University of Manouba
National School of Computer Science

Summer Internship Report

Implementing an end-to-end Machine Learning Pipeline for Time Series, Health Data

---

Realized by :
Walid CHTIOUI

Organization : PPR Technologies inc.
Supervised by : Mariem ABID, Amal KHABOU
Address : 1120 rue Hélène-Boullé Boucherville (Québec) J4B 2A7 Canada
Phone : (+1) 514 654 4200
Email : abid.mariem@gmail.com, a.khabou@gmail.com

Academic Year :
2021-2022

Table des matières

Introduction

Building an efficient real-world machine learning model is a tedious task that is costly to realise not only in terms of time but also in terms of required expertise. For instance, figuring the right set of hyperparameters for a deep learning model is a difficult task that requires an expertise in the field of deep learning. Other than that, the right data preprocessing techniques should be set properly as these tend to heavily influence the performance of a machine learning model. Moreover, building such models involves an iterative process of machine learning operations such as data preprocessing, fitting models, tuning parameters, until specific acceptance criteria(s) is met. Thus, having a system that supports the entire modelling process by automating some parts of it and making it cheap to iterate can help data scientists to quickly perform experiments and deploy efficient models. The suggested solution for the hosting organism is an end-to-end, highly configurable machine learning pipeline that is specifically designed to deal with time-series health data. This solution efficiently automates various machine learning operations. The project also puts a high emphasis on extensibility, modularity and high configurability. An extensive application programming interface (API) that implements numerous data-science tools is provided for better flexibility.

Chapitre 1

Project General Context

This chapter aims to provide a general presentation of the project starting with the hosting organism then a discussion about the problematic we're aiming to solve. After that, existing solutions are discussed and the reasons why they can't be used for this problematic are provided. We conclude this chapter with a description of the provided solution.

1.1 Hosting Organism

PPR Technologies inc. is a digital health startup, based in Montreal, Canada, developing algorithms for predicting the risk of complications for people with chronic diseases using machine learning techniques on data collected via monitoring medical devices.

1.2 Project Presentation

The realisation of this project took place remotely from June to August of the year 2022 as a summer internship for the second year of Computer Science studies in the National School of Computer Science in Tunisia.

1.2.1 Problem Statement

Machine learning model development is an extremely time-consuming, iterative task that requires an expertise in the field of deep learning in order to generate high-performance and high-quality models. An automated machine learning pipeline helps automate these tasks and allows data scientists, analysts and developers to build ML models with high scale, efficiency and productivity all while sustaining model quality. A pipeline in machine learning is a technical infrastructure that allows an organization to organize and automate machine learning operations. The logic of the pipeline and the range of tools it incorporates varies based on the business requirements. In our case, the automated ML pipeline has to specifically work on health data, mostly in the form of time-series data, to generate deep learning classifiers.

1.2.2 Existing Solutions

There are numerous existing automated machine learning (AutoML) solutions such as Auto-sklearn, Auto-PyTorch and AutoKeras. Although these projects are mature and have proven their effectiveness in automatically yielding high quality models, they are designed to work on a wide variety of data sets thus making them too generic to perform very well on health data sets, on particular, time-series data sets. Moreover, working on health data sets requires a

specific set of data preprocessing techniques and machine learning workflows that may not be fully integrated in these projects.

1.2.3 Provided Solution

The objective is to design and build an end-to-end, fully automated machine learning pipeline that significantly reduces the costs of generating and deploying high-quality deep learning models. The pipeline should mainly be used by both non-expert users and developers. A user provides a data set and the pipeline automatically generates a high quality deep learning model. The developers should be easily able to customize the pipeline. Project requirements are further discussed in chapter 3.

Chapitre 2

Preliminary Study

This chapter aims to provide a very brief explanation of the most important fundamental concepts and technologies. It starts by providing a short explanation of deep learning. Then a section about hyperparameter optimization is provided. The chapter ends with a description of electrocardiogram data which is the main type of health data this project works on.

2.1 Deep Learning Basics

2.1.1 Definition

Deep learning (DL) is a subset of machine learning (ML), which uses multiple artificial neural network (ANN) layers to extract high-level features from raw input data. In DL each layer transforms the input data into a more abstract and composite representation. Generally, the more hidden layers a DL model has the better, more refined and more accurate the predictions it makes. DL has seen significant success in various applications that require very minimal to none human intervention. Computer vision, speech recognition and natural language processing (NLP) are among the fields where DL has seen the most success [2].

