Master's thesis: NLU for integral management of watch functionalities

Master's thesis

María de las Nieves Úbeda Castañeda

Research thesis for

Máster en Lenguajes y Sistemas Informáticos

Universidad Nacional de Educación a Distancia

Supervised by

Professor Anselmo Peñas

February 2023

Acknowledgement

I am very grateful to my supervisor, Anselmo Peña, for his guidance and suggestions, and I value the experience and knowledge I have acquired when working on this thesis. I thank Paulo Villegas and Nerea Suárez for the co-supervision in terms of Telefónica's supervisors for their assistance on the thesis. I am grateful to my colleagues, Hugo Fernández, José A. Carballo, Carlos Rodríguez, Daniel M. García Ocaña and Óscar Miranda for their valuable remarks and feedback regarding my code. I want to express my gratitude to Ester Vaquero for all the support and help concerning the linguistic part. My biggest thanks belong to my family for all the encouragement that they have given me throughout the years. I am particularly grateful to Javier Ortiz for his love, patience, valuable feedback, help and support.

Abstract

Conversational assistants and chatbots have become popular tools that can quickly answer users' requests to help them solve simple tasks. One of the most important goals of a conversational assistant is to understand a user's intent, that is, NLU. To carry out this aim, intent classification is one of the most important tasks. Thus, this thesis aims to design and implement a benchmark to test different models that can be included for a intent recognition system in order to improve a stage that already exists in Aura (Telefónica's virtual assistant). The resulting module can classify user utterances regarding watch actions. Recent advances in natural language by using labels from a predefined set of intents. The best model achieves an improvement of accuracy concerning the solution that is currently used by the company.

Keywords

Intent classification, Machine Learning, Natural Language Processing, transformers, classes.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, people have undergone a new way of life. Virtual assistants have appeared in our daily life. The idea of being surrounded by robots that can chat with humans and ease human tasks is a reality. Definitely, this is a new age, the period of virtual assistant technology.

Some years ago, the fact of coexisting with virtual assistants such as Google Home, Amazon's Alexa or Siri developed by Apple was unbelievable. Devices, generally, were conceived as tools to carry out some tasks, for instance, making phone calls, sending messages or emails, searching interesting things on the Internet, etc., although they were not customised or easy-to-use. Even though this could appear incredible, 12 years ago, bots did not even exist. That is why, all technological developments as well as algorithms are quite recent.

However, technology has evolved considerably in the last few years thanks to research, development and innovation efforts. Furthermore, some of the most astonishing recent functionalities are, for example, telling stories, answering questions, singing songs, telling jokes, setting up reminders, playing a content on the TV, showing routes, or even doing the shopping. Most homes, at least, count with one of these famous devices. To cover all these tasks, intent classification is the basis. Firstly, the user says or writes a sentence that is interpreted by an algorithm, it can be an ASR[1] system if it is voice-based, or text processing algorithm if it is text-based. Then, once this is processed and cleaned, it goes through a classification model module, ge-

[1] Automatic Speech Recognition: also known as Speech to Text (STT), is the task of transcribing a given audio to text. It has many applications, such as voice user interfaces (Jurafsky & Martin, 2022).

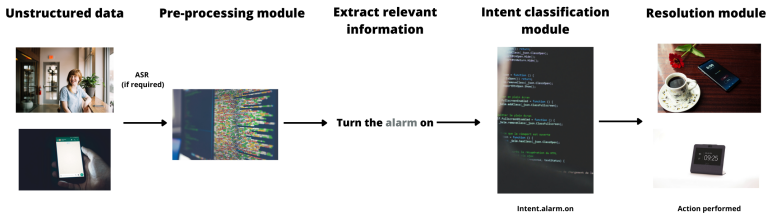

Figure 1.1: *Steps in the unstructured data processing.*

nerally an intent classification module where it carries out concrete actions. Once the utterance is classified, it is mapped to an action that the device used performs. Thus, if the user says "Please, set an alarm at 10 a.m.", it would be recognised as "alarm.on" and the action that the user will see would be ringing an alarm at 10 a.m.

In this context, there is a vast amount of information saved in several data bases. In fact, this diversity entails a variety of formats that will imply "cleaning" data and interpreting it correctly. Thus, Natural Language Processing's main aim is making machines understand unstructured texts and extracting the most relevant information from these texts. To carry out this, the pre-processing phase is somehow as important as building a great intent-classification module. Indeed, with quality data, many algorithms and technologies can be applied. Nevertheless, one may have the best classification algorithm, but without quality data, the recognition and classification would be unfeasible.

Finally, several challenges are raised: converting speech into text, understanding unstructured data, converting this unstructured data into structured data, extracting the most relevant information, classifying this information correctly and mapping this classification to an action to perform. In this thesis, the focus is the intent-classification module. Notwithstanding, the pre-processing data phase and the multiple evaluation of different algorithms will be also significant parts in this project.

1.1. Motivation

Grounds for this thesis lie in the attempt to continue improving the classification model problem. That is, most of the existing virtual assistants as well as artificial intelligences try to help users to solve domestic-related requests such as the management of the watch. Concretely, in this thesis, the Use Cases that are going to be analysed are the concept of turning on and off the alarm, e.g. “Activa la alarma” and “Quita la alarma” respectively, the timer, for example, “Pon un temporizador” and “Elimina el temporizador” respectively, and the reminder, e.g. “Pon un recordatorio” and “Quítame el recordatorio” respectively. The key point consists in evaluating different types of technologies such as algorithms based on Long Short-Term Memory Model[2], Support-Vector Machines Model[3], Automated Machine Learning Models, and Transformers Models to see which algorithm retrieves the best results taking into account the same data set.

Given that computer science is a large field of study, this thesis is based on NLP, which is a subfield of it aiming at understanding human language, or, at least, to understanding text and spoken words in much the same way as human beings do. Therefore, this subfield combines computing and language, for example, in the use of linguistic-rule-based models of human language with statistical, Machine Learning or Deep Learning models.

It is important to highlight that in this work, Spanish language is the basis to choose a concrete model to classify users' utterances due to the fact that the vast majority of models that are already developed were thought for English language utterances. Thus, this work provides a new approach and/or advance regarding those classification models.

Finally, the evaluation of these models means that the main goal of this thesis is not just using several algorithms given a concrete data set concerning the Watch Management domain, but identifying which model improves the results to see if it could be included in a real virtual assistant such as Aura[4].

[2]Long-Short-Term-Memory, also known as LSTM. It is an artificial neural network used in the fields of artificial intelligence and deep learning.
[3]Support-Vector Machines Model, also known as SVM.
[4]Aura is the Artificial Intelligence of Telefónica, a real virtual assistant.

1.2. Proposal and Objectives

The main goal of this project consists in developing different classification models to see which behaves best taking into account the data set that is based on multiple use cases under the Watch Management domain in Spanish. The final aim is selecting the algorithm that recognises most classes correctly to understand why a technology is more useful to solve this problem than the others.

To carry it out, these specific objectives have been proposed:

- Define a specific domain that is going to be used as the basis of the data, taking into account the data should be anonymised and real.
- Define the number of use cases that comprises the data set (training set and test set) required to develop this project.
- Analyse the present state-of-the-art of the classification model problem and transformer-based embedding algorithms. An essential part of the intent detection module of a virtual assistant such as Aura is an algorithm that transforms sentences into vectors. Thus, similar sentences have similar transformations. Therefore, an important part of this project is dedicated to semantic textual similarity and the evaluation of different encoding algorithms.
- Search on Aura's logs to discover real users' requests that will compose the complete corpus. One of the premises of this project is that data must not be synthetic but real.
- Define which types of technologies and algorithms will be used to carry out this work by taking into account which are the most recent technologies and the conventional ones.
- Create a pre-processing module to structure data extracted from Aura's logs and cleaning it to extract and detect the most relevant information.
- Build an architecture that allows changing the training, the testing and the algorithm parameters, so that it is possible to do tests with different configurations in order to obtain comparable measures when extracting the results. Moreover, getting both low latency and memory

efficiency will be taken into account due to the fact that these data come from an existing virtual assistant, Aura.

- Identify useful libraries that will allow developing these classification models.
- Iterate algorithms by using all the data from the training set and the test set or just some parts of it. This would measure how the algorithm behaves with a greater or smaller numbers of classes.
- Evaluate each selected algorithm by using the same data set to obtain final conclusions about its behaviour in comparison with the rest of technologies used, i.e., classify input text with reasonable accuracy. In fact, the classification model problem deals with mapping a sentence's numerical representation to an intent or class. Several approaches have been developed during this work such as the use of Long Short-Term Memory model, Support Vector Machines Model, Auto Machine Learning models, and transformers models.

The present NLP research is a collaborative work with the company Telefónica, who provides the data (using Aura's logs) and supervises the approach of this project. This work aims at providing an improvement of the intent classification and recognition module by incorporating state-of-the-art encoding models.

1.3. Structure of the thesis

This work is divided into six chapters and will describe the research done in the field of intent classification problem regarding a specific data set. Each chapter provides a general background, methods, evaluation, and results section.

Chapter 1. Introduction. This chapter introduces the underlying motivation of this thesis, the research questions to solve the problem herewith presented as well as the proposal and objectives of this thesis.

Chapter 2. State-of-the-art. This chapter focuses on the techniques used to solve this classification model problem by explaining all the

possible possible types of algorithms that have been used in the field so far.

Chapter 3. Proposed case study. In this chapter, the data used in this thesis and a brief approach of the project are presented. Furthermore, some issues found while carrying out this work related to the data handled will be also described.

Chapter 4. Development. This chapter focuses on the development of the project. Firstly, the pre-processing module and the use of the whole data set or partial data set would be indicated in each experiment to see how algorithms behave in these particular tests or challenges. Then, the algorithms used for this project, their configuration and parameters will be also explained. Finally, in the evaluation section, each test will be explained as well as the challenge that pretends to be addressed, the evaluation that has been carried out in this project and the metrics used.

Chapter 5. Evaluation and Results. In this chapter, the results of the previous chapter are analysed and compared.

Chapter 6. Conclusions and future work. This chapter collects several conclusions extracted from the proposed problem. In addition, new lines of work are set out for future course of action.

Chapter 7. Bibliography. This chapter comprises all the bibliography used and referenced in this thesis.

Chapter 2

Preliminaries

This chapter summarizes the theoretical background and history that makes automatic intent classification possible. The overall work connects the multiple disciplines of Linguistics (human language, basically the input of the project), Computer Science (Artificial Intelligence), and Data Science (Machine Learning).

2.1. The Science of Linguistics

Linguistics is the study of the origin, evolution and structure of the language. Thus, this science studies fundamental structures of human language, its variations throughout all the language families and its conditions that make possible the understanding and communication by using a natural language.

There are different approaches to study Linguistics. Some of them are theoretical linguistics, applied linguistics, diachronic and synchronic linguistics, comparative linguistics and computational linguistics. This last concept is the most relevant one for this project.

Computational Linguistics is an interdisciplinary field that is based on computing, that is, computational modelling and linguistics, e.g. natural language. In this way, Computational Linguistics can be defined as the study of computational approaches to linguistic questions.

2.2. Artificial Intelligence

Computer Science is the study of formal science that comprises Information theory and computing just as their application in computational systems.

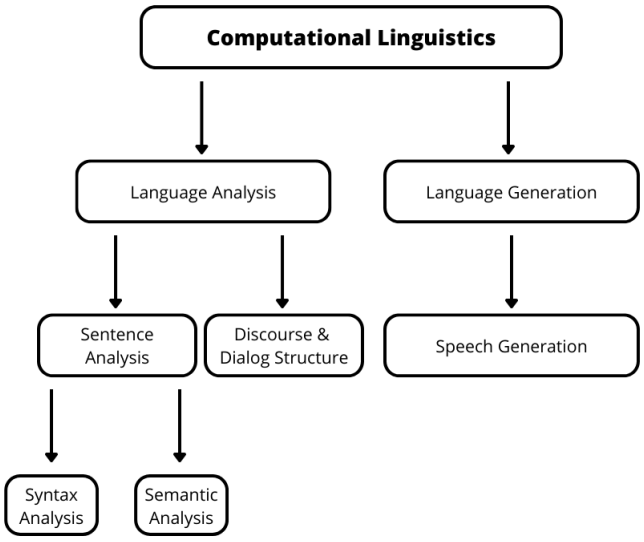

Figure 2.1: *Computational Linguistics tasks.*

This science can be described as the systematic study of algorithmic processes that transform information: their theory, analysis, design, efficiency, implementation and application.

One of the main subfields of Computer Science is Artificial Intelligence[5]. In 1950, Turing presented a paper called "Computing Machinery and Intelligence" in which the Turing Test[6] was established. This fact could be considered as the goal and vision of Artificial Intelligence (Jurafsky & Martin, 2022). That is the reason why Turing could be considered as the father of Artificial Intelligence.

To define this concept correctly, it is important to understand that it is a branch of Computer Science with the aim of creating machines that are able to perform tasks that typically require humans. Some applications of this field can be Natural Language Processing tasks, Speech recognition and Machine Vision. Generally, an Artificial Intelligence program focuses on the cognitive skills of learning, reasoning and self-correction.

[5] also known as AI.

[6] It tested the machine's ability to show intelligent behaviour equivalent to, or indistinguishable from a human.

- **Learning process:** This part is based on collecting data and creating rules to convert the data into relevant information. Rules, on the other hand, normally known as algorithms (Heeyoung Lee et al., 2017), provide computing devices with instructions to finish the task.
- **Reasoning process:** This part is based on choosing the right algorithm to get the desired output.
- **Self-correction process:** This part is based on fine-tuned algorithms to ensure that they retrieve the most accurate result.

2.2.1. Machine Learning in NLP

Machine Learning is a subfield of the Artificial Intelligence and Computer Science which is focused on the use of data and algorithms to imitate the way that humans learn by improving their performance. It is also a relevant component in the field of Data Science.

When using statistical methods, algorithms are trained to classify or predict so that they can discover key information inside data mining projects.This useful information eases the decision-making process inside applications directly affecting the growth of companies which also affects directly the growth.

Linked to this field, Data Science is a field that belongs to Computer Science but also includes scientific methods, processes and systems to extract knowledge and information to improve the understanding of data in different ways. In fact, it is relevant when carrying out Machine Learning tasks.

Thus, Data science developers apply Machine Learning algorithms to numbers, text, images, video, audio, and more to produce Artificial Intelligence systems to perform tasks that typically required humans.

Machine Learning has many applications such as search engines, medical diagnoses, fraud detection, market analysis, classification of DNA sequences, speech recognition, text recognition and robotics.

The most relevant Machine Learning techniques are those shown below:

- **Decision tree learning:** It is a decision support tool that uses a tree-like model of decisions.
- **Association rule learning:** It is a rule-based Machine Learning method used to discover interesting relations between variables in large

databases.

- **Genetic algorithm:** It is a meta-heuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms.
- **Artificial neural networks:** They are computing systems inspired by the biological neural networks that constitute human brains.
- **Support Vector Machines:** They are supervised learning models with associated learning algorithms that analyse data for classification and regression analysis.
- **Clustering:** It is the classification in different groups or clusters of data inputs that are similar among them according to a specific criterion or criteria.
- **Bayesian Networks:** It is a probabilistic graphical model that represents a set of variables and their conditional dependencies via a directed acyclic graph.

2.2.2. Deep Learning

Deep learning can be considered as a type of Machine Learning that trains a machine to carry out tasks as humans do such as Speech to Text transformation, identification of images, or making predictions. Deep Learning uses the parameter configuration regarding data and trains the machine to learn on its own with the use of patterns and several layers of processing. Its methods are based on artificial neural network with representation learning. Thus, learning can be supervised, semi-supervised or even unsupervised.

The most relevant deep learning models are those shown below:

- **Convolutional Neural Networks:** It is a type of artificial neural network in which artificial neurons corresponds to responsive fields, similar to neurons inside the biological brain.
- **Recurrent Neural Networks[7]:** It is a type of artificial neural network in which connections between nodes can create a cycle, allowing outputs from some nodes affect subsequent inputs of the same nodes. Some popular RNN types are those below:

[7] Recurrent Neural Networks, also known as RNN.

    - **Long Short-Term Memory Networks:** It is an artificial neural network used in the fields of Artificial Intelligence and Deep Learning.
    - **Gated Recurrent Units**[8]**:** They are a variation on LSTMs. They have one less gate and are wired slightly differently: instead of an input, output and a forget gate, they have an update gate (Chung et al., 2014)

- **Generative Adversarial Networks:** It is a class of artificial neural network which uses unsupervised learning. They are implemented by a system of two neural network that compete mutually.

- **Radial Basis Function Networks:** It is a type of artificial neural network that calculates the output of the function by calculating the distance to the center node.

- **Feed Forward Neural Networks**[9] **and Perceptrons**[10]**:** It is a type of artificial neural network that feeds information from the front to the back.

- **Attention Networks**[11]**:** It is a type of artificial neural network that includes Transformer architecture. In addition, as this type of network counts with enough hidden neurons, theoretically, it can model the relationship between the input and the output.

2.2.3. Transformers

In 2017, in a paper called *Attention is All You Need* (Vaswani et al., 2017) a new architecture was presented. This consisted in a Transformer, a new deep learning model. This model has the main and new characteristic of replacing LSTMs with attention layers.

These layers code each word in a sentence taking into account the rest of the sequence. Thus, it allows introducing the context in the mathematical representation of the text. That is the reason why Transformers are also known as Contextual Embeddings.

[8] Gated Recurrent Units, known as GRU.
[9] Feed Forward Neural Networks, known as FF or FFNN.
[10] Perceptrons, also known as P.
[11] Attention Networks, also known as AN.

Some of the most relevant transformer models are going to be explained in this section.

- **BERT:** It is a model proposed in *Open Sourcing BERT: Preliminaries Pre-training for Natural Language Processing* (Devlin et al., 2018). It is a transformer-based model for Natural Language Processing tasks. In fact, the original BERT has two different models: BERTBASE (composed of 12 encoders with 12 bidirectional self-attention heads) and BERTLARGE (24 encoders with 16 bidirectional self-attention heads). One of the most relevant characteristics of this model is the large amounts of training data used, that is, a data set of 3.3 billion words. The fact of training this huge data set was possible due to the novel *Transformer architecture and sped up by using TPUs*. BERT is a MLM[12]. This model type enables bidirectional learning from text by hiding a word in a sentence and forcing BERT's to bidirectionally use the words on either side of the covered word to predict the masked word. As this model type was not developed previously, this was a milestone in modeling.

- **RoBERTa:** It is a model proposed in *RoBERTa: A Robustly Optimised BERT Pretraining Approach* (Liu et al., 2019). It consists in a new training model that improves on BERT. It is built upon BERT's algorithm but the key hyperparameters are modified, that is, the next-sentence pre-training objective is removed and its training contains much larger mini-batches and learning rates. Furthermore, RoBERTa has a similar architecture to BERT's, although it uses a byte-level BPE as a tokeniser as GPT-2[13].

- **XLNet:** It is a model proposed in *XLNet: Generalised Autoregressive Pretraining for Language Understanding* (Yang et al., 2019). It is an extension of the Transformer-XL model pre-trained using an autoregressive method to learn bidirectional contexts, by maximizing the expected likelihood over all permutations of the input sequence factorisation order. As well as other transformers models, XLNet achieves

[12] Masked Language Model.

[13] Generative Pre-trained Transformer 2. It is an open-source artificial intelligence created by OpenAI . It is in charge of translating text, answering questions, summarizing passages, and generating text output.

better performance than pre-training approaches based on autoregressive language modeling. When using this model, the most relevant advantages are the ability of learning bidirectional contexts by maximizing the expected likelihood over all permutations of the factorisation order and avoiding BERT's limitations due to its autoregressive formulation.

