Question extraction and intent classification using COVID-19 Twitter data

Tuesday 25th March, 2025 - 09:40

Ivaylo Krumov
University of Luxembourg
Email: ivaylo.krumov.001@student.uni.lu

This report has been produced under the supervision of:
Sviatlana Höhn
University of Luxembourg
Email: sviatlana.hoehn@uni.lu

Abstract

The purpose of this Bachelor Semester Project is to analyze the potential feasibility for an autonomous algorithmic approach to process and classify data. The project utilizes the domain of intent classification to conduct this study. To produce concrete results, several keywords, related to the concerned project subject, have been given clear definitions, thus establishing the scientific context, in which the project is centered. Moreover, an intent-classifying program has been developed with the aim of providing a practical example for this research. All accomplished results of the project are thoroughly presented in this report. The ultimate goal of the report and the project as a whole is to give useful insight into the feasibility of algorithm-based classification of data.

1. Plagiarism statement

This 350 words section without this first paragraph must be included in the submitted report and placed after the conclusion. This section is not counting in the total words quantity.

I declare that I am aware of the following facts:

- As a student at the University of Luxembourg I must respect the rules of intellectual honesty, in particular not to resort to plagiarism, fraud or any other method that is illegal or contrary to scientific integrity.
- My report will be checked for plagiarism and if the plagiarism check is positive, an internal procedure will be started by my tutor. I am advised to request a pre-check by my tutor to avoid any issue.
- As declared in the assessment procedure of the University of Luxembourg, plagiarism is committed whenever the source of information used in an assignment, research report, paper or otherwise published/circulated piece of work is not properly acknowledged. In other words, plagiarism is the passing off as one's own the words, ideas or work of another person, without attribution to the author. The omission of such proper acknowledgement amounts to claiming authorship for the work of another person. Plagiarism is committed regardless of the language of the original work used. Plagiarism can be deliberate or accidental. Instances of plagiarism include, but are not limited to:
  1) Not putting quotation marks around a quote from another person's work
  2) Pretending to paraphrase while in fact quoting
  3) Citing incorrectly or incompletely
  4) Failing to cite the source of a quoted or paraphrased work
  5) Copying/reproducing sections of another person's work without acknowledging the source
  6) Paraphrasing another person's work without acknowledging the source
  7) Having another person write/author a work for oneself and submitting/publishing it (with permission, with or without compensation) in one's own name ('ghost-writing')
  8) Using another person's unpublished work without attribution and permission ('stealing')
  9) Presenting a piece of work as one's own that contains a high proportion of quoted/copied or paraphrased text (images, graphs, etc.), even if adequately referenced

  Auto- or self-plagiarism, that is the reproduction of (portions of a) text previously written by the author without citing that text, i.e. passing previously authored text as new, may be regarded as fraud if deemed sufficiently severe.

2. Introduction

In the current digital era we all live in, technological progress has become inevitable. In recent times, the subject of artificial intelligence (AI) has been a central topic of discussion. The utilization of AI in order to assist in making

technological breakthroughs has become a necessity, as all kinds of research and development processes become more complex, thus giving rise to a need for fast, efficient and automated methods, which is what AI offers.

Specifically, AI is a key aspect of the field of natural language processing (NLP). Many applications rely on AI assistants that are able to process spoken or written requests made by humans and follow them up with suitable responses according to the particular situation. This is a case where the AI's efficiency, as determined by the quality of the method used to train it to perform a certain task, is vital to its success.

A common scenario in NLP, and more specifically in text or intent classification, is the processing of data that requires some machine to identify the context this data belongs to in order to produce the most appropriate results. In such a situation it is vital to have a well-defined and properly ordered basis for comparison, from which the AI can quickly know how some particular data should be classified.

This report presents an in-depth response to the question of how raw disorganized data can be properly sorted in groups with the help of AI. To show this, a program, which uses intent classification to classify a large set of data, is described. Moreover, a scientific piece of text provides context to the topic of the program and this project in general. With all given information, it is expected that the report manages to give useful insight into the feasibility of algorithm-based classification of data as a whole.