The figure 2.1 describes the global architecture of DL models. The input layer is where the input data is fed to the model. The output layer outputs model predictions. The layers in between are called hidden layers, each one of these layers is responsible for transforming the layer's input data into a more abstract representation.

2.1.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a deep learning algorithm that can capture the spatial and temporal dependencies in the input data, usually in the form of an image, through the application of relevant filters. While in primitive methods filters are hand-engineered, with enough training, CNNs have the ability to learn these filters and extract high-level, important features. The architecture performs a better fitting to the input dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the input data better. In conclusion, CNN reduces the input into a form which is easier to process, without losing features which are critical for getting good predictions [6].

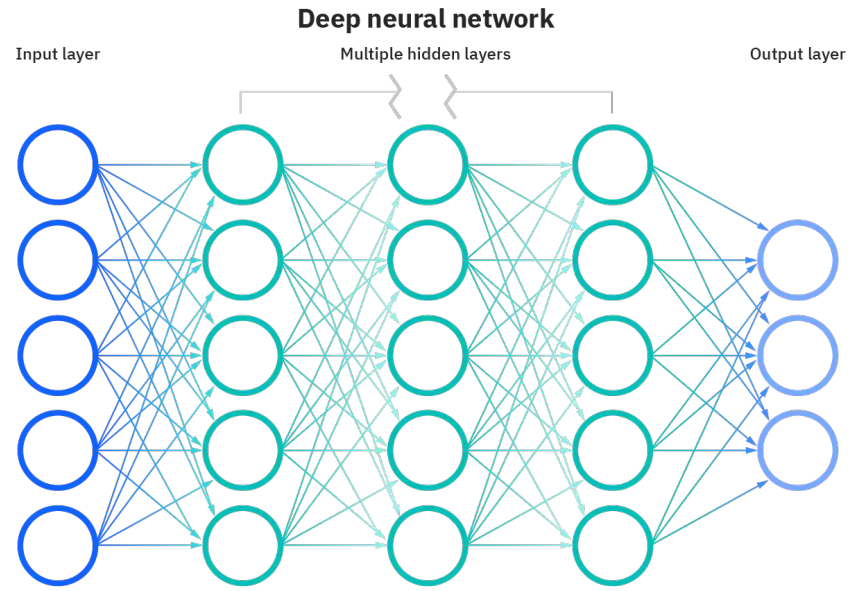

FIGURE 2.1 – Architecture of a Deep Learning Model.

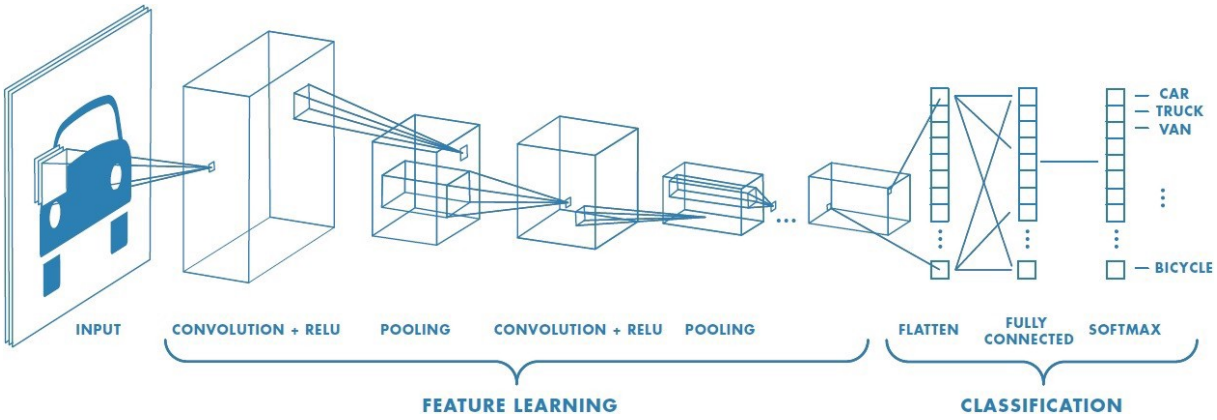

FIGURE 2.2 – CNN Architecture.

