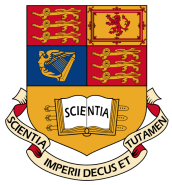


Imperial College London

“Finding the assay - natural language processing for biomedical scientific literature”

Project 1 thesis, Data Science stream

Master of Research in Biomedical Research (Data Science)

14th of March 2021

Supervisors: Joram M Posma, Tim Beck

Author: Filip Makraduli

Acknowledgments

I would like to thank my principal mentor Joram M. Posma, and secondary mentor Tim Beck for their tireless support and thoughtful insight throughout this project. Also, Yan Hu, Joy Li and the other members in our research group.

All code can be accessed at https://github.com/fm1320/IC_NLP.

The link for the website is:

https://share.streamlit.io/fm1320/ic_nlp/main/streamlit_appv1.py

Sometimes there is a possibility that the hosting service shuts down or there could be a server error. If this is the case, or for any other questions, feel free to contact the author of this report at: f.makraduli20@imperial.ac.uk

Originality statement

‘I hereby declare that this submission is my own work, product of my own work, conducted during the current year of the MRes in Biomedical Research at Imperial College London.
To the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at Imperial College London or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at Imperial College London or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.’

Abbreviations

MWAS - Metabolome-wide association studies
GWAS - Genome-wide association studies
NLP - Natural language processing
SVM - Support vector machine
RBM - Reduced boltzmann machines
RNN - Recurrent neural network
LSTM - Long short term memory (network architecture)
GPT - Generative Pre-trained Transformer
BERT - Bidirectional Encoder Representations from Transformers
PMC - PubMed Central
NER - Named entity recognition
NMR - Nuclear magnetic resonance
CNN - Convolutional neural network
SGD - Stochastic gradient descent
CSV - Comma separated values (file type)
JSON - JavaScript Object Notation (file type)
NLI - Natural Language Inference
MNLI - Multi-(Genre) Natural Language Inference
LC - Liquid chromatography
MS - Mass spectrometry
IAO - Information Artefact Ontology
OBI - Open Biomedical Investigations

Table of contents

Abstract

Assays are a key element in many biomedical research studies, notably in MWAS (Metabolome-wide association studies) and GWAS (Genome-wide association studies). Assays convey the information about reproducibility and replicability of the studies, as it is important to know with what technology data was acquired. These aspects are crucial in any scientific research. Furthermore, data curation and database creation efforts are being made to classify GWAS and MWAS in databases. Accelerating these efforts of database creation is important too because of the large number of studies published in the domain. Natural language processing techniques can be utilized to automatically classify these scientific papers. In this thesis, with the help of biomedical domain experts, a list of assays was compiled and used as a reference. The main aim was to create a robust, adaptable, and easy to use text-processing pipeline tool that would recognize assays in text, but also serve as a good starting point for the implementation of other algorithms and entity recognition tasks. In the thesis, this was achieved by performing data annotation and data preprocessing of over 2500 biomedical articles, exploring and implementing both string matching and deep learning-based models for named entity recognition, and using transformer-based architecture to augment the usefulness of assay recognition. Everything was deployed to a working ready-to-use public website. In this project, it was hypothesized that deep learning algorithms would perform better in named entity recognition due to their ability to generalize. However, the results did not prove this hypothesis, and it was shown that simple regular expression and string matching algorithms can perform better and be more adaptable and easier to use for the task of named entity recognition. It was also revealed that using more complex transformer-based and deep learning algorithms can be of better use in other natural language processing tasks such as summarization, question answering, natural language inference, and topic modeling. These findings also opened up a lot of possibilities for future work in natural language processing for biomedical texts.

1. Introduction

1.1 Background and related work

Machine learning models became available to bigger audiences as the result of the developments in computer hardware in the early twenty-first century. This was, to a great extent, due to the accessibility and price of computing power which became cheaper thus allowing machine learning algorithms to be tested and explored. Such developments can be attributed to better computer hardware design, computer chips were able to process more and more operations and got smaller and smaller in size. A pivotal moment was the design of the graphical processing unit (McClanahan, 2010), a hardware architecture still being used today. In the past, before these computing resources became widespread, deep learning algorithms were mostly just theoretical concepts in mathematics, the applied and experimental nature of these subjects was pretty much non-existent. Only certain machine learning paradigms such as SVMs (support vector machines) were in popular use. Concepts like neural networks and perceptrons were present, but not in broader use. RBM (reduced boltzmann machines) too popularized the take off of deep learning. The wider usages and opportunities to gain empirical knowledge about the deep learning aspect of machine learning proved to be crucial in the development of these fields and the birth of Data Science. This rise of various machine learning methods had spread to many industries and research areas. One such area of application is the biomedical sciences where these methods can solve different problems. A popular field of research and application is natural language processing (NLP). Natural language processing stems from human curiosity to not only process text but understands human language. During the early years of machine learning and artificial intelligence, understanding human language was deemed a very difficult task. At the beginning of the 20th-century, natural language processing revolved around simple prediction tasks.

A development in natural language processing was the “word-to-vector” algorithm in 2013 (Mikolov et al., 2013). During these years most of the state-of-the-art research in machine

learning had been centered in computer vision, but this discovery of the word-to-vector algorithm was the beginning of the rise of natural language processing. It is the first paper that tries to contextualize human language in terms of understanding relations between words. The meaning of this is that vectors that signify a similar word would also have a similar numeric representation. For example, the word “dog” and the word “cat” would have high similarity values compared to “waterfall” and “carrot”. This architecture was a step in the right direction in understanding human language, but human language and sentiment are too complex to only be communicated through words. A lot of the meaning in language is represented with sentence structures, metaphors, idiomatic expressions, and the majority of these meanings cannot be deduced from only words. Efforts to tackle this problem were made using Recurrent Neural Network (RNN) architectures (Liu et al., 2016), which can encode temporal information to a certain degree. This had helped in the understanding of sentence structures but there were still limitations. Most notably, situations where words at the beginning of the sentences referred to ones in the end. An improvement was the introduction of Long Short Term Memory (LSTM) networks which enabled “memory”. This meant there was now a way to encode information from words at the beginning of the sentences so they could later be remembered about other words.

These networks were quite successful for certain tasks in NLP and are still used today in a lot of applied or industry-related projects. This architecture was first introduced in the 1990s (Hochreiter & Schmidhuber, 1997), but only decades later was the scientific community able to widely use it in NLP tasks, due to the limitations of computing power already mentioned in the introduction above. The key feature of these networks (RNN and LSTM) is the sequential processing of words. These networks process the text word by word, and each encoding of a word is dependent on the encoding of the previous word. The encoding of the previous word is called a hidden state. The difference between RNNs and LSTMs is that the latter has a characteristic where the hidden states influence words further in the sentence. An obvious problem with this architecture is the fact these hidden states deteriorate over time. This means

that words at the beginning of the sentences lose their influence towards the end of the sentences. There are ways to minimize this effect and one option is to use bi-directional architectures. One such example used in one of the discussed solutions in this paper is the BiRNN architecture (Schuster & Paliwal, 1997) presented in figure 1.1.

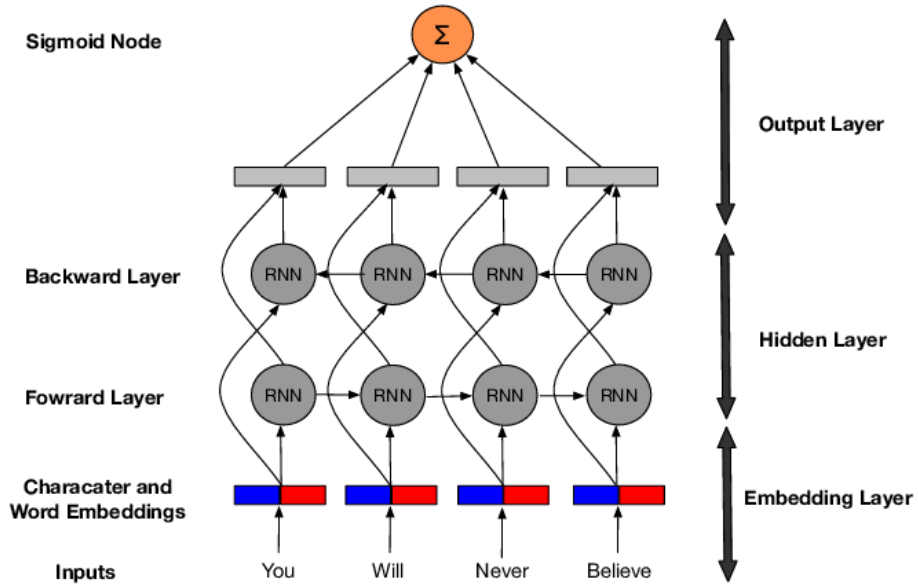


Fig 1.1- A bidirectional recurrent neural network architecture,
source: Anand, Ankesh & Chakraborty, Tanmoy & Park, Noseong. (2016).
We used Neural Networks to Detect Click baits: You won't believe what happened next!.

In this neural network, the sentences are encoded from both directions, end to beginning and beginning to end, this enables a more generalized understanding of the words which better models language. Also, before this encoding happens, word embeddings are calculated using algorithms similar to the already mentioned word-to-vector algorithm.