2.3. Natural Language Processing

Natural Language Processing is a subfield of Artificial Intelligence and Linguistics Science. The main goal is to make machines and computers understand natural language. It uses several techniques to understand human language, for instance, statistical methods, Machine Learning, learning based on rules and algorithms. There are different methods because human language is very extensive and different. Therefore, speech and text are processed in a different way, and probably, with different processing methods. In fact, they differ not only in the format but also in lexicon, for instance, in the usage of spoken expressions, abbreviations, slang, argot and so on, in grammar and syntax, as the spoken and written language do not use the same grammatical and syntactic constructions.

Although there are many NLP tasks, the basic and most relevant ones are going to be explained below:

- **Tokenisation:** It consists in segmentation of a text in constituent parts that can be, for instance, words. This kind of tokenisation is carried out by using separators such as punctuation signs, spaces and so on. In addition, regarding this task, segmentation is very similar to tokenisation because it consists of tokenising texts into phrases also by using punctuation signs.

- **Lemmatisation:** It consists in obtaining the basic form of a word, the "lemma". For example, for the word "casarnos", the lemma is "casar". Consequently, in some languages such as Spanish that have several verb tenses, this tool is quite useful to simplify texts for understanding purposes.

- **NER:** Also known as Named Entity Recognition. It is primarily used to extract relevant information and to classify them in different catego-

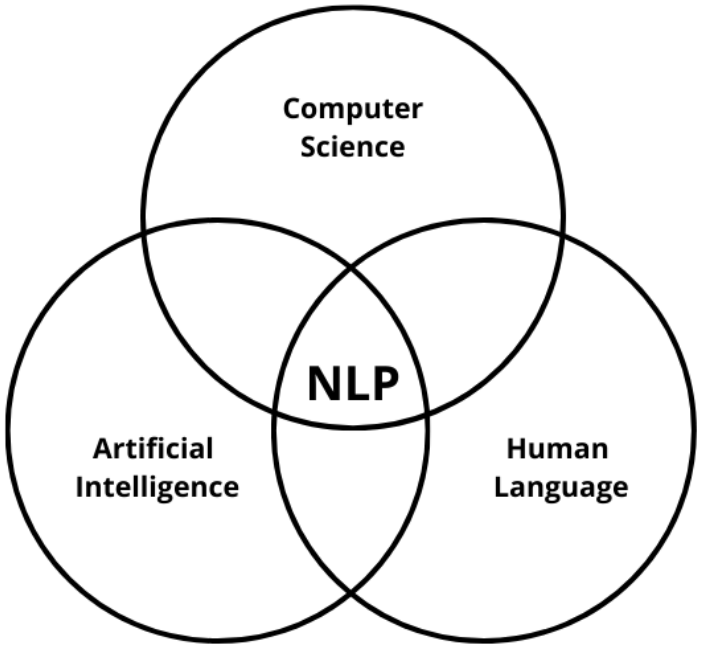

Figure 2.2: *Relation between NLP, Computer Science, Artificial Intelligence and Human Language (adapted from What is Natural Language Processing (NLP)? (Gopalan, 2021)).*

ries such as person names, locations, time expressions, organisations, etc.

- **Labeling:** It comprises the task of assigning labels to documents or some parts of it. It can be also known as text classification. Linked to this task, when creating a model, labeling is used before training, which allows an update and improvement of predictions for statistic models.

- **WSD**[16]**:** It is a process of identifying the sense of a word or sentence

[16]Word Sense Disambiguation.

given its context and it is particularly helpful in cases of polysemy or homophony.

- **Sentiment Analysis:** It is a task that identifies the emotional tone behind a text o part of the text.

- **Summarisation:** It deals with reducing the number of sentences or words of a text without changing its meaning.

- **Machine translation:** It is the automatic process of translating a text into a target language.

There are several factors in NLP making it a most notably complex process (Liu et al., 2021). For instance, there are hundreds of languages, somehow different or totally different syntactic and grammatical rules as well as ambiguity of words can make these tasks so complicated. In these section, these NLP challenges as well as others will be laid out.

- **Several languages:** There are about 7.000 languages in the world, although just 28 are the most spoken ones. However, even if 28 is a relative low number, it is still not manageable number to carry out tasks based on NLP. That is the reason why only NLP tasks are carried out in the most useful and talked languages, that is, each language is different (not only in semantics, but also in syntax and other components), then, NLP tasks will differ too.

- **Ambiguity:** This is one of the hardest parts of NLP. In many languages, there are words that can be ambiguous at many levels.

  - **Polysemy:** It occurs when a word has several meanings and the way to select the correct meaning is done by using the context of the sentence which it is used in. For example, “Es una persona noble”. It can be *noble* as noble from nobility, or it can refer to “noble” as a synonym of kind, kindness.
  - **Homonymy:** It occurs when a word sounds or is written exactly the same as another one but the origin and etymology is completely different. For example, “banco” (bank) and “banco” (bench).

  - **Homophones:** It occurs when a word sounds exactly the same as another one, such as "horca/orca". That is, "horca" refers to gallows while "orca" is the animal orca, the killer whale.
  - **Homographs:** It occurs when a word is written exactly the same as other, for instance, "llama/llama". One of the words "llama" refers to a verb (call), whilst the other "llama" can refer to the animal (llama) or a noun related with fire (flame).
  - **Amphibology:** Also known as syntactic ambiguity. It occurs when it is not clear which element is being referring to inside a sentence. For example, "Cuando dejamos el cristal en la mesa, se rompió". The action of breaking could refer to the table or to the crystal, because the reference is not clear at all.

- **Tokenisation:** In spoken language, it is hard to detect the moment in which a word starts or ends. Therefore, this problem is generally solved by using a good ASR that by probability detects several words and put them in a context with sense. However, there are languages such as Mandarin Chinese, in which this task is not only difficult in spoken language but in written language too. This happens due to the fact that there is no distance between a word and the following and previous one. Therefore, tokenisation in Mandarin Chinese is a great challenge.

- **Input data errors:** When processing natural language, some problems regarding input data collection are faced. For example, there are some accents, dialects, regionalisms, misspellings or argot that make this task more difficult. Furthermore, the wrong use of grammar as well as the use of abbreviations or slang create difficulties in NLP tasks.

- **Irony and sarcasm:** These natural aspects of the language can cause problems for Machine Learning models because it is difficult for them to detect if something is positive or negative, or is the human wants to say strictly what he has said or even the contrary.

- **Context:** The use of context is the basis of Natural Language Processing tasks. Not only for some difficulties previously seen such as ambiguity, but others such as identifying what the speaker really wants

to express. Therefore, the use of context is essential for NLP, not just rules and regexes[17] can be used to solve NLP tasks.

- **Tropes and literary figures:** The use of literary figures such as tropes, metaphors, metonymies or allegories raise a significant difficulty for NLP due to the need of the context that it requires to interpret a concrete part of a text.

2.4. Platforms for Dialogue Management

Other useful resources when developing Intent Classification modules are platforms, the use of third-party platforms[19].

By using third-party platforms, users solve the problem of creating an architecture from scratch, because it can be integrated in an in-house development. However, the price of the use as well as the potential latency when sending requests out of a local environment are the two main negative aspects of them. Most significant IA platforms can be shown below.

2.4.1. Dialogflow

Dialogflow is a Google's Natural Language Understanding platform that makes designing and integrating a conversational user interface into a mobile app, web application, bots, and so on easy. Thus, there are two versions, Dialogflow CX (the advanced version) and Dialogflow ES (the standard one).

Most of the advantages of using Dialogflow are that is quite user-friendly and very easy to learn how to train intents and entities. On the one hand, there is some basis already developed that can be used when introducing your own data. On the other hand, it is easy to deploy as well as easily scalable by adding 20 independent flows and 40.000 intents in each agent. Nonetheless, it is a black box which means that any user is completely blind when training because its code is not accessible to be changed or modified.

[17] Also known as Regular Expression. It is a sequence of characters that specifies a search pattern in text.

[19] A platform is an application or website that serves as a base from which a service is provided (Merriam-Webster, n.d.).

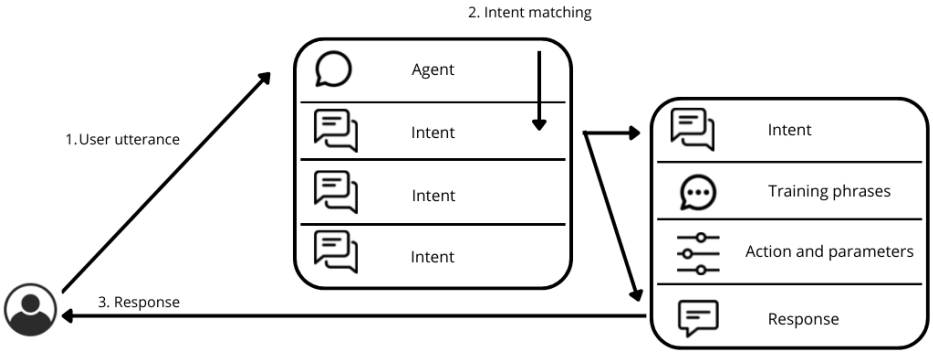

Figure 2.3: *Dialogflow diagram (adapted from Google Cloud Documentation (Google, n.d.).*

2.4.2. LUIS

LUIS, also known as Language Understanding, is a Microsoft's Artificial Intelligence Conversational cloud-based service that allows users adding customised Machine Learning to a conversation. That is, it is a model that is trained to predict the global meaning of a text, extract its most relevant information and map it to an intention. The way to use it is throughout its portal.

As Dialogflow, it is a system that can be integrated in an independent project by sending requests to this portal and results being retrieved and interpreted by the own code. In fact, it is generally used in BOTs, devices, web applications, etc. Another similarity is that this system also has pretrained intents and entities that can be used if desired. Nevertheless, it also includes the possibility of creating custom entities (LUIS NER) by using patterns, regexes, and synonyms. Additionally, LUIS has a special mode called "LUIS as a container" that allows users download a trained model in a Docker container.

As for the disadvantages of this platform, apart from the latency because it is a cloud system and sending requests and waiting for the output can delay the answer to the user, it is also a black box in which many parameters or features cannot be customised. Therefore, most users when having problems

while training can suspect what is happening, but cannot know it certainly.

Finally, LUIS will be deprecated in two years and will be replaced with "CLU - Conversational Language Understanding" (Microsoft, n.d.).

2.4.3. RASA

RASA is a Machine Learning framework used to create conversational chatbots. Furthermore, it is possible to integrate RASA easily and automatically. One positive aspect differentiating it from the rest of platforms is that it is flexible and can be modified if required. Moreover, the most relevant advantage that RASA has, is the possibility to work on three different elements: Natural Language Understanding, Natural Language Generation and Dialog Management. In contrast to other platforms, RASA allows users to have access to the configurations, that is, it is not a matter of including input data to train a model, but the user must know what he wants to configure and would be able to modify many parameters if required. According to the article *A Comparison of Natural Language Understanding Platforms for Chatbots in Software Engineering* (Abdellatif, A., 2021), RASA outperforms other platforms in intent recognition tasks: IBM Watson and Google Dialogflow.

On the contrary, one of the negative aspects of using RASA is that it is not free with large volumes of data are not free. However, there is a free version that the user can install and use locally. Apart from this, due to the possibility of accessing the parameter settings, it also requires from user's hand to work more on the training models to get better results than out-of-the-box platforms such as Dialogflow or LUIS.

2.4.4. Watson

Watson is an IBM's analysis and visualisation platform that allows users to carry out tasks related with Natural Language Processing. In fact, it not only includes the ability of classifying relevant data, identifying keywords or sentiments but it also counts on an ASR which provides users to have a speech-to-text Engine (Watson Speech to Text). In addition, it is possible to add specific vocabulary such as industry argot to train correctly models and be able to understand specific utterances.

On the downside, latency as well as indexing time are greater than other platforms. In addition, as all is inside the same collection, the use of me-

mory and the size of collection can be also a problem. Thus, applications of Watson Explorer Engine use a database that, in some cases of large collections, can be very large. Finally, the use of one collection will increase the probabilities of not obtaining a satisfactory backup.

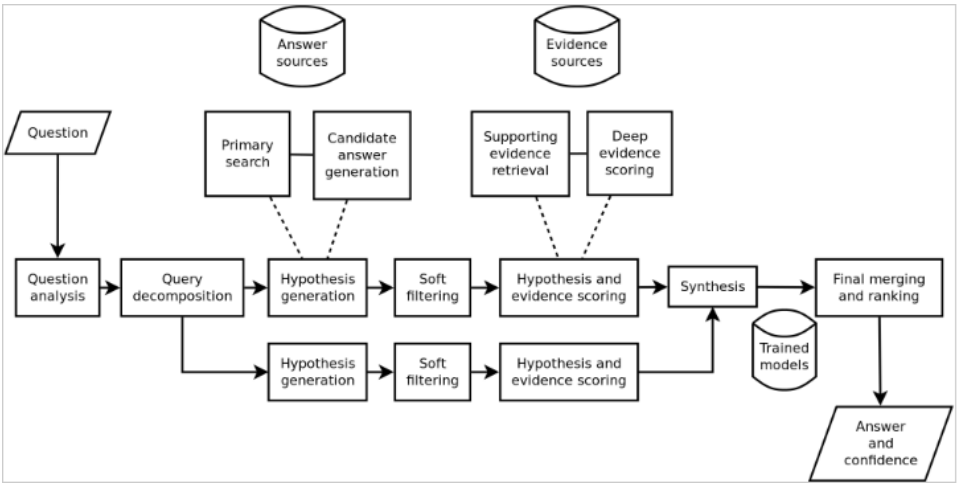

Figure 2.4: *Watson architecture diagram (extracted from Building Watson: An Overview of the DeepQA Project (Ferrucci et al., 2010)*

2.5. Intent Classification

Intent Classification or Recognition is a technique used in NLP to determine the underlying intention in a speaker's request. According to Jurafsky, it is also called intent determination and consists in obtaining the goal that the user is trying to accomplish (Jurafsky & Martin, 2022).

Intent Classification is carried out by taking the input data.In speech, data is pre-processed using an ASR system to convert and transform it into text. In these pre-processing stage, tokenisation,and keyword detection are needed to provide the system with evidence and clues so that it can classify texts correctly, this action is also known as "slot filling"[18].

Then, intent classifiers are trained and texts are labelled and mapped to intents. For example, in the request "Activa la alarma" the speaker is trying

[18]Extraction of the particular slots and fillers that the user intends the system to understand from their utterance with respect to their intent (Jurafsky Martin, 2022).

to convey the idea of turning the alarm on.

One of the main benefits of an Intent Classification module is that it is generally used for conversational AI platforms or frameworks to obtain customised conversational experiences.

Thus, many businesses could understand the needs of their customers and, sometimes, can solve customers' issues without the need of a person which is, indeed, cheaper for companies. Therefore, companies not only obtain information from their customers but also save money.

Consequently, companies that invest in getting information from their customers will be the most powerful and the richest.

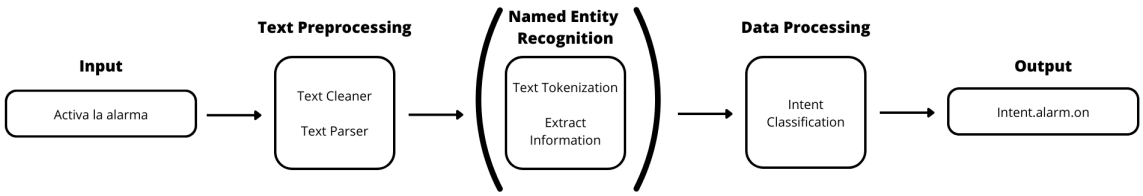

Figure 2.5: *Intent Classification diagram.*

Thus, in this work, this module has been developed to improve a real virtual assistant, Aura. Therefore, the challenge will be improving the present architecture by using new models to get better results. In order to accomplish this task, several technologies have been used. Thus, in this section, the most recent and useful technologies for intent classification will be shown.

Finally, to build an Intent Classification module, there are several methods, for example, the use of Natural Language Processing libraries or even the use of third-party platforms and/or frameworks. It is important to mention that these libraries are not only used particularly for this module but they can be also used in the input data pre-processing stage.

On the one hand, while platforms and frameworks are generally out-of-the-box products that can be integrated with in-house developments by sending requests to them and their results will be retrieved and serve as an input for the in-house code, libraries and toolkits are groups of functions that are normally used to create a script to process data from scratch.

On the other hand, there are different libraries that are mainly thought

for developing and training models, whereas others are conceived to preprocess input data.

2.6. Libraries and tools

A library is a collection of programs and packages that are made available for common use within some environment. A typical library might contain compilers, utility programs, packages for mathematical operations, etc. Usually, it is only necessary to import the library program to be automatically incorporated in a user's program (Jalli, A., 2022).

In this section, some relevant Python programming language libraries will be briefly explained in order to obtain a basic notion of them.

2.6.1. Data Science and Machine Learning

As it has been explained in 2.2.1, Data Science and Machine Learning are interlinked. Thus, the main libraries that are used to cover tasks regarding these two fields are shown below:

- **Matplotlib:** It is a Python programming language library used to generate graphics from data included in lists or arrays. In addition, it allows users to create quality plots or interactive figures that can be exported in several file formats. It is very useful to visualize and display data.

- **NumPy:** It is a Python programming language library that helps users create vectors and large multidimensional matrixes on the basis of a large group of mathematical functions.

- **Pandas:** It is an extension of NumPy used to manipulate and analyse data in Python programming language. Concretely, it offers data structures and operations to read and work with numeric tables, data frames and series. In fact, this library is generally used to read and load the input data that scripts will later process.

2.6.2. NLP

There are some tools that are used to create intent classification scripts. These tools allow users or students having an environment that covers all the needs of the script to be run.

- **Google Colab:** It is a product of Google Research also known as "Colaboratory". It allows each user writing and executing any Python code in a browser. It is specially useful in Machine Learning tasks as well as big data analytics. In addition, it has the perfect environment to run scripts that require high GPU memory usage.

- **Hugging Face Hub:** It is a data science community and platform that allow users building, training and deploying models based on several technologies. This platform is very useful for users that want to review some examples of models already developed with specific technologies. The focus of this platform is on Natural Language Processing tasks. Thus, users can have access to free courses on Deep Learning, theory, tutorial and hands-on guides. In addition, it is also considered a Web application to create and share computational documents: code analysis and development, visualisation and creation of models. It is generally used as a development and test environment before releasing a program to be used. It supports 40 programming languages such as R, Python and so on.

- **Kaggle:** Kaggle is a customisable Jupyter Notebooks environment whose GPUs[14] are free and in which there is a huge community of published data and code. Its main aim is a challenge collector in which many tasks are published and users are allowed to complete the challenge and take part in competitions. On the other hand, when starting to create a model, it can be very useful for users to have a look at the models already coded. Thus, its available input data can be very helpful in case of searching for large amounts of training data. One relevant advantage is that it can be considered as a part of portfolio for data scientists. In fact, it can be used for free.

- **PyCharm:** It is one of the most relevant IDE[15] to develop Python

[14] Graphics Processing Units.
[15] Integrated Development Environment.

code. Furthermore, PyCharm allows users having tools to debug or inspect the code, it also indicates errors while programming, and fast fixings such as the refactor of automatic code or complete browser functionalities.

When developing any module of Natural Language Processing such as an Intent Classification module, it is very useful to count on several Python libraries that automate tedious and iterative tasks. Although the most relevant Python libraries are shown below, only some of them have been used in this project. The specification of the use of each library will be described in section *Implementation*.