The figure 2.2 describes the global architecture of a CNN model. As depicted in this figure, there are two main stages : **Feature Learning** and **Classification**. Feature learning stage is where the model extracts the most important features through a series of filters and pooling layers. Then, these extracted features are fed to the classification stage where the layers are trained on the training data set to produce predictions.

2.1.3 Hyperparameters

Hyperparameters are parameters whose values are set before starting the model training process and are used to control it. Hyperarameters should not be confused with parameters that are set during the training process, e.g. node weights. DL models, can have anywhere from a few hyperparameters to a few hundred hyperparameters [3].

The following list provides some commonly-used examples of hyperparameters :

— Choice of optimization algorithm (e.g., gradient descent, stochastic gradient descent, or Adam optimizer)

— Learning rate in optimization algorithms
— Choice of activation function in a NN layer (e.g. Sigmoid, ReLU, Tanh)
— Number of hidden layers in a NN
— Number of activation units in one particular layer
— Kernel size in a convolutional layer
— Filer size in a convolutional layer
— Pooling size

The following list lists the most important characteristics of hyperparameters :

— Hyperparameters can significantly influence the time required to train and test a model, e.g. the hyperparameter representing the size of a layer affects the speed at which the model is trained.
— Hyperparameters may be conditionally provided upon the value of other hyperparameters. e.g. the size of a hidden layer can be conditionally provided depending on whether the layer is added to the model or not.
— Hyperparameters are usually of continuous or integer type (categorical), leading to mixed-type optimization problems.

2.2 Hyperparameter Optimization

Definition

Hyperparameter optimization (HPO) objectively searches different values for model hyperparameters and chooses a subset that results in a model that achieves the best performance on a given dataset. The result of a HPO is a single set of well-performing hyperparameters that you can use to configure your model. An optimization procedure involves defining a **search space**. This can be thought of geometrically as an n-dimensional volume, where each hyperparameter represents a different dimension and the scale of the dimension are the values that the hyperparameter may take on, such as real-valued, integer-valued, or categorical. A point in the search space is a vector with a specific value for each hyperparameter value. The goal of the optimization procedure is to find a vector that results in the best performance of the model after learning according to a specified metric such as accuracy or minimum error [1].

There are several approaches to HPO including [4] :

— **Manual Search** : select hyperparameters based on intuition/experience/guessing, train the model with the selected hyperparameters, and score on the validation data. Repeat process until you run out of patience or are satisfied with the results.
— **Grid Search** : set up a grid of hyperparameter values and for each combination, train a model and score on the validation data. In this approach, every single combination of hyperparameters values is tried which can be very inefficient.
— **Random Search** : set up a grid of hyperparameter values and select random combinations to train the model and score. The number of search iterations is set based on time/-resources.
— **Guided Search Algorithms** : use methods such as gradient descent, Bayesian Optimization, or bandit-based approaches to conduct a guided search for the best hyperparameters.

2.3 Electrocardiogram Data

This section briefly describes the most used type of data set in this project. The most important type of heath data set used in this project is electrocardiogram signal which is represented in the form of time-series data sets.

Definition

The electrocardiogram (ECG) is an electrophysiological signal that contains a large amount of valuable information about the electrical activity of the heart. ECG waveforms are seen in clinical assessments of heartbeats and include P-waves, QRS complexes, and T-waves. The amplitudes and time intervals of ECG waveforms provide insight on heart rhythm abnormalities and heart diseases such as ischemia [5].

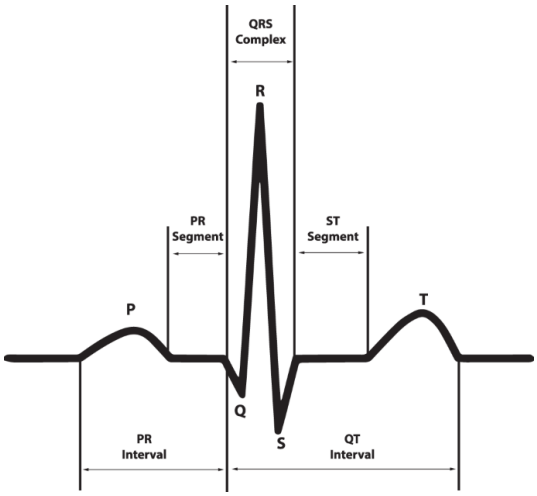

FIGURE 2.3 – Schematic representation of an ECG wave.