3. Project description

3.1. Domains

3.1.1. Scientific.

- **Natural Language Processing.** Natural Language Processing (NLP) is a field in Computer Science that is strongly related to the fields of Artificial Intelligence and Linguistics. Essentially, NLP deals with the ways computers can process human languages on their own, allowing machines to be used for analysis of usually large volumes of human language data, either in written or spoken form. More often than not, data needed to be analyzed via NLP is text-based and is frequently unorganized, making this field ideal for the task of producing structured chunks of data based on analysis results.
- **Natural Language Understanding.** Natural Language Understanding (NLU) is a subfield of NLP which specifically concerns a computer's ability to understand human language data. Similarly to NLP, NLU is also used for analyzing and structuring data, though it is focused on the analysis of the meaning of given data, e.g. semantics of a sentence. In practice, NLU is a way for machines to comprehend human language, which allows them to organize data based on the meaning it conveys or even interact with humans by having question-and-answer type conversations with them in real time.
- **Intent classification.** Intent classification is a part of NLU which deals with the task of a computer analyzing a piece of human speech and labeling it according to the meaning it intends to convey. Using intent classification, large amounts of unstructured data can be organized into several collections of smaller data chunks, where each collection represents the intent which is common for all elements in that collection.

3.1.2. Technical.

- **Python.** Python is the programming language of choice for the realization of the technical part of this project. The reasoning behind this is simple - Python is a very popular and versatile programming language and therefore supports numerous modern libraries used for dealing with a variety of tasks such as NLP, NLU, visual data representation and many more.
- **Spacy.** Spacy is a Python library designed for the purposes of NLP. Several human languages are supported to work with Spacy. It has numerous features that allow for the extraction of different kinds of information from texts in those languages by performing tasks such as named entity recognition (NER), part-of-speech (POS) tagging, morphological analysis, text classification, etc.
- **Rasa.** Rasa is a Python library specifically used for NLU. In particular, it is very well fit for training a machine to be able to perform intent classification on text data. Oftentimes this is further applied in the development of conversational AI assistants.
- **Gensim.** Gensim is a Python library for topic modelling. Essentially, it provides tools that can be used to analyze text data and find certain patterns in it based on semantic similarities between different texts. These patterns are then used to construct a certain amount of topics, which represent common semantic meanings between texts in the data. The structured information provided in the form of topics can make it much easier to do further semantic text analysis.
- **Pandas and pyLDAvis.** Both Pandas and pyLDAvis are Python libraries useful for certain kinds of visual data representation. In particular, Pandas provides access to the DataFrame data structure for tabular representation of data, while pyLDAvis is used to plot graphs which concern topic modelling and show information about topic clusters and common topic keywords.
- **Tweepy.** Tweepy is a Python library that provides developers with access to the Twitter API. Although there are 3 different levels of access depending on the needs of the developer, for the purposes of this project the only necessary access level is the most basic one - Essential access. As with the higher access levels, Essential access still grants the ability to call the API to retrieve Tweet data by using its Tweet ID, which is the only required API call in the project's technical part.
- **Google Colaboratory.** Google Colaboratory (Colab for short) is an online coding environment designed specif-

ically for writing Python code in Jupyter notebooks. Thanks to its built-in Jupyter notebook service, Colab is suitable to use for the purposes of machine learning and data analysis.

3.2. Targeted Deliverables

3.2.1. Scientific deliverables. The produced scientific deliverable attempts to find an answer to the question "Can a program process human-written questions to identify their intent?". To accomplish this, it thoroughly analyzes the most crucial keywords of the question and provides concise definitions for each one. With this understanding, a conclusive answer to the scientific question is formulated at the end. Finally, there is a brief discussion to assess the extent to which all requirements for the deliverable have been satisfied.

3.2.2. Technical deliverables. The produced technical deliverable is a program with the purpose to identify questions from within a large set of text data and classify them by their intent. It is presented by a thorough description of the different algorithms that operate within its flow of execution, followed by a short discussion about its underlying unsolved problems affecting its optimal productivity.

4. Pre-requisites

The complete realization of this project, both the scientific and the technical part, takes advantage of several skills and competencies that the author already is in possession of before the start of the project research process. These pre-requisites are respectively identified in the 2 subsections that follow.

4.1. Scientific pre-requisites

The author has a solid understanding about how to adequately write and structure a piece of work focused on a scientific topic. Furthermore, he is fully capable of comprehending meaningful information from scientific sources, as long as the terminology used is at a reasonable level of complexity. Lastly, he possesses some knowledge about the real-life applications of the project's scientific domains, particularly in the development of chatbots and virtual assistants.

4.2. Technical pre-requisites

The author has prior experience in programming in several languages. Because of this, he is very much aware of the possibility of encountering technical obstacles during the development of the technical deliverable and is prepared to utilize suitable methods to overcome them. The author also possesses decent knowledge in programming using the Python programming language, is able to install and use multiple Python packages and is capable of working with the different file formats required for this project.