- **Gensim:** It is a Python programming language library for topic modelling, document indexing, and similarity retrieval with corpora. It includes memory-independent corpus algorithms, an intuitive interface, distributed computing, etc.
- **NLTK:** It is a Python programming language toolkit, that is, a group of libraries and programs regarding Natural Language Processing. It also includes libraries that allows tokenizing, detecting stop-words, semantic and syntactic analysis, classification, detection of synonyms with wordnets, etc. Therefore, many pre-processing tasks make use of a toolkit or some libraries.
- **TextBlob:** It is a Python programming language library with the aim of processing textual data. It is useful to extract noun phrases, tag part-of-speech tagging, sentiment analysis, tokenisation, intent classification, parsing, n-grams detection, spelling correction, lemmatisation, and wordnet integration.
- **AutoGluon:** It is a Python programming language library that allows users to have an easy way to use an Auto Machine Learning by focusing on automated stack ensembling, deep learning, and real-world applications spanning image, text, and tabular data. In addition, it shows a quick way to prototype deep learning and classical Machine Learning solutions with a few lines of code. Therefore, many users that are not expert can use this easily. In fact, all hyperparameters of tuning, model selection, architecture search, and data processing are automatic, therefore they cannot be changed.

- **PyTorch:** It is a Python programming language library for Machine Learning tasks. It is used for tasks regarding Artificial Visualisation as well as Natural Language Processing. It also has a C++ interface but the focus is on Python. Moreover, it includes deep neural network built on tape-based automatic differentiation system and tensor computing.

- **Scikit-learn:** It is a Python programming language library of Machine Learning. It includes classification algorithms, support-vector machines, random forests, K-means and Gradient boosting. It is designed to be used with other libraries such as NumPy and SciPy. It is generally used to get the evaluation of models as well as the confusion matrixes to analyse all the errors and confusions.

- **Spacy:** It is a Python programming language library to carry out Natural Language Processing tasks such as tokenisation in different languages, importing and training models for different languages too and multi-task learning with transformers such as BERT. Additionally, it is also compatible with PyTorch and TensorFlow.

- **TensorFlow:** It is a programming language library of Machine Learning and Artificial Intelligence that can be used in several programming languages such as Python, C++ and Java. It is normally used to carry out some tasks by focusing on training models and inferring deep neural network. Furthermore, some useful tasks that can be done are tokenisation, training model, obtaining metrics and results, etc.

Chapter 3

Proposal

In this chapter, research questions, their approaches, and difficulties found during the experiment will be described in depth.

3.1. Research questions

As it has been mentioned, intent classification tasks raise a problem in terms of using the most accurate model to obtain the best results.

Thus, how can it be determined if a model outperforms the other? Does it depend on the intents used to train and test? In addition, the number of classes can affect the accuracy of a classification. It is said that the greater number of classes to be grouped by, the less accurate a model will be. What is the best approach for measuring performance?

On the other hand, there are different paradigms that can be addressed. Furthermore, it has been mentioned that when changing the criteria of classification, even if the data set is the same, results may vary. How different are the results taking into account two different approaches when classifying? Can an attribute be decisive when classifying?

3.2. Approach

To solve these paradigms, several technologies, algorithms and platforms will be used.

Thus, to answer these research questions two paradigms will be tackled within the project.

- **Intent classifier:** The idea of this challenge is to compare different technologies to classify users' utterances by intents. This quandary will be developed by using different data sets. The results of both data sets for each technology used will be compared to see how they evolve and which technology works the best.

- **Attribute classifier:** As both data sets are based on intents related to perform actions "turning on" and "turning off", these characteristics will be the focus of this challenge. Therefore, all technologies by using these two data sets but defining two new intents by classifying utterances by concept or attribute "on" and "off" will be the focus of this research question. Thus, the key objective of this research is detecting which the best technology is to solve this problem.

The main approach to answer these questions will be explained below:

Firstly, all datasets will be pre-processed. Then, these data will be classified by two different model types:

Then, these data will be classified by two different model types: intent classification based on performing different actions depending on the selected class; and attribute classification, based on the concept of turning on or off any attribute. Theoretically, regarding the second paradigm, by adding a secondary module, for example a NER module, first research question will be addressed by identifying keywords such as "alarm", "timer" and/or "reminder" as named entities (NER). This secondary module is a theoretical approach because it is not going to be developed throughout the project.

Finally, the outputs of these models will be evaluated by the evaluation module. After that, a comparison between technologies and algorithms will be done to check which model performs best for each paradigm and to see how models behave using different datasets.

3.3. Difficulties found

During the development of this project, several problems have been found. The main difficulties found will be described in this section:

- **Imbalance of classes within the data sets:** When facing real data, as Aura's real logs have been used in this thesis, there are several difficulties such as defining use cases to be used that have a similar

number of utterances. For example, *alarm.on* has about 2.000 logs while the rest of intents have only around 200 logs.
To solve this problem, there are two different approaches: reduce the multitudinous logs sample of *alarm.on* randomly and leave just 200 logs sample as the rest of intents; or increase the number of logs for the rest of intents to have the same sample as *alarm.on*.
However, this last alternative is not viable because one premise of this work was that data sets should be real. Therefore, the first option was selected.

- **Transformers environment required:** When trying to train models of transformers such as *BERT*, *RoBERTa* or *XLNet*, the problem of not having a proper GPU arose. Therefore, after researching how to train these models in machines without the specific requirements that transformers need, the use of Google Colab was the solution that was provided.

- **Spanish language as the basis:** As Aura's logs are in Spanish, the language framework was Spanish language. However, most of technologies as well as libraries in NLP are just thought for English language. Therefore, this was one of the difficulties found when choosing technologies and libraries for this project.

- **Evaluating with the same metrics:** Since several technologies and libraries are used, although the metrics used are the same, their output may be totally different. Thus, to compare all these technologies by using the same data set, it was very important to define the concrete metrics that the output must include. Therefore, apart from default metrics that each technology has, a library has been chosen to be used for the output of all the different technologies to calculate the same metrics and confusion matrixes.

Chapter 4

Dataset

This chapter describes the datasets used in this project, their details, a comparison between them and all their details. All these datasets have been selected from Aura's logs (Telefónica's virtual assistant). These logs have been tagged manually to be able to train the system. Once they have been tagged, they have been randomly divided in training and test sets. The distribution of these datasets are: 80 % for training and 20 % for test set. For each classification model, the same datasets are going to be used but the paradigm changes. The table below describes a summary of all datasets used in this project.

Datasets	Intent names	Training	Testset
Dataset A	alarm.off, alarm.on, timer.off, timer.on	640	160
Dataset B	alarm.off, alarm.on, timer.off, timer.on, reminder.off, reminder.on	960	240
Dataset C	on, off	640	160
Dataset D	on, off	960	240

Table 4.1: *Datasets summary*

4.1. Intent classifier Datasets

Datasets A and B are going to be used for intent classification.

4.1.1. Dataset A

Samples of Training set and Test set from this dataset have been extracted randomly from the whole sample of Aura's logs from the last year. In includes several linguistic patterns to identify each action. For each intent, there are specific linguistic expressions that determine if an utterance should be classified under a concrete class:

alarm.off: The keyword is "alarma" or its synonyms and plural form and some verbs that map the intent to the action of "turning off" would be "quitar, cancelar, posponer, retirar, eliminar", etc. Some examples of this use case are: "Anular servicio alarma", "Quitar la alarma a las 3.00 de la mañana" or "Cancela alarma".

alarm.on: The keyword is "alarma" or its synonyms and plural form and some verbs that map the intent to the action of "turning on" would be "activar, encender, poner, establecer, prender", etc. Some examples of this use case are: "Pon una alarma a las 7 menos 10 minutos", "Quiero poner una alarma a las 9.50" or "Activa la alarma".

timer.off: The keyword is "temporizador" or its synonyms and plural form and some verbs that map the intent to the action of "turning off" would be "quitar, cancelar, posponer, retirar, eliminar", etc. Some examples of this use case are: "Apaga temporizador", "Cancelar el temporizador de 1 minuto 40 segundos" or "Detener el temporizador de cinco minutos".

timer.on: The keyword is "temporizador" or its synonyms and plural form and some verbs that map the intent to the action of "turning on" would be "activar, encender, poner, establecer, prender", etc. Some examples of this use case are: "Pon el temporizador de medio minuto", "Activar el temporizador" or "Pon temporizador diez minutos".

Intent Name	Training samples	Testing samples
alarm.off	160	40
alarm.on	160	40
timer.off	160	40
timer.on	160	40

Table 4.2: *Training and Test set details of Dataset A*

4.1.2. Dataset B

Samples of Training set and Test set from this dataset have been selected randomly from the whole sample of Aura's logs from the last year. In includes several linguistic patterns to identify each action as in the previous dataset. In fact, this dataset is the same as the previous one but adding two more intent names. In this section, the specific linguistic expressions that determine if an utterance should be classified under one of the two additional intents are explained below.

- **reminder.off:** The keyword is "recordatorio" or its synonyms and plural form and some verbs that map the intent to the action of "turning off" would be "quitar, cancelar, posponer, retirar, eliminar", etc. Some examples of this use case are: "Quiero anular recordatorio", "Quita el recordatorio que hay" or "Quiero apagar el recordatorio audio".

- **reminder.on:** The keyword is "recordatorio" or its synonyms and plural form and some verbs that map the intent to the action of "turning on" would be "activar, encender, poner, establecer, prender", etc. Some examples of this use case are: "¿Puedes añadir un recordatorio?", "Pon un recordatorio para mañana a las 10.00 de la mañana" or "Crea el recordatorio".

Intent Name	Training samples	Testing samples
alarm.off	160	40
alarm.on	160	40
timer.off	160	40
timer.on	160	40
reminder.off	160	40
reminder.on	160	40

Table 4.3: *Training and Test set details of Dataset B*

4.2. Attribute classifier Datasets

Dataset C and D are going to be used for attribute classification.

4.2.1. Dataset C

This dataset comprises a group of utterances that are the same that Dataset A but they have been tagged according to two specific attributes or new intents: "on" and "off".

- **off:** The keyword is "off" or its synonyms. This group of utterances is composed of Dataset A *alarm.off* and *timer.off*. Some examples of this use case are: "Quitar el temporizador de 10 minutos", or "¿Puedo esquivar la alarma a las 8.00?".
- **on:** The keyword is "on" or its synonyms. This group of utterances is composed of Dataset A *alarm.on* and *timer.on*. Some examples of this use case are: "Pon temporizador", or "¿Puedes conectar una alarma?".

Attributes	Training samples	Testing samples
Off	320	80
On	320	80

Table 4.4: *Training and Test set details of Dataset C*

4.2.2. Dataset D

This dataset comprises a group of utterances that are the same that Dataset B but they have been tagged according to two specific attributes or new intents: "on" and "off". The difference between this dataset and the previous one is the addition of two more use cases: *reminder.on* and *reminder.off*.
This dataset comprises a group of utterances that are the same that Dataset B but they have been tagged according to two specific attributes: "on" and "off".

- **off:** The keyword is "off" or its synonyms. This group of utterances is composed of Dataset A *alarm.off*, *reminder.off* and *timer.off*. Some examples of this use case are: "Elimina temporizador", "Recordatorio detenido" or "Anula la alarma".
- **on:** The keyword is "on" or its synonyms. This group of utterances is composed of Dataset A *alarm.on*, *reminder.on* and *timer.on*. Some examples of this use case are: "Pon temporizador", "Poner recordatorio" or "¿Puedes conectar una alarma?".

Attributes	Training samples	Testing samples
Off	480	120
On	480	120

Table 4.5: *Training and Test set details of Dataset D*

Chapter 5

Experimentation

In this section, the development of all the experiments will be described by including the metrics used and the modules that compose the main development of the work. Thus, this section includes, evaluation setting, which is a point that highlights the metrics used and why they have been used; the implementation, which is a point that includes the three different modules that entail the whole development of the experiment; and the results in which the output of the different experiments will be described.

5.1. Evaluation Setting

For all datasets, intent classification was evaluated by using the same metrics: precision, recall and F1 in all the models.

The choice of these metrics is a primary key for model selection. Thus, this selection is based on the type of classification: binary classification or multilabel classification, and the model that is going to be used for this classification.

Therefore, precision measures the ability of a classifier to identify the correct utterances in each class while recall is the ability of a classifier to find all the correct utterances per class. Finally, F1 is the harmonic mean of the two previous measures (Vilalta, 2000).

To define and understand these metrics, it is important to explain the following concepts:

- **TP:** A true positive is a sample that is correctly classified as belonging to a positive class by the model. For example, if the actual class of a

sample is class A and the model also predicts class A, then this is considered a true positive for class A.

- **TN:** A true negative is a sample that is correctly classified as belonging to a negative class by the model. For example, if the actual class of a sample is not class A and the model also predicts a class different from class A, then this is considered a true negative for class A.

- **FP:** A false positive is a sample that is incorrectly classified as belonging to a positive class by the model. For example, if the actual class of a sample is not class A but the model predicts class A, then this is considered a false positive for class A.

- **FN:** A false negative is a sample that is incorrectly classified as belonging to a negative class by the model. For example, if the actual class of a sample is class A but the model predicts a class different from class A, then this is considered a false negative for class A.

Once these definitions are clear, the metrics can be described as below:

- **Precision:** It is explained as how good the model is when predicting a class. For each class, there is a metric of precision.

$$\textbf{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}}$$

Figure 5.1: *Precision metric (adapted from Explaining Accuracy, Precision, Recall, and F1 Score (Bhadouria, V., 2020).*

- **Recall:** It is explained as the percentage of the correct predictions of the model regarding the actual labels or classes. For each class, there is a metric of recall.

$$\textbf{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

Figure 5.2: *Recall metric (adapted from Explaining Accuracy, Precision, Recall, and F1 Score (Bhadouria, V., 2020).*

- **F1:** It is explained as the balanced metric between precision and recall. For each class, there is a metric of F1.

$$\textbf{F1 Score} = \frac{\text{2 x (Precision x Recall)}}{\text{Precision + Recall}}$$

Figure 5.3: *F1 metric (adapted from Explaining Accuracy, Precision, Recall, and F1 Score (Bhadouria, V., 2020).*

- **Accuracy:** It is explained as how good our model is at predicting the correct category.

$$\text{Accuracy} = \frac{\text{True Positives + False Positives}}{\text{All the sample}}$$

Figure 5.4: *Accuracy metric (adapted from Explaining Accuracy, Precision, Recall, and F1 Score (Bhadouria, V., 2020).*

Once these metrics are available, a confusion matrix[34] can be created to see how classes have diverted to others.

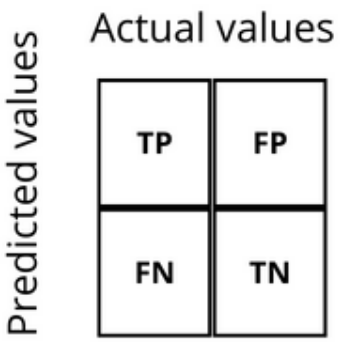

Figure 5.5: *Understanding Confusion Matrix (Narkhede, S., 2018)*

[34] A confusion matrix is a performance measurement for Machine Learning Classification problem of at least two classes. It is a table with four different combinations of predicted and actual values (Narkhede, S., 2018).

5.2. Implementation

In this section, all the modules related to the implementation will be explained: pre-processing module, that is common to all the experiments; classification module, which include the explanation of the libraries and parameters used in all the experiments and models used; and evaluation module, which is the explanation of the code that has been developed for the metrics explained in 5.1.

5.2.1. Pre-processing Module

A pre-processing module can be defined as a group of operations that is applied to each string to delete irrelevant and undesired information, to standardize the format and to extract relevant and useful characteristics or attributes for the following stages or modules. In this section, all the operations used to pre-process Aura's logs will be described in depth.

- **Loading and reading data:** To load data from Aura's logs, a CSV file was the format chosen for data sets. These CSV files (training and test sets) include two columns: "texts or utterances" and "class name or intent name". To load and read these files, the Pandas library, concretely, *read_csv* has been used.

- **Tokenize:** Once the files have been read, the column "texts or utterances", row by row, that is, each user's utterance is split in words. This tokenisation is carried out by using from the NLTK library, *word_tokenize*.

- **Lowercase:** After that, all the result input, already tokenised, is lowercased. To perform this action, a lambda function is used to lower-case users' utterances.

- **Stopwords cleaning:** Then, once the input is in lowercase and already tokenised, the process of stopwords[20] removal begins. Fortunately, NLTK counts with a stopword list for Spanish. Therefore, the library used has been this one.

[20]Stopwords, also known as "empty words" are words that have no meaning by themselves, that is, they are used to pair them with other words. For instance: articles, pronouns, adverbs, prepositions and even some verbs. These words, generally, in NLP are filtered not to be taken into account when processing them.

- **Stemming:** Once these operations are carried out, the last but not least operation is performed, that is, the stemming[21]. As Porter Stemmer has been also used for Spanish language texts, this algorithm has been selected to carry out this task. Therefore, from the library NLTK, *PorterSteemer* has extracted the stem word of each token within the utterance. The difference between stemming and lemmatisation is that stemming uses the stem of the word while lemmatisation transforms the word into its base form.

Once these operations have been accomplished, the input for the following module has been obtained.

5.2.2. Classification Module

In this section, all the algorithms and technologies used to answer and solve the research questions laid out in section 3.1 will be described, its configuration and parameters used as well as all the requirements met to use them to achieve these challenges.
For each model, there will be two descriptions due to the fact that there are two different research questions: intent classification and attribute classification. Most of the models have been developed and evaluated in Python, and the ones developed in dialogue platforms have been also evaluated in Python.

- **Models from scratch:**

  All the models developed from scratch represent some of the most relevant models for intent classification. LSTM and SVM are used to classify users' utterances in defined intents.

  - **LSTM Model:** In this model, firstly, once the pre-processing module processed the training and the test set files, the Pandas library has been used to load and read training and test set files. Secondly, the module preprocessing from the Scikit-learn library is loaded to use OneHotEncoder to convert classes to labels. Then, the TensorFlow library is imported to use its tokenizer. After that, the tokenised files are padded by importing

[21] Stemming is the process to reduce a word to its word stem or root form. It is normally used in information retrieval processes. In addition, the most relevant algorithm for stemming is Porter Stemmer

pad_sequences from TensorFlow.
Later, a function is created to transform data into a matrix. Next, by importing TensorFlow Dense, LSTM, BatchNormalisation, Dropout, Input from layers module; Adam from optimizers module, CategoricalCrossentropy from losses module, *he_uniform* and *glorot_uniform* from initializers module, AUC from metrics module, Model from Keras module and l2 from regularizers module, a class called LSTM model has been created. Thus, it has been used to build the model by using the following configuration:

- **Dropout rate**[22]**:** The set value is 0.5.
- **Bias regularizer**[23]**:** The set value is 0.3.
- **Kernel regularizer**[24]**:** The set value is 0.3.

Finally, to train the model, the parameters that have been configured are:

- **Validation split**[25]**:** The set value is 0.2.
- **Epochs**[26]**:** The set value is 60.

According to previous researches such as *Deep Bi-Directional LSTM Network for Query Intent Detection* (Sreelakshmi et al., 2018), the semantic enrichment and the proposed deep learning model implemented when using LSTM improves the results of intent detection.

- **SVM Model:** In this model, firstly, once the pre-processing module processed the training and the test set files, the Pandas library has been used to load and read training and test set files.