The figure 2.3 represents a very simplified version of an ECG signal with the most important segments labeled. The diagnosis of the signal relies on the morphology of the waves, as well as the duration of each peak and the segments that make it up. Therefore, detection of each section of the ECG signal is essential for health professionals in screening, diagnosis, and monitoring of several heart conditions.

Conclusion

Although the most important concepts are explained, there is still a lot to cover to fully understand even the most basic concepts. Fortunately, along each discussed fundamental concept, there is a reference to an external resource that dives much deeper into explaining them. In the next chapter, project requirements are discussed along with a brief description of the chosen system design.

Chapitre 3

Requirements and System Design

In this chapter a very brief discussion about the functional and non-functional requirements is provided. The system design is then presented.

3.1 Functional Requirements

The figure 3.1 describes the general functionalities of the provided solution. As depicted in this diagram, there are two main actors :

- **User** : is a consumer of the application.
- **Developer** : is a user that has some minimal required knowledge to adjust the ML pipeline.

And one secondary actor :

- **AI Agent** : is the environment that performs the processing. For example : Cloud Computing, Local Multi-GPU setup, etc.

This is the non-exhaustive list of the main functionalities provided by this solution :

- **Automatic Model Generation** : automatically choose a good DL model and data preprocessing steps for a new dataset at hand.
- **Add New Models** : the developer can easily add new DL models to the list of models to try by the ML pipeline.
- **Add New Data Preprocessing Pipeline** : the developer can easily add new data preprocessing pipelines to the list of data preprocessing pipelines to try by the ML pipeline.
- **Add New Data Transformer** : the developer can easily add new individual data transformers to the ML pipeline.
- **Profile Code Performance** : the developer can easily profile sections of the project's source code thanks to integrated profiling tools.

It should be noted that a data preprocessing pipeline is composed of a sequence of data transformers. As its name suggests, data transformers simply transform input data by performing operations on it and outputs it.

3.2 Non-functional requirements

This projects highly emphasises on various non-functional requirements, most importantly it takes software engineering principles and best practices extremely seriously. This is the non-exhaustive list of the main non-functional requirements :

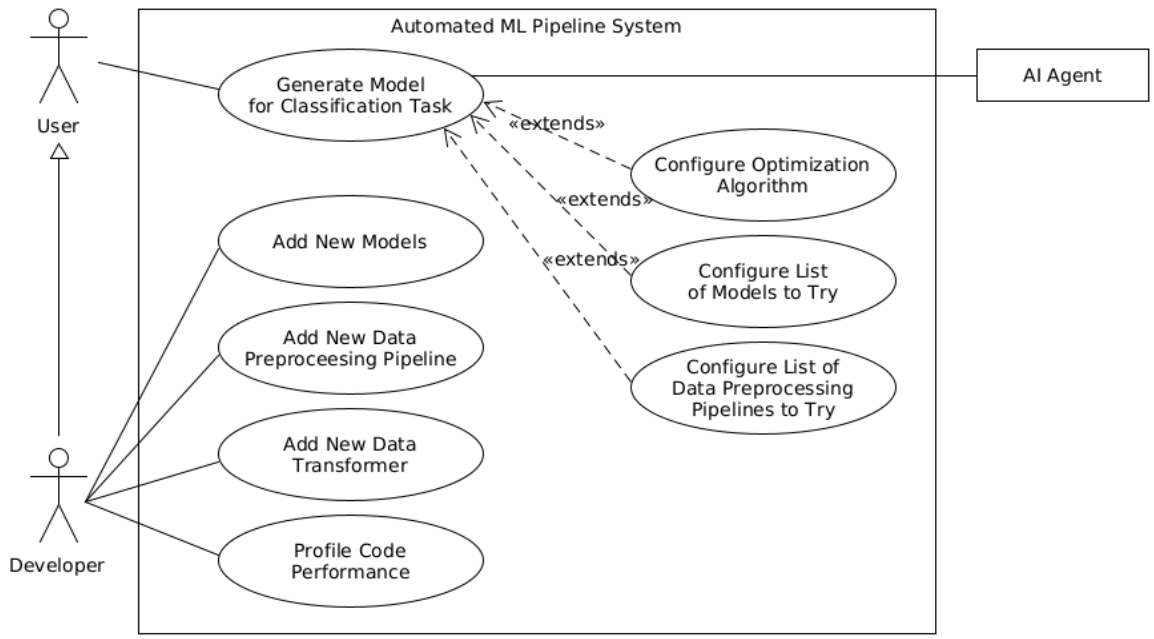

FIGURE 3.1 – Global Use Case Diagram.

