove

Fix missing values Discard Using the most common Custom

Encode Label/Integer encoding One-Hot encoding

New modified dataset name

Configuration name

SUBMIT

Figure 27: Application - modify preprocessing.

Creating a model is usually a long term run, so you test a few hyperparameters and compare what gives you better results. Test run is supposed to be some sort of trial run. Model is not stored anywhere. When you are satisfied with the model's accuracy, you can fill model's name and click create. It creates again a new model that may be a slightly different than in test run. Keep in mind that learning a neural network is not deterministic since we always start with random weights.

Both test run and create shows plots of metrics (accuracy and loss), predictions as bar plot (correct and incorrect bars for each class) and confusion matrix.

Cross validate runs a stratified k-fold cross-validation. Predefined options for k are 3, 4, 5 and 10. For each fold a model is created, trained and tested. Accuracy for both train and test sets are shown in plots. Final accuracy is calculated as mean of achieved accuracies on test sets.

List

Models list shows all created models. There are two buttons next to each model, so you can quickly go to model's detail or remove the model.

Detail

Model detail shows a summary of a model. There is the dataset name that was used as training set, configuration, label, input dimension (number of features), output dimension (number of classes), train score and hyperparameters. There are also plots that were shown during creation, so accuracy and loss metrics, predictions.

Predict

Model predict is used to test a model against an unseen data. Dataset and model have to be selected and then you click predict button. The application loads the model and dataset from disc into memory and passes the data into model. Result is again overall accuracy, predictions plot and confusion matrix.

5.5 Storage

Storage is designed using directories on the underlying file system. There are following folders:

```
/data
├── datasets
├── models
│   └── classification
├── preprocessing
└── statistics-cache
```

All datasets are located in *datasets* folder. Folder *models* has *classification* subfolder. Each model that is saved creates its own subfolder there. Name of the model is the folder's name

and there are always 2 files, namely *metadata.json* and *model.h5*. Preprocessing configurations are stored in *preprocessing* folder. There is one subfolder per dataset and then there are its corresponding configurations. The last folder is *statistics-cache* that consist of cache files only.

Dataset formats

The application supports following dataset formats, allowed file extensions are in brackets:

- CSV (*csv*)
- Excel (*xlsx*, *xlsm*)
- Pickle (*pickle*)
- Feather (*feather*)
- Parquet (*parquet*)
- HDF (*hdf*, *hdf5*, *h5*)

We have made a quick comparison between the formats, results are presented in Figure 6. The dataset we used is MNIST, it is database of handwritten digits, 70000 instances and 785 features. First 784 features are representing pixels, so values are 0 – 255 and the last feature is label that is 0 – 9.

First of all, we could not even generate such big Excel file, so it was excluded from the comparison. CSV is a standard text format that is being used a lot. However, as data gets bigger, it starts getting slow. We must mention that once we preprocess the dataset it suddenly takes 362 MB. The rest are binary formats. Pickle is Python format, it is just a serialized object²⁶. It performed quite fine. Feather format provides binary columnar serialization²⁷. In our case it has the fastest write and read. Parquet²⁸ is a bit slower but its precedence is in stored size, it takes 20 times less space on disc. Small file size is achieved using multiple layers of encoding and compression. It is really efficient for low entropy data. HDF performs quite similar compared to others. It is mainly designed for extremely large and complex data.

Let's sum up the dataset formats. If you want to have the best performance and do not care about storage space, use Feather. Otherwise, use Parquet, you are likely to save a lot of storage space.