  Secondly, the library Spacy has been imported. This library has been used to load the pipeline *es_core_news_md* that includes

[22] It is the probability of training a given node in a layer (Srivastava et al., 2014).
[23] It applies a penalty only on the layer's bias.
[24] It applies a penalty on the layer's kernel (weight) but excluding bias.
[25] Keras can separate a portion of the training data into a validation set and then, evaluate the performance of the model used on that validation dataset in each epoch.
[26] It is a parameter that defines the number of times that the learning algorithm will work through the training dataset.

the following components: tok2vec, morphologizer, parser, senter, NER, attribute_ruler and lemmatizer. It is in Spanish and counts with vocabulary, syntax and entities extracted from written texts of pieces of news and media.
After that, the dimensionality of the loaded pipeline is calculated. Later, by defining a function, the sentences used in Spacy NLP model are encoded to obtain a encoded training set and a encoded test set. Then, by importing Scikit-learn preprocessing module and using LabelEncoder, classes assigned to training and test set files are also encoded. Finally, module SVM from the Scikit-learn library is imported to define the training and validation functions that only requires the following parameters:

- **training function:** model, encoded training set and encoded labels of training set file.
- **validation function:** model, encoded test set and encoded labels of test set file.

In addition, one of the advantages of using SVM to classify intents is its effectiveness both in high dimensional spaces (it is relatively memory efficient], and in cases in which the number of dimensions is greater that the number of samples, and it works relatively well when classes are clearly distinguishable.

- **Auto Machine Learning:** The development of AutoGluon, this Auto Machine Learning library, is based on their modules and all the models that are trained when using it. To begin with, the library AutoGluon module *tabular*, by importing *TabularDataset* allows loading and reading training and test datasets. After that, the TabularPredictor from *tabular* module is also imported to create the *predictor*. Once it is done, by using the feature *TabularPredictor* and adding our expected intents as labels and using the training data already loaded, the models are trained. The libraries used to train models made by AutoGluon are those below described:

  - **CatBoost:** It is a high-performance open source Python library for gradient boosting on decision trees (AutoGluon, n.d.).

- **ExtraTreesEntr:** It is a class that implements a meta estimator that fits a number of randomised decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting (Scikit-learn, n.d.). Its supported criteria is Entropy[27].

- **ExtraTreesGini:** It is a class that implements a meta estimator that fits a number of randomised decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting (Scikit-learn, n.d.). Its supported criteria is Gini impurity[28].

- **KNeighborsDist:** It is a classifier that implements the k-nearest neighbors vote by using the parameter of weight with value *distance* (Scikit-learn, n.d.).

- **KNeighborsUnif:** It is a classifier that implements the k-nearest neighbors vote by using the parameter of weight with value *uniform* (Scikit-learn, n.d.).

- **LightGBM:** It is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed. (Microsoft Corporation, n.d.).

- **LightGBMLarge:** It is a gradient boosting framework that uses tree based learning algorithms with a custom configuration which enables larger models but trains slower (Microsoft Corporation, n.d.).

- **LightGBMXT:** It is a gradient boosting framework that uses tree based learning algorithms and using the parameter *extra_trees* is "True" (Microsoft Corporation, n.d).

- **NeuralNetFastAI:** It is a class for Fast AI v1 neural network model that operate on tabular data (AutoGluon, n.d.).

[27]It is a measurement used to build Decision Trees to determine how the features of a dataset should split nodes to form the tree (Mohri, M., et al., 2018).

[28]It is the measurement of disorder or impurities in the information processed in Machine Learning (Zhou, V., 2019).

- **NeuralNetTorch:** It is a class for neural network model that operates on tabular data. These networks use different types of input layers to process different types of data in various columns (AutoGluon, n.d.).
- **RandomForestEntr:** It is a random forest classifier that is considered a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Its supported criteria is Entropy (Scikit-learn, n.d.).
- **RandomForestGini:** It is a random forest classifier that is considered a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control.
- **WeightedEnsemble_L2:** It is a weighted ensemble meta-model that implements Ensemble Selection.
- **XGBoost:** It is an optimised distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements Machine Learning algorithms under the Gradient Boosting framework (XGBoost, n.d.).

For the three models regarding transformers that have been developed, the code is practically the same. The changes that are performed differ in the modules used concerning the transformers library and some parameters of the model. In this section, the code is going to be explained and the differences of the configuration parameters will be detailed in each model paragraph.
Thus, in these models, once the pre-processing module processed the training and the test set files, the Pandas library has been used to load and read training and test set files. Secondly, labels of training and test set files are encoded by using *LabelEncoder* from the Scikit-learn preprocessing module. Later, some libraries are imported such as the TensorFlow library, concretely, Adam and SGD from optimizers module.
Furthermore, by importing *ClassificationModel* from the simpletransformers library classification module, the model is created by adding

a concrete setting up that will be shown in the following paragraphs. Finally, by using the module train_model and the training file loaded, the selected model is trained.

Next, the common configuration for the three models will be shown:

- **GPU**[29] The set value is enabled.
- **num_labels**[30]**:** The set value depends on the dataset used.

Dataset used	num_labels
Dataset A	4
Dataset B	6
Dataset C	2
Dataset D	2

Table 5.1: *Labels distribution for transformers' model.*

- **args**[31]**:** The set value is num_train_epochs, that describes the number of epochs selected for training, whose value in these experiments with this type of model is 30.

To conclude, the concrete setting up parameters and libraries used for the three models will be shown below:

- **BERT's model:** The only difference concerning the transformers library is the use of modules *BertTokenizer* and *TFBertModel* to tokenize and build the model based on BERT. In addition, regarding the concrete parameters used in the *ClassificationModel* library to create and train the model are those below:
  - **Model type**[32]**:** The set value is bert.
  - **Model name**[33]**:** The set value is bert-base-cased.

[29]Graphics Processing Unit.
[30]It is an optional argument, but useful one when the number of classes should be taken into account.
[31]It refers to optional arguments, for example num_train_epochs.
[32]It refers to the type of model used (Rajapakse, 2022).
[33]It specifies the exact architecture and trained weights to use. This may be a Hugging Face Transformers compatible pre-trained model, a community model, or the path to a directory containing model files (Rajapakse,2022).

- **RoBERTa's model:** The one difference concerning the transformers library is the use of modules DistilBertTokenizer and RobertaTokenizer to tokenize and build the model based on RoBERTa. In addition, regarding the concrete parameters used in the *ClassificationModel* library to create and train the model are those below:
  - **Model type:** The set value is roberta.
  - **Model name:** The set value is roberta-base.
- **XLNet's model:** The one difference concerning the transformers library is the use of modules TFXLNetModel and XLNetTokenizer to tokenize and build the model based on XLNet. In addition, regarding the concrete parameters used in *ClassificationModel* library to create and train the model are those below:
  - **Model type:** The set value is xlnet.
  - **Model name:** The set value is xlnet-base-cased.

Transformer model	Model type	Model name	num_labels	num_train_epochs
BERT	bert	bert-base-cased	Dataset A: 4, Dataset B: 6, Dataset C: 2, Dataset D: 2	30
RoBERTa	roberta	roberta-base	Dataset A: 4, Dataset B: 6, Dataset C: 2, Dataset D: 2	30
XLNet	xlnet	xlnet-base-cased	Dataset A: 4, Dataset B: 6, Dataset C: 2, Dataset D: 2	30

Table 5.2: *Transformers configuration summary*

- **Dialogue Management Platforms Models:** Third party platform models have been developed in their own platform and evaluated in a Python module.

- **Dialogflow Model:** As there are four different data sets, four agents in Dialogflow have been built. Although they have the same configuration, the data imported in these agents is different due to the four data sets. The agent configuration is the following one:

  - **Default time zone:** GMT+1 (Madrid zone).
  - **Beta Features:** Enable beta features and APIs.
  - **Language:** Spanish ES.
  - **ML Classification Threshold:** 0.5.
  - **Default entities:** Disable.

  For each agent, different entities and intents are registered:

Agents	Intent names	Entities
Dataset A	alarm.off, alarm.on, timer.off, timer.on	"alarma", "temporizador"
Dataset B	alarm.off, alarm.on, timer.off, timer.on, reminder.off, reminder.on	"alarma", "temporizador", "recordatorio"
Dataset C	on, off	"encender", "apagar"
Dataset D	on, off	"encender", "apagar"

Table 5.3: *Dialogflow agents summary*

- **LUIS Model:** Four different applications in LUIS have been created because four different experiments are going to be carried out. Although they have the same configuration, the training data included in these applications is different due to the four data sets. The application setting up is the following one:

  - **Culture:** es-ES.
  - **Prediction resource:** Enable.
  - **Domain:** Selected a pre-built domain.
  - **Default entities:** Disable.

  For each application, different entities and intents are registered:

Agents	Intent names	Entities
Dataset A	alarm.off, alarm.on, timer.off, timer.on	“alarma”, “temporizador”
Dataset B	alarm.off, alarm.on, timer.off, timer.on, reminder.off, reminder.on	“alarma”, “temporizador”, “recordatorio”
Dataset C	on, off	“encender”, “apagar”
Dataset D	on, off	“encender”, “apagar”

Table 5.4: *LUIS applications summary*

5.2.3. Evaluation Module

All the models developed in 5.2.2 have the same evaluation module. This module consists of, firstly, extracting labels expected from the test set files and, secondly, predicting labels for each utterance that appears in each test set file by each model.

Then, once the two arrays have been obtained (predicted test set and labeled test set), by importing the module metrics of Scikit-learn, concretely, *classification_report* and *confusion_matrix*, all the metrics explained previously have been obtained (precision, recall, F1, total accuracy and confusion matrix), by model and data set.

Thus, for models developed from scratch such as LSTM and SVM models, there will be four different results one per each dataset and model (eight in total).

The same will happen with transformers' models (BERT, RoBERTa, and XLNet), there will be four different results one by each dataset. Thus, twelve different results will be obtained regarding transformers.

Concerning AutoGluon Machine Learning models, as there are fourteen models (CatBoost, ExtraTreesEntr, ExtraTreesGini, KNeighborsDist, KNeighborsUnif, LightGBM, LightGBMLarge, LightGBMXT, NeuralNetFastAI, NeuralNetTorch, RandomForestEntr, RandomForestGini, WeightedEnsemble L2, XGBoost), there will be fifty-six different results, taking into account the four datasets used.

Finally, regarding dialog management platforms (LUIS and Dialogflow), there will be also eight results: four from each platform including the four different datasets that have been tested.

In addition, it is relevant to mention that the evaluation of dialog management platform modules has a singularity. This consists in creating a

previous submodule in which POST requests are sent to each platform and the obtained results are saved to generate the array required of predicted test set labels.

This submodule has been also developed by using Python. It requires the training set, the test set (input files), a configuration file and the output file. In addition, it performs tests against the different services (Dialogflow and LUIS), and depending on each service, SDKs or requests to other applications that allow testing the model are used. Once the information of these requests is retrieved, it is saved and printed in the output file.

5.3. Results

Results will be shown in different ways. Firstly, a display of the average accuracy of all the models will be shown to be able to compare these models in general.

Then, the results will be shown by dataset and model used, splitting them by type of development: models developed from scratch, models based on transformers, models with AutoGluon and models developed with dialogue management platforms. Finally, the results will be compared and analysed regarding the different datasets in the following chapter.

5.3.1. Average accuracy results by model

In this section, the results regarding average accuracy will be shown in a table to be compared below:

Models	Dataset A	Dataset B	Dataset C	Dataset D	Dataset Average
XLNet	0.98	0.97	0.97	0.94	0.965
Dialogflow	0.96	0.97	0.97	0.96	0.965
LUIS	0.94	0.95	0.96	0.97	0.955
RoBERTa	0.98	0.97	0.97	0.88	0.95
BERT	0.97	0.97	0.98	0.88	0.95
SVM	0.89	0.87	0.94	0.93	0.9075
LightGBMXT	0.89	0.86	0.89	0.87	0.8775
ExtraTreesGini	0.87	0.87	0.89	0.86	0.8725
ExtraTreesEntr	0.87	0.87	0.88	0.86	0.87
WeightedEnsemble L2	0.86	0.87	0.89	0.86	0.87
LSTM	0.83	0.87	0.94	0.84	0.87
CatBoost	0.88	0.85	0.89	0.85	0.8675
XGBoost	0.87	0.87	0.88	0.85	0.8675
LightGBM	0.88	0.82	0.88	0.85	0.8575
LightGBMLarge	0.86	0.85	0.87	0.85	0.8575
RandomForestGini	0.86	0.84	0.86	0.85	0.8525
RandomForestEntr	0.86	0.83	0.86	0.86	0.8525
KNeighborsDist	0.74	0.69	0.81	0.71	0.7375
KNeighborsUnif	0.74	0.67	0.81	0.72	0.735
NeuralNetTorch	0.63	0.42	0.73	0.71	0.6225
NeuralNetFastAI	0.62	0.41	0.76	0.68	0.6175

Table 5.5: *Average accuracy by model summary*

On the one hand, based on the results obtained in the table above, the best average accuracy results regarding dataset A are those obtained by models based on transformers: RoBERTa and XLNet have obtained the same metric and BERT has obtained a similar result. The same situation has occurred with dataset B. However, in this case, all the models based on transformers have obtained the same result. In addition, Dialogflow has also reached the same punctuation.

Concerning dataset C, BERT has obtained a slightly better result. Close to this score, there are other models such as RoBERTa, XLNet, and Dialogflow.

Regarding dataset D, the best results are obtained by models developed with dialogue management platforms: LUIS in this case is minimally better

than Dialogflow.

Finally, reviewing the average dataset column, it can be observed that Dialogflow and XLNet have obtained the best results followed by LUIS, RoBERTa and BERT. Therefore, both technologies transformers and dialog management platforms are the models that seem to be the most appropriate ones to classify these types of Use Cases.

Thus, according to the previous conclusions, it can be said that, generally, models based on transformers and models developed with dialogue management platforms are more complete and accurate than the rest of model types.

To summarize, XLNet would be the best option to develop these types of Use Cases; firstly, because it has obtained one of the best results; and, secondly, because it allows developers modifying parameters to continue improving the results of the classification.

On the other hand, it can be observed that, generally, the type of model that works the worst is models that perform worst are those developed with AutoGluon. However, it is true that not all the models developed with this auto Machine Learning tool work in the same way. That is, CatBoost, ExtraTreesEntr, ExtraTreesGini, LightGBM, LightGBMLarge, LightGBMXT, RandomForestEntr, RandomForestGini, WeightedEnsemble_L2 and XGBoost are better that the rest of the models. Thus, it can be confirmed that some algorithms such as CatBoost, ExtraTrees variations, LightGBM variations, RandomForest variations and XGBoost work better than KNeighbors variations, NeuralNet FastAI and Torch.

As a result of that, it can be concluded that transformers, that is one of the last algorithm types that have been developed, have supposed a great progress concerning intent classification tasks. Although this model type is a great progress, it is evident that models that have been developed with dialogue management platforms are also great solutions to carry out intent classification tasks quickly and easily.

Although both model types have obtained similar results, it is clear that one of the advantages of transformers is that the developer can alter and modify whatever they need to optimize these results, while concerning the dialogue platforms, developers can only modify training to make the results improve, which is a clear limitation.

5.3.2. Results based on model type and datasets

In this section, results regarding model types will be described and shown in a table to be compared below:

Models	Dataset A	Dataset B	Dataset C	Dataset D
XLNet	alarm.off (0.99), alarm.on (0.99)	alarm.off (0.99), alarm.on (0.99)	on (0.97), off (0.97)	on (0.97), off (0.97)
Dialogflow	alarm.off (1.00), alarm.on (1.00)	alarm.off (0.99), alarm.on (0.99)	on (0.97), off (0.97)	on (0.96), off (0.96)
LUIS	alarm.off (0.99), alarm.on (0.99)	alarm.off (0.97), alarm.on (0.97)	on (0.96), off (0.96)	on (0.97), off (0.97)
RoBERTa	alarm.on (0.99), timer.on (0.99)	reminder.off (1.00), reminder.on (1.00)	on (0.97), off (0.97)	off (0.89)
BERT	alarm.on (0.98)	alarm.off (0.99), alarm.on (0.99)	on (0.98), off (0.98)	on (0.89)
SVM	alarm.on (0.94)	alarm.on (0.93)	on (0.94), off (0.94)	on (0.93), off (0.93)
LightGBMXT	alarm.off (0.92)	alarm.off (0.93)	off (0.90)	on (0.87), off (0.87)
ExtraTreesGini	alarm.off (0.93)	alarm.off (0.93)	on (0.89), off (0.89)	off (0.87)
ExtraTreesEntr	alarm.off (0.92)	alarm.off (0.94), alarm.on (0.94)	off (0.88)	off (0.87)
WeightedEnsemble L2	alarm.off (0.88), alarm.on (0.88)	alarm.off (0.95), alarm.on (0.95)	off (0.90)	off (0.87)

Table 5.6: *Classes with best F1 result for each dataset by model summary*

Models	Dataset A	Dataset B	Dataset C	Dataset D
LSTM	timer.off (0.95)	alarm.off (1.00), alarm.on (1.00)	on (0.95), off (0.95)	off (0.86)
CatBoost	alarm.off (0.94), timer.on (0.94)	alarm.on (0.88)	off (0.90)	off (0.87)
XGBoost	alarm.off (0.91)	alarm.off (0.95), alarm.on (0.95)	off (0.88)	off (0.86)
LightGBM	alarm.off (0.93)	alarm.off (0.92)	off (0.89)	off (0.86)
LightGBMLarge	alarm.off (0.89), alarm.on (0.89)	alarm.off (0.94), alarm.on (0.94)	off (0.88)	off (0.86)
RandomForestGini	alarm.off (0.90), alarm.on (0.90)	alarm.off (0.90)	on (0.86), off (0.86)	off (0.87)
RandomForestEntr	alarm.off (0.90), alarm.on (0.90)	alarm.off (0.89)	on (0.86), off (0.86)	off (0.88)
KNeighborsDist	alarm.off (0.80)	alarm.on (0.79)	off (0.82)	off (0.73)
KNeighborsUnif	alarm.off (0.77)	alarm.on (0.75)	off (0.83)	off (0.74)
NeuralNetTorch	timer.on (0.68)	alarm.on (0.60)	on (0.73)	off (0.73)
NeuralNetFastAI	timer.on (0.67)	timer.on (0.57)	off (0.76)	on (0.69)

Table 5.7: *Continuation: Classes with best F1 result for each dataset by model summary*

Models	Dataset A	Dataset B	Dataset C	Dataset D
XLNet	timer.off (1.00), alarm.on (1.00)	reminder.off (1.00), reminder.on (1.00), alarm.on (1.00)	off (1.00)	off (0.98)
Dialogflow	alarm.off (1.00), alarm.on (1.00)	alarm.off (1.00), timer.off (1.00)	on (0.99)	on (0.96), off (0.96)
LUIS	alarm.on (1.00)	alarm.off (0.97), alarm.on (0.97), timer.off (0.97)	off (0.97)	on (0.98)
RoBERTa	alarm.on (1.00)	alarm.on (1.00), reminder.off (1.00), reminder.on (1.00)	on (0.97), off (0.97)	on (0.90)
BERT	alarm.off (1.00), timer.off (1.00)	alarm.off (1.00), timer.off (1.00), reminder.off (1.00)	off (1.00)	off (0.96)
SVM	alarm.on (0.93)	reminder.on (0.94)	on (0.95)	off (0.96)
LightGBMXT	alarm.on (0.97)	alarm.on (0.97)	on (0.96)	on (0.89)
ExtraTreesGini	alarm.on (0.97)	alarm.on (0.97)	on (0.89), off (0.89)	on (0.95)
ExtraTreesEntr	alarm.on (0.97)	alarm.on (0.97)	on (0.88)	on (0.95)
WeightedEnsemble L2	alarm.off (0.88), alarm.on (0.88)	alarm.on (0.97)	on (0.96)	on (0.94)

Table 5.8: *Classes with best Precision result for each dataset by model summary*