— **Performance** : model generation should be performed within a fixed computational budget.
— **High Configurability** : the parameters are exposed to allow for custom configuration.
— **Extensibility** : adding new functionalities to the source code is straightforward and easy to do.
— **Modularity** : strong cohesion within modules and weak coupling between them.
— **Extensive Documentation** : each function and class should be documented following standard documentation format.

3.3 System Architecture

In this section the end-to-end machine learning pipeline architecture is described in detail. Following that, the chosen DL models are also briefly described.

3.3.1 Pipeline Architecture

The figure 3.2 describes the chosen ML pipeline architecture. Each step is explained **sequentially**.

— **Step 1** : Raw training and testing data is entered into the pipeline. The pipeline doesn't perform any data preparation, thus it is assumed that the input data is already in the right format.
— **Step 2** : A data preprocessing pipeline that consists of chained pre-defined data transformation techniques is selected, raw input data is transformed into processed data.
— **Step 3** : For each model in a predefined list of models to try, hyperparameter optimization is performed and the hyperparameters that lead to the highest score are saved. Although,

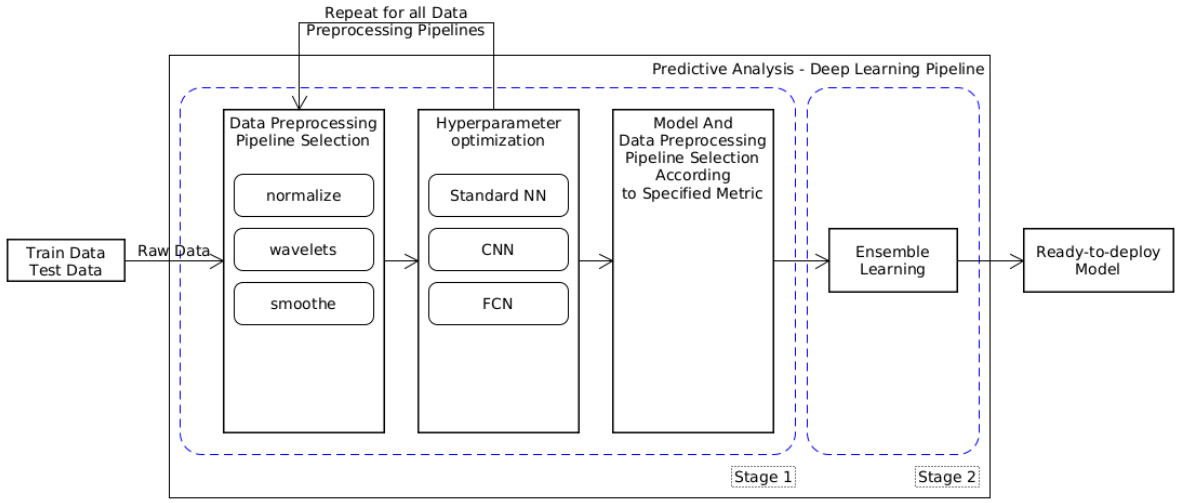

FIGURE 3.2 – Implemented ML Pipeline Architecture.

currently, the model only trained on a subset of the training data, it is saved in memory for potential future use.

— **Step 4** : Repeat **Step 2** to **Step 3** until all data preprocessing pipelines are tried. The user can provide optional arguments that change this behaviour, for instance, a maximum-execution timer can be set.

— **Step 5** : The tuned models and their data preprocessing pipelines are ranked in a **leaderboard**. The user has full access to this leaderboard.

— **Step 6** : If specified, ensemble learning could be used to obtain better performance than that obtained from any of the constituent learning algorithms alone.

— **Step 7** : The user has the option to either export the model that was already trained on a subset of the training dataset, or retrain the model on the full dataset then export it.

3.3.2 Models Architecture

This section describes the architecture of the chosen deep learning models. The term "Models Architecture" is a bit misleading since we are actually not providing concrete model architectures for the pipeline. Instead, many hyperparameters are included in the search space for HPO.