5. Can a program process human-written questions to identify their intent?

5.1. Requirements

The scientific deliverable of this project aims to satisfy several requirements, which are presented in this section. In order to provide a better quality for the structure of their presentation, they are split into two different categories, namely functional and non-functional requirements. The functional requirements are described first, followed by a description of the non-functional ones.

Functional requirements:
The scientific deliverable should give a complete answer to the scientific question of the project - "Can a program process human-written questions to identify their intent?". For this purpose, it should be able to provide clear explanations of the key terminology directly related to the topic of the question. Furthermore, the deliverable should make use of this terminology to logically reach a conclusive answer to the question. It should also utilize external scientific sources and the information they may contain as support whenever it is suitable to do so.

Non-functional requirements:
Provided that the necessary scientific terms are well defined, the ideas in the scientific deliverable should also be expressed in such a manner that makes it possible for a non-professional reader to be able to grasp them. In regard to the overall structure of the deliverable, its contents should logically flow well together and they should all have either a direct or an indirect relation to the main topic of the question. It is important that all information provided is essential to ultimately achieving the final goal of the scientific deliverable, namely giving a definite answer to the scientific question. Furthermore, wherever citations from other scientific sources are made, credit should always be given to the referenced material and its author.

5.2. Design

The scientific deliverable is a text that is supposed to provide context to the ideas behind the produced technical deliverable. It is intended to serve as supportive material to the technical deliverable. Therefore, less focus has been placed on the scientific deliverable as the overall success of the project relies more on the results of the technical deliverable. For this reason, the time spent in research for the scientific deliverable and its creation is roughly 3 weeks, which is significantly less than what might otherwise be expected. Nevertheless, during this time noteworthy effort has been made to recognize and study the key subtopics of the scientific question so that it is answered as clearly and as accurately as possible.

The scientific deliverable has a straightforward structure - it is split into several small parts, each of which defines a

certain keyword or subtopic related to the scientific question. Once all of these definitions are made, the text concludes with the formulation of the final answer to the question, which is a result of the compilation of all important information presented in the previous parts.

With the intention of providing clear and concrete information, 2 main scientific sources are used as points of reference throughout the text. These sources are scientific publications published by researchers from 2 universities in Europe - one is Radboud University in the Netherlands while the other is the University of Luxembourg. Both papers are completely independent works, created 10 years apart, but each of them holds valuable information related to answering this project's scientific question.

5.3. Production

The following text aims to accomplish the goal of finding a concrete answer to the scientific question "Can a program process human-written questions to identify their intent?". Given the way in which this question is formulated, a possible answer can be provided once the meaning of each of its included terms are understood clearly. Therefore, for the purpose of achieving full clarity, and thus directly developing a good understanding of the project's technical deliverable, every important aspect related to the question is now going to be defined, no matter whether a definition may look trivial or not.

To begin with, since the scientific deliverable's aim is to establish the context for the technical one, which by itself is a computer program, it is vital to first understand the concept of a program. It can be seen as a series of instructions (code) that, once executed, is able to produce output based on externally provided input. In this sense, a program is very similar to a mathematical function, such as $x \rightarrow f(x)$ for example, where the input value of x is mapped to the corresponding output value of $f(x)$. In the core of this mapping procedure lies an algorithm which is defined by the specific function and is able to process the initial value of x and produce the final value of $f(x)$. A program behaves in exactly the same manner. In regard to the program of the project, this can be observed very easily: first it takes a set of written questions as input, then an algorithm, consisting of several smaller individual algorithms, does the required processing of the questions (intent classification), and finally outputs a result, in which each question from the input is mapped to a corresponding question intent. Since the algorithmic part of the program is largely technical rather than scientific, in this particular deliverable only remain to be viewed the definitions of a human-written question and an intent, which will be enough work to provide a stable scientific basis for the context of the technical deliverable.

In order to define a human-written question (from now on just referred to as "question(s)" for simplicity), there needs to be an exploration of the various ways in which it can be structured. Fortunately, Tanya Stivers and N.J. Enfield have already done thorough research in this regard [1]. Their study suggests that questions can come in several forms or shapes due to the innate ability of humans to linguistically express the same meaning but with seemingly different sentences. An interesting finding is that while most questions stand out from other sentence types because of the typical question mark at the end, a question is not necessarily defined by it and thus some sentences ending with a full stop may be implicit questions (e.g. "I guess you aren't coming tonight." can have the same meaning as "Aren't you coming tonight?"). In terms of meaning, questions are not always direct, a good example being rhetorical questions, which are defined as questions that do not actually expect a response as the answer is already implicitly contained inside them. This information is sufficient to give a definition for a question. However, there is one key limit that still needs to be mentioned. Since this project is not directly focused on the questions themselves, it is not necessary to take into consideration all possible ways to structure them. For our specific goals, it suffices to define a question by its most common characteristic trait - the question mark. Therefore, within the scope of this project, a question is going to be considered as any sentence, whose final punctuation point is a question mark.