LSTM	alarm.on (1.00)	alarm.off (1.00), alarm.on (1.00), reminder.on (1.00)	on (0.99)	on (0.97)
Models	Dataset A	Dataset B	Dataset C	Dataset D
CatBoost	alarm.on (0.94), timer.on (0.94)	reminder.on (0.91)	on (0.96)	on (0.98)
XGBoost	alarm.on (0.97)	alarm.on (0.97)	on (0.94)	on (0.89)
LightGBM	alarm.on (0.97)	reminder.on (0.90)	on (0.96)	on (0.89)
LightGBMLarge	alarm.on (0.90)	alarm.on (0.97)	on (0.94)	on (0.90)
RandomForestGini	alarm.on (0.92)	alarm.on (0.94)	on (0.87)	on (0.96)
RandomForestEntr	alarm.on (0.92)	alarm.on (0.94)	on (0.87)	on (0.96)
KNeighborsDist	alarm.on (0.82)	alarm.on (0.78)	on (0.87)	on (0.74)
KNeighborsUnif	timer.on (0.88)	timer.on (0.75)	on (0.90)	on (0.77)
NeuralNetTorch	timer.on (0.72)	alarm.on (0.55)	off (0.73)	on (0.74)
NeuralNetFastAI	timer.on (0.77)	alarm.on (0.56)	on (0.76)	off (0.69)

Table 5.9: *Continuation: Classes with best Precision result for each dataset by model summary*

Models	Dataset A	Dataset B	Dataset C	Dataset D
XLNet	alarm.off (1.00), timer.on (1.00)	alarm.off (1.00), reminder.on (1.00), timer.on (1.00)	on (1.00)	on (0.98)
Dialogflow	alarm.off (1.00), alarm.on (1.00)	timer.on (1.00), alarm.on (1.00)	off (0.99)	on (0.96), off (0.96)
LUIS	alarm.off (1.00)	alarm.off (0.97), alarm.on (0.97), timer.on (0.97)	off (0.97)	off (0.98)
RoBERTa	alarm.on (1.00), timer.on (1.00)	reminder.off (1.00), reminder.on (1.00), alarm.off (1.00)	on (0.97), off (0.97)	off (0.90)
BERT	alarm.on (1.00), timer.on (1.00)	alarm.on (1.00), reminder.on (1.00), timer.on (1.00)	on (1.00)	on (0.97)
SVM	alarm.off (0.95), alarm.on (0.95)	alarm.off (0.95), alarm.on (0.95)	off (0.95)	off (0.96)
LightGBMXT	alarm.off (0.97)	alarm.off (0.97)	off (0.96)	off (0.90)
ExtraTreesGini	alarm.off (0.97)	alarm.off (0.97)	on (0.89), off (0.89)	off (0.96)
ExtraTreesEntr	alarm.off (0.97)	alarm.off (0.97)	off (0.89)	off (0.96)
WeightedEnsemble L2	alarm.off (0.88), alarm.on (0.88)	alarm.off (0.97)	off (0.96)	off (0.95)

Table 5.10: *Classes with best Recall result for each dataset by model summary*

Models	Dataset A	Dataset B	Dataset C	Dataset D
LSTM	alarm.off (1.00), timer.off (1.00)	alarm.off (1.00), alarm.on (1.00), reminder.off (1.00)	off (0.99)	off (0.97)
CatBoost	alarm.off (0.95), timer.off (0.95)	reminder.off (0.93)	off (0.96)	off (0.98)
XGBoost	alarm.off (0.97)	alarm.off (0.97)	off (0.95)	off (0.90)
LightGBM	alarm.off (0.97)	alarm.off (0.95)	off (0.96)	off (0.90)
LightGBMLarge	alarm.off (0.90)	alarm.off (0.97)	off (0.95)	off (0.92)
RandomForestGini	alarm.off (0.93)	alarm.off (0.95)	off (0.88)	off (0.97)
RandomForestEntr	alarm.off (0.93)	alarm.off (0.95)	off (0.88)	off (0.97)
KNeighborsDist	alarm.on (0.82)	alarm.on (0.80), reminder.off (0.80)	off (0.87)	off (0.78)
KNeighborsUnif	timer.off (0.90)	alarm.on (0.78), reminder.off (0.78)	off (0.93)	off (0.81)
NeuralNetTorch	timer.off (0.70)	alarm.on (0.65)	on (0.74)	off (0.78)
NeuralNetFastAI	timer.on (0.77)	timer.on (0.62)	off (0.76)	on (0.70)

Table 5.11: *Continuation: Classes with best Recall result for each dataset by model summary*

According to the tables above, models of each type are going to be compared to show the differences and similarities concerning results of each dataset. In addition, to review the results detailed by precision, recall and F1, appendix **??** includes the data analysed in depth.

To begin with, regarding models from scratch, it can be observed that both models have obtained similar results in terms of average accuracy, although SVM model has outperformed LSTM model in dataset A and dataset D. That is, regarding dataset A and B, he lesser number of classes for SVM model the better results it obtains. However, for LSTM, the contrary happens. When adding more training, concretely, two more classes "reminder.on" and "reminder.off", results improve in relation to dataset A. Nevertheless, the opposite case occurs for dataset C and D. The greater number of samples classified in the same number of classes "on" and "off", the worse results are obtained by both models. Additionally, the key difference of this case is that LSTM does not assimilate the additional sample added to dataset D (sample reclassified "reminder.on" as "on" and "reminder.off" as "off").

Furthermore, which is interesting to highlight is the the best class for each dataset in both models. Generally, classes coincide in two datasets: A and C. However, when adding more samples to the training and test sets, datasets B and D, results are different in datasets B and D. On the one hand, in dataset B, "alarm.on" coincides as the best class. However, for LSTM, also "alarm.on" works as well as "alarm.off". On the other hand, in dataset D, both classes have obtained the same punctuation in SVM model, while in LSTM model, "off" is minimally better than "on".

That is the reason why, it can be observed that both models work in a very similar way. Nonetheless, when adding more samples that have the same concept but with different keywords, SVM models are able to assimilate this problem appropriately.

Secondly, regarding models based on transformers, which in general are one of the best model types, it can be noticed that there is hardly any difference in terms of average accuracy results for the three models, although it is clear that just XLNet has been able to grasp dataset D. That is, the the samples of the classes that belong to "reminder.on" and "reminder.off".

Furthermore, to check if the best class for each dataset in the three models is the same can be relevant when comparing results. Generally, classes coincide in two datasets: A and C. However, when adding more samples to the training and test sets, datasets B and D, results are different in datasets B and D. On the one hand, in dataset B, "alarm.on" coincides as the best class. However, for LSTM, also "alarm.on" works as well as "alarm.off". On

the other hand, in dataset D, both classes have obtained the same score in SVM model, while in LSTM model, “off” is minimally better than “on”.

Thus, it can be seen that as there is a minimum difference in terms of average accuracy, so there is also a minimum difference in terms of dataset A, B and C because results for the different classes and the three models are quite close, although classes with the best results cannot coincide. Despite this fact, the notable difference between these models occurs in dataset D for XLNet model in which both classes have obtained the same good result, while in the rest of models, just one class has obtained better results than the other.

Thirdly, regarding models developed with AutoGluon, that, clearly, some of them are much better than others in these experiments in terms of average accuracy, it can be noticed that dataset C has obtained the best results in comparison with the rest of datasets for all the models under this category. It is also evident that, dataset B has obtained the worst results in relation to the rest of the datasets regardless of the model used under this type.

As there are many models that have been tested under this category, the summary of the best classes concerning datasets and models are going to be split by subtype. Thus, six different tables will be shown according to the similarities and differences between the algorithms used.

Models based on ExtraTrees have obtained practically the same results. Therefore, it is expected that their best classes for each dataset and model are the same for almost all the cases. In fact, for these cases that are not the same, the result for the class that is classified as the best one is very close to the following best result.

As the two models based on KNeighbors have obtained similar results, it is expected that their best classes for each dataset and model are the same.

As the three models based on LightGBM have obtained similar results, it is expected that their best classes for each dataset and model are similar too,although not completely the same in all cases.

Concerning NeuralNet models, results are very similar. However although for dataset A and B the best classes are the same, for dataset C and D, these algorithms have behaved in a different way and the best classes are the contrary in each case. This means that both models interpret intent classification task for the concepts “on” and “off” in a different way.

Regarding RandomForest models, results in terms of selecting the best

class for each dataset are the same. This explains that general results such as the average accuracy metric obtained by these models is practically the same for all the datasets. Thus, differences are minimum.

Finally, these three models, CatBoost, XGBoost and WeightedEnsemble_L2 have been included in the same group to be analysed in depth due to their similar results concerning average accuracy. Therefore, it is not surprising that the best classes for these three models are practically the same.

Finally, in relation to the models developed with dialogue management tools, it can be observed that both models have obtained similar results in terms of average accuracy, although Dialogflow model has outperformed LUIS model in all the datasets except for dataset D, in which LUIS has obtained a slightly better result.

It can be said that both technologies work in a similar way. Different from the rest of model types, these models work better in dataset C, and between A and B, there is a minimum variance, obtaining B better results than A.

Furthermore, it is relevant to see if the best classes for each dataset and model are also the same. The best classes coincide in all the datasets which means that the differences between these models are minimum.

Chapter 6

Conclusions

In this thesis, two different approaches regarding intent classification have been presented. In fact, both approaches have been tested by using the different types of models shown in the previous section.
At the beginning of this thesis, the following research questions were formulated:

- How can it be determined if a model works better than other? Does it depend on the intents used to train and test?
- What is the best approach for measuring performance?
- How different are the results taking into account two different approaches when classifying? Can an attribute be decisive when classifying?

To try to answer these questions, the intent classification of four different classification model types were evaluated on four different datasets that include different intents. One of the datasets, called dataset A, was created by using logs from Aura with four different intents: “alarm.off”, “alarm.on”,“timer.off” and “timer.on”. Then, dataset B includes the same classes as A but adding two more: “reminder.off” and “reminder.on”. Finally, datasets C and D are the same sample as datasets A and B respectively but being classified in two different classes by concept or attribute: “off” and “on”. The four types of models compared were models developed from scratch, models based on transformers, models developed with AutoGluon and models developed with dialogue management platforms. They have been chosen based on having previously been successfully used for intent classification as well as representing different categories of classifiers.

The results obtained in the experiments presented in this thesis indicate that most of the intent classification models perform quite similarly, that is, they retrieve relatively high classification accuracies. Concretely, transformers, dialogue management platforms and models from scratch perform better than those developed with AutoGluon. Indeed, XLNet and Dialogflow would be better options than the rest taking into account their average dataset results.

Additionally, three more general conclusions can be addressed:

Firstly, out of the classification methods evaluated in this thesis, models based on transformers generally perform the best on intent classification. However, models developed by using dialogue management platforms were not far behind and sometimes even performed better than transformer models. This can be determined when evaluating them using the following metrics: precision, recall, F1 and accuracy.

Secondly, according to the previous conclusions and analysis, dataset C, that is the approach of classifying "on" and "off" with a training and test set sample that include similar use cases such as "alarm.on", "alarm.off", "timer.on" and "timer.off" retrieves better results that the rest of dataset.

Thirdly, when including more samples with other use cases "reminder.on" and "reminder.off" that, a priori, can be conceived as the same concept "on" and "off" but not only the keyword "reminder" is different but the verbs and expressions used for turning on and off a reminder are also different, some classifiers have been not able to interpret that the concept was the same and it has penalised the results in some cases such as in RoBERTa and BERT models.

Finally, to be able to answer with more certitude if an attribute or keyword can be decisive when classifying, further research need to be done along these lines and, for instance, address the potential optimization in classification when including a NER module in the equation.

Chapter 7

Bibliography

Abdellatif, A., Badran, K., Costa, D.E., & Shihab, E. (2021). *A Comparison of Natural Language Understanding Platforms for Chatbots in Software Engineering.* arXiv.org., from *https://arxiv.org/abs/2012.02640*

AutoGluon. (n.d.). *Autogluon Official Documentation*, from *https://auto.gluon.ai/stable/index.html*

Bhadouria, V. S. (2020). *Explaining Accuracy, Precision, Recall, and F1 Score*, from *https://medium.com/swlh/explaining-accuracy-precision-recall-and-f1-score-f29d370caaa8*

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.* arXiv.org., from *https://arxiv.org/abs/1412.3555*

Devlin, J. & Chang, M. (2018). *Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing.*Google AI Blog, from *https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html*

Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A. A., Lally, A., Murdock, J. W., Nyberg, E., Prager, J., Schlaefer, N., & Welty, C. (2010). *Building Watson: An Overview of the DeepQA Project.* AI Magazine.

Google. (n.d.). *Dialogflow ES basics*, from *https://cloud.google.com/dialogflow/es/docs/basics*

Gopalan, B. (2021). *What is Natural Language Processing(NLP)?*, from *https://www.numpyninja.com/post/what-is-natural-language-processing-nlp*

Heeyoung Lee, H., Surdeanu, M. & Jurafsky, D. (2017). *A scaffolding approach to coreference resolution integrating statistical and rule-based models. Natural Language Engineering.*

Jalli, Artturi. (2022). *What Is a Library in Programming? A Complete Guide*, from *https://www.codingem.com/what-is-a-library/*

Jurafsky, D. & Martin, J. H. (2022). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.* Pearson.

Liu, X., Eshghi, A., Swietojanski, P., & Rieser, V. (2021). *Benchmarking natural language understanding services for building conversational agents.* Lecture Notes in Electrical Engineering, 165183, from *https://doi.org/10.1007/978-981-15-9323-9_15*

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). *Roberta: A robustly optimized Bert pretraining approach.* arXiv.org., from *https://arxiv.org/abs/1907.11692*

Merriam-Webster. (n.d.). *Platform. In Merriam-Webster.com dictionary*, from *https://www.merriam-webster.com/dictionary/platform*

Microsoft Corporation. (n.d.). *Welcome to LightGBM's documentation!*, from *https://lightgbm.readthedocs.io/en/latest/*

Microsoft. (n.d.). *What is conversational language understanding?*, from *https://learn.microsoft.com/en-us/azure/cognitive-services/language-service/conversational-language-understanding/overview*

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). *Foundations of Machine Learning.* The MIT Press.

Narkhede, S. (2018). *Understanding Confusion Matrix*, from *https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62*

Rajapakse, T. (2022). *Simple Transformers.Classification Specifics*, from *https://simpletransformers.ai/docs/classification-specifics/supported-model-types*

Scikit-learn. (n.d.). *Scikit-learn Documentation*, from *https://scikit-learn.org/stable/*

Sreelakshmi, K., Rafeeque, P. Ca, & Gayathri, E.S. (2018). *Deep bi-directional LSTM network for query intent detection* Procedia Computer Science, 143, pp. 939_946., from *https://doi.org/10.1016/j.procs.2018.10.341*

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, 15 (56), pp. 1929_1958.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). *Attention is all you need.* arXiv.org., from *https://arxiv.org/abs/1706.03762*

Vilalta, R. (2000). *Evaluation metrics in classification: A quantification of distance-bias.* Yorktown Heights, NY: IBM T.J. Watson Research Center.

XGBoost. (n.d.). *XGBoost Documentation*, from *https://xgboost.readthedocs.io/en/latest/*

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. V. (2020). *XLNet: Generalized autoregressive pretraining for Language Understanding.* arXiv.org., from *https://arxiv.org/abs/1906.08237v2*

Zhou, V. (2019). *A Simple Explanation of Gini Impurity*, from *https://victorzhou.com/blog/gini-impurity/*

Appendix A

Acronyms

AI Artificial Intelligence.
AN Attention Networks.
ASR Automatic Speech Recognition.
BERT Bidirectional Encoder Representations from Transformers.
CNN Convolutional Neural Networks.
FFNN Feed Forward Neural Networks.
GRU Gated Recurrent Units.
IDE Integrated Development Environment.
LSTM Long-Short-Term-Memory.
ML Machine Learning. **NER** Named Entity Recognition.
NLP Natural Language Processing.
NLU Natural Language Understanding.
RNN Recurrent Neural Networks.
ROBERTA Robustly Optimised BERT pre-training Approach.
STT Speech To Text.
SVM Support-Vector Machines Model.
WSD Word Sense Disambiguation.

Appendix B

Analysis of results by model type

This appendix includes the specific information for the analysis of results by model type based on the four datasets used. For each dataset, two tables will be provided: one for the results and another one for the confusion matrix.

- **Models developed from scratch**
  In this section, LSTM model and SVM model results will be shown and described.

  - **LSTM:** In this section, results for LSTM model are going to be described and shown in a table.

    - **Dataset A**
      In dataset A, regarding precision, the class that yielded the best result is "alarm.on" (1.00). However, its recall result is the worst (0.42). In addition, the best recall results have been obtained by "alarm.off" (1.00) and "timer.off" (1.00) classes. Taking into account the F1 metric, the class that yielded the best result is "timer.off" (0.95). These results can be checked in tables C.1 and C.2.

    - **Dataset B**
      In dataset B, regarding precision, the classes with the best results are "alarm.on" (1.00), "alarm.off" (1.00) and "reminder.on" (1.00). In addition, the best recall results ha-

ve been obtained by "alarm.on" (1.00), "alarm.off" (1.00), "timer.off" (1.00) and "reminder.off" (1.00) classes. Taking into account the F1 metric, the classes with best results are "alarm.off" (1.00) and "alarm.on" (1.00). However, the class with the worst F1 result has been "reminder.on" (0.52) although its precision is one of the best (1.00). These results can be checked in tables C.3 and C.4.

- **Dataset C**
  In dataset C, regarding F1, both classes have obtained the same results which is a good score (0.95). However, the best recall result has been obtained by "Off" (0.99) class while the best precision result has been obtained by "On" (0.99) class. These results can be checked in tables C.5 and C.6.

- **Dataset D**
  In dataset D, regarding F1, both classes have obtained similar results (0.86 - 0.82). However, the best recall result has been obtained by "Off" (0.97) class while the best precision result has been obtained by "On" (0.97) class. These results can be checked in tables C.7 and C.8.

- **SVM:** In this section, results for SVM model are going to be described and shown in a table.

  - **Dataset A**
    In dataset A, regarding precision, the class that yielded the best result is "alarm.on" (0.93). In addition, its recall result is one of the best results beside "alarm.off" (0.95). However, the worst recall result has been obtained by "timer.off" (0.82) class. Taking into account the F1 metric, the class that yielded the best result is "alarm.on" (0.94). These results can be checked in tables C.9 and C.10.

  - **Dataset B**
    In dataset B, regarding precision, the class that yielded the

best result is “reminder.on” (0.94). However, the class with less recall result is also “reminder.on” (0.78). On the other hand, the best recall results have been obtained by classes “alarm.off” (0.95) and “alarm.on” (0.95). Taking into account the F1 metric, the class that yielded the best result is “alarm.on” (0.93) class. These results can be checked in tables C.11 and C.12.

- **Dataset C**
  In dataset C, the F1 metric is the same for both classes (0.94). However, while “Off” class has obtained a better result regarding recall (0.95), “On” class has obtained a better result concerning precision (0.95). These results can be checked in tables C.13 and C.14.

- **Dataset D**
  In dataset D, the F1 metric is the same for both classes (0.93). However, while “Off” class has obtained a better result regarding recall (0.96), “On” class has obtained a better result concerning precision (0.96). These results can be checked in tables C.15 and C.16.

- **Models developed with transformers** In this section, transformer models such as BERT, RoBERTa and XLNet will be shown and described.

  - **BERT:** In this section, results for BERT model are going to be described and shown in a table.

    - **Dataset A**
      In dataset A, the best recall results have been obtained by classes “alarm.on” (1.00) and “timer.on” (1.00). However, the best precision results have been obtained by the opposite classes: “alarm.off” (1.00) and “timer.off” (1.00). Finally, F1 metrics are pretty similar but the best result has been obtained by “alarm.on” class (0.98). These results can be

checked in tables C.17 and C.18.