Standard Neural Networks

A generic standard neural networks model (NN) is implemented in the list of models the pipeline tries because it acts as a simple reference for other sophisticated models to compare with. In other terms, it helps answer the question of how much better a sophisticated model is compared to a simple neural networks model. The table 3.1 describes the architecture of this model with its search space.

Convolutional Neural Networks

Since convolutional neural networks (CNN) model proved to provide excellent performance for time series classification (TSC) problems, a generic CNN model architecture is added to the

Hyperparameters	Definition	Search Space
layer00 units	the number of neurons in the first hidden layer	{32, 64, ..., 480, 512}
layer01 units	the number of neurons in the second hidden layer	{16, 24, ..., 112, 128}
layer02 units	the number of neurons in the third hidden layer	{16, 24, ..., 112, 128}

TABLE 3.1 – Standard Neural Networks Hyperparameters Search Space.

list of models this pipeline tries. The table 3.2 describes the architecture of this model with its search space.

Hyperparameters	Definition	Search Space
Convolution1D_00 filters	number of output filters in the first convolution kernel	{32, 48, ..., 112, 128}
Convolution1D_00 size	size of the first one-dimentional convolution kernel	{3, 4, 5}
MaxPool1D_00 size	size of the first one-dimentional max-pooling kernel	{3, 4, 5}
Convolution1D_01 filters	number of output filters in the second convolution kernel	{32, 48, ..., 112, 128}
Convolution1D_01 size	size of the second one-dimentional convolution kernel	{3, 4, 5}
MaxPool1D_01 size	size of the second one-dimentional max-pooling kernel	{3, 4, 5}
Convolution1D_02 filters	number of output filters in the third convolution kernel	{16, 20, ..., 28, 32}
Convolution1D_02 size	size of the third one-dimentional convolution kernel	{2, 3}
layer00 units	the number of neurons in the first hidden layer	{32, 64, ..., 480, 512}
layer01 units	the number of neurons in the second hidden layer	{16, 32, ..., 224, 256}

TABLE 3.2 – Convolutional Neural Networks Hyperparameters Search Space.

Conclusion

This chapter only focused on the most important functional and non-functional requirements. It also presented a non-detailed version of the chosen system design. The next chapter will dive a bit into how the previously discussed solution is implemented.

Chapitre 4

Implementation

This chapter starts off with a very brief description of the hardware environment and software development environment(s), then the most important technology tools that are used in this project are presented. The next section provides a thorough description about architecture of the implemented, end-to-end machine learning pipeline. After that, a great emphasis is put on the source code structure by showcasing the functions of each Python module and the relations between them. Following that, performed tests and experiments on the implemented pipeline and their results are described. Finally, this chapter is closed by listing the most important APIs provided for the user(s).

4.1 Work Environment

4.1.1 Hardware Environment

Project development was mostly done on personal laptop with these specifications :

Almost all testing and experiments were executed on the convenient, free-to-use and interactive environment Google Colab that runs on the cloud. GPU runtime type was the hardware environement on which all tests are conducted.

Google Colab

- CPU : Intel(R) Xeon(R) CPU @ 2.30GHz
- GPU : Tesla K80, 12GB VRAM
- RAM : 12GB
- OS : Linux

Dell Inspiron Gaming 7577

- CPU : Intel(R) Core i7-7700hq CPU @ 2.80GHz
- GPU : Geforce GTX 1050 Ti, 4GB VRAM
- RAM : 16GB
- OS : Ubuntu 21.10

4.1.2 Software Environment

- **Git** : It is a free and open source distributed version control system (VCS) designed to handle everything from small to very large projects with speed and efficiency. Git is the go-to VCS in this project for coordinating work among us, developers, collaboratively developing the source code for this project.

— **GitHub** : It is an Internet hosting service for software development and version control using Git. GitHub makes the process of developing this project much easier by providing features such as : access control, bug tracking, software feature requests, task management, continuous integration, etc.

— **Visual Studio Code** : It is a code editor with support for numerous development operations like debugging, task running, intelligent code completion, snippets, code refactoring, etc. Visual Studio Code's embedded Git proved to be extremely helpful in the development process of this project.

— **Google Colab** : It is an easy-to-configure, convenient and interactive environment to run Jupyter notebooks on the cloud. Google Colab is used in this project for two main reasons, the first is to access high-performance hardware such as the GPU. The second is rapidity of accessing large databases due to extremely good internet speeds.