Finally, it should be discussed what is meant by intent and how to recognize one. The word itself is a synonym for intention, implying that a sentence's intent is basically the meaning which the particular sentence conveys. Knowing this, it can be seen that an intent is dependent on the topic of the context it is used in. This relation has also been noticed by Ninghan Chen, Zhiqiang Zhong and Jun Pang in their study which, similarly to this project, deals with classifying COVID-19 Twitter data based on intents [2]. Their research method includes the manual creation of data-specific intents with the use of topics recognized by a machine algorithm. The algorithm manages to process each Tweet's particular topic and combine the most related ones into a separate cluster, thus creating 7 distinct clusters of Tweets with similar topics. The data contained in each cluster determines the overall topic for that cluster. Since every cluster has some topic that defines it, a corresponding intent can also be identified for it. In this case, Chen, Zhong and Pang have indeed extracted 7 different intents from this data, 1 for each topic cluster. While this method of recognizing intents may be far from perfect, due to the fact that the topic classification algorithm is prone to errors and inaccuracies, it has proven to be good enough to achieve successful results for the purpose of the three researchers' study. This makes it clear that, when working with a large dataset, defining common topics can be quite useful for the manual identification of intents. As a sidenote, this is why the method described above also largely serves as an influence for the implementation of the intent classification algorithm for this project's technical deliverable.

With all of this information in mind, the moment has come to give a conclusive answer to the scientific question. In theory, yes, a program is most definitely able to perform processing on an input question to output its corresponding intent. In fact, even in practice, this has already been shown

to be possible by the aforementioned research of Chen, Zhong and Pang, in which they accomplish the task of classifying different kinds of sentences, not only questions, by intent. In spite of all that, it is important to mention that there exist some limitations. Most notably, a program may not be capable of fully autonomously doing this job. As seen in the aforementioned research work [2], the method used for intent classification included manually detecting and extracting the intents themselves. Nevertheless, this does not interfere with the work of the autonomous computer algorithms and results are still successfully produced. Therefore, from a theoretical standpoint, there appear to be no real boundaries that can disprove a program's overall ability to do such processing. The practical feasibility of the task defined in the scientific question remains to be seen in the following technical deliverable section.

5.4. Assessment

The aim of the scientific deliverable was to provide context to the purpose of the technical deliverable by providing clear definitions for the essential terminology as well as outlining the potential limits of an intent-classifying program, if any. To that end, several requirements were established prior to the development of the scientific deliverable itself.

The production of the scientific deliverable has satisfied all defined requirements for it. Most importantly, the deliverable clearly conveys the necessary information and utilizes external scientific sources to affirm its ideas. These sources and their respective authors are properly credited as required and the logical flow of the contents of the text is sufficiently good to be considered as related to the main topic of the question while being easily comprehensible.

Overall, the scientific deliverable fully manages to reach the expected goals set for it. The provided scientific context to the technical deliverable ensures that the reader is prepared to explore the production of the latter by understanding the major concepts included in it and the reasoning for their utilization in its implementation.

6. Intent classification program

6.1. Requirements

The technical deliverable of this project aims to satisfy several requirements, which are presented in this section. In order to provide a better quality for the structure of their presentation, they are split into two different categories, namely functional and non-functional requirements. The functional requirements are described first, followed by a description of the non-functional ones.

Functional requirements:

The technical deliverable should be able to read a large amount of unorganized Tweet data and process it in such a way that it produces a structured result which consists of extracted questions grouped by the similarity of their intents. To do so, the developed program should be capable of performing pre-processing on the initial dataset, which only consists of the ID of each Tweet that is to be analyzed. These IDs should therefore be used by the pre-processing algorithm to retrieve the texts of the corresponding Tweets, which should in turn be utilized in identifying and extracting the contained questions. The program should also be able to semantically analyze the meaning of each question and determine the intent group to which it most likely belongs.

Non-functional requirements:

The technical deliverable should display important information in a suitable and easily comprehensible visual form. More specifically, the grouping of questions by intent after the classification process should be clearly represented as simple visual output in the program's working terminal. Moreover, the program should take all of its necessary input by reading other files containing the required data and whenever data must be stored for later use, it should be saved in a separate file as well.