- **Dataset B**
  In dataset B, the best recall results have been obtained by classes "alarm.on" (1.00), "reminder.on" (1.00) and "timer.on" (1.00). However, the best precision results have been obtained by the opposite classes: "alarm.off" (1.00), "reminder.off" (1.00) and "timer.off" (1.00). Finally, F1 metrics are pretty similar but the best results have been obtained by "alarm.on" (0.99) and "alarm.off" (0.99). These results can be checked in tables C.19 and C.20.

- **Dataset C**
  In dataset C, F1 metric results are the same for both classes (0.98). The one difference between both classes is that the recall result is higher in class "On" (1.00), while precision result is higher in "Off" class (1.00). These results can be checked in tables C.21 and C.22.

- **Dataset D**
  In dataset D, "Off" class has obtained better result regarding precision (0.96), while "On" class has obtained better result regarding recall (0.97). However, the best F1 result has been obtained by "On" class (0.89). These results can be checked in tables C.23 and C.24.

- **RoBERTa:** In this section, results for RoBERTa model are going to be described and shown in a table.

  - **Dataset A**
    In dataset A, the best recall results have been obtained by classes "alarm.on" (1.00) and "timer.on" (1.00). However, the best precision result has been obtained by one of the opposite classes: "alarm.off" (1.00). Finally, F1 metrics are pretty similar but the best results have been obtained by classes "alarm.on" (0.99) and "timer.on" (0.99). These re-

sults can be checked in tables C.25 and C.26.

- **Dataset B**
  In dataset B, the best recall results have been obtained by classes "alarm.off" (1.00), "reminder.off" (1.00) and "reminder.on" (1.00). However, the best precision results have been obtained by the following classes: "alarm.on" (1.00), "reminder.off" (1.00) and "reminder.on" (1.00). Finally, the best F1 results have been obtained by classes "reminder.off" (1.00) and "reminder.on" (1.00). These results can be checked in tables C.27 and C.28.

- **Dataset C**
  In dataset C, both classes have obtained the same results concerning recall (0.97), precision (0.97) and F1 (0.97). These results can be checked in tables C.29 and C.30.

- **Dataset D**
  In dataset C, both classes have obtained pretty similar results (0.89 - 0.88). However, "Off" class is slightly better concerning recall (0.90) and "On" class is slightly better regarding precision (0.90). These results can be checked in tables C.31 and C.32.

- **XLNet:** In this section, results for XLNet model are going to be described and shown in a table.

  - **Dataset A**
    In dataset A, the best recall results have been obtained by classes "alarm.off" (1.00) and "timer.on" (1.00). However, the best precision result has been obtained by "alarm.on" (1.00) and "timer.off" (1.00) classes. Finally, F1 metrics are pretty similar but the best results have been obtained by classes "alarm.on" (0.99) and "alarm.off" (0.99). These results can be checked in tables C.33 and C.34.

- **Dataset B**
  In dataset B, the best recall results have been obtained by classes "alarm.off" (1.00), "reminder.on" (1.00) and "timer.on" (1.00). However, the best precision result has been obtained by "alarm.on" (1.00) and "timer.off" (1.00) classes. Finally, F1 metrics are pretty similar but the best results have been obtained by classes "alarm.on" (0.99) and "alarm.off" (0.99). These results can be checked in tables C.35 and C.36.

- **Dataset C**
  In dataset C, the F1 result is the same for both classes (0.97). The one difference between these classes is that "Off" class is better regarding precision result (1.00), while "On" class is better regarding recall result (1.00). These results can be checked in tables C.37 and C.38.

- **Dataset D**
  In dataset D, the F1 result is the same for both classes (0.94). The one difference between these classes is that "Off" class is better regarding precision result (0.98), while "On" class is better regarding recall result (0.98). These results can be checked in tables C.39 and C.40.

- **Models developed with AutoGluon** In this section, results of the different models generated by AutoGluon will be shown and described.

  - **KNeighborsUnif:** In this section, results for KNeighborsUnif model are going to be described and shown in a table.

    - **Dataset A**
      In dataset A, regarding precision, the class that yielded the best result is "timer.on" (0.88). However, its recall result is the worst (0.57). In addition, the best recall result has been obtained by "timer.off" (0.90). Taking into account the F1 metric, the class that yielded the best result is "alarm.off"

(0.77). These results can be checked in tables C.41 and C.42.

- **Dataset B**
  In dataset B, regarding precision, the classes with the best results are "timer.on" (0.75) and "alarm.on" (0.72) classes. In addition, the best recall results have been obtained by "alarm.on" (0.78), "reminder.off" (0.78) and "alarm.off" (0.72) classes. Taking into account the F1 metric, the classes with best results are "alarm.on" (0.75) and "reminder.off" (0.73). These results can be checked in tables C.43 and C.44.

- **Dataset C**
  In dataset C, regarding F1, the class that yielded the best result is "Off" (0.83). However, the worst precision result is also obtained by "Off" (0.75) while the best precision result has been obtained by "On" (0.90). These results can be checked in tables C.45 and C.46.

- **Dataset D**
  In dataset D, regarding F1, "Off" class has obtained better result (0.74). However, the best precision result has been obtained by "On" (0.77) while the best recall result has been obtained by "Off" (0.81). These results can be checked in tables C.47 and C.48.

- **KNeighborsDist:** In this section, results for KNeighborsDist model are going to be described and shown in a table.

  - **Dataset A**
    In dataset A, regarding precision, the class that yielded the best result (0.82) is "alarm.on". In addition, the best recall results have been obtained by classes "timer.off" (0.82) and "alarm.off" (0.82). Taking into account the F1 metric, the class that yielded the best result is "alarm.off" (0.80)

class. These results can be checked in tables C.49 and C.50.

- **Dataset B**
  In dataset B, regarding precision, the class with the best result is "alarm.on" (0.78) class. In addition, the best recall results have been obtained by "alarm.on" (0.80) and "reminder.off" (0.80) classes. Taking into account the F1 metric, the class that yielded the best result is "alarm.on" (0.79 class. These results can be checked in tables C.51 and C.52.

- **Dataset C**
  In dataset C, regarding F1, the class that yielded the best result is "Off" (0.82) class. However, the worst recall result (0.72) has been obtained by "On" while the best precision result (0.87) has been obtained by "On". These results can be checked in tables C.53 and C.54.

- **Dataset D**
  In dataset D, regarding F1, "Off" class has obtained better result (0.73). However, the best precision result (0.74) has been obtained by "On" class while the best recall result (0.78) has been obtained by "Off" class. These results can be checked in tables C.55 and C.56.

- **LightGBMXT:** In this section, results for LightGBMXT model are going to be described and shown in a table.

  - **Dataset A**
    In dataset A, regarding precision, the class that yielded the best result is "alarm.on" (0.97). In addition, the best recall results have been obtained by classes "timer.off" (0.95) and "alarm.off" (0.97). Taking into account the F1 metric, the class that yielded the best result is "alarm.off" (0.92). These results can be checked in tables C.57 and C.58.

- **Dataset B**
  In dataset B, regarding precision, the class with the best result (0.97) has been obtained by "alarm.on" class. In addition, the best recall result (0.97) has been obtained by "alarm.off" class. Taking into account the F1 metric, the class that yielded the best result is "alarm.off" (0.93). These results can be checked in tables C.59 and C.60.

- **Dataset C**
  In dataset C, regarding F1, the class that yielded the best result is "Off" (0.90). However, the worst recall result (0.81) has been obtained by "On" class while the best precision result (0.96) has been obtained by "On" class. These results can be checked in tables C.61 and C.62.

- **Dataset D**
  In dataset D, regarding F1, both classes have obtained the same results (0.87). However, the best precision result (0.89) has been obtained by "On" class while the best recall result (0.90) has been obtained by "Off" class. These results can be checked in tables C.63 and C.64.

- **LightGBM:** In this section, results for LightGBM model are going to be described and shown in a table.

  - **Dataset A**
    In dataset A, regarding precision, the class with the best result is (0.97) "alarm.on". In addition, the best recall results have been obtained by classes "timer.off" (0.90) and "alarm.off" (0.97). Taking into account the F1 metric, the class that yielded the best result is "alarm.off" (0.93) class. These results can be checked in tables C.65 and C.66.

  - **Dataset B**
    In dataset B, regarding precision, the class with the best result (0.95) has been obtained by "alarm.on" class. In ad-

dition, the best recall result (0.95) has been obtained by "alarm.off" class. Taking into account the F1 metric, the class that yielded the best result is "alarm.off" (0.92). These results can be checked in tables C.67 and C.68.

- **Dataset C**
  In dataset C, regarding F1, the class with the best result is "Off" (0.89). However, the worst recall result (0.80) has been obtained by "Off" class while the best precision result (0.96) has been obtained by "On" class. These results can be checked in tables C.69 and C.70.

- **Dataset D**
  In dataset D, regarding F1, class "Off" has obtained better result (0.86) than "On" (0.82). However, the best precision result (0.89) has been obtained by "On" class while the best recall result (0.90) has been obtained by "Off" class. These results can be checked in tables C.71 and C.72.

- **RandomForestGini:** In this section, results for RandomForestGini model are going to be described and shown in a table.

  - **Dataset A**
    In dataset A, regarding precision, the class that yielded the best result is "alarm.on" (0.92). In addition, the best recall result (0.93) has been obtained by class "alarm.off". Taking into account the F1 metric, the class that yielded the best result is "alarm.off" (0.93). These results can be checked in tables C.73 and C.74.

  - **Dataset B**
    In dataset B, regarding precision, the class with the best result (0.94) has been obtained by "alarm.on" class. In addition, the best recall result (0.95) has been obtained by "alarm.off" class. Taking into account the F1 metric, the class that yielded the best result is "alarm.off" (0.90). The-

se results can be checked in tables C.75 and C.76.

- **Dataset C**
  In dataset C, regarding F1, both classes have obtained the same result (0.86). However, the worst recall result (0.85) has been also obtained by "Off" class while the best precision result (0.87) has been obtained by "On" class. These results can be checked in tables C.77 and C.78.

- **Dataset D**
  In dataset D, regarding F1, class "Off" has obtained a better result (0.87) than "On" (0.84) class. However, the best precision result (0.96) has been obtained by "On" class while the best recall result (0.97) has been obtained by "Off" class. These results can be checked in tables C.79 and C.80.

- **RandomForestEntr:** In this section, results for RandomForestEntr model are going to be described and shown in a table.

  - **Dataset A**
    In dataset A, regarding precision, the class that yielded the best result is "alarm.on" (0.92). In addition, the best recall result (0.88) has been obtained by class "alarm.off". Taking into account the F1 metric, the class that yielded the best results are "alarm.off" (0.90) and "alarm.on" (0.90) classes. These results can be checked in tables C.81 and C.82.

  - **Dataset B**
    In dataset B, regarding precision, the class with the best result is "alarm.on". In addition, the best recall result (0.95) has been obtained by "alarm.off". Taking into account the F1 metric, the class that yielded the best results are "alarm.off" (0.89) and "alarm.on" (0.88) classes. These results can be checked in tables C.83 and C.84.

- **Dataset C**
  In dataset C, regarding F1, both classes have obtained the same result (0.86). However, the worst recall result (0.85) has been also obtained by "Off" class while the best precision result has been obtained by "On" (0.87) class. These results can be checked in tables C.85 and C.86.

- **Dataset D**
  In dataset D, regarding F1, class "Off" has obtained better result (0.88) than "On" class (0.85). However, the best precision result has been obtained by "On" class (0.96) while the best recall result has been obtained by "Off" class (0.97). These results can be checked in tables C.87 and C.88.

- **CatBoost:** In this section, results for CatBoost model are going to be described and shown in a table.

  - **Dataset A**
    In dataset A, regarding precision, the class that yielded the best results are "alarm.on" (0.94) and "timer.on" (0.94). In addition, the best recall result have been obtained by classes "alarm.off" (0.95) and "timer.off" (0.95). Taking into account the F1 metric, the class that yielded the best result is "alarm.off" (0.90). These results can be checked in tables C.89 and C.90.

  - **Dataset B**
    In dataset B, regarding precision, the class with the best result is "reminder.on" (0.91). In addition, the best recall result (0.93) has been obtained by "reminder.off" class. Taking into account the F1 metric, the class that yielded the best result is "alarm.on" (0.88). These results can be checked in tables C.91 and C.92.

  - **Dataset C**

In dataset C, regarding F1, class "Off" has obtained better result (0.90) than class "On" (0.88). However, the best recall result (0.96) has been obtained by "Off" class while the best precision result (0.96) has been obtained by "On" class. These results can be checked in tables C.93 and C.94.

- **Dataset D**
  In dataset D, regarding F1, class "Off" have obtained better results (0.87) than "On" (0.83). However, the best precision result (0.98) has been obtained by "On" class while the best recall result (0.98) has been obtained by "Off" class. These results can be checked in tables C.95 and C.96.

- **ExtraTreesGini:** In this section, results for ExtraTreesGini model are going to be described and shown in a table.

  - **Dataset A**
    In dataset A, regarding precision, the class that yielded the best result is "alarm.on" (0.97). In addition, the best recall result (0.97) has been obtained by class "alarm.off". Taking into account the F1 metric, the class that yielded the best result is "alarm.off" (0.93). These results can be checked in tables C.97 and C.98.

  - **Dataset B**
    In dataset B, regarding precision, the class with the best result is "alarm.on" (0.97). In addition, the best recall result (0.97) has been obtained by "alarm.off" class . Taking into account the F1 metric, the class that yielded the best result is "alarm.off" (0.93). These results can be checked in tables C.99 and C.100.

  - **Dataset C**
    In dataset C, regarding F1, both classes have obtained the same result (0.89). However, the best recall result (0.89) has been obtained by "Off" class while the best precision result

has been obtained by "On" class (0.89). These results can be checked in tables C.101 and C.102.

- **Dataset D**
  In dataset D, regarding F1, class "Off" has obtained better result (0.87) than "On" class (0.84). However, the best precision result (0.95) has been obtained by "On" class while the best recall result (0.96) has been obtained by "Off" class. These results can be checked in tables C.103 and C.104.

- **ExtraTreesEntr:** In this section, results for ExtraTreesEntr model are going to be described and shown in a table.

  - **Dataset A**
    In dataset A, regarding precision, the class that yielded the best result is "alarm.off" (0.97). In addition, the best recall result has been obtained by class "alarm.off" (0.97). Taking into account the F1 metric, the class that yielded the best result is "alarm.off" (0.92). These results can be checked in tables C.105 and C.106.

  - **Dataset B**
    In dataset B, regarding precision, the class with the best result is "alarm.on" (0.97). In addition, the best recall result (0.97) has been obtained by "alarm.off". Taking into account the F1 metric, the classes with best results are "alarm.off" (0.94) and "alarm.on" (0.94). These results can be checked in tables C.107 and C.108.

  - **Dataset C**
    In dataset C, regarding F1, "Off" class has obtained a better result (0.88) than "On" class (0.87). However, the best recall result (0.89) has been obtained by "Off" class while the best precision result (0.88) has been obtained by "On" class. These results can be checked in tables C.109 and

C.110.

- **Dataset D**
  In dataset D, regarding F1, class "Off" has obtained better result (0.87) than "On" class (0.85). However, the best precision result (0.95) has been obtained by "On" class while the best recall result (0.96) has been obtained by "Off" class. These results can be checked in tables C.111 and C.112.

- **NeuralNetFastAI:** In this section, results for NeuralNetFastAI model are going to be described and shown in a table.

  - **Dataset A**
    In dataset A, regarding precision, the class that yielded the best result is "timer.on" (0.77). In addition, the best recall result (0.72) has been obtained by class "alarm.on". Taking into account the F1 metric, the class that yielded the best result is "alarm.on" (0.67). These results can be checked in tables C.113 and C.114.

  - **Dataset B**
    In dataset B, regarding precision, the class with the best result (0.56) has been obtained by "alarm.on". In addition, the best result (0.62) has been obtained by "timer.on". Taking into account the F1 metric, the class that yielded the best result is "timer.on" (0.57). These results can be checked in tables C.115 and C.116.

  - **Dataset C**
    In dataset C, regarding F1, both classes have obtained practically the same result. The difference is minimum (0.76 - 0.75). However, the best recall result (0.76) has been obtained by "Off" class while the best precision result (0.76) has been obtained by "On" class. These results can be checked in tables C.117 and C.118.

- **Dataset D**
  In dataset D, regarding F1, both classes have obtained practically the same result. The difference is minimum (0.68 - 0.69). However, the best precision result (0.69) has been obtained by "Off" class while the best recall result (0.70) has been obtained by "On" class. These results can be checked in tables C.119 and C.120.

- **XGBoost:** In this section, results for XGBoost model are going to be described and shown in a table.

  - **Dataset A**
    In dataset A, regarding precision, the class that yielded the best result is "alarm.on" (0.97). In addition, the best recall result (0.97) has been obtained by class "alarm.off". Taking into account the F1 metric, the class that yielded the best result is "alarm.off" (0.91). These results can be checked in tables C.121 and C.122.

  - **Dataset B**
    In dataset B, regarding precision, the class with the best result (0.97) is "alarm.on". In addition, the best recall result (0.97) is "alarm.off". Taking into account the F1 metric, the classes with best results are "alarm.off" (0.95) and "alarm.on" (0.95). These results can be checked in tables C.123 and C.124.

  - **Dataset C**
    In dataset C, regarding F1, the class that yielded the best result is "Off" (0.88). However, the best recall result (0.95) has been obtained by "Off" class while the best precision result (0.94) has been obtained by "On" class. These results can be checked in tables C.125 and C.126.

- **Dataset D**
  In dataset D, regarding F1, both classes have obtained practically the same result. The difference is minimum (0.86 - 0.85). However, the best precision result (0.89) has been obtained by "On" class while the best recall result (0.90) has been obtained by "Off" class. These results can be checked in tables C.127 and C.128.

- **NeuralNetTorch:** In this section, results for NeuralNetTorch model are going to be described and shown in a table.

  - **Dataset A**
    In dataset A, regarding precision, the class that yielded the best result is "timer.on" (0.72). In addition, the best recall result has been obtained by class "timer.off" (0.70). Taking into account the F1 metric, the class that yielded the best result is "timer.on" (0.68). These results can be checked in tables C.129 and C.130.

  - **Dataset B**
    In dataset B, regarding precision, the class with the best result is "alarm.on" (0.55). In addition, the best recall result is "alarm.on" (0.65). Taking into account the F1 metric, the class that yielded the best result is "alarm.on" (0.60). These results can be checked in tables C.131 and C.132.

  - **Dataset C**
    In dataset C, regarding F1, the class that yielded the best result is "On" (0.73). However, the best recall result (0.74) has been obtained by "On" class while the best precision result (0.73) has been obtained by "Off" class. These results can be checked in tables C.133 and C.134.

  - **Dataset D**
    In dataset D, regarding F1, the "Off" class has obtained a better result (0.73) than "On" class (0.69). However, the

best precision result (0.74) has been obtained by "On" class while the best recall result (0.78) has been obtained by "Off" class. These results can be checked in tables C.135 and C.136.