4.1.3 Technology Tools

4.1.3.1 Programming Languages

— **Python** : It is an interpreted, object-oriented and general purpose programming language. The number of mature, easy-to-use and very-well written packages for Python is what makes it the best programming language for this project.

4.1.3.2 Libraries and Frameworks

— **Keras** : It is an open-source software library that provides a Python API for artificial neural networks. Keras acts as an interface for the TensorFlow library. We chose Keras as a deep learning library not only because it is flexible and powerful but also because it has a well-designe library for tuning deep learning models implemented using Keras.

— **KerasTuner** : It is a hyperparameter optimization framework that allows for automatic hyperparameter search. KerasTuner can be used to optimize any function by just defining the search space and the evaluation metric. This one of the main reasons why KerasTuner is so suitable for this projet.

— **CookieCutter** : It is an open-source tool to creates projects from project templates. In this project, CookieCutter is used to initially structure the project according to a generic template useful for Python data science packages.

— **Sphinx** : It is a documentation generator written and used by the Python community. Sphinx is used in this project to generate beautiful looking documentation from Python code source files.

4.2 Source Code Structure

not DONE yet

4.3 User Manual

4.3.1 Automated Classifier API

The user can generate a classifier model on a newly provided dataset in a few lines of code. The code 4.1 describes a minimal classification use-case example where the user doesn't provide any custom configuration.

```
from predictive_analysis.core.core import AutoClassifier
from predictive_analysis.data.prepare_data import prepare_classification_data

# loads mitbih data from data/raw folder
train_data = pd.read_csv('/content/datasets/mitbih_train.csv', header=None)
test_data = pd.read_csv('/content/datasets/mitbih_test.csv', header=None)

# prepares the previously loaded data for classification
X_train, X_test, y_train, y_test, nbr_outputs = prepare_classification_data(
train_data, test_data)

# AutoClassifer instance to automatically determine the best pipeline to use on
data
classifier = AutoClassifier()

# run fit method to start the search process
classifier.fit(X_train, y_train, X_test, y_test, nbr_outputs)

# re-train the best found model on the full dataset and export it to the
filesystem
classifier.export_model(X_train, y_train, retrain_model=True, filename='
mitbih_test')
```

FIGURE 4.1 – Automated Classifier API.

4.3.2 Profiler API

This project comes with integrated profiling tools for developers to asses the performance of the source code. The profiler could be easily called with a few lines of code as depicted by the figure 4.2. The profiling data is then saved into a predefined directory.

```
ProfilerWrapper.enable()
## CODE SECTION HERE ##
ProfilerWrapper.disable()
```

FIGURE 4.2 – Integrated Profiler API.

4.3.3 Leaderboard

After finishing execution, the ML pipeline outputs a leaderboard : a ranked list of best tuned models with their corresponding data preprocessing pipelines. The figure 4.3 showcases an example of this leaderboard. The user has full access to it, thus they can, among other things, chose whichever model to export or to re-train. The column name represents the name of the data preprocessing pipeline and the name of the model. The score is the chosen based on which models are ranked. The mse column is the mean squared error of the model on the test dataset.

4.4 Tests and Results

In this section, a thorough description of how tests and experiments are done is presented. First of all, the contents and the format of the used dataset are briefly described. Following that, the configuration of the ML pipeline is provided. Evaluation metrics describes the used metrics to evaluate the performance of the ML pipeline. This section is then closed with a brief analysis of the obtained results.

name	score	mse
W3_4_N with cnn	0.948383	0.163894
W4_5_N with cnn	0.941394	0.174397
W3_5_N_S with cnn	0.9297	0.196009
W3_5_N_S with simple_nn	0.917276	0.258276
W3_4_N with simple_nn	0.877261	0.341163
W4_5_N with simple_nn	0.854285	0.417871

FIGURE 4.3 – An example of leaderboard output.

4.4.1 Used Dataset

The input data used for the implemented ML pipeline is the **ECG Heartbeat Categorization Dataset**. This dataset is composed of signals in the form of time-series data which correspond to electrocardiogram (ECG) shapes of heartbeats for the normal case and the cases affected by different arrhythmias and myocardial infarction. These signals are preprocessed and segmented, with each segment corresponding to a heartbeat. The figure 4.4 showcases one sample of the dataset for each class.