6.2. Design

The technical deliverable of this project is designed as a program that attempts to classify questions in a given dataset as accurately as possible based on their intent. The output of the program, which is a visual representation of the mapping of each question to its most probable intent, implicitly provides an answer to the question whether or not it is able to successfully recognize a question's intent.

In regard to its visual design, the program is heavily focused on its data pre-processing and intent classification algorithms, and does not aim to visually appeal to the user. As a result, the program's visual design only exists in the working terminal, where important data is output mostly in the form of Pandas DataFrames (which are essentially a tabular way to display data) or cluster graphs (only for displaying topic clusters in the dataset).

The program is dependent on several external files in order to run as intended. Namely, the most crucial ones are the initial dataset file, which contains the raw Tweet IDs, the file created after the Tweet pre-processing and question extraction, containing the set of questions to be classified, the training model file needed for the intent classification algorithm as well as its configuration file. Due to these dependencies, all of those files including the main program file should be stored in a common directory path, which ensures that all algorithms are properly executed.

The program has been developed over the course of roughly 10 weeks. A large portion of this time has been spent on technical research and optimization of the intent classification algorithm. The latter involves tuning the training model to incorporate more suitable training examples in regard to the given dataset. Meanwhile, the former consists of finding the most useful Python packages that effectively allow for a

successful achievement of the technical deliverable's goals.

6.3. Production

The technical deliverable is a program composed of several algorithms that are used in conjunction with each other to produce a final result. As each of the algorithms depends on the previous ones to run properly, their working principles are now going to be described by order of their execution, starting with the pre-processing algorithms and finishing with the intent classifier itself.

As a first step, the initial dataset file, which is a .pkl file that stores the Tweet ID data in binary format, is converted into a human-readable format. This is done with the help of the Pickle package by using its load() function to read the binary data and generate the DataFrame of IDs represented by it. This DataFrame is then converted into a list in preparation for the the algorithm that follows, namely the Tweet text extraction by ID (Fig. 1).

Before the execution of this algorithm, however, there are a few things that need to be set up. Firstly, a Tweepy client object is initialized, providing access to the necessary functions for the extraction process (Fig. 2). Since the dataset Tweets are expected to be written in various languages (English, French, German, etc.) and because for the purpose of this project it is not necessary to work with different languages at the same time, some simplification has been done in that regard. In particular, following the client object, a Spacy language detector object is initialized with the goal of identifying English Tweets during the extraction process (Fig. 3). By doing this, the algorithm is restricted to extracting Tweets in English only, effectively producing a Tweet sample size smaller than the original dataset. In spite of this, it is still large enough to produce meaningful results while also being easier to work with.

The API access level of the Twitter developer account used for this project does have some limitations in regard to the amount of times the API is allowed to provide Tweet access within a certain period. Specifically, each 15 minutes it can be called for getting particular Tweet data up to roughly 200 times. For the extraction algorithm this means that there is a limited number of loop iterations before reaching a "cooldown point" for the API. Given the large number of the ID sample (35 283 IDs to be exact), a limitation such as this one implies that with 200 Tweets per 15 minutes the entire process would be extremely time inefficient. Fortunately, the get_tweets() function, which is the one that provides Tweet access by a specific ID, allows the usage of a list containing up to 100 reference IDs per call. In essence, this greatly increases the amount of possible extracted Tweets per 15 minutes - up to 20 000. Even though this number does not include all of the IDs from our dataset, it still represents more than 57% of it, which is more than enough to accomplish the program's goal.

With all of this explicitly mentioned, the Tweet extraction algorithm works as follows (Fig. 4): it does 200 iterations in total, where each iteration takes a set of 100 IDs from the ID list and reads each one, returning the corresponding Tweet data. Next, the extracted Tweets from the current set are checked whether or not the language in which each one is written is English. This is done with Spacy using the aforementioned English language detector by keeping only those texts whose language attribute corresponds to "en" (English) while also having a confidence level of at least 0.9 (90%), ensuring that the detected language is indeed English. Each text string that successfully passes these checks is appended to a list where all the filtered Tweet texts are stored for later use. After the complete execution of the algorithm, out of all 20 000 extracted Tweets, only 6014 separate text strings remain in the final list, which is still a good enough working sample size.

The next part of the data pre-processing is the extraction of questions from the Tweet texts (Fig. 5). This algorithm is much more straightforward to implement since it uses this project's definition of a question as explained in the scientific deliverable. Therefore, its working principle is simple: it iterates through each string in the Tweet text list and checks if a sentence in it contains a question mark. For that purpose, Spacy is utilized once more by filtering only those sentences, which contain a sentence token in the form of "?" (question mark). Each identified question is then converted to lowercase in preparation for an easier analysis later on and is appended to a list containing all of the previously extracted questions. The final list of questions is comprised of 490 different question strings. With all questions finally in possession, the pre-processing phase of the program is over and all that remains is the actual intent classification process.