- **LightGBMLarge:** In this section, results for LightGBMLarge model are going to be described and shown in a table.
  - **Dataset A**
    In dataset A, regarding precision, the class that yielded the best result is "alarm.on" (0.90). In addition, the best recall result has been obtained by class "alarm.off" (0.90). Taking into account the F1 metric, the classes with best results are "alarm.off" (0.89) and "alarm.on" (0.89). These results can be checked in tables C.137 and C.138.
  - **Dataset B**
    In dataset B, regarding precision, the class with the best result is "alarm.on" (0.97). In addition, the best recall result is "alarm.off" (0.97). Taking into account the F1 metric, the classes with best results are "alarm.on" (0.94) and "alarm.off" (0.94). These results can be checked in tables C.139 and C.140.
  - **Dataset C**
    In dataset C, regarding F1, the class that yielded the best result is "Off" (0.88). However, the best recall result (0.95) has been obtained by "Off" class while the best precision result (0.94) has been obtained by "On" class. These results can be checked in tables C.141 and C.142.
  - **Dataset D**
    In dataset D, regarding F1, the "Off" class result (0.86) is better than "On" class result (0.84). However, the best precision result (0.90) has been obtained by "On" class while the best recall result (0.92) has been obtained by "Off"

class. These results can be checked in tables C.143 and C.144.

- **WeightedEnsemble_L2:** In this section, results for WeightedEnsemble_L2 model are going to be described and shown in a table.
  - **Dataset A**
    In dataset A, regarding precision, the classes with best results are "alarm.on" (0.88) and "alarm.off" (0.88). In addition, the best recall results have been obtained by classes "alarm.off" (0.88) and "alarm.on" (0.88). Taking into account the F1 metric, the classes with best results are "alarm.off" (0.88) and "alarm.on" (0.88). These results can be checked in tables
  - **LUIS:** In this section, results for LUIS model are going to be described and shown in a table.
    - **Dataset A**
      In dataset A, regarding precision, the class that yielded the best result is "alarm.on" (1.00). However, the best recall result has been obtained by "alarm.off" (1.00). Taking into account the F1 metric, the class that yielded the best results are "alarm.on" (0.99) and "alarm.off" (0.99). These results can be checked in tables C.161 and C.162.

    - **Dataset B**
      In dataset B, regarding precision, the classes with best results are "alarm.on" (0.97) and "alarm.off" (0.97). However, the classes with best recall results are also "alarm.on" (0.97), "alarm.off" (0.97), beside "timer.on" (0.97). Taking into account the F1 metric, the class that yielded the best results have been obtained by "alarm.on" (0.97) and "alarm.off" (0.97). These results can be checked in tables C.163 and C.164.

◇ **Dataset C**

In dataset C, the F1 metric is the same for both classes (0.96). However, while "Off" class has obtained a better result regarding recall (0.97), "On" class has obtained a better result concerning precision (0.97). These results can be checked in tables C.165 and C.166.

◇ **Dataset D**

In dataset D, the F1 metric is the same for both classes (0.97). However, while "Off" class has obtained a better result regarding recall (0.98), "On" class has obtained a better result concerning precision (0.98). These results can be checked in tables C.167 and C.168.

Appendix C

Tables

Class	Precision	Recall	F1	Sample
alarm.off	0.63	1.00	0.78	40
alarm.on	1.00	0.42	0.60	40
timer.off	0.91	1.00	0.95	40
timer.on	0.97	0.90	0.94	40

Table C.1: *Precision, recall and F1 by class for LSTM model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	40	0	0	0
alarm.on	22	17	0	1
timer.off	0	0	40	0
timer.on	0	0	4	36

Table C.2: *Confusion matrix for LSTM model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	1.00	1.00	1.00	40
alarm.on	1.00	1.00	1.00	40
reminder.off	0.62	1.00	0.76	40
reminder.on	1.00	0.35	0.52	40
timer.off	0.87	1.00	0.93	40
timer.on	0.97	0.88	0.92	40

Table C.3: *Precision, recall and F1 by class for LSTM model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	40	0	0	0	0	0
alarm.on	0	40	0	0	0	0
reminder.off	0	0	40	0	0	0
reminder.on	0	0	5	35	0	0
timer.off	0	0	0	0	40	0
timer.on	0	0	0	1	25	14

Table C.4: *Confusion matrix for LSTM model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.91	0.99	0.95	80
On	0.99	0.91	0.95	80

Table C.5: *Precision, recall and F1 by class for LSTM model and Dataset C*

	Off	On
Off	79	1
On	7	73

Table C.6: *Confusion matrix for LSTM model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.77	0.97	0.86	120
On	0.97	0.72	0.82	120

Table C.7: *Precision, recall and F1 by class for LSTM model and Dataset D*

	Off	On
Off	117	3
On	34	86

Table C.8: *Confusion matrix for LSTM model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.88	0.95	0.92	40
alarm.on	0.93	0.95	0.94	40
timer.off	0.89	0.82	0.86	40
timer.on	0.88	0.88	0.88	40

Table C.9: *Precision, recall and F1 by class for SVM model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	38	2	0	0
alarm.on	2	38	0	0
timer.off	2	0	33	5
timer.on	1	1	3	35

Table C.10: *Confusion matrix for SVM model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.81	0.95	0.87	40
alarm.on	0.90	0.95	0.93	40
reminder.off	0.88	0.88	0.88	40
reminder.on	0.94	0.78	0.85	40
timer.off	0.92	0.82	0.87	40
timer.on	0.81	0.88	0.84	40

Table C.11: *Precision, recall and F1 by class for SVM model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	38	2	0	0	0	0
alarm.on	2	38	0	0	0	0
reminder.off	2	0	33	5	0	0
reminder.on	1	1	3	35	0	0
timer.off	4	0	0	0	35	1
timer.on	0	1	0	3	5	31

Table C.12: *Confusion matrix for SVM model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.93	0.95	0.94	80
On	0.95	0.93	0.94	80

Table C.13: *Precision, recall and F1 by class for SVM model and Dataset C*

	Off	On
Off	76	4
On	6	74

Table C.14: *Confusion matrix for SVM model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.91	0.96	0.93	120
On	0.96	0.90	0.93	120

Table C.15: *Precision, recall and F1 by class for SVM model and Dataset D*

	Off	On
Off	115	5
On	12	118

Table C.16: *Confusion matrix for SVM model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	1.00	0.95	0.97	40
alarm.on	0.95	1.00	0.98	40
timer.off	1.00	0.93	0.96	40
timer.on	0.93	1.00	0.96	40

Table C.17: *Precision, recall and F1 by class for BERT model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	38	2	0	0
alarm.on	40	0	0	0
timer.off	0	0	37	3
timer.on	0	0	0	40

Table C.18: *Confusion matrix for BERT model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	1.00	0.97	0.99	40
alarm.on	0.98	1.00	0.99	40
reminder.off	1.00	0.95	0.97	40
reminder.on	0.95	1.00	0.98	40
timer.off	1.00	0.93	0.96	40
timer.on	0.93	1.00	0.96	40

Table C.19: *Precision, recall and F1 by class for BERT model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	39	1	0	0	0	0
alarm.on	0	40	0	0	0	0
reminder.off	0	0	38	2	0	0
reminder.on	0	0	0	40	0	0
timer.off	0	0	0	0	37	3
timer.on	0	0	0	0	0	40

Table C.20: *Confusion matrix for BERT model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	1.00	0.96	0.98	80
On	0.96	1.00	0.98	80

Table C.21: *Precision, recall and F1 by class for BERT model and Dataset C*

	Off	On
Off	77	3
On	0	80

Table C.22: *Confusion matrix for BERT model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.96	0.78	0.86	120
On	0.82	0.97	0.89	120

Table C.23: *Precision, recall and F1 by class for BERT model and Dataset D*

	Off	On
Off	94	26
On	4	116

Table C.24: *Confusion matrix for BERT model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	1.00	0.95	0.97	40
alarm.on	0.98	1.00	0.99	40
timer.off	0.97	0.97	0.97	40
timer.on	0.98	1.00	0.99	40

Table C.25: *Precision, recall and F1 by class for RoBERTa model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	38	1	1	0
alarm.on	0	40	0	0
timer.off	0	0	39	1
timer.on	0	0	0	40

Table C.26: *Confusion matrix for RoBERTa model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.98	1.00	0.99	40
alarm.on	1.00	0.97	0.99	40
reminder.off	1.00	1.00	1.00	40
reminder.on	1.00	1.00	1.00	40
timer.off	0.91	0.97	0.94	40
timer.on	0.97	0.90	0.94	40

Table C.27: *Precision, recall and F1 by class for RoBERTa model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	40	0	0	0	0	0
alarm.on	1	39	0	0	0	0
reminder.off	0	0	40	0	0	0
reminder.on	0	0	0	40	0	0
timer.off	0	0	0	0	39	1
timer.on	0	0	0	0	4	36

Table C.28: *Confusion matrix for RoBERTa model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.97	0.97	0.97	80
On	0.97	0.97	0.97	80

Table C.29: *Precision, recall and F1 by class for RoBERTa model and Dataset C*

	Off	On
Off	78	2
On	2	78

Table C.30: *Confusion matrix for RoBERTa model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.87	0.90	0.89	120
On	0.90	0.87	0.88	120

Table C.31: *Precision, recall and F1 by class for RoBERTa model and Dataset D*

	Off	On
Off	108	12
On	16	104

Table C.32: *Confusion matrix for RoBERTa model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.98	1.00	0.99	40
alarm.on	1.00	0.97	0.99	40
timer.off	1.00	0.95	0.97	40
timer.on	0.95	1.00	0.98	40

Table C.33: *Precision, recall and F1 by class for XLNet model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	40	0	0	0
alarm.on	1	39	0	0
timer.off	0	0	38	2
timer.on	0	0	0	40

Table C.34: *Confusion matrix for XLNet model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.98	1.00	0.99	40
alarm.on	1.00	0.97	0.99	40
reminder.off	1.00	0.93	0.96	40
reminder.on	0.93	1.00	0.96	40
timer.off	1.00	0.95	0.97	40
timer.on	0.95	1.00	0.98	40

Table C.35: *Precision, recall and F1 by class for XLNet model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	40	0	0	0	0	0
alarm.on	1	39	0	0	0	0
reminder.off	0	0	37	3	0	0
reminder.on	0	0	0	40	0	0
timer.off	0	0	0	0	38	2
timer.on	0	0	0	0	0	40

Table C.36: *Confusion matrix for XLNet model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	1.00	0.94	0.97	80
On	0.94	1.00	0.97	80

Table C.37: *Precision, recall and F1 by class for XLNet model and Dataset C*

	Off	On
Off	75	5
On	0	80

Table C.38: *Confusion matrix for XLNet model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.98	0.90	0.94	120
On	0.91	0.98	0.94	120

Table C.39: *Precision, recall and F1 by class for XLNet model and Dataset D*

	Off	On
Off	108	12
On	2	118

Table C.40: *Confusion matrix for XLNet model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.72	0.82	0.77	40
alarm.on	0.82	0.68	0.74	40
timer.off	0.65	0.90	0.76	40
timer.on	0.88	0.57	0.70	40

Table C.41: *Precision, recall and F1 by class for KNeighborsUnif model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	33	4	3	0
alarm.on	12	27	0	1
timer.off	1	1	36	2
timer.on	0	1	16	23

Table C.42: *Confusion matrix for KNeighborsUnif model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.66	0.72	0.69	40
alarm.on	0.72	0.78	0.75	40
reminder.off	0.69	0.78	0.73	40
reminder.on	0.61	0.57	0.59	40
timer.off	0.60	0.62	0.61	40
timer.on	0.75	0.53	0.62	40

Table C.43: *Precision, recall and F1 by class for KNeighborsUnif model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	29	9	0	1	1	0
alarm.on	6	31	0	0	2	1
reminder.off	2	0	31	6	0	1
reminder.on	2	2	11	23	0	2
timer.off	5	0	1	6	25	3
timer.on	0	1	2	2	14	21

Table C.44: *Confusion matrix for KNeighborsUnif model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.75	0.93	0.83	80
On	0.90	0.69	0.78	80

Table C.45: *Precision, recall and F1 by class for KNeighborsUnif model and Dataset C*

	Off	On
Off	74	6
On	25	55

Table C.46: *Confusion matrix for KNeighborsUnif model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.69	0.81	0.74	120
On	0.77	0.63	0.69	120

Table C.47: *Precision, recall and F1 by class for KNeighborsUnif model and Dataset D*

	Off	On
Off	97	23
On	44	76

Table C.48: *Confusion matrix for KNeighborsUnif model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.77	0.82	0.80	40
alarm.on	0.82	0.70	0.76	40
timer.off	0.65	0.82	0.73	40
timer.on	0.78	0.62	0.69	40

Table C.49: *Precision, recall and F1 by class for KNeighborsDist model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	33	4	3	0
alarm.on	9	28	1	2
timer.off	1	1	33	5
timer.on	0	1	14	25

Table C.50: *Confusion matrix for KNeighborsDist model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.71	0.72	0.72	40
alarm.on	0.78	0.80	0.79	40
reminder.off	0.71	0.80	0.75	40
reminder.on	0.61	0.62	0.62	40
timer.off	0.62	0.60	0.61	40
timer.on	0.70	0.57	0.63	40

Table C.51: *Precision, recall and F1 by class for KNeighborsDist model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	29	8	0	2	1	0
alarm.on	5	32	0	0	2	1
reminder.off	1	0	32	6	0	1
reminder.on	2	0	11	25	0	2
timer.off	4	0	0	6	24	6
timer.on	0	1	2	2	12	23

Table C.52: *Confusion matrix for KNeighborsDist model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.76	0.89	0.82	80
On	0.87	0.72	0.79	80

Table C.53: *Precision, recall and F1 by class for KNeighborsDist model and Dataset C*

	Off	On
Off	71	9
On	22	58

Table C.54: *Confusion matrix for KNeighborsDist model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.69	0.78	0.73	120
On	0.74	0.65	0.69	120

Table C.55: *Precision, recall and F1 by class for KNeighborsDist model and Dataset D*

	Off	On
Off	93	27
On	42	78

Table C.56: *Confusion matrix for KNeighborsDist model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.87	0.97	0.92	40
alarm.on	0.97	0.85	0.91	40
timer.off	0.81	0.95	0.87	40
timer.on	0.94	0.78	0.85	40

Table C.57: *Precision, recall and F1 by class for LightGBMXT model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	39	1	0	0
alarm.on	6	34	0	0
timer.off	0	0	38	2
timer.on	0	0	9	31

Table C.58: *Confusion matrix for LightGBMXT model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.89	0.97	0.93	40
alarm.on	0.97	0.88	0.92	40
reminder.off	0.77	0.90	0.83	40
reminder.on	0.89	0.82	0.86	40
timer.off	0.79	0.82	0.80	40
timer.on	0.91	0.78	0.84	40

Table C.59: *Precision, recall and F1 by class for LightGBMXT model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	39	1	0	0	0	0
alarm.on	5	35	0	0	0	0
reminder.off	0	0	36	4	0	0
reminder.on	0	0	7	33	0	0
timer.off	0	0	4	0	33	3
timer.on	0	0	0	0	9	31

Table C.60: *Confusion matrix for LightGBMXT model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.84	0.96	0.90	80
On	0.96	0.81	0.88	80

Table C.61: *Precision, recall and F1 by class for LightGBMXT model and Dataset C*

	Off	On
Off	77	3
On	15	65

Table C.62: *Confusion matrix for LightGBMXT model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.85	0.90	0.87	120
On	0.89	0.84	0.87	120

Table C.63: *Precision, recall and F1 by class for LightGBMXT model and Dataset D*

	Off	On
Off	108	12
On	19	101

Table C.64: *Confusion matrix for LightGBMXT model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.89	0.97	0.93	40
alarm.on	0.97	0.88	0.92	40
timer.off	0.80	0.90	0.85	40
timer.on	0.89	0.78	0.83	40

Table C.65: *Precision, recall and F1 by class for LightGBM model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	39	1	0	0
alarm.on	5	35	0	0
timer.off	0	0	36	4
timer.on	0	0	9	31

Table C.66: *Confusion matrix for LightGBM model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.88	0.95	0.92	40
alarm.on	0.95	0.88	0.91	40
reminder.off	0.73	0.93	0.81	40
reminder.on	0.90	0.68	0.77	40
timer.off	0.76	0.70	0.73	40
timer.on	0.74	0.78	0.76	40

Table C.67: *Precision, recall and F1 by class for LightGBM model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	38	2	0	0	0	0
alarm.on	5	35	0	0	0	0
reminder.off	0	0	37	3	0	0
reminder.on	0	0	13	27	0	0
timer.off	0	0	1	0	28	11
timer.on	0	0	0	0	9	31

Table C.68: *Confusion matrix for LightGBM model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.83	0.96	0.89	80
On	0.96	0.80	0.87	80

Table C.69: *Precision, recall and F1 by class for LightGBM model and Dataset C*

	Off	On
Off	77	3
On	16	64

Table C.70: *Confusion matrix for LightGBM model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.82	0.90	0.86	120
On	0.89	0.80	0.84	120

Table C.71: *Precision, recall and F1 by class for LightGBM model and Dataset D*

	Off	On
Off	108	12
On	24	96

Table C.72: *Confusion matrix for LightGBM model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.88	0.93	0.90	40
alarm.on	0.92	0.88	0.90	40
timer.off	0.82	0.80	0.81	40
timer.on	0.80	0.82	0.81	40

Table C.73: *Precision, recall and F1 by class for RandomForestGini model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	37	3	0	0
alarm.on	5	35	0	0
timer.off	0	0	32	8
timer.on	0	0	7	33

Table C.74: *Confusion matrix for RandomForestGini model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.86	0.95	0.90	40
alarm.on	0.94	0.85	0.89	40
reminder.off	0.81	0.88	0.84	40
reminder.on	0.86	0.80	0.83	40
timer.off	0.81	0.72	0.76	40
timer.on	0.75	0.82	0.79	40

Table C.75: *Precision, recall and F1 by class for RandomForestGini model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	38	2	0	0	0	0
alarm.on	6	34	0	0	0	0
reminder.off	0	0	35	5	0	0
reminder.on	0	0	8	32	0	0
timer.off	0	0	0	0	29	11
timer.on	0	0	0	0	7	33

Table C.76: *Confusion matrix for RandomForestGini model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.85	0.88	0.86	80
On	0.87	0.85	0.86	80

Table C.77: *Precision, recall and F1 by class for RandomForestGini model and Dataset C*

	Off	On
Off	70	10
On	12	68

Table C.78: *Confusion matrix for RandomForestGini model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.79	0.97	0.87	120
On	0.96	0.74	0.84	120

Table C.79: *Precision, recall and F1 by class for RandomForestGini model and Dataset D*

	Off	On
Off	116	4
On	31	89

Table C.80: *Confusion matrix for RandomForestGini model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.88	0.93	0.90	40
alarm.on	0.92	0.88	0.90	40
timer.off	0.84	0.80	0.82	40
timer.on	0.81	0.85	0.83	40

Table C.81: *Precision, recall and F1 by class for RandomForestEntr model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	37	3	0	0
alarm.on	5	35	0	0
timer.off	0	0	32	8
timer.on	0	0	6	34

Table C.82: *Confusion matrix for RandomForestEntr model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.84	0.95	0.89	40
alarm.on	0.94	0.82	0.88	40
reminder.off	0.81	0.88	0.84	40
reminder.on	0.86	0.80	0.83	40
timer.off	0.81	0.72	0.76	40
timer.on	0.75	0.82	0.79	40

Table C.83: *Precision, recall and F1 by class for RandomForestEntr model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	39	1	0	0	0	0
alarm.on	7	33	0	0	0	0
reminder.off	0	0	35	5	0	0
reminder.on	0	0	8	32	0	0
timer.off	0	0	0	0	29	11
timer.on	0	0	0	0	7	33

Table C.84: *Confusion matrix for RandomForestEntr model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.85	0.88	0.86	80
On	0.87	0.85	0.86	80

Table C.85: *Precision, recall and F1 by class for RandomForestEntr model and Dataset C*

	Off	On
Off	70	10
On	12	68

Table C.86: *Confusion matrix for RandomForestEntr model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.80	0.97	0.88	120
On	0.96	0.76	0.85	120

Table C.87: *Precision, recall and F1 by class for RandomForestEntr model and Dataset D*