A major advancement in the NLP world came with the introduction of Transformer architectures in late 2017. The main goal of this architecture is to avoid the recurrent nature of the RNNs and LSTMs and instead process text in a parallel manner instead of the sequential step-by-step manner. This meant that Transformer networks do not exhibit the problem of vanishing influence

of words. Transformers work using a three-step process. First, the words in a sentence are processed in parallel feeding the whole sentence or passage in the model. Then, positional embeddings are assigned to words, to signify their position in the sentence. Third, the similarity between the words is calculated using a mechanism which the authors have called self-attention (Vaswani et al., 2017). This mechanism additionally helps the algorithm understand which part of the sentence is important and which is not. Transformers has proven to be a state-of-the-art solution for many NLP tasks, usually in tasks that require broad language modeling such as summarization, translation, question answering, etc. Using the concepts of this paper, a lot of other popular transformer-based algorithms have been developed, most influential being BERT, GPT, XLNet (Devlin et al., 2019),(Brown et al., 2020),(Yang et al., 2020). One negative aspect of these architectures is their data “hungry” nature. This is evident if we take a look at the computing resources needed to achieve state-of-the-art results in all the papers that involve transformers. A lot of processing power and data is needed to train these architectures from scratch. Efforts have been made to mitigate this problem with various architectures like DistilBERT and Reformer (Sanh et al., 2020),(Kitaev et al., 2020) that try to make transformers less costly to replicate. A paper worth mentioning is the T5 transformer - Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer (Raffel et al., 2020) which eases the task of pre-training and fine-tuning transformers by making a lot of NLP tasks a text-to-text architecture. To put it simply, instead of training a few architectures for various tasks, only one architecture is trained and then the same one is fine-tuned for different tasks. No need to use one model for summarization and another for question answering. This improves the versatility of using transformer architectures, and a pre-trained model of this architecture is used for a summarization task in this report. Most results in deep learning and machine learning are empirical. This means that besides the theoretical study of new algorithms and the chase of “state-of-the-art” results, there is substantial progress to be made in the applied areas of this science. One prominent example is Data Science applied in the area of biomedical sciences. In

these sciences, machine learning and deep learning methods need to be adapted to specific tasks suited for each field with unique terminology.

Many of the state-of-the-art NLP algorithms are computationally intensive and require lots of data to train. This makes the entry margins of their reproduction quite narrow. Also, for a lot of scientific use cases, scientists want to be able to try things from scratch and not depend on a ready-made solution. Another difficulty with biomedical research papers is the sheer number of studies conducted with millions of papers into publishing right now. A robust and easy-to-use processing tool, which would filter out these papers according to certain characteristics, would tackle this difficulty. Another reason for such a tool is the efforts in GWAS and MWAS scientific communities in making scientific paper databases and logs. Natural language processing techniques have high applicability in such a situation.

Problem statement introduction

In this report, the main task is automatic named entity recognition (NER). This is a common task in the realm of NLP and has been researched thoroughly in the world of NLP. Many NER datasets have generic entity labels like people, companies, countries, etc. Many solutions in this area with these datasets achieve state-of-the-art results. However, when specific case entities in a given domain are discussed, these generic systems don't perform as well. For unique applications, like the biomedical domain in this report, certain alterations, tuning, and data-preprocessing are needed for a working solution. Framing the problem of this report as a NER task is done because once it is solved, it can be used to perform other NLP tasks where specific domain knowledge plays less of a role. Also, more complex algorithms can be applied on top of NER, which would augment the usefulness of solving this task. For example, once the desired entities are recognized, it is possible to do other NLP tasks including, but not limited to: contextual summarization, question answering, topic classification. This is why the work in this report is centered around NER, but it does not limit itself to only NER. The

solution would enable scientists to cycle through the results of millions of research papers and categorize them by topic, style, results, expertise, etc.

The approach was to use specific scientific domain knowledge of the subject to implement an NLP solution that can be computationally efficient, reproducible, and requires less data to train. In this case, the entity that is being recognized is what type of laboratory assay is used in the study. Assays are analytical or laboratory procedures used to recognize, quantify or detect a compound, element, or substance. They are usually used in sciences related to physics, chemistry, biochemistry, and pharmacology. Prominent examples here are spectrometry and spectroscopy techniques such as Proton nuclear magnetic resonance (PNMR), Liquid chromatography-mass spectrometry (LC-MS), Fourier-transform infrared spectroscopy (FTIR), Raman spectroscopy, etc. Used in the context of GWAS and MWAS articles, assays are crucial for their categorization (Bush & Moore, 2012), (Goodwin et al., 2016). One of the most important aspects of these studies, but also science in general, is the ability to accurately reproduce and replicate the generated results.

By understanding which assays are used, researchers know which papers are closely related to their work thus enabling filtering by their preferences.

Hypothesis

1. The **first hypothesis** is that for a task like entity recognition, simpler (RNN, LSTM, or CNN based) deep learning models will outperform string matching/regular expressions-based algorithms because of their ability to generalize.

2. The **second hypothesis** states that these simpler deep learning models also perform better compared to more complex architectures (transformers) that require a lot of compute and large training sets.

Aims & Objectives

The estimate is that the task of named entity recognition of assays can be executed to a satisfactory degree of accuracy and reproducibility with the combination of deep learning and rule-based approaches while also exploring other NLP tasks in efforts to create a scientific text processing pipeline that would aid scientists in their search and curation needs. Identifying assays is a central task of this report, but other NLP tasks are also explored. Also, simple string matching-based algorithms are explored, as some have been used in the context of medical sciences (Lovis & Baud, 2000), as predecessors to complex deep learning algorithms.

- **The first aim** is to preprocess and extract relevant data from 2500+ research articles and papers from various scientific journals.

    An objective is to use the AutoCORPus library (Hu et al., 2021) for the extraction.

- **The second aim** is to pre-process the data in ways to be suitable as an input for the named entity recognition algorithms.

    To simplify the process of replicating our model, an automatic annotator tool was created. This was the main **objective as part of the second aim**. The annotation of unlabeled data is an issue in many data science applications. Recognizing this need, a data annotator using regular expressions was created. This significantly increases the speed of deploying a named entity recognition enabling users to start creating their model with their data if desired. This annotator can be upgraded with a manual annotation tool but serves as a great starting point for this type, and other types of tasks.

- **The third aim** is to train (1) deep learning and (2) rule-based string matching able to solve the NER task to a satisfactory level of accuracy while also maintaining the ability to replicate and modify the models.

The objective was to explore different deep learning models and choose a model that is easy to adapt and deploy while not compromising accuracy. Also, good documentation with a variety of pre-trained options is available. That is the reason for the choice of Spacy v2.2.4

- **The fourth aim** was to expand and augment the usefulness of named entity recognition by leveraging that information for more complex problems.

  An objective here is to use transformer networks as a proof of concept that they can be used on top of other tasks to expand the information conveyed by named entities.

  In the end, other NLP tasks like summarization are added, and as a sub-aim, everything is deployed to an interactive web app for users to get a practical feel of the algorithms. It is important to note that recently, transformer-based architectures have been a “go-to” solution for any NLP task.

This report presents an NLP pipeline for biomedical text processing, centered to laboratory assay recognition while also keeping into consideration the necessity of replication, efficiency, and ease of access in the scientific community.

2. Data and models

2.1 Data

2.1.1 Data source

Data were obtained from almost 2500 scientific papers in various peer-reviewed medical journals in the area of biomedical sciences, predominantly scientific papers in genomics, metabolomics, and cancer medicine. More precisely there are 1200 PubMed Central (PMC) full-text publications of GWAS (Genome-wide association studies) whose summary level data is incorporated in GWAS-Central and 1241 PMC full-text publications on MWAS (Metabolome wide association studies).

2.1.2 Data pre-processing

Throughout this project, the programming language is Python for all the tasks. Besides the functionalities of the native Python language, other Python libraries and packages are used. They will be covered in more detail for each area of use accordingly. The text from the scientific papers was not used fully, but sections of interest were extracted. In this case, the sections of interest were those where laboratory assays are mentioned. These sections have headers such as: “Materials and methods” or “Methods” depending on the journal. As the title of these sections can have many variations of the words “materials” and “methods" in the title, a Python package was used in solving the problem. It uses a combination of rule-based matching synonyms from the Information Artefact Ontology (IAO) and a digraph model to predict the most likely header for a given section. The Information Artefact Ontology (IAO) is an ontology of information entities in Biomedical investigations. This ontology was created and is maintained by the Open Biomedical Investigations (OBI) consortium. The Python package is called Auto-CORPus (Hu et al., 2021) and was developed by students from the same research group where this project is. Here it has been applied to relevant research papers. The package, once configured and run on a dataset, outputs three folders: “abbreviations”, “main text” and “tables”.

This means that for each paper that is processed, three JSON text files are generated: a file of the abbreviations in the text, a file of the main text divided into sections, and a file of the tables. Here, only the JSON file for the main text is used. In this folder JSON files for each scientific paper are generated and text is divided into sections by tracking IAO (Information Artefact Ontology) terms for each section. These terms make the division of paragraphs easier as they provide universal identifiers for the sections and section headings. The output JSON file format is presented in image 2.1.1.


```
"paragraphs": [
 {
  "section_heading": "Abstract",
  "subsection_heading": "Abstract",
  "body": "Advanced glycation end-products (AGEs) are a
  "IAO_term": [
   "textual abstract section"
  ]
 },
 {
  "section_heading": "Introduction",
  "subsection_heading": "Introduction",
  "body": "Advanced glycation end products (AGEs) are a
  "IAO_term": [
   "introduction section"
  ]
 },
 {
  "section_heading": "Introduction",
  "subsection_heading": "Introduction",
  "body": "While there are numerous association studies
  "IAO_term": [
   "introduction section"
  ]
 },
```


2.1.1(a) - Main Text output file


```
{
 "title": "Genetic Analysis of Advanced Glycation End P
 "abbreviations": {
  "DHS": "Diabetes heart study",
  "AGE": "Advanced Glycation end product",
  "ELISA": "Enzyme linked immunosorbant assay",
  "T2D": "Type 2 diabetes",
  "SNP": "Single nucleotide polymorphism",
  "GWAS": "Genome Wide Association Study",
  "eGFR": "Estimated glomerular filtration rate",
  "HbA1c": "Glycosylated hemogobin",
  "SOLAR": "Sequential Oligogenic Linkage Analysis",
  "BMI": "Body mass index",
  "CVD": "Cardiovascular disease",
  "RAGE": "Receptor of advanced glycation end products"
 },
```


2.1.1(b) - Abbreviations output file

This enables filtering by IAO terms. For assay recognition in this report, the relevant sections that are extracted from the papers are the ones where the materials and methods are discussed. They can usually either be found either after the introduction or at the end of the paper, but are not always in the same location. This points to the fact why a python package is needed and why standardization is important. This point holds for all MWAS and GWAS papers. By locating areas of interest, less text is processed. This saves computing power and makes the text more information-dense because only parts where assays are mentioned and discussed are sampled.

2.1.3 Data annotation

One recurring issue in NLP but also machine learning, in general, is the low availability of annotated datasets. The annotation step is essential for designing deep learning algorithms but it is time-consuming and a tedious task. This step is a major hurdle for developing machine learning applications - even prototypes. In this project there was no annotated data available, hence solving the problem of data annotation was instrumental. A situation like this is an issue before creating a deep learning model of any kind. One common approach is doing the process of annotation manually, where humans label training data by hand. Many tools assist this process, such as Prodigy (*Prodigy · An Annotation Tool for AI, Machine Learning & NLP*, n.d.) and (*Doccano/Doccano*, 2018/2021). While this process is a viable solution for many applications, manual annotation takes a lot of time and needs to be done from the beginning for each new task (Neves & Ševa, 2021). Manual annotation generates good quality examples, but for simpler tasks like Named entity recognition, having a system that automatically does the process of annotation can be an advantage. Having these problems in mind, as part of this project, an automatic annotation method was designed by us. It offers:

- Quick annotation of thousands of examples in a matter of minutes
- Robust and easy to use and/or implement for any entity type and number
- Useful with different annotation formats

These annotations were created to be used with the python library Spacy (Honnibal & Montani, 2017). In the later chapters of this report, the deep learning aspect of this library will be discussed. This library mainly uses two annotation schemes, spacy annotations, and beginning inside-outside (BIO) annotations.

The annotator in this paper can output all three types of annotated data. The steps for data annotation are as follows:

1. A list of laboratory assays was compiled by experts and researchers in the areas of MWAS and GWAS. This list is used for annotating the assays as entities but it is also in the “Models” section where it is further referenced. This list is easily updatable, appendable, and modifiable. It is saved as a comma-separated (CSV) file and available in Appendix A.
2. The text is imported and relevant sections of interest are extracted with the Auto Corpus package as explained above. Afterward, as per Spacy’s data annotation requirements, the text is divided into sentences. This can be done either using various rule-based approaches or unsupervised learning techniques (Kiss & Strunk, 2006). In this report, both a Spacy pipeline component for rule-based sentence boundary detection and a regular expression rule-based approach was used (Regular expressions are explained more thoroughly in the next chapter but also Appendix A).
3. The entities that need to be recognized are saved to a .csv file and handled using the Pandas python library. This is done because Pandas works well with large files posing fewer limitations in the .csv file size. Afterward, string matching techniques are used to locate the entities and the character range of the entity match counted from the beginning of the sentence. This is done using the regular expression library re in python. One problem that arises in this step is the situation of overlapping entities. For example, the entity “1H-NMR Spectroscopy” would be recognized as multiple inter-entity matches (depending on how many are mentioned in the .csv file): “1H-NMR”,” NMR”,” NMR Spectroscopy”, etc. This will generate overlapping character spans for each match and this is against the annotation rules of the Spacy library. An example of this situation is shown in figure 2.1.3.1. (Source: algorithm is applied to a dummy sentence)

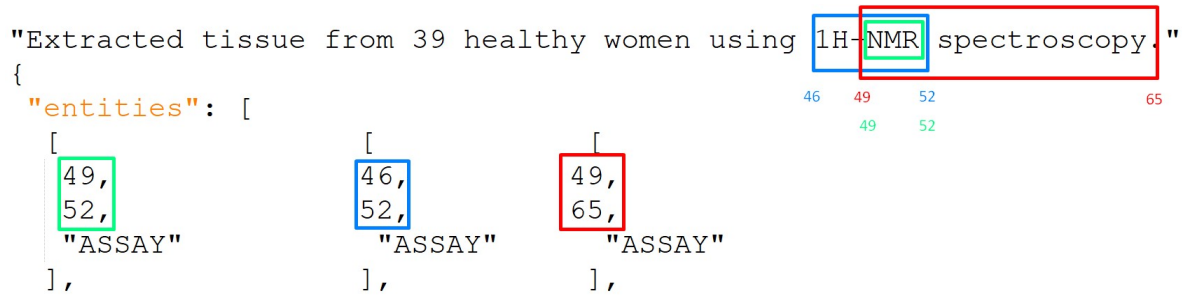


Figure 2.1.3.1 - An illustration of the problem of overlapping entities

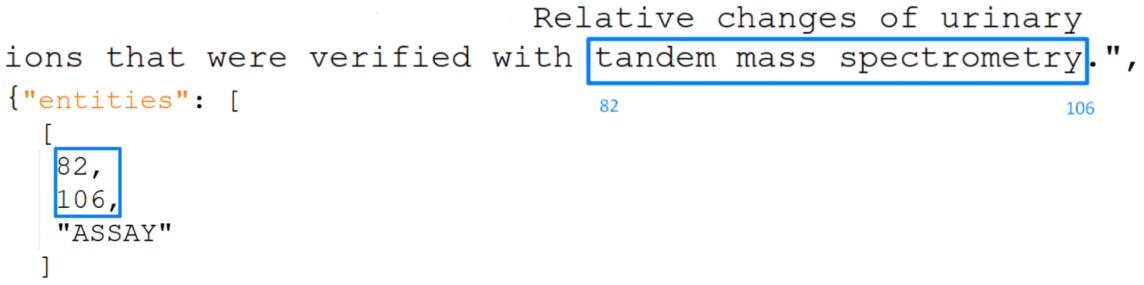
This is solved by using a statistical method called the Jaccard index (Jaccard, 1901). This is a method that calculates the similarity between two non infinite sets using the ratio between the absolute values of the intersection(∩) and union(∪) of two sets. This method is also known as the Tanimoto index. The mathematical expression for this method is illustrated with the equation:

$$J(A,B) = |A \cap B| / |A \cup B|$$

The index ranges between $0 \leq J(A,B) \leq 1$meaning that an index of 1 is calculated when there is an overlap, and 0 when there is no overlap. This method has the advantage of also calculating the level of similarity. The index can have values between 0 and 1, and the closer the value is to 1, the higher the similarity between the sets. Set is a distinct collection of elements. To utilize the Jaccard index, each match is assigned to a set where the first element is the first character matched and the last element is the last element matched. Following the example in figure 1.3.1, the following sets can be created: *"NMR"={50,51,52}*, *"1H-NMR"={47,48,49,50,51,52}*, and *"NMR spectroscopy"={50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65}*. Once the character spans of the matches are in this format, the set operations specified in the equation above can be performed. This is implemented using native Python.

4. The final part of the data annotator is the specified output. Files are outputted in a JSON format for the spacy annotation scheme. Scripts are also available to convert this

annotation to BIO format or gold standard annotations. In picture 2.1.3.2 the snapshot of the output format is shown.



2.1.3.2 - Final output format where the model annotates “tandem mass spectrometry”

This automatic annotator procedure can serve as a starting point even for application beyond this report where the precise human manual annotation is needed. There it can serve as a baseline for judging the human annotations and provide valuable inputs for scientists.

2.2 Models

In this report, three types of model paradigms were tried. A rule-based model using regular expressions, a Spacy-based deep learning model, and a transformer-based model. The first two tackle the problem of named entity recognition directly, whereas the transformer-based models are used after the named entity recognition to augment the results and get more insight into the data. The idea is to use simple, quick, and versatile algorithms for the named entity recognition and only afterward use complex transformer-based networks. This is to leverage their power and suitability for more complex tasks that require contextualization. All code details are provided on the Github account of this project at www.github.com/fm1320/IC_NLP

2.2.1 Regular expressions and string matching model

Regular expressions are a sequence of characters that define a search pattern in strings. These regular expressions and string matching models were chosen because of the computational efficiency and easy adaptability compared to a deep learning model. Although Python-based regular expressions are slower than the Thompsons NFA algorithm (Appendix A), they are still better in terms of time complexity compared to deep learning models. Furthermore, if there is a change in the data the whole model can be easily adapted and there is no need for re-training. Besides the already mentioned regular expressions algorithm, a native string matching python algorithm is also used in this report. The python string matching algorithm works by using comparison operators at a bytecode level. In figure 2.2.1.1. a time complexity test carried out using the native python timeit library is presented. To emulate the use case in this paper, an example chosen was the recognition of “NMR" in the made-up sentence *“LC/MS is a popular technique, however more prevalent in the scientific community is the NMR”*. In this report, both string matching algorithms and regular expressions algorithms were used. Although python’s string matching is considerably faster, regular expressions were used for their features of non-literal string matching.

Algorithm:	Number of loops:	Time:	Best of how many:
Regular expressions	1 000 000	195 ***ns*** per loop	5 iterations
String matching	5 000 000	53.5 ***ns*** per loop	5 iterations

Fig 2.2.1.1 - Time complexity comparison of string matching and Regular expressions

The named entity recognition using these algorithms is performed in two ways.

1. The entity recognition is done by matching the entities needed to be recognized from the .csv file (mentioned in step 1. of “Data Annotation”) to the target text. In this case, python native string matching can be used as well as regular expressions. The matching returns all occurrences of the entities and also is case insensitive.
2. The entity recognition is done by extracting the whole sentences that have at least one of the entities. This is implemented only using regular expressions. Whole sentences are extracted because later on, they serve as an input to transformer summarization models. Also, sentences are visually intuitive for the user.

2.2.2 Spacy deep learning-based model

The deep learning model used in this project is Spacy’s NER model (Honnibal & Montani, 2017). This model is a variation of a residual convolutional neural network with bloom embeddings (Serrà & Karatzoglou, 2017). It is adapted for custom entity recognition, certain hyperparameters are tuned, and different pre-training models are tried. This NER deep learning model is one piece of a text processing pipeline with which Spacy operates. Before presenting the details of the deep learning model, Spacy’s library text processing structure is explained. Spacy version last tested with: v2.2.4 and compatible with: spaCy v2.1.0+. Spacy was chosen for its ease of upgrading, modifying, and reputation as a production-grade software library.

2.2.2.a Spacy library structure

The Language class, the Vocab, and the Doc object are the core data structures of spacy. To process a text and convert it into a Doc object, the Language class is used. It is stored as a variable called “nlp”. The sequence of tokens and all their annotations are operated by the Doc object. This general structure is shown in figure 2.2.2.1.

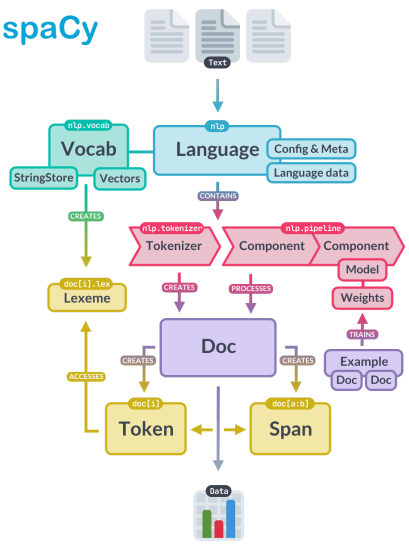


Fig 2.2.2.1 - Spacy library structure, source: Spacy's official documentation

The Doc object is built by the Tokenizer, and then can be modified by the pipeline components. These components are coordinated by the Language Object. It takes raw text as input and returns an annotated document via the pipeline. It also orchestrates serialization and preparation.

Spacy's processing pipeline can contain many components. The order of these components can be seen in figure 2.2.2.2. Before the application of any of the pipelines, the input text is tokenized with the "tokenizer" component. Tokenization means dividing a text into meaningful chunks. Spacy uses a tokenization rule-based algorithm that is specific to the language of the text. The NLTK v3.2.5 library also works well as a tokenizer. That is why, separate models exist for English, Spanish, Macedonian, etc. First, the text is split on whitespace characters, and then rules about prefixes and suffixes are checked to divide the text into tokens. The "tagger" and "parser" components are used to identify parts of speech and their relations with one another. In the case of this report, these pipelines can be used in conjunction with regular expressions. Since the assays that need to be recognized are nouns, instead of just dividing the text into sentences, as explained in the previous chapter 1. Data, the tagger, and parser can be

utilized to extract noun chunks and then the entity recognition can be applied to a smaller amount of text. This is a good choice if the user wants to apply a computationally intensive task after the noun chunk extraction. In the context of named entity recognition, it does not change the result evaluation, so this is left as a feature of the project.

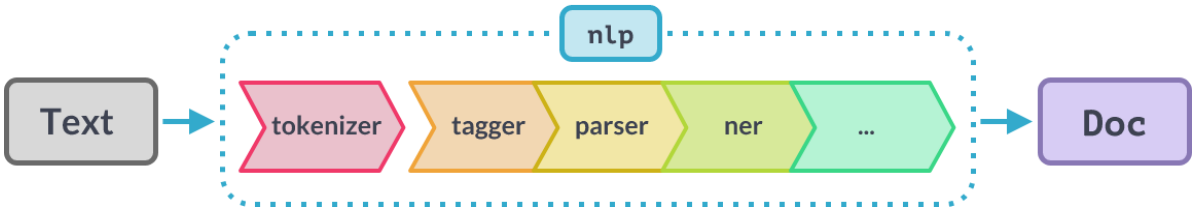


Fig 2.2.2.2 - Spacy pipeline structure, source: Spacy's official documentation

The "ner" component is the one of most interest. In this project, a custom "ner" pipeline component is added and trained on the annotated data from the previous chapters.

2.2.2.b Spacy named entity recognition

Spacy uses a named entity recognition system that combines multiple technologies. It uses something called "bloom" embeddings, which is a more compact way of embedding words. Furthermore, a residual convolutional neural network is used for encoding words. This combination of different concepts is in order to get high efficiency, accuracy, and adaptability. Spacy's NER model works according to these steps:

Embed: Embedding of words is done with bloom filters. This means that instead of saving words, the words are hashed and the hashes are saved in a dictionary. This makes word embeddings to have better compactness, but the downside is the possibility of some words having the same vector representations. Different default Spacy models have different amounts of words that are represented with the same vector.

Encode: Context of words is taken into account by encoding word lists to sentence matrices. Convolutional neural networks are used for the encoding. This step can be done by LSTMs or RNNs too. However Spacy's creators have chosen a Convolutional neural network and one of the reasons is speed (Honnibal & Montani, 2017).

Attend: The attention mechanism helps in understanding which parts are more information dense given a query. The output is a feature vector with all the important features included from the other vectors. (Similar to a weighted sum)

Predict: Spacy uses a multi-layer feed forward network for inferring to which class the word belongs.

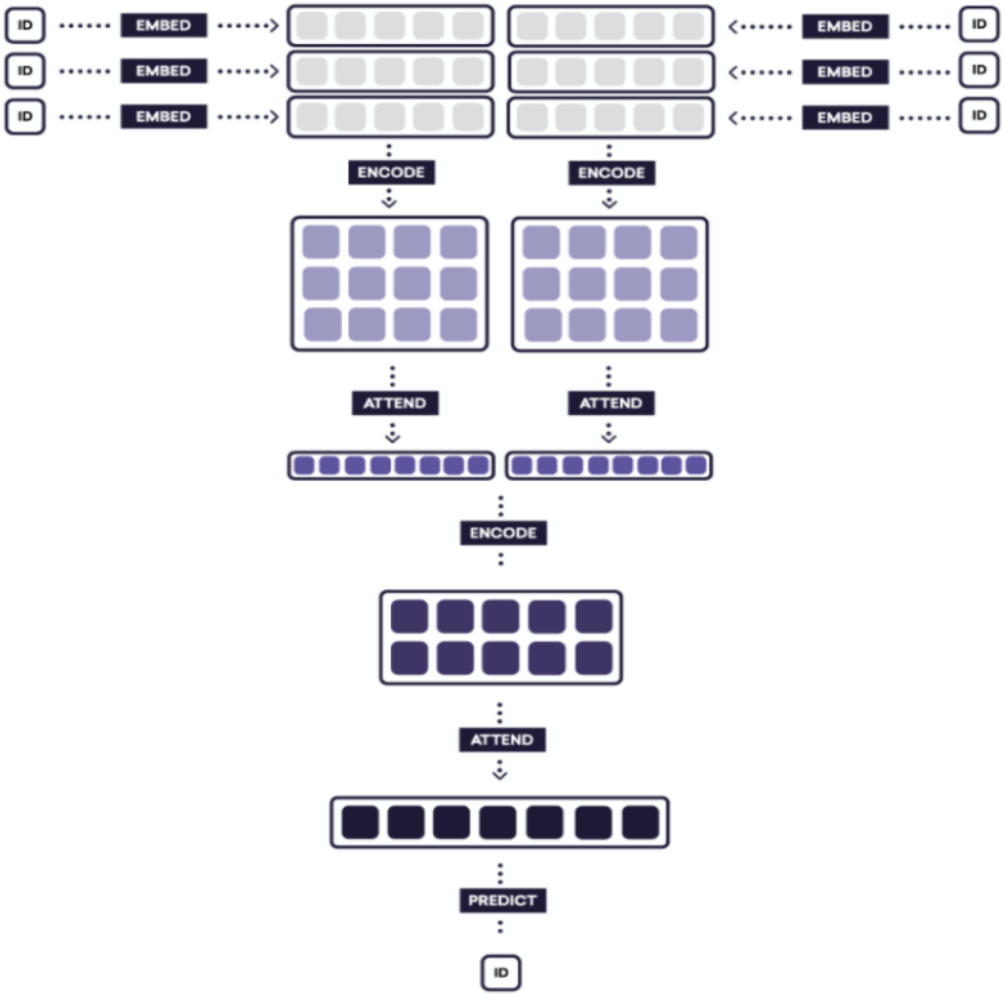


Fig 2.2.2.3 - The architecture of Spacy's NER model, source: Spacy's documentation

Spacy has 3 pre-trained English models: en_core_web_sm, en_core_web_md, en_core_web_lg. These models can predict the following default entities:

cardinal number, date , event, facility, geo-political entity, language, law, location, money, nationalities or religious or political groups, ordinal numbers, organisations, persent, person, product, quantity, time, work of art. For these named entity recognition tasks they perform quite well.

According to their tests (Honnibal & Montani, 2017), a reported accuracy of at least 85% in NER on the Onto Notes release 5.0 is achieved. (Weischedel, Ralph et al., n.d.). However, in the case of predicting custom entities, this level of accuracy is difficult to maintain.

2.2.2.c Custom named entity recognition

To leverage the abilities of transfer learning an approach, a pre-trained model was used. However, due to the occurrence of the “catastrophic forgetting problem” (McCloskey & Cohen, 1989). This is a problem when neural networks, when assigned to learn new tasks, “forget" what they have learned in the tasks before. There is no need to recognize default entities, the pre-training models had no previous knowledge of recognizing any entity. There are ways to circumvent the “catastrophic forgetting problem”, such as pseudo rehearsal (Atkinson et al., 2021), this was not needed in the context of assay recognition. It would be needed if multiple assays are recognized. The parameters tuning of the model was set-up following the recommendations in (Honnibal & Montani, 2017). The process can be summed up with the following settings:

Data and hardware

Almost the entirety of the data mentioned in the Data section has been used for training, a certain amount of publications (<50) from the domain were chosen for testing the model. This was done to train a model on the biggest amount of data available. The training hardware that was used was Google Colaboratory. This is a free service that offers GPU computation power in a Jupyter notebook editing interface. The types of GPUs available in Colab vary over time. The available GPUs include Nvidia K80s, T4s,

P4s, and P100s. RAM size of 12GB is included. As this is a free service, choosing explicitly which resources are available at a given point in time is not possible. The hardware provided was sufficient to train the models with an average training time of 64 min per 45 epochs.

The training was done by dividing the data into 7 batches with the first two batches trained for 80 epochs and the other 5 with 45 epochs.

Pre-training

The model weights were initialized trying each of the three Spacy trained English models: en_core_web_sm, en_core_web_md, en_core_web_lg. Afterward, experimentally, the best performing pre-trained model was chosen.

Parameters

Compounding affects the calculation of the loss function, it is advised that the values move from lower values to higher values. In this way, the loss is calculated on a few examples first and then slowly transitions towards calculations on more examples. This helps the model generalize better. The learning rate was kept as advised by the documentation at a value of 0.001. The **optimizer** that was used was Stochastic gradient descent (SGD). Due to the sparse nature of the text, SGD based optimizers often perform well in natural language processing tasks

Random shuffle is performed at the beginning of training to increase variance in the data and eliminate the chance for a bias towards a certain entity type

A dropout of 35% is used as a regularization technique and prevention of overfitting. The range between 20% and 40% can also be used as it yields similar results when tested in this project. It is important to note that the dropout is dependent on the size of the dataset. A high dropout percentage might inhibit the learning of the model. This was not the case in this report. In figure 2.2.2.4 these parameters are highlighted.

```
sizes = compounding(1., 16., 1.001)
# batch up the examples using spaCy's minibatch
for itn, i in enumerate(range(n_iter)):
    random.shuffle(TRAIN_DATA)
    batches = minibatch(TRAIN_DATA, size=sizes)
    losses = {}
    for batch in batches:
        texts, annotations = zip(*batch)
        nlp.update(texts, annotations, sgd=optimizer, drop=0.35, losses=losses)
    print("iter:",i+1,"Losses:",losses)
```

Fig 2.2.2.4 - Code snippet of tuned hyper-parameters

2.2.3 Transformer based models

Many previous studies have used transformers for named entity recognition and the task of fine-tuning them has been a common approach in scientific work. One notable example is the area of biomedical sciences is BioBERT (Lee et al., 2020). This was part of the reason why a different path was chosen in this project, approaching it in a different less explored way.

2.2.3a Natural Language Inference

Natural language processing is a very exciting area. Some fairly effective methods of learning from the vast amount of unmarked data available have been identified.

One big drawback of transformer models in NLP is their size. They have a large number of parameters and this makes them difficult to apply in practical scenarios, especially when annotated data is scarce. However, recent advancements have shown that language models encode a lot of information in their weights (Brown et al., 2020) and can perform well on downstream tasks with less specific training data. Or as quoted in (Petroni et al., 2019) "*The surprisingly strong ability of these models to recall factual knowledge without any fine-tuning demonstrates their potential as unsupervised open-domain QA systems".*

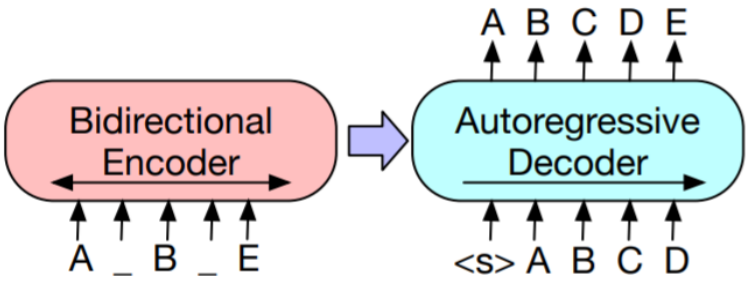
2.2.3b Zero-Shot learning

Zero-shot learning can broadly be defined as the “application of a model to solve a task for which it has received no specific training before”. A prominent example is (Radford et al., 2019) where the model is tested on tasks it hasn’t been trained in.

The model in use in this report does this by creating a pair of two words, a hypothesis, and a premise. Afterward, it is evaluated whether the pairs are a: contradiction(false), entailment(truth), or neutral. To understand multiple topics, a lot of word pairs are modeled. The idea is to take the sequence of interest and label it as the "premise" and turn each candidate label into a "hypothesis." If the model predicts that the premise "entails" the hypothesis, the label is taken to be true. This is used “out of the box” as proposed by (Yin et al., 2019) and using the Hugging face and Transformers python library. In this way, a multiple candidate label natural language inference is performed.

2.2.3c Usage

A zero-shot pipeline using the Bart-large-MNLI model (Lewis et al., 2019) is implemented. BART was used because Inputs to the encoder do not need to be aligned to the outputs of the decoder thus enabling the addition of noise. If this is done in other languages, it should be done with other models because BART is only trained in the English language.



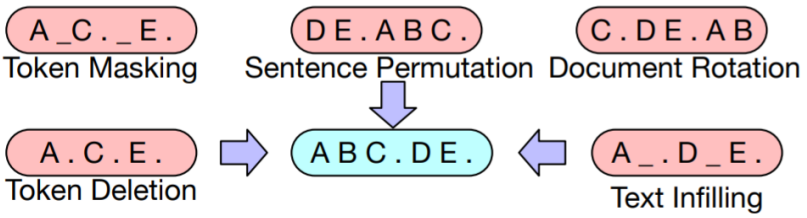


Fig 2.2.2.5 - Architecture and training sequence of BART, source: Lewis et al. 2019

This makes it easier to train the model for more tasks which means it has a less limited applicability to other transformer-based models. It is also suitable for zero-shot learning (Yin et al., 2019).

The extracted entities using the previously mentioned models are used as candidate labels for topic modeling in the biomedical text. In this way, named entities serve as topics to classify their relevance in the texts.

2.2.3d Summarization

The task of text summarization can be divided into two types of summarization:

- Extractive
- Abstractive

The former uses already existing parts of the text to summarize the meaning while the latter generates new text to summarize the meaning. In the report, extractive summarization was performed. Contextualized summarization of sections where entities are present or even summarization of whole paragraphs in scientific literature is a useful and user-friendly way to encode information about scientific entities.

This summarization is done by the gensim python library and the Text rank extractive text summarization algorithm (Mihalcea, 2004).

3. Results

3.1 Models results

In this chapter, the results in the task of recognition of assays are presented.

3.1.1 Regular expressions model results

This model was evaluated using a confusion matrix. Various metrics were also calculated from the confusion matrix. The testing text sequence was from the “Materials and Methods” section of 5 MWAS papers. The evaluation metrics are listed in figure 3.1.1.1

*True Positive (TP): Correctly predicting a label (predicted “1”, true value “1”); True Negative (TN): Correctly predicting the other label (predicted “0”, true value “0”), False Positive (FP): Falsely Predicting a label (predicted “1”, true value “0”), False Negative (FN): Missing and incoming label (predicted “0”, true value “1”).

	TRUE POSITIVES	TRUE NEGATIVES
PREDICTED POSITIVES	57	0
PREDICTED NEGATIVES	8	805

Measure	Value	Derivations
Sensitivity	0.8769	TPR = TP / (TP + FN)
Specificity	1.0000	SPC = TN / (FP + TN)
Precision	1.0000	PPV = TP / (TP + FP)
Negative Predictive Value	0.9902	NPV = TN / (TN + FN)
False Positive Rate	0.0000	FPR = FP / (FP + TN)
False Discovery Rate	0.0000	FDR = FP / (FP + TP)
False Negative Rate	0.1231	FNR = FN / (FN + TP)
Accuracy	0.9908	ACC = (TP + TN) / (P + N)

F1 Score	0.9344	F1 = 2TP / (2TP + FP + FN)
Matthews Correlation Coefficient	0.9318	TP*TN - FP*FN / sqrt((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))

Fig 3.1.1.1 - Confusion matrix and evaluation metrics for regular expression model, graphing source: onlineconfusionmatrix.com

The test set was in a total of 870 words of which 61 were words that are assays(positive class) and 809 non-assay words (negative class). The full test set is available in the repository. The confusion matrix is calculated based on how many of the 61 words that are included in an essay class the model got right. A metric of “full misses” was added. This includes the cases when a model entirely misses an entity or entirely spots a new false positive entity. Usually,transformer-based models guess the true positive class but don’t include all words that construct the entity. This model has: 0 full misses. In figure 3.1.1.2 an excerpt of the testing data is shown and the recognized entities by the regex model are shown.

The output of the regular expression model lists which entities have been recognized. The list is not in order, and if there is a repeating entity it is listed once.

For the direct infusion analysis of the fractions, a robotic nanoflow ion source TriVersa NanoMate (Advion BioSciences, Ithaca, NY, USA) was coupled to a high-field Q Exactive HF™ quadrupole-Orbitrap mass spectrometer (Thermo Fisher Scientific, Bremen, Germany). The dried samples were diluted in isopropanol (IPA)/MeOH/CHCl3 4:2:1 (v/v/v) containing 7.5 mM ammonium formate and 30 µL was placed in a 96-well twin.tec® plate (Eppendorf, Hamburg, Germany). Nano-electrospray ionization (nano-ESI) chips

```
0 : "Electrospray ionization"
1 : "ionization"
2 : "Orbitrap"
3 : "quadrupole"
4 : "spectrometer"
5 : "Thermo Fisher"
6 : "Thermo"
7 : "ESI"
8 : "mass spectrometer"
```

Fig 3.1.1.2 - Regular expressions model example of evaluated data (*the blue highlights are drawn for demonstration purposes, the regular expression model only outputs the entity list).

3.1.2 Spacy based deep learning model results

The loss function in the training of the deep learning model is shown on graph 3.1.2.1 This is the last batch, but the pattern of loss minimization was similar when training other batches. The exact values of the losses and the validation set are provided in the repository.

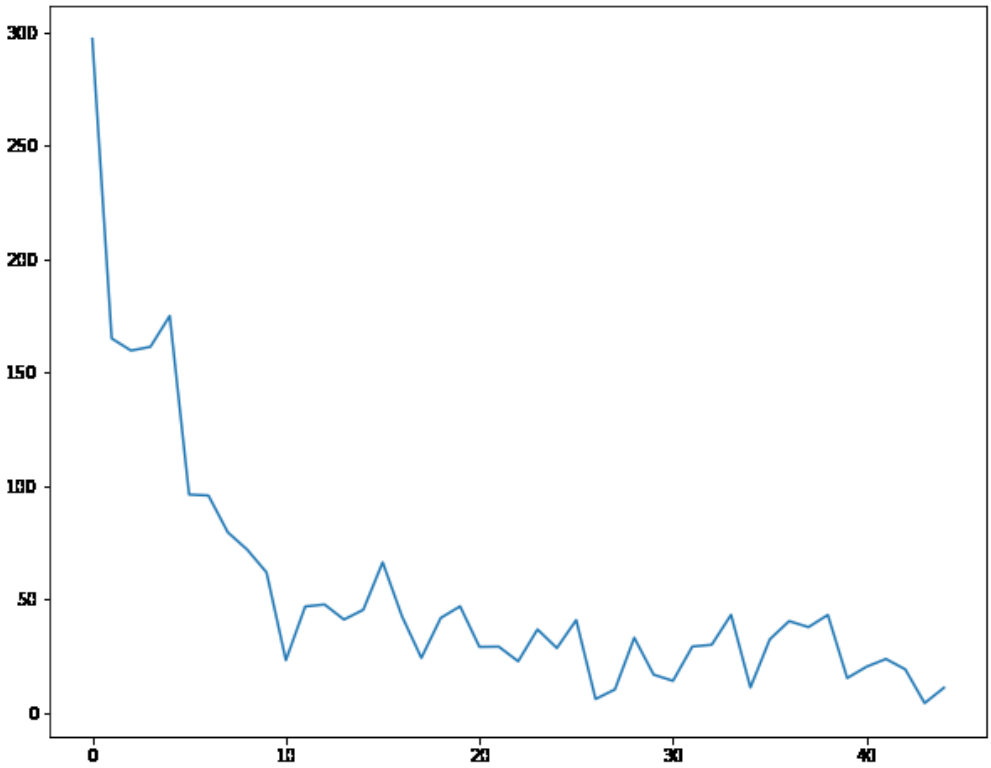


Fig 3.1.2.1 - Loss graph of last training batch; x-axis: Epochs , y-axis: Loss value

While training, the model was tested using a validation data set of MWAS paper. The model starts overfitting when there is a sharp decline in the loss values but also a decline in the F-score. This is when training is stopped. In figure 3.1.2.2 the confusion matrix and metrics are provided.

	TRUE POSITIVES	TRUE NEGATIVES
PREDICTED POSITIVES	40	4
PREDICTED NEGATIVES	20	805

Measure	**Value**	**Derivations**
Sensitivity	0.6667	TPR = TP / (TP + FN)
Specificity	0.9951	SPC = TN / (FP + TN)
Precision	0.9091	PPV = TP / (TP + FP)
Negative Predictive Value	0.9758	NPV = TN / (TN + FN)
False Positive Rate	0.0049	FPR = FP / (FP + TN)
False Discovery Rate	0.0909	FDR = FP / (FP + TP)
False Negative Rate	0.3333	FNR = FN / (FN + TP)
Accuracy	0.9724	ACC = (TP + TN) / (P + N)
F1 Score	0.7692	F1 = 2TP / (2TP + FP + FN)
Matthews Correlation Coefficient	0.7652	TP*TN - FP*FN / sqrt((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))

Fig 3.1.2.2 - Confusion matrix and evaluation metrics for custom deep learning model

Figure 3.1.2.3 shows how this model performs on the same excerpt on which the previous (regular expression/string matching) model was evaluated.

For the direct infusion analysis of the fractions, a robotic nanoflow ion source TriVersa ASSAY NanoMate (Advion BioSciences, Ithaca, NY, USA) was coupled to a high-field Q Exactive HF™ quadrupole ASSAY - Orbitrap ASSAY mass spectrometer ASSAY (Thermo Fisher ASSAY Scientific, Bremen, Germany). The dried samples were diluted in isopropanol (IPA)/MeOH/CHCl3 4:2:1 ASSAY (v/v/v) containing 7.5 mM ammonium formate and 30 µL was placed in a 96-well twin.tec® plate (Eppendorf, Hamburg, Germany). Nano-electrospray ionization ASSAY (nano-ESI ASSAY) chips

	text	label_	start	end	start_char	end_char
0	TriVersa	ASSAY	14	15	81	89
1	quadrupole	ASSAY	37	38	180	190
2	Orbitrap	ASSAY	39	40	191	199
3	mass spectrometer	ASSAY	40	42	200	217
4	Thermo Fisher	ASSAY	43	45	219	232
5	4:2:1	ASSAY	63	64	326	331
6	ionization	ASSAY	98	99	474	484
7	ESI	ASSAY	102	103	491	494

Fig 3.1.2.3 - Examples of guesses by the deep learning model

The model also outputs a table where entities are listed in chronological order and character spans are defined. It is important to note that some companies like “Thermo Fisher” still belong to the “ASSAY” class, because these companies are involved in production of equipment related to assays.

3.1.2 Transformer based models results

In this section, the results of the transformer-based models on the tasks of summarization and zero-shot topic modeling.

3.1.2a Summarization results

The summarization summarizes the input text from 689 words to 187 words.

The example output of extractive summarization is of the introduction section in (Garcia-Perez et al., 2020). The extracted summarized text is:

"As a result, many published NMR-based metabolic profiling studies for modeling continue to include putatively identified metabolites and unknown features without providing unequivocal proof of assignment, or they simply label peaks as 'unknown', thereby potentially missing key mechanistic information. To avoid the problem of multiple entries for the same compound in databases under different names, a community-wide effort is underway to develop better, faster and more standardized metabolite identification strategies, such as implementing standard nomenclature for newly identified metabolites using the International Chemical Identifier (InChI)17. proposed a four-level system18 for assigning a confidence level to newly identified metabolites in metabolic profiling studies: 1) positively identified compounds (with a name, a known structure, a CAS number or an InChI); 2) putatively annotated compounds using spectral similarity with databases but without chemical reference standard; 3) putatively identified chemicals within a compound class; and 4) unknown compounds. Commercial packages, such as Bruker's AMIX TM software, and open-source software20, such as COLMAR (http://spinportal.magnet.fsu.edu/), can help with identifying these 'known unknowns', and some of these software applications are capable of automatically or semi-automatically annotating a limited number of compounds in a biological sample."

3.1.2b Zero-shot learning results

The resulting example output from the zero-shot learning is given in figure 3.1.2b.1 The text used was from the abstract section of (Garcia-Perez et al., 2020). These scores are visually represented in 3.1.2b.1, and they give additional information about the context and importance of the entities as topics.

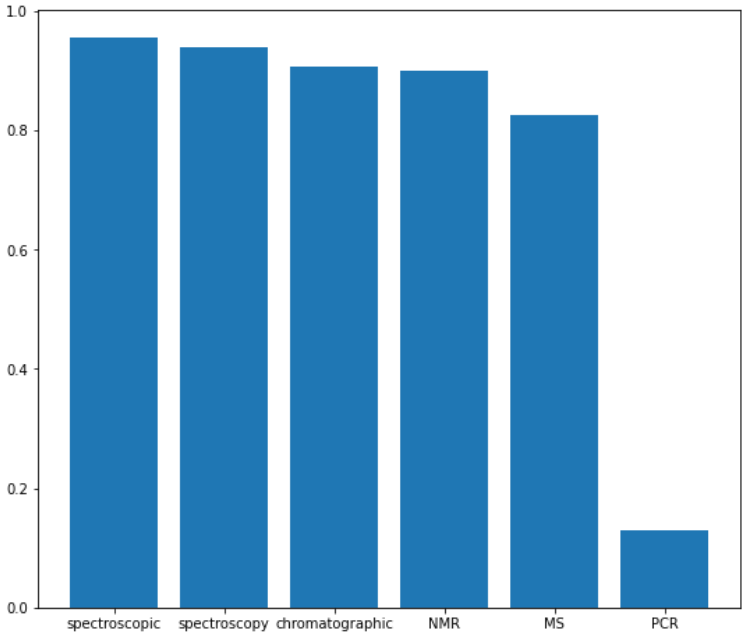
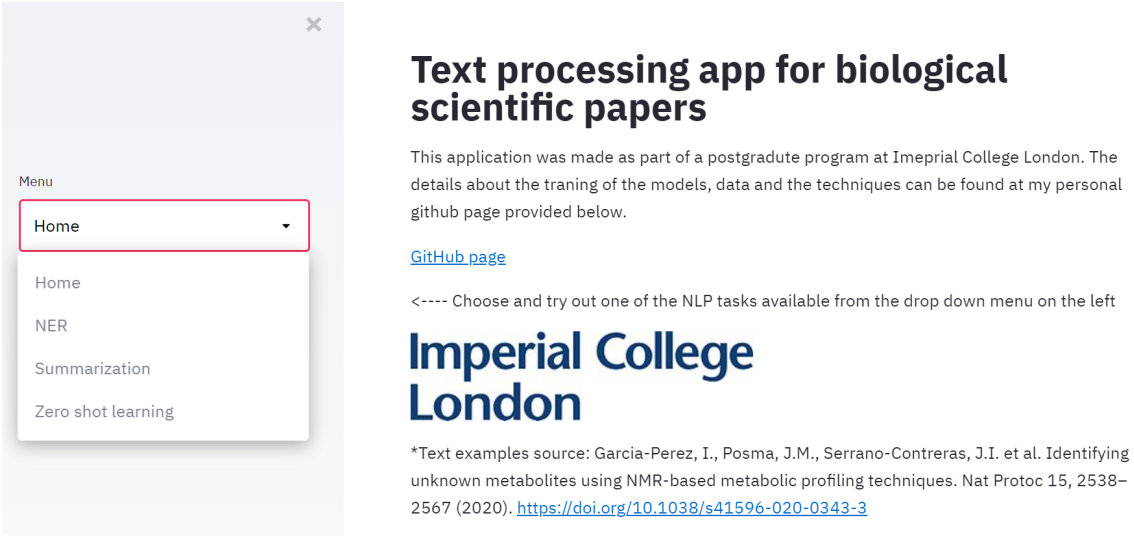


Fig 3.1.2b.1 - Scores of each recognized assay

3.2 Deployment

3.2.1 Web application

The fully functional website link is available at the GitHub repository as well as all of the code used in this report. Images 3.2.1.1, 3.2.2.2, and 3.2.3.3 outlines the main structures of the website.



3.2.1.1 - The home page of the website, the task can be chosen from the left drop-down menu

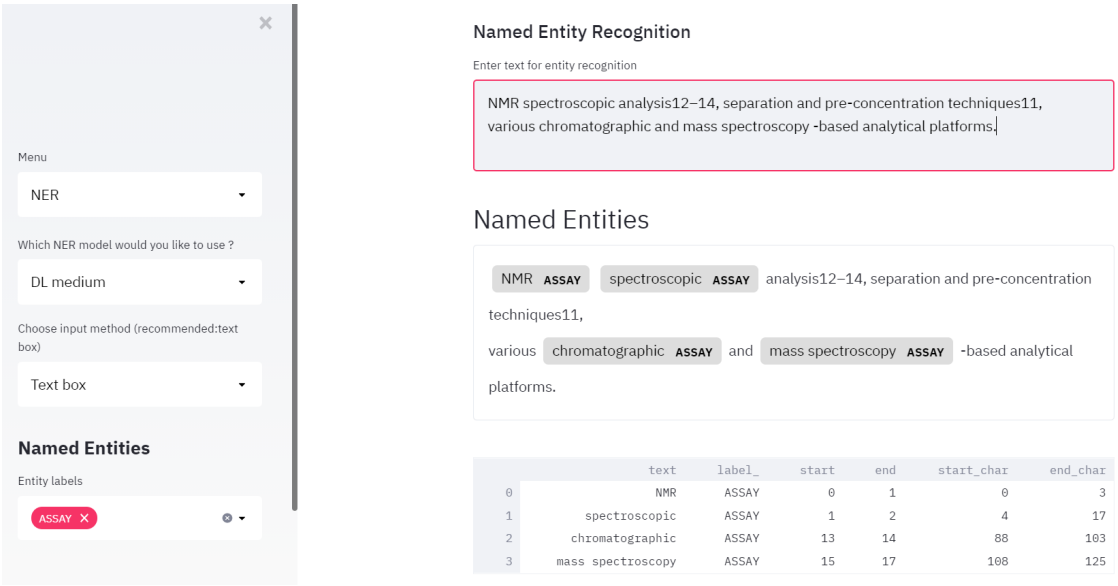


Fig 3.2.1.2 - Once a task is chosen, a type of model and input method can be chosen

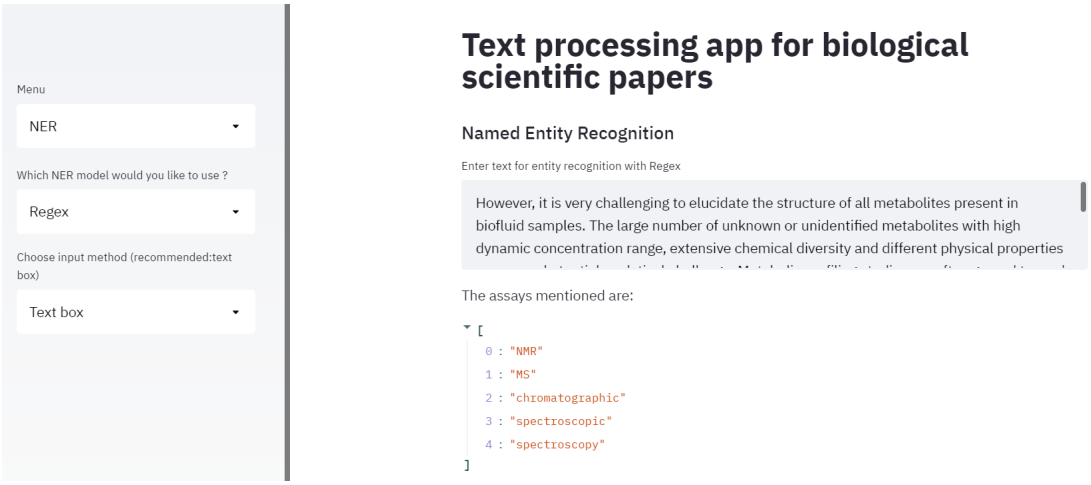


Fig 3.2.1.3 - Deep learning algorithms also provide the option of entity choice for recognition

3.2.2. Adapted application for GWAS studies

The regular expression-based model was also applied to GWAS studies as an example of the adaptability of the model. As ground truth, the table in figure 3.2.2.1 is taken. This is a table with manually extracted values.

PMID	PMCID	Platform	QC SNPs	Imputation
19648918	PMC2839871	Illumina	558542	
19118814	PMC2668056	Illumina	~ 2500000	yes
19330027	PMC2748125	Illumina, Perlegen	up to 206586	
15761122	PMC1512523	Affymetrix	103611	

Fig 3.2.2.1 - Manually extracted table of important information in GWAS studies, the highlighted row is the one presented in greater detail in figure 3.2.2.2.Source: GWAS Central

The first three columns PMID, PMCID, and Platform, can be tackled as a named entity recognition task and solved with our string matching/regular expression models by appending these entities to the .csv entity list and editing regular expression patterns. The last two columns signify how many single-nucleotide polymorphisms (SNPs) have passed quality control and whether there was imputation present. However,

understanding this requires context beyond named entity recognition. This was tested on the 5 publications above in Fig 3.2.2.1 where the model could recognize PMC, PMCID, and Platform type. For QC of SNPs and imputation, only parts of the text where some of these entities are mentioned could be extracted. An example of these techniques applied to a paper is given in figure 3.2.2.2. The exact recognition of number of SNPs

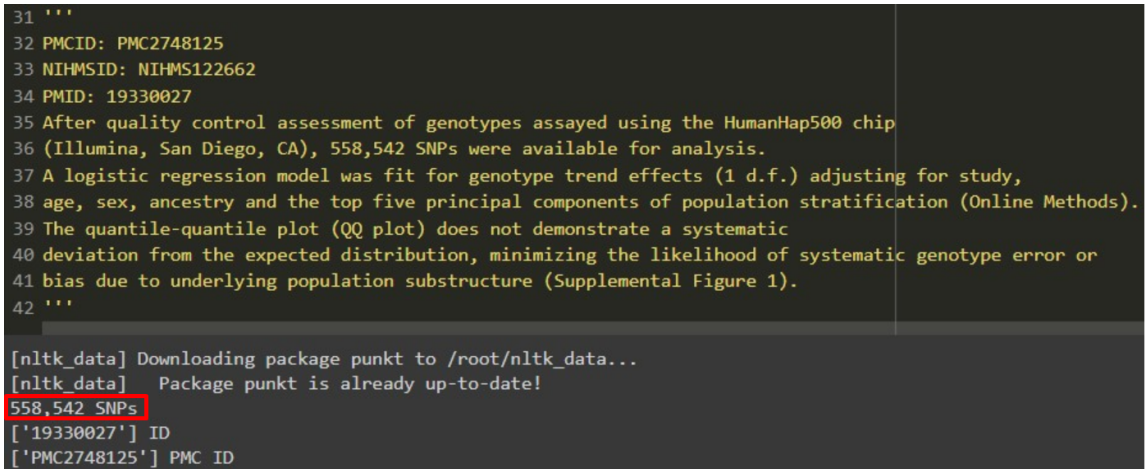


Fig 3.2.2.2 In the lower part of the figure, highlighted in red, a successful extraction of the amount of SNPs after quality control. This extraction worked only in 1 out of 10 papers tested.

In the upper part with yellow letters is the relevant passage of the paper as a reference.

4. Discussions

4.1 Regular expressions/string matching model discussion

This model has showcased great accuracy of more than 97% and an F1-score of more than 93%. This is a compelling result compared to other named entity recognition systems. These results come with drawbacks. The list of entities that need to be recognized has to be updated in order for the model to work on new entities. The model would not be able to adapt to new examples without these adjustments. However, given these metrics, for many applications, this is a small price to pay, especially if regular expression models are coupled with other more complex algorithms on top of one another (as was the case in this report). Improvements in terms of the computational

efficiency of regular expressions could be made. This would be useful for big data applications where the python regular expression library would be too slow.

An illustration of a regular expression string matching task for different regular expression formulations is given in figure 4.1.1. It is important to note that the task given in this figure is not the one used in this report but a computationally intensive task is purposely presented. This graph illustrates that if the task is defined specifically, even the worst regular expression algorithms can achieve times less than a second.

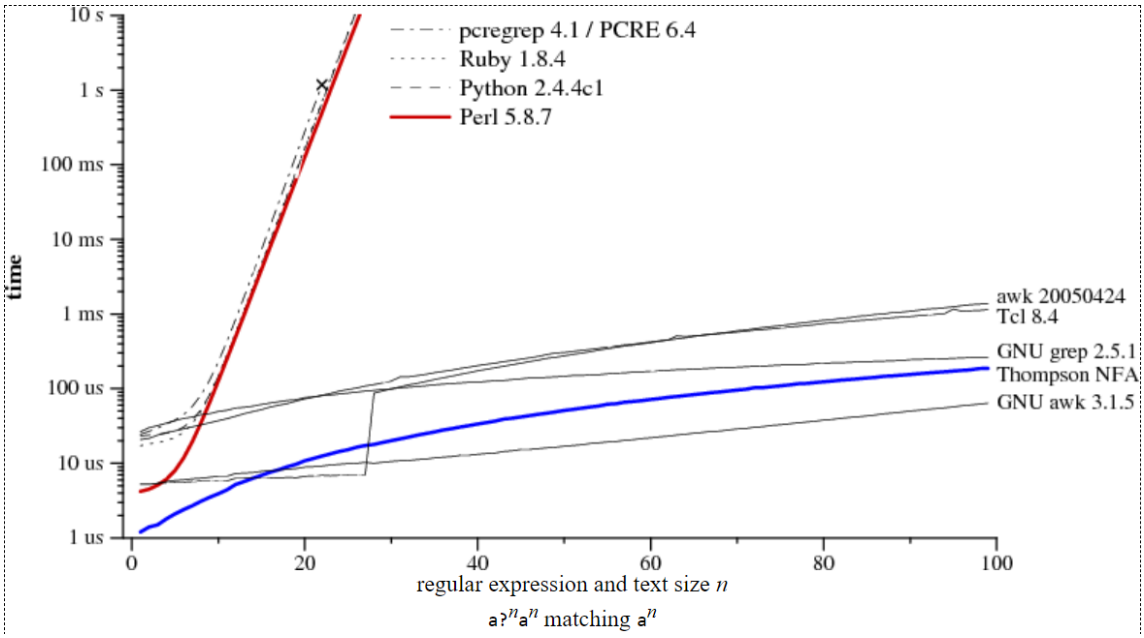


Fig 4.1.1 - Regular expressions formulations and time complexity,

source: Russ Cox, 2007

This possibility for adaptation further reinforces regular expressions as a fitting first step solution in the task of entity recognition and topic modeling. For best efficiency, other string matching algorithms can be implemented, such as the Knuth, Morris, Pratt (KMP) algorithm for string matching (Knuthf et al., 1974). If matching a given pattern to a given text, this algorithm achieves linear time complexity (the complexity increases linearly proportionally to the size of the input text).

4.2 Spacy based deep learning model discussion

The deep learning model meets lower standards judging by the calculated metrics.

As the highlighted example in figure 4.2.2.1

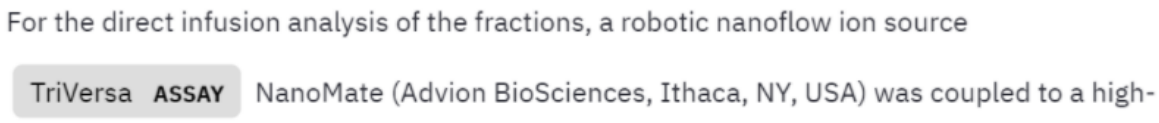


Fig 4.2.2.1 - Recognition of an entity not from the list of assays

These results suggest that the model has an ability to generalize. The proof is the recognition of entities, not part of the entities in the annotated data, such as “TriVersa”. This entity is not recognized by the regular expression-based model because it is not part of the entity list for string matching. This example highlights the main compromise points of each of the models. Nevertheless, the generalization can sometimes lead to false-positives such as the example in figure 4.2.2.2.

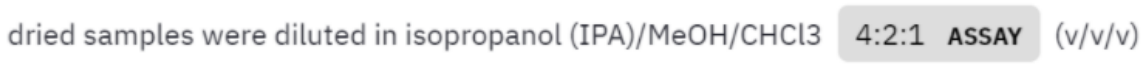


Fig 4.2.2.2 - The deep learning model recognizing a nonexisting assay as a false positive

The ambiguous nature of assays also suggests that maybe a better framing of the problem statement would yield better results.

Training. A possible improvement here would be to train using BIO annotated data. This would convey better information about spans of entities and the model should be able to understand the exact spans of the entities. Also accessing lower levels of the model network and pre-training on fewer layers of the network will lower the inductive bias. This will have to be done using a different library that allows lower-level access.

Data and algorithm. The named entity recognition task can be divided into smaller subgroups. Not just recognizing “assay” as an entity, but more different variations of assays. Transformer networks were not used directly here for named entity recognition

because they are resource demanding and many projects such as BioBERT(Lee et al., 2020) have already undertaken a similar approach. Also, transformers are expected to make a greater impact in more complex tasks like the ones mentioned in the next heading.

4.3 Transformer based networks discussion

The results of the topic modeling and summarization tasks perform well and prove the point that solving named entity recognition can serve as a baseline for more advanced models.

The intuitive improvement to do here would be to fine-tune the transformers to the downstream task of the given problem of assay recognition. Furthermore, certain unsupervised techniques for topic modeling as proposed by (Schick & Schütze, 2020) can be tried. Also, using an abstractive summarization model compared to the extractive is possible. A good starting point would be (Zhang et al., 2020) where abstractive summarization of scientific literature is performed.

4.4 Deployment discussion

The website serves as a great visualization tool and by offering the ability to try the code in action. The ease of access, reproducibility, and robustness is more so demonstrated by the custom data annotator, which in terms of deployment can be used for many custom named entity recognition tasks without the human work of data annotation. The results on GWAS show that although the regular expression/string matching model does not perform to a satisfactory standard, it can still be utilized as a first step, even if the end goal is another algorithm (as shown with the topic modeling task). This was one of the key aims of this report, to create code that offers the means but is not the end.

5.Future work

Data. The quality of data annotation can be improved if this annotation pipeline and website are coupled with a crowdsourcing annotation. In this way, annotators can have a starting point done by the algorithms and this would make their task easier and less time-consuming.

Named entity recognition. Divide the entity “assay” into more sub-groups and fine-tune a transformer-based architecture for this specific problem as a multiple entity recognition problem. Many studies have taken up this approach for different domains. A relevant project fine-tuning the BERT transformer is BioBERT (Lee et al., 2020) for recognizing biomedical entities. Also, Spacy’s new version 3.0 library which was recently published could be explored.

Problem statement. The results of the topic modelling and summarization tasks perform well and prove the point that solving named entity recognition can serve as a baseline for more advanced models which go beyond the task of named entity recognition. One group of models like this would be question answering transformer models. The entities recognized through named entity recognition could be used to contextualize the training data.

Another future opportunity is graph topic modeling. In this way, relationships between assays could be extracted to better understand the context. Also representing the learned graph and explaining the graph structures is a riveting challenge. These architectures have been used in drug discovery (Kwak et al., 2020) and electronic health records. A similar approach to (Zhu & Razavian, 2021) can be a fitting starting point.

6. Conclusion

The first hypothesis that a deep learning model would perform better than a string matching/regular expression algorithm has not been proven. In regards to the second hypothesis, the findings of this study suggest that complex models would be better suited for other tasks that extract more information about the named entities rather than entity recognition itself. The entity recognition can be performed by simpler and computationally less intensive algorithms to a satisfactory degree.

The principal aim of this project was to create a simple, robust, adaptable text processing pipeline. This pipeline does assay named entity recognition but also uses transformer networks to augment the entity recognition. In addition to this, the automatic data annotation tool saves time and resources. This tool can be used as a starting point for many different natural language processing tasks beyond the scope of the thesis.

The work in this project is valuable for GWAS and MWAS studies for future data curation and creation of backlog databases. These efforts are important because they help researchers know which papers are closely related to their work thus enabling filtering by their preferences.

In Data Science there is this tendency of going for the most complex and advanced deep learning algorithm straight away. Sometimes, simpler and older algorithms can do as well and even better. They clear the way for other advanced deep learning architectures on top of their work. In this way, the complex architectures are used in problems for which they are truly needed.

References

1. Aho, A. V., & Corasick, M. J. (1975). Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, *18*(6), 333–340. https://doi.org/10.1145/360825.360855
2. Atkinson, C., McCane, B., Szymanski, L., & Robins, A. (2021). Pseudo-Rehearsal: Achieving Deep Reinforcement Learning without Catastrophic Forgetting. *Neurocomputing*, *428*, 291–307. https://doi.org/10.1016/j.neucom.2020.11.050
3. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020). Language Models are Few-Shot Learners. *ArXiv:2005.14165 [Cs]*. http://arxiv.org/abs/2005.14165
4. Bush, W. S., & Moore, J. H. (2012). Chapter 11: Genome-Wide Association Studies. *PLOS Computational Biology*, *8*(12), e1002822. https://doi.org/10.1371/journal.pcbi.1002822
5. Deutsch, L. P., & Lampson, B. W. (1967). An online editor. *Communications of the ACM*, *10*(12), 793–799. https://doi.org/10.1145/363848.363863
6. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *ArXiv:1810.04805 [Cs]*. http://arxiv.org/abs/1810.04805
7. *Doccano/doccano*. (2021). [Python]. doccano. https://github.com/doccano/doccano (Original work published 2018)
8. Garcia-Perez, I., Posma, J. M., Serrano-Contreras, J. I., Boulangé, C. L., Chan, Q., Frost, G