For the question intent classification, the program follows a cyclic procedure which is not fully automated, as it requires some manual regulation. While the chosen approach may be far from being the fastest, it is simple to implement and also allows a lot of control over the entire process due to the manual portion of the work. This procedure is split into two main parts, both of which are now going to be described in detail.

The goal of the first half of the process is to distinguish the most common topics and keywords throughout the question sample. As shown in the scientific deliverable, this is an effective way to reveal the different possibilities for question intents and identify which ones are more likely to be present than others. To accomplish this, a list of each question's main part-of-speech keywords (namely any contained nouns, adjectives, verbs, adverbs or pronouns) is taken and is processed into a training corpus (in other words, training data) for the topic modelling algorithm. The list of keywords itself stores sublists as elements, each of which in turn contains the corresponding question's part-of-speech keywords in the form of strings. The sublists are populated by a simple algorithm (Fig. 6) that takes each question string from the questions list, converting it to a Spacy doc object, and checks whether or not each token matches the keyword criteria defined earlier. Whenever a match is found, that specific word string is appended to the corresponding sublist. After iterating over every question, they keywords list is then converted into a special dictionary

format (Fig. 7), specifically used to create a corpus for the modelling algorithm, which uses the standard Latent Dirichlet allocation (LDA) as a model for its topic modelling. For achieving accurate results while at the same time producing topics distinguished from one another, the modelling algorithm completes 300 iterations and attempts to recognize 5 distinct topics before it finally generates the LDA topic model for this particular set of keywords (Fig. 8). Using the utility of the pyLDAvis package, the resulting topics and their most common keywords are visualized in a graphical form (Fig. 9) convenient for a follow-up data analysis, which is essential for establishing the behaviour of the intent classifier.

The second half of the classification process consists of the implementation of the intent classification algorithm itself with the help of the Rasa NLU package. From a high level point of view, its working principle is simple: the training data is first loaded from a JSON-type file, containing example question-to-intent mappings intended to train the classification model (Fig. 10). The training questions and their respective intents are manually written beforehand using the results of the topic model's data analysis. This is done by a simple approach consisting of logically determining the most relevant possibilities for questions in the dataset based on keyword frequency in each topic. Once loaded, the training data is then configured for use via a file defining the model itself and the language in which it is to be trained. The model's behaviour is then trained with the given training data using Rasa's train function. With the classification model now fully set up, the only remaining task is to apply it onto the extracted question data. This is done with the assistance of Rasa's built-in model interpreter, which is able to parse each question from the questions list and map it to its most probable intent as determined by the classifier. Following this, the results of the classification process are conveniently presented in a DataFrame (Fig. 11), with each row displaying a question string, the corresponding intent it has been mapped to and the classifier's level of confidence (ranging between 0.0 and 1.0) regarding its decision to assign it to that specific intent. Lastly, a data analysis takes part, where the data is manually checked if each question is correctly mapped to the intent it is intended to represent. The reasoning behind the manual process' decision-making is not important to understand here and so it is further described is in the following Assessment section. During the analysis the correctly classified questions per a certain intent form a "correctness threshold" for that intent, which exists between the question with the lowest and the one with the highest confidence score out of all "correct" ones. Within this threshold fall only those correctly classified questions whose confidence level is considered high enough based on the average confidence score in that intent group. After the analysis process each question from each intent's threshold is removed from the dataset (Fig. 12) and the resulting DataFrame is converted back into a file format. This decreases the working dataset's size so that for their next iteration the topic modelling and classification algorithms only work with questions that are yet to be properly classified. The cycle then repeats itself, starting from the topic modelling for the updated dataset file and ending with the data classification and analysis process, until the file containing the dataset becomes empty, which indicates that each question in it has been properly classified by intent.

6.4. Assessment

The goal of the technical deliverable was to show in practice if and how a program can independently process a large amount of data and sort its contents out by certain similarities between different data chunks in the set. This has certainly been achieved despite the few shortcomings that lie in the method used to accomplish this task.

Perhaps the most critical flaw in this method is its significant dependency on external manual data processing. Its existence implies that the entire process is not fully automated and so the program cannot completely reach its goal all by itself. Due to the natural limits of human speed and work productivity, this means that the whole classification process is expected to be slower than in the case where the training data and classified data are automatically analyzed and updated. Despite this, manual data processing allows for a more thorough analysis than a machine might be able to do independently (at least without requiring several training iterations), potentially leading to more precisely defining the required updates than need to be made in the specific data.