	Off	On
Off	116	4
On	29	91

Table C.88: *Confusion matrix for RandomForestEntr model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.86	0.95	0.90	40
alarm.on	0.94	0.85	0.89	40
timer.off	0.81	0.95	0.87	40
timer.on	0.94	0.78	0.85	40

Table C.89: *Precision, recall and F1 by class for CatBoost model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	38	2	0	0
alarm.on	6	34	0	0
timer.off	0	0	38	2
timer.on	0	0	9	31

Table C.90: *Confusion matrix for CatBoost model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.89	0.85	0.87	40
alarm.on	0.86	0.90	0.88	40
reminder.off	0.77	0.93	0.84	40
reminder.on	0.91	0.72	0.81	40
timer.off	0.83	0.88	0.85	40
timer.on	0.87	0.82	0.85	40

Table C.91: *Precision, recall and F1 by class for CatBoost model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	34	6	0	0	0	0
alarm.on	4	36	0	0	0	0
reminder.off	0	0	37	3	0	0
reminder.on	0	0	11	29	0	0
timer.off	0	0	0	0	35	5
timer.on	0	0	0	0	7	33

Table C.92: *Confusion matrix for CatBoost model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.84	0.96	0.90	80
On	0.96	0.81	0.88	80

Table C.93: *Precision, recall and F1 by class for CatBoost model and Dataset C*

	Off	On
Off	77	3
On	15	65

Table C.94: *Confusion matrix for CatBoost model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.78	0.98	0.87	120
On	0.98	0.72	0.83	120

Table C.95: *Precision, recall and F1 by class for CatBoost model and Dataset D*

	Off	On
Off	118	2
On	34	86

Table C.96: *Confusion matrix for CatBoost model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.89	0.97	0.93	40
alarm.on	0.97	0.88	0.92	40
timer.off	0.80	0.82	0.81	40
timer.on	0.82	0.80	0.81	40

Table C.97: *Precision, recall and F1 by class for ExtraTreesGini model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	38	2	0	0
alarm.on	6	34	0	0
timer.off	0	0	38	2
timer.on	0	0	9	31

Table C.98: *Confusion matrix for ExtraTreesGini model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.89	0.97	0.93	40
alarm.on	0.97	0.88	0.92	40
reminder.off	0.88	0.88	0.88	40
reminder.on	0.88	0.88	0.88	40
timer.off	0.83	0.75	0.79	40
timer.on	0.77	0.85	0.81	40

Table C.99: *Precision, recall and F1 by class for ExtraTreesGini model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	39	1	0	0	0	0
alarm.on	5	35	0	0	0	0
reminder.off	0	0	35	5	0	0
reminder.on	0	0	5	35	0	0
timer.off	0	0	0	0	30	10
timer.on	0	0	0	0	6	34

Table C.100: *Confusion matrix for ExtraTreesGini model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.89	0.89	0.89	80
On	0.89	0.89	0.89	80

Table C.101: *Precision, recall and F1 by class for ExtraTreesGini model and Dataset C*

	Off	On
Off	71	9
On	9	71

Table C.102: *Confusion matrix for ExtraTreesGini model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.80	0.96	0.87	120
On	0.95	0.76	0.84	120

Table C.103: *Precision, recall and F1 by class for ExtraTreesGini model and Dataset D*

	Off	On
Off	115	5
On	29	91

Table C.104: *Confusion matrix for ExtraTreesGini model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.89	0.97	0.92	40
alarm.on	0.97	0.88	0.91	40
timer.off	0.81	0.85	0.83	40
timer.on	0.84	0.80	0.82	40

Table C.105: *Precision, recall and F1 by class for ExtraTreesEntr model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	38	2	0	0
alarm.on	5	35	0	0
timer.off	0	0	34	6
timer.on	0	0	8	32

Table C.106: *Confusion matrix for ExtraTreesEntr model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.91	0.97	0.94	40
alarm.on	0.97	0.90	0.94	40
reminder.off	0.90	0.88	0.89	40
reminder.on	0.88	0.90	0.89	40
timer.off	0.81	0.75	0.78	40
timer.on	0.77	0.82	0.80	40

Table C.107: *Precision, recall and F1 by class for ExtraTreesEntr model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	38	2	0	0	0	0
alarm.on	5	35	0	0	0	0
reminder.off	0	0	35	5	0	0
reminder.on	0	0	4	36	0	0
timer.off	0	0	0	0	30	10
timer.on	0	0	0	0	7	33

Table C.108: *Confusion matrix for ExtraTreesEntr model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.87	0.89	0.88	80
On	0.88	0.86	0.87	80

Table C.109: *Precision, recall and F1 by class for ExtraTreesEntr model and Dataset C*

	Off	On
Off	71	9
On	11	69

Table C.110: *Confusion matrix for ExtraTreesEntr model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.80	0.96	0.87	120
On	0.95	0.77	0.85	120

Table C.111: *Precision, recall and F1 by class for ExtraTreesEntr model and Dataset D*

	Off	On
Off	115	5
On	28	92

Table C.112: *Confusion matrix for ExtraTreesEntr model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.54	0.53	0.53	40
alarm.on	0.63	0.72	0.67	40
timer.off	0.60	0.68	0.64	40
timer.on	0.77	0.57	0.66	40

Table C.113: *Precision, recall and F1 by class for NeuralNetFastAI model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	21	14	4	1
alarm.on	6	29	3	2
timer.off	8	1	27	4
timer.on	4	2	11	23

Table C.114: *Confusion matrix for NeuralNetFastAI model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.50	0.45	0.47	40
alarm.on	0.56	0.55	0.56	40
reminder.off	0.28	0.20	0.23	40
reminder.on	0.30	0.33	0.31	40
timer.off	0.27	0.30	0.28	40
timer.on	0.52	0.62	0.57	40

Table C.115: *Precision, recall and F1 by class for NeuralNetFastAI model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	18	13	0	5	2	2
alarm.on	8	22	2	5	0	3
reminder.off	2	0	8	7	16	7
reminder.on	3	4	5	13	8	7
timer.off	3	0	11	10	12	4
timer.on	2	0	3	3	7	25

Table C.116: *Confusion matrix for NeuralNetFastAI model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.75	0.76	0.76	80
On	0.76	0.75	0.75	80

Table C.117: *Precision, recall and F1 by class for NeuralNetFastAI model and Dataset C*

	Off	On
Off	61	19
On	20	60

Table C.118: *Confusion matrix for NeuralNetFastAI model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.69	0.67	0.68	120
On	0.68	0.70	0.69	120

Table C.119: *Precision, recall and F1 by class for NeuralNetFastAI model and Dataset D*

	Off	On
Off	61	19
On	28	92

Table C.120: *Confusion matrix for NeuralNetFastAI model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.85	0.97	0.91	40
alarm.on	0.97	0.82	0.89	40
timer.off	0.80	0.90	0.85	40
timer.on	0.89	0.78	0.83	40

Table C.121: *Precision, recall and F1 by class for XGBoost model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	39	1	0	0
alarm.on	7	33	0	0
timer.off	0	0	36	4
timer.on	0	0	9	31

Table C.122: *Confusion matrix for XGBoost model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.91	0.97	0.95	40
alarm.on	0.97	0.90	0.95	40
reminder.off	0.90	0.88	0.89	40
reminder.on	0.88	0.90	0.89	40
timer.off	0.81	0.75	0.78	40
timer.on	0.77	0.82	0.80	40

Table C.123: *Precision, recall and F1 by class for XGBoost model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	39	1	0	0	0	0
alarm.on	3	37	0	0	0	0
reminder.off	0	0	36	4	0	0
reminder.on	0	0	6	34	0	0
timer.off	0	0	2	0	30	8
timer.on	0	0	0	0	8	32

Table C.124: *Confusion matrix for XGBoost model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.83	0.95	0.88	80
On	0.94	0.80	0.86	80

Table C.125: *Precision, recall and F1 by class for XGBoost model and Dataset C*

	Off	On
Off	76	4
On	16	64

Table C.126: *Confusion matrix for XGBoost model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.82	0.90	0.86	120
On	0.89	0.81	0.85	120

Table C.127: *Precision, recall and F1 by class for XGBoost model and Dataset D*

	Off	On
Off	108	12
On	23	97

Table C.128: *Confusion matrix for XGBoost model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.55	0.53	0.54	40
alarm.on	0.63	0.65	0.64	40
timer.off	0.62	0.70	0.66	40
timer.on	0.72	0.65	0.68	40

Table C.129: *Precision, recall and F1 by class for NeuralNetTorch model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	21	14	4	1
alarm.on	6	26	3	5
timer.off	7	1	28	4
timer.on	4	0	10	26

Table C.130: *Confusion matrix for NeuralNetTorch model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.49	0.50	0.49	40
alarm.on	0.55	0.65	0.60	40
reminder.off	0.19	0.07	0.11	40
reminder.on	0.37	0.25	0.30	40
timer.off	0.29	0.45	0.35	40
timer.on	0.51	0.60	0.55	40

Table C.131: *Precision, recall and F1 by class for NeuralNetTorch model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	20	14	0	0	0	0
alarm.on	6	26	2	3	0	3
reminder.off	5	1	3	1	23	7
reminder.on	3	6	3	10	10	8
timer.off	3	0	5	10	18	4
timer.on	4	0	3	2	7	24

Table C.132: *Confusion matrix for NeuralNetTorch model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.73	0.71	0.72	80
On	0.72	0.74	0.73	80

Table C.133: *Precision, recall and F1 by class for NeuralNetTorch model and Dataset C*

	Off	On
Off	57	23
On	21	59

Table C.134: *Confusion matrix for NeuralNetTorch model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.68	0.78	0.73	120
On	0.74	0.64	0.69	120

Table C.135: *Precision, recall and F1 by class for NeuralNetTorch model and Dataset D*

	Off	On
Off	93	27
On	43	77

Table C.136: *Confusion matrix for NeuralNetTorch model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.88	0.90	0.89	40
alarm.on	0.90	0.88	0.89	40
timer.off	0.82	0.82	0.82	40
timer.on	0.82	0.82	0.82	40

Table C.137: *Precision, recall and F1 by class for LightGBMLarge model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	36	4	0	0
alarm.on	5	35	0	0
timer.off	0	0	37	3
timer.on	0	0	7	33

Table C.138: *Confusion matrix for LightGBMLarge model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.91	0.97	0.94	40
alarm.on	0.97	0.90	0.94	40
reminder.off	0.75	0.95	0.84	40
reminder.on	0.94	0.72	0.82	40
timer.off	0.79	0.75	0.77	40
timer.on	0.80	0.80	0.80	40

Table C.139: *Precision, recall and F1 by class for LightGBMLarge model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	39	1	0	0	0	0
alarm.on	4	36	0	0	0	0
reminder.off	0	0	38	2	0	0
reminder.on	0	0	11	29	0	0
timer.off	0	0	2	0	30	84
timer.on	0	0	0	0	8	32

Table C.140: *Confusion matrix for LightGBMLarge model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.82	0.95	0.88	80
On	0.94	0.79	0.86	80

Table C.141: *Precision, recall and F1 by class for LightGBMLarge model and Dataset C*

	Off	On
Off	57	23
On	21	59

Table C.142: *Confusion matrix for LightGBMLarge model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.81	0.92	0.86	120
On	0.90	0.79	0.84	120

Table C.143: *Precision, recall and F1 by class for LightGBMLarge model and Dataset D*

	Off	On
Off	110	10
On	25	95

Table C.144: *Confusion matrix for LightGBMLarge model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.88	0.88	0.88	40
alarm.on	0.88	0.88	0.88	40
timer.off	0.85	0.85	0.85	40
timer.on	0.85	0.85	0.85	40

Table C.145: *Precision, recall and F1 by class for WeightedEnsemble_L2 model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	35	5	0	0
alarm.on	5	35	0	0
timer.off	0	0	34	6
timer.on	0	0	6	34

Table C.146: *Confusion matrix for WeightedEnsemble_L2 model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.93	0.97	0.95	40
alarm.on	0.97	0.93	0.95	40
reminder.off	0.85	0.88	0.86	40
reminder.on	0.87	0.85	0.86	40
timer.off	0.83	0.75	0.79	40
timer.on	0.77	0.85	0.81	40

Table C.147: *Precision, recall and F1 by class for WeightedEnsemble_L2 model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	39	1	0	0	0	0
alarm.on	4	36	0	0	0	0
reminder.off	0	0	35	5	0	0
reminder.on	0	0	6	34	0	0
timer.off	0	0	0	0	30	10
timer.on	0	0	0	0	6	34

Table C.148: *Confusion matrix for WeightedEnsemble_L2 model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.84	0.96	0.90	80
On	0.96	0.81	0.88	80

Table C.149: *Precision, recall and F1 by class for WeightedEnsemble_L2 model and Dataset C*

	Off	On
Off	77	3
On	15	65

Table C.150: *Confusion matrix for WeightedEnsemble_L2 model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.81	0.95	0.87	120
On	0.94	0.78	0.85	120

Table C.151: *Precision, recall and F1 by class for WeightedEnsemble_L2 model and Dataset D*

	Off	On
Off	114	6
On	27	93

Table C.152: *Confusion matrix for WeightedEnsemble_L2 model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	1.00	1.00	1.00	40
alarm.on	1.00	1.00	1.00	40
timer.off	0.89	0.97	0.93	40
timer.on	0.97	0.88	0.92	40

Table C.153: *Precision, recall and F1 by class for Dialogflow model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	40	0	0	0
alarm.on	0	40	0	1
timer.off	0	0	39	1
timer.on	0	0	5	35

Table C.154: *Confusion matrix for Dialogflow model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	1.00	0.97	0.99	40
alarm.on	0.98	1.00	0.99	40
reminder.off	0.91	0.97	0.94	40
reminder.on	0.97	0.90	0.94	40
timer.off	1.00	0.95	0.97	40
timer.on	0.95	1.00	0.98	40

Table C.155: *Precision, recall and F1 by class for Dialogflow model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	39	1	0	0	0	0
alarm.on	0	40	0	0	0	0
reminder.off	0	39	1	0	0	0
reminder.on	0	0	4	36	0	0
timer.off	0	0	0	0	38	2
timer.on	0	0	0	1	25	14

Table C.156: *Confusion matrix for Dialogflow model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.95	0.99	0.97	80
On	0.99	0.95	0.97	80

Table C.157: *Precision, recall and F1 by class for Dialogflow model and Dataset C*

	Off	On
Off	79	1
On	4	76

Table C.158: *Confusion matrix for Dialogflow model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.96	0.96	0.96	120
On	0.96	0.96	0.96	120

Table C.159: *Precision, recall and F1 by class for Dialogflow model and Dataset D*

	Off	On
Off	115	5
On	5	115

Table C.160: *Confusion matrix for Dialogflow model and Dataset D*

Class	Precision	Recall	F1	Sample
alarm.off	0.98	1.00	0.99	40
alarm.on	1.00	0.97	0.99	40
timer.off	0.88	0.93	0.90	40
timer.on	0.92	0.88	0.90	40

Table C.161: *Precision, recall and F1 by class for LUIS model and Dataset A*

	alarm.off	alarm.on	timer.off	timer.on
alarm.off	40	0	0	0
alarm.on	1	39	0	0
timer.off	0	0	37	3
timer.on	0	0	5	35

Table C.162: *Confusion matrix for LUIS model and Dataset A*

Class	Precision	Recall	F1	Sample
alarm.off	0.97	0.97	0.97	40
alarm.on	0.97	0.97	0.97	40
reminder.off	0.88	0.93	0.90	40
reminder.on	0.92	0.88	0.90	40
timer.off	0.97	0.95	0.96	40
timer.on	0.95	0.97	0.96	40

Table C.163: *Precision, recall and F1 by class for LUIS model and Dataset B*

	alarm.off	alarm.on	reminder.off	reminder.on	timer.off	timer.on
alarm.off	39	1	0	0	0	0
alarm.on	1	39	0	0	0	0
reminder.off	0	0	37	3	0	0
reminder.on	0	0	5	35	0	0
timer.off	0	0	0	0	38	2
timer.on	0	0	0	0	1	39

Table C.164: *Confusion matrix for LUIS model and Dataset B*

Class	Precision	Recall	F1	Sample
Off	0.94	0.97	0.96	80
On	0.97	0.94	0.96	80

Table C.165: *Precision, recall and F1 by class for LUIS model and Dataset C*

	Off	On
Off	78	2
On	5	75

Table C.166: *Confusion matrix for LUIS model and Dataset C*

Class	Precision	Recall	F1	Sample
Off	0.96	0.98	0.97	120
On	0.98	0.96	0.97	120

Table C.167: *Precision, recall and F1 by class for LUIS model and Dataset D*

	Off	On
Off	118	2
On	5	115

Table C.168: *Confusion matrix for LUIS model and Dataset D*

Dataset	LSTM	SVM
Dataset A	timer.off	timer.off
Dataset B	alarm.off, alarm.on	alarm.on
Dataset C	both classes	both classes
Dataset D	off	both classes

Table C.169: *Summary of the best classes concerning all the datasets and the different models from scratch*

Dataset	BERT	RoBERTa	XLNet
Dataset A	alarm.on	alarm.on, timer.on	alarm.off, alarm.on
Dataset B	alarm.off, alarm.on	reminder.off, reminder.on	alarm.off, alarm.on
Dataset C	both classes	both classes	both classes
Dataset D	on	off	both classes

Table C.170: *Summary of the best classes concerning all the datasets and the different models based on transformers*

Dataset	ExtraTreesEntr	ExtraTreesGini
Dataset A	alarm.off	alarm.off
Dataset B	alarm.off, alarm.on	alarm.off
Dataset C	off	both classes
Dataset D	off	off

Table C.171: *Summary of the best classes concerning all the datasets and ExtraTrees algorithms*

Dataset	KNeighborsDist	KNeighborsUnif
Dataset A	alarm.off	alarm.off
Dataset B	alarm.on	alarm.on
Dataset C	off	off
Dataset D	off	off

Table C.172: *Summary of the best classes concerning all the datasets and KNeighbors algorithms*

Dataset	LightGBM	LightGBMLarge	LightGBMXT
Dataset A	alarm.off	alarm.on, alarm.off	alarm.off
Dataset B	alarm.off	alarm.on, alarm.off	alarm.off
Dataset C	off	off	off
Dataset D	off	off	both classes

Table C.173: *Summary of the best classes concerning all the datasets and LightGBM algorithms*

Dataset	NeuralNetFastAI	NeuralNetTorch
Dataset A	timer.on	timer.on
Dataset B	alarm.on	alarm.on
Dataset C	off	on
Dataset D	on	off

Table C.174: *Summary of the best classes concerning all the datasets and ExtraTrees algorithms*

Dataset	RandomForestEntr	RandomForestGini
Dataset A	alarm.off, alarm.on	alarm.off, alarm.on
Dataset B	alarm.off	alarm.off
Dataset C	both classes	both classes
Dataset D	off	off

Table C.175: *Summary of the best classes concerning all the datasets and ExtraTrees algorithms*

Dataset	CatBoost	WeightedEnsemble_L2	XGBoost
Dataset A	alarm.off	alarm.off, alarm.on	alarm.off
Dataset B	alarm.on	alarm.off, alarm.on	alarm.off, alarm.on
Dataset C	off	off	off
Dataset D	off	off	off

Table C.176: *Summary of the best classes concerning all the datasets and ExtraTrees algorithms*

Dataset	Dialogflow	LUIS
Dataset A	alarm.off, alarm.on	alarm.off, alarm.on
Dataset B	alarm.off, alarm.on	alarm.off, alarm.on
Dataset C	both classes	both classes
Dataset D	both classes	both classes

Table C.177: *Summary of the best classes concerning all the datasets and the different models from scratch*