5.6 API

Backend exposes API using REST. Overall, API is exposed on path */api/<version>*, so far there is only one version named *v1*. OpenAPI specification is available on path */api/v1/swagger.json*

²⁶<https://docs.python.org/3.6/library/pickle.html>

²⁷<https://github.com/wesm/feather>

²⁸<https://arrow.apache.org/docs/python/parquet.html> or <https://fastparquet.readthedocs.io/>

| Format | Size [MB] | Write [s] | Read [ms] |
|---------|-------------|-------------|------------|
| CSV | 232 | 51 | 6930 |
| Pickle | 429 | 3,83 | 632 |
| Feather | 429 | 0,48 | 247 |
| Parquet | 20,7 | 2,22 | 387 |
| HDF | 430 | 2,04 | 321 |

Table 6: Dataset format comparison.

and Swagger UI on path `/api/v1/docs`. File `swagger.json` can be used, for instance, to generate client code, there are over 50+ supported clients now²⁹. Swagger UI is handy for development because it offers a list of all available routes, including its parameters, methods (GET, POST, PUT, DELETE), responses and so on. Each route is also a form where you can fill parameters and execute it, see Figure 28.

The screenshot displays the Swagger UI interface for a REST API endpoint. At the top, the endpoint is identified as `GET /dataset/{dataset}`. Below this, the 'Parameters' section shows a single parameter: `dataset`, which is a required string (path). A text input field contains the value 'dataset'. A blue 'Execute' button is positioned below the parameter field. To the right of the parameters section is a red 'Cancel' button. The 'Responses' section shows a response with a status code of 200 and a description of 'Success'. The response content type is set to 'application/json'. At the bottom of the interface, the `DELETE /dataset/{dataset}` endpoint is partially visible.

Figure 28: Application - Swagger UI.

²⁹<https://editor.swagger.io>

5.7 Experiments

5.7.1 Iris experiment

Iris flower dataset³⁰, sometimes known as Iris dataset, is a dataset of flowers. It consists of following attributes:

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class (Iris Setosa, Iris Versicolour and Iris Virginica)

There are 150 instances in total, 50 for each class. We normalize all attributes and encode class using label encoding. We use 100 epochs and two hidden layers, each with 16 neurons, see Figure 29. Results of test run are presented in Figure 30 and results of 10-fold cross validation are presented in Figure 31.

³⁰<http://archive.ics.uci.edu/ml/datasets/iris>

Dataset Analyser

Classification model creation

Dataset
iris.csv

Configuration
 normalized and encoded

Label
class

Model name

SIMPLE **ADVANCED**

Time
- + 100

Size
- + - + 16 - 16

TEST RUN **CROSS VALIDATE** CREATE

Figure 29: Application - Iris experiment - settings.

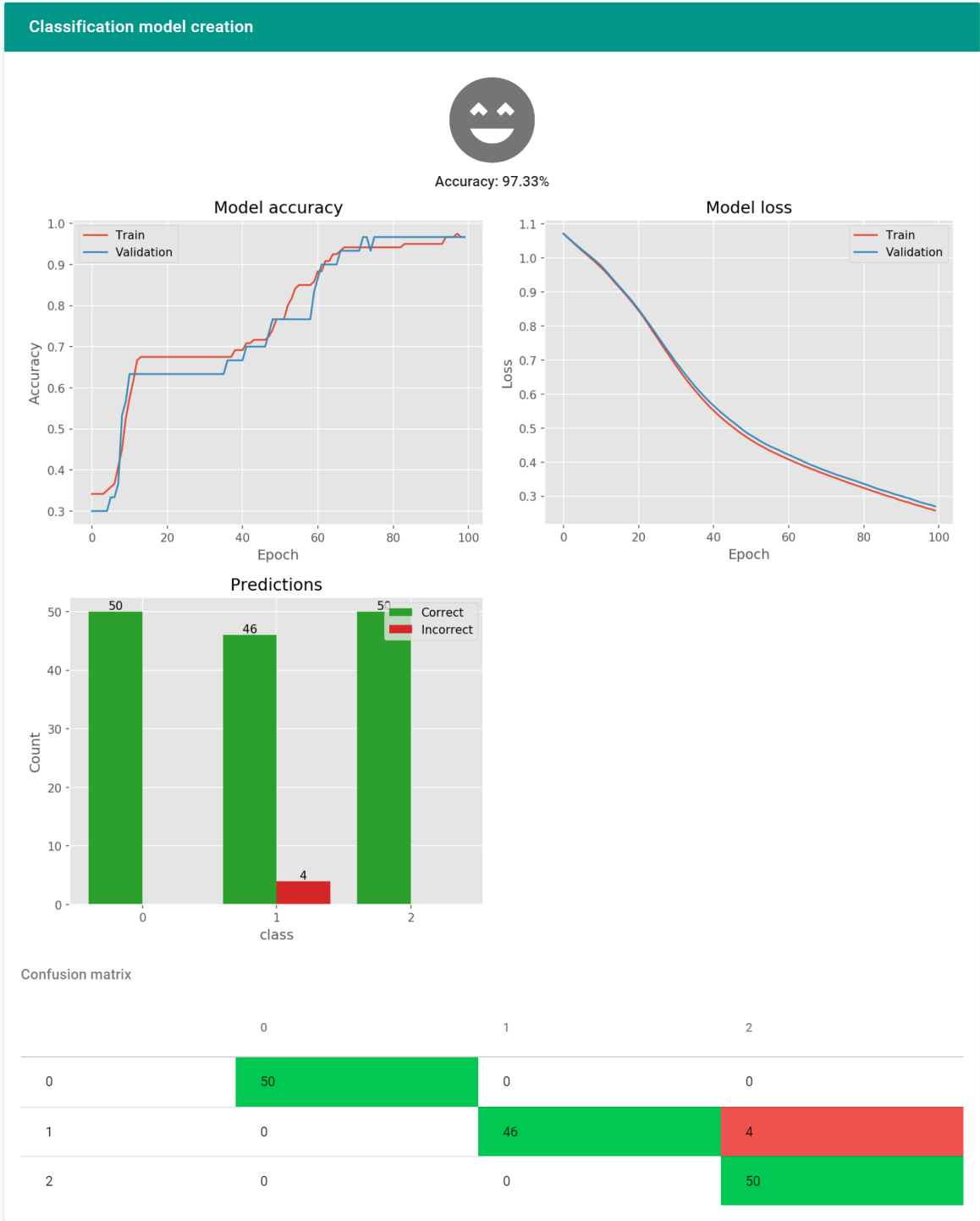


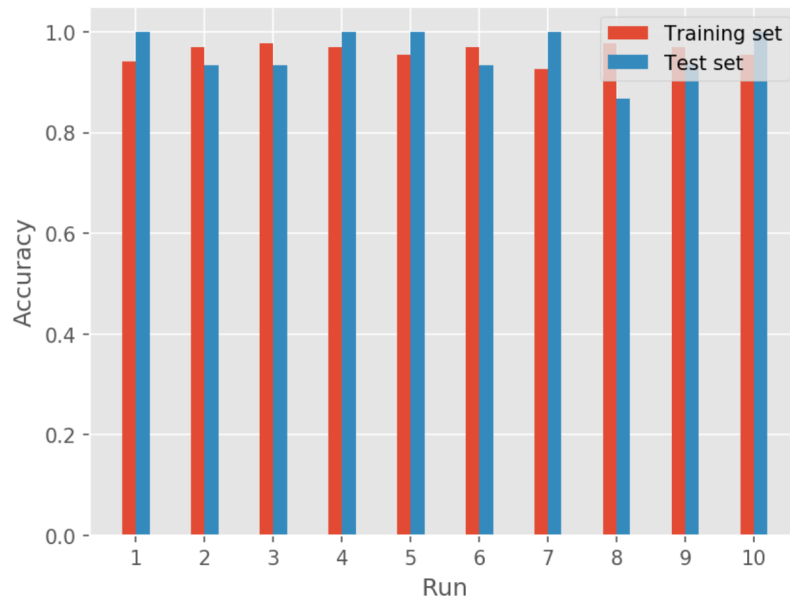
Figure 30: Application - Iris experiment - test run results.

Classification model creation



Cross-validation accuracy: 96%

Cross-validation



Cross-validation

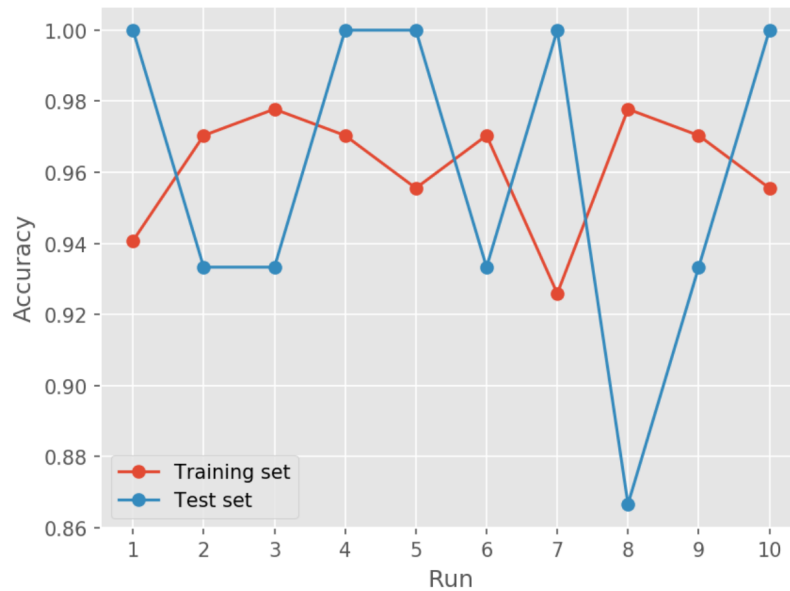


Figure 31: Application - Iris experiment - 10-fold cross-validation results.

5.7.2 Leukemia experiment

Leukemia dataset consists of 187 patients. There were 5 different categories in label, namely: CLL, MCL, MZL, DLBCL and SLL but we have removed the DLBCL and merged together CLL and SLL. So, we have CLL/SLL, MCL and MZL labels which we try to predict. More details about the data are presented in [24].

In all experiments we have removed meaningless features like Sample, second class type, and DoublePair. We have also removed all instances with missing values and normalized all features. Label called class is encoded using label encoding. The data is unbalanced, there are 160 instances of the CLL/SLL, 20 instances of the MCL and 7 instances of the MZL.

Classification of all classes at once. We used all 14 features, 40 epochs, two hidden layers, each with 64 neurons and 10-fold cross-validation results into 95% accuracy. We used subset of features, namely: CD200, CD23, CD79b, FMC7. Neural network was trained on 70 epochs, two hidden layers were used, each with 16 neurons and 10-fold cross-validation results into 95% accuracy. Then, we removed FMC7 feature and 10-fold cross-validation results into 94,5% accuracy.

Classification of the CLL/SLL vs. MCL. We used all features, 80 epochs, two hidden layers, each with 32 neurons and 10-fold cross-validation results into 98,3% accuracy.

Classification of the CLL/SLL vs. MZL. We used all features, 80 epochs, two hidden layers, each with 32 neurons and 10-fold cross-validation results into 97,6% accuracy.

Classification of the MCL vs. MZL. We used all features, 120 epochs, two hidden layers, each with 32 neurons and 10-fold cross-validation results into 80% accuracy.

Experiments were performed multiple times, the most common result was taken into account. All results are expected and comparable to the results in the aforementioned article, see Table 7.

| Experiment | All classes [%] | CLL/SLL vs. MCL [%] | CLL/SLL vs. MZL [%] | MCL vs. MZL [%] |
|--------------------------|-----------------|---------------------|---------------------|-----------------|
| Article ¹ | - | 98,8 | 98,5 | 76,7 |
| Application ¹ | 95 | 99,4 | 97,6 | 80 |
| Application ² | 94,5 | 98,8 | 98,8 | 80 |
| Application ³ | 95 | 98,3 | 97,6 | 80 |

¹ Used attributes: CD200, CD23, CD79b, FMC7

² Used attributes: CD200, CD23, CD79b

³ All attributes

Table 7: Leukemia experiment results.

6 Conclusions and future work

In this thesis we have reviewed core concepts of machine learning and neural networks. We have developed the software environment for researchers and domain experts in biology and biochemistry specializing in the field of biomedical data analysis. This software environment allow them prepare and visualize data, plan experiments with exchangeable configurations and see results using appropriate visualizations. The software environment does not require any programming or machine learning in-depth knowledge.

Thanks to the integration with the OpenML environment and dataset generators the new software environment capabilities have been created that were not initially required. The software environment can be used as a tool to teach/practice neural networks or machine learning in general.

The software environment can be extended in several ways. Other machine learning tasks such as clustering or feature selection should be supported. Traditional machine learning algorithms such as decision tree may be added. Static plots generated on backend can be replaced with interactive visualization that all modern browsers are capable of.

References

- [1] Yun Liu et al. “Detecting Cancer Metastases on Gigapixel Pathology Images”. In: *CoRR* abs/1703.02442 (2017). arXiv: 1703.02442. URL: <http://arxiv.org/abs/1703.02442>.
- [2] Andre Esteva et al. “Dermatologist-level classification of skin cancer with deep neural networks”. In: *Nature* 542.7639 (2017), p. 115.
- [3] Paras Lakhani and Baskaran Sundaram. “Deep learning at chest radiography: automated classification of pulmonary tuberculosis by using convolutional neural networks”. In: *Radiology* 284.2 (2017), pp. 574–582.
- [4] Joaquin Vanschoren et al. “OpenML: Networked Science in Machine Learning”. In: *SIGKDD Explorations* 15.2 (2013), pp. 49–60. DOI: 10.1145/2641190.2641198. URL: <http://doi.acm.org/10.1145/2641190.2641198>.
- [5] Max Bramer. “Introduction to Data Mining”. In: *Principles of Data Mining*. London: Springer London, 2013. ISBN: 978-1-4471-4884-5. DOI: 10.1007/978-1-4471-4884-5_1. URL: https://doi.org/10.1007/978-1-4471-4884-5_1.
- [6] R. Bellman, Rand Corporation, and Karreman Mathematics Research Collection. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957. ISBN: 9780691079516. URL: <https://books.google.it/books?id=wdtoPwAACAAJ>.
- [7] R. Rosenblatt. “The perceptron: A perceiving and recognizing automaton”. In: *Report* 85.460 (1957), p. 1.
- [8] Charu C Aggarwal. *Neural networks and deep learning*. Springer, 2018.
- [9] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 1969.
- [10] P. J. Werbos. “Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences”. PhD thesis. Harvard University, 1974.
- [11] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: <https://doi.org/10.1038/323533a0>.
- [12] Norman P Jouppi et al. “In-datacenter performance analysis of a tensor processing unit”. In: *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2017, pp. 1–12.
- [13] John J Hopfield. “Neural networks and physical systems with emergent collective computational abilities”. In: *Proceedings of the national academy of sciences* 79.8 (1982), pp. 2554–2558.
- [14] Teuvo Kohonen. “The self-organizing map”. In: *Proceedings of the IEEE* 78.9 (1990), pp. 1464–1480.

- [15] J Sola and Joaquin Sevilla. “Importance of input data normalization for the application of neural networks to complex industrial problems”. In: *IEEE Transactions on nuclear science* 44.3 (1997), pp. 1464–1468.
- [16] Haifeng Jin, Qingquan Song, and Xia Hu. “Auto-Keras: Efficient Neural Architecture Search with Network Morphism”. In: *arXiv preprint arXiv:1806.10282* (2018).
- [17] Hanxiao Liu, Karen Simonyan, and Yiming Yang. “Darts: Differentiable architecture search”. In: *arXiv preprint arXiv:1806.09055* (2018).
- [18] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.
- [19] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [20] Lars Buitinck et al. “API design for machine learning software: experiences from the scikit-learn project”. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.
- [21] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: <http://tensorflow.org/>.
- [22] François Chollet et al. *Keras*. <https://keras.io>. 2015.
- [23] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).
- [24] David Starostka et al. “Quantitative assessment of informative immunophenotypic markers increases the diagnostic value of immunophenotyping in mature CD5-positive B-cell neoplasms”. In: *Cytometry Part B: Clinical Cytometry* 94.4 (2018), pp. 576–587.

A How to develop

Prerequisites

You need to have installed Yarn or npm, and Python 3.5 or 3.6. I personally developed on two Windows 10 machines with the following versions

1. Yarn 1.12.3, npm 6.5.0 and Python 3.6.6.
2. Yarn 1.9.4, npm 6.2.0 and Python 3.6.8.

Note: As a package manager I used PIP that should be already packed together with Python.

Backend

Open terminal in the backend folder. First, you have to download libraries that the application relies on. If you want to use virtual environment, you run following commands

```
python -m venv venv
venv\Scripts\activate
pip install -r requirements.txt
```

Otherwise, if you do not want to use virtual environment, you just run

```
pip install -r requirements.txt
```

Then, you can run the backend

```
python app.py
```

You should see couple of application logs and then Flask's information about runtime info such as environment type, debug mode and the URL it is listening on. It should be running on `http://127.0.0.1:5000`. Verify using curl or browser that backend is up and running. Backend API documentation is located on `http://localhost:5000/api/v1/docs`, there you can see Swagger documentation of exposed REST API.

Frontend

Open terminal in the frontend folder. I use yarn, so following examples are in yarn. If you are using npm, just change yarn to npm in commands. You need to download all dependencies first.

```
yarn install
```

Then, you can run the frontend using

```
yarn serve
```

This starts a development server using vue-cli-service that is listening on `http://localhost:8080`. Internally, it is also a proxy for a backend, so you usually develop against 8080 port. If you wish the opposite way you have to uncomment `frontend_proxy` function located in `backend/app.py`, then you can visit backend port 5000.

B How to build and install

Prerequisites

You need to have Docker only. I had Docker version 18.09.2, build 6247962.

How to build a docker image

You need terminal and in the root folder and you just run

```
docker build -t thesis:latest .
```

Note: you may rename image name to whatever you like.

This command builds the application, the build commands are located in the Dockerfile, see Listing 1. It uses multi-stage build, so we have the final image as small as possible. First, we use node:lts-alpine as base image for the frontend build. We install dependencies using yarn and build the frontend. Then we can start building the main image. As the base image we use tensorflow/tensorflow:1.13.1-py3. We copy frontend artifacts to this image. We install all the Python libraries and we start the gunicorn web server with 4 workers.

```
FROM node:lts-alpine as frontend-build
WORKDIR /app
COPY frontend .
RUN yarn install
RUN yarn build

FROM tensorflow/tensorflow:1.13.1-py3
RUN apt-get update && \
    apt-get install -y --no-install-recommends && \
    pip install --upgrade pip setuptools gunicorn
WORKDIR /app
COPY --from=frontend-build /app/dist frontend
COPY backend .
RUN pip install -r requirements.txt
CMD [ "gunicorn", "-w", "4", "-b", ":5000", "--log-level", "debug",
      "--timeout", "3600", "app:app" ]
```

Listing 1: Dockerfile.

How to run the docker image

Once you have your image in your local repository, you can run it. The simplest version is following

```
docker run --name thesis -p 5000:5000 thesis
```

This starts a container named thesis from the thesis image, it also binds port host's port 5000 to the port 5000 of the container.

This is definitely not good for the production environment though. All the data is persisted in the container, so once you shut it down and remove, they are gone. So, we need volume binding also.

```
docker run --name thesis -p 5000:5000 -v "YOUR_CUSTOM_PATH":/app/data thesis
```

For the Windows machine it can be something like this

```
docker run --name thesis -p 5000:5000 -v C:\thesis-data:/app/data thesis
```

This persists all the application data on the host machine, so if you remove the container and start it again with volume binding, the data is still there.

How to export and import the docker image

Sometimes you do not want to download the image from the official Docker Hub because it has roughly 1,5GB. So, for that case it can be handy to export the image as a physical file that we can copy to a different machine.

So, we can save it to thesis.tar file

```
docker save -o thesis.tar thesis
```

Then, we can also load it from the file into our local repository

```
docker load -i thesis.tar
```