The benefits of manual data processing regarding the training data can be seen in the freedom of choice for number of intents and examples per intent. In this particular dataset 10 intents have been recognized with the majority of the examples belonging to the intents related to measures, symptoms and COVID-19 spread. In hindsight, the amount of examples for some intents should have been more than it actually is, not only to balance the overall example number between intents but also to contribute to the improvement of the intent classifier's precision. In theory, this imbalance should affect the manual data analysis negatively afterwards since it can cause confusion regarding the correctness of a question's classification. In practice, however, the decision-making process for correctness, which consists of manually recognizing a question's actual most probable intent and comparing it to the intent it was assigned by the classification algorithm, has proven to be more resilient in this scenario. This is very likely due to the fact that it abstracts itself from external information and depends only on the choices made by the user responsible for deciding the question correctness.

A problem with the final results of the program are the occurrences of "senseless" questions. These are the types of questions for which the presence of context is necessary to understand their meaning, for example one-word questions like "really?". These kinds of questions exist usually because they are found in replies to other Tweets for which the rest of the questions in the dataset are unable to provide context themselves. As a result, it can be very challenging or even impossible to identify such a question's intent correctly. Unfor-

tunately this problem was noticed very late in production and because of that no measures have been taken to prevent it from happening. However, in theory, this could have been solved by manually looking through all the questions, identifying those that are ”senseless” and removing each one of them.

Overall, the technical deliverable manages to satisfy most of the needed requirements and produce the expected results. It clearly proves that a program is indeed able to classify large datasets by a certain type of criterion, albeit not entirely independently. It is likely that the entire process is quite possibly able to be fully automated once enough manual training has been provided for its algorithms. However, this would require fixing some of the discussed flaws and problems as well as more time for further work. Nevertheless, the program still accomplishes its main goals, which is enough to deem it as a success.

Acknowledgment

The authors would like to thank the BiCS management and education team for the amazing work done.

7. Conclusion

To sum up, the project most definitely manages to satisfy its defined requirements and live up to the expected results. Therefore, it can be considered successful, despite the encountered problems described earlier. As it has already been mentioned, these flaws are certainly removable and, once enough necessary related knowledge is acquired, the entire technical process can be optimized to run more efficiently. But even in the current situation these imperfections do not pose a large issue, since the project is still very much able to accomplish its main goal - to provide useful insight into the overall feasibility of algorithm-based classification of data.

References

[1] Tanya Stivers, N.J. Enfield, ”A coding scheme for question–response sequences in conversation”, Journal of Pragmatics 42 (2010) 2620–2626

[2] Chen, N.; Zhong, Z.; Pang, J. ”An Exploratory Study of COVID-19 Information on Twitter in the Greater Region”, Big Data Cogn. Comput. 2021,5,5. https://doi.org/10.3390/bdcc5010005

8. Appendix

```
with open('COVID19-GR-0122-0605.pkl', 'rb') as f:
    data = pickle.load(f)
id_list = data.values.tolist()
```

Fig. 1. Binary file to ID list conversion

Fig. 2. Setting up a Tweepy client

```
def get_lang_detector(nlp, name):
    return LanguageDetector()

spacy.cli.download('en_core_web_md')
nlp = spacy.load('en_core_web_md')
Language.factory('language_detector', func=get_lang_detector)
nlp.add_pipe('language_detector', last=True)
```

Fig. 3. Creation of English language detector

```
texts = []
for i in range(100,20000,100):
    tweets = client.get_tweets(id_list[i-100:i])
    for j in range(len(tweets.data)):
        tweets.data[j] = str(tweets.data[j])
        tweets.data[j] = tweets.data[j].replace('\n', ' ')
        doc = nlp(tweets.data[j])
        if doc._.language['language'] == 'en' and doc._.language['score'] >= 0.9:
            texts.append(tweets.data[j])

df = pd.DataFrame(texts, columns=['text'])
df
df.to_csv('tweets.csv', sep='\t', index=False)
```

Fig. 4. Tweet extraction by ID (note that the Tweets are initially stored in a .csv file for later use)

```
df = pd.read_csv('tweets.csv', sep='\t')
tweets = df.text.tolist()
questions = []
for tweet in tweets:
    doc = nlp(tweet)
    for sent in doc.sents:
        for token in sent:
            if token.shape_ == '?':
                questions.append(str(sent).strip().lower())
questions = list(dict.fromkeys(questions))
questions = {'text': questions}
df = pd.DataFrame(questions)
df.to_csv('questions.csv', '\t', index=False)
```

Fig. 5. Question detection and extraction (note that the questions are initially stored in a .csv file for later use)

```
df = pd.read_csv('questions.csv', sep='\t')
questions = df.text.tolist()

keywords = [[] for item in range(len(questions))]

pos = ['NOUN', 'ADJ', 'VERB', 'ADV', 'PRON']
for question in questions:
  doc = nlp(question)
  if len(doc) > 1:
    for token in doc:
      if token.pos_ in pos:
        keywords[questions.index(question)].append(str(token))
```

Fig. 6. Creation of keywords list

```
dictionary = corpora.Dictionary(keywords)
corpus = [dictionary.doc2bow(word) for word in keywords]
```

Fig. 7. Corpus initialization for topic modelling

```
from gensim.models import ldamodel
lda = ldamodel.LdaModel(corpus, num_topics=5, random_state=1, iterations=300, passes=10)
```

Fig. 8. Topic modelling function

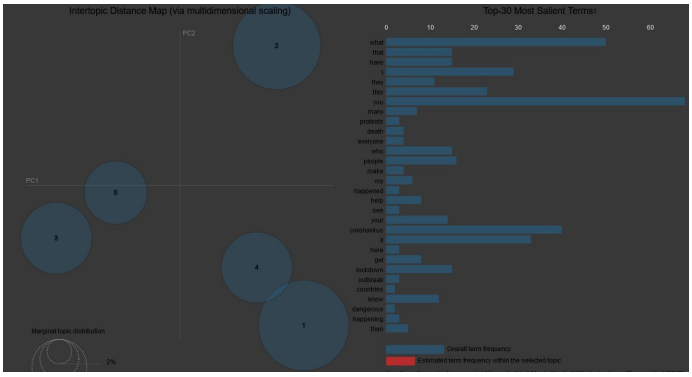

Fig. 9. Visual representation of topic modelling results (each topic cluster can be selected to display its particular set of most common keywords)

```
training_data = load_data('training_data.json')
trainer = Trainer(config.load('config.yaml'))
trainer.train(training_data)
model_directory = trainer.persist('/content/')
interpreter = Interpreter.load(model_directory)
```

Fig. 10. Training data configuration and interpreter initialization

```
intents = []
confidence = []
for question in questions:
  intents.append(interpreter.parse(question).get('intent').get('name'))
  confidence.append(interpreter.parse(question).get('intent').get('confidence'))
df['intent'] = intents
df['confidence'] = confidence
df
```

	text	intent	confidence
0	rt @pokrath: q: what is the missing word in @w...	self protection	0.317613
1	malaysian nctzens/wayzenni, are you interested...	self protection	0.208597
2	how many miles is that wall supposed to be?	self protection	0.262183
3	the pathetic thing here is "if china has not s...	self protection	0.216824
4	but seriously, how are they going to clean you...	symptoms	0.310260
...	...	...	...
485	droogies?	self protection	0.283520
486	rt @protohedgehog: "copyright on the health of...	self protection	0.266717
487	1 wildcard??...	symptoms	0.212978
488	it may not be a smart long-term economic stimu...	self protection	0.192494
489	rt @billbrowder: what is the us doing acceptin...	self protection	0.329363

490 rows × 3 columns

Fig. 11. Intent classification applied onto the set of questions (including sample DataFrame output)

```
df.drop(df[(df.intent == 'coronavirus spread') & (df.confidence >=0.33) & (df.confidence <= 0.43)].index, inplace=True)
df
```

	text	intent	confidence
0	rt @pokrath: q: what is the missing word in @w...	self protection	0.327402
1	malaysian nctzens/wayzenni, are you interested...	symptoms	0.201581
2	how many miles is that wall supposed to be?	self protection	0.242887
3	the pathetic thing here is "if china has not s...	self protection	0.194905
4	but seriously, how are they going to clean you...	symptoms	0.305482
...	...	...	...
485	droogies?	self protection	0.303077
486	rt @protohedgehog: "copyright on the health of...	self protection	0.209190
487	1 wildcard??...	symptoms	0.218849
488	it may not be a smart long-term economic stimu...	measures announcement	0.236920
489	rt @billbrowder: what is the us doing acceptin...	self protection	0.270829

484 rows × 3 columns

Fig. 12. Example of removal of correctly classified questions within a correctness threshold (notice the reduced number of rows in the resulting DataFrame, compared to the one in Fig. 11)