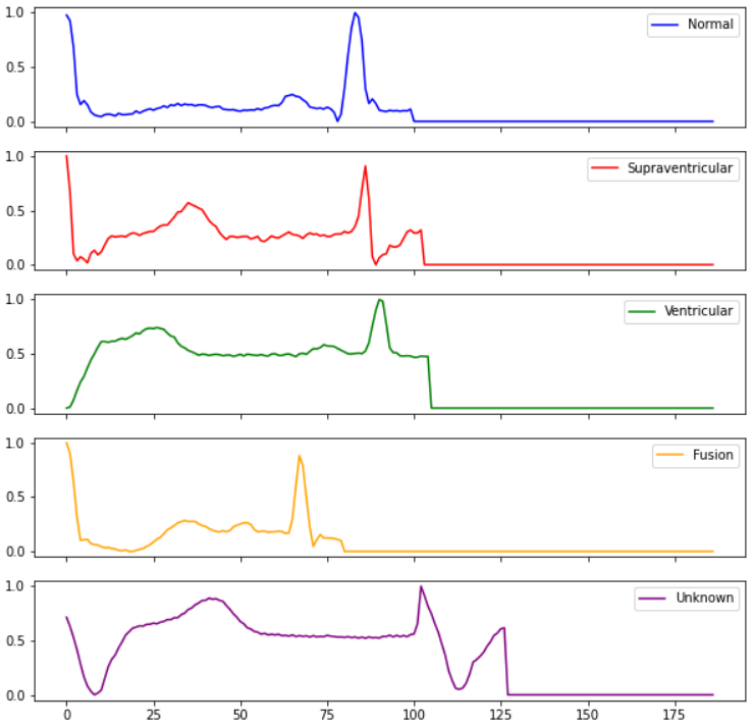

FIGURE 4.4 – ECG Signal of Each Class Label.

4.4.2 Results and Analysis

After feeding the ECG heartbeat dataset to the automated ML pipeline, the best data preprocessing pipeline and model combination, after being re-trained on the full dataset, resulted in an accuracy of about **0.9434**. The execution time of the whole automated classifier took around **42.352 minutes**. This required execution time should be kept at a minimum since a lot of factors may heavily influences it, e.g. input data size, hardware work environment, etc.

Conclusion

Although this chapter described the main implementation aspects of this project, there are many other implementation details that were not discussed here. This project implements many other helpful tools that makes it, to some degree, a complete package for data scientists to use and experiment with. Nonetheless, there is a large room for improvements, for instance execution times could be greatly reduced by using grid search for searching best combinations of data preprocessing pipelines and models.

Acknowledgement

I cannot express enough thanks to my supervisors Dr. Mariem ABID and Dr. Amal KHABOU for their continued support and encouragement. I would also like to thank Mr. Med Aziz Driss with whom I worked on this project, I genuinely couldn't have asked for a better coworker. I offer my sincere appreciation for the learning opportunities provided by all of them.

Bibliographie

[1] Jason Brownlee. Hyperparameter Optimization With Random Search and Grid Search. https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/, 2022. [Online ; accessed 31-August-2022].

[2] IBM Cloud Education. What is deep learning ? https://www.ibm.com/cloud/learn/deep-learning, 2022. [Online ; accessed 31-August-2022].

[3] IBM. Hyperparameter tuning. https://www.ibm.com/docs/en/wmla/1.2.3?topic=features-hyperparameter-tuning, 2022. [Online ; accessed 31-August-2022].

[4] Will Koehrsen. Intro to Model Tuning : Grid and Random Search. https://www.kaggle.com/code/willkoehrsen/intro-to-model-tuning-grid-and-random-search/notebook, 2022. [Online ; accessed 02-September-2022].

[5] Siti Nurmaini, Alexander Edo Tondas, Annisa Darmawahyuni, Muhammad Naufal Rachmatullah, Jannes Effendi, Firdaus Firdaus, and Bambang Tutuko. Electrocardiogram signal classification for automated delineation using bidirectional long short-term memory. *Informatics in Medicine Unlocked*, 22 :100507, 2021.

[6] Sumit Saha. A Comprehensive Guide to Convolutional Neural Networks. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, 2022. [Online ; accessed 01-September-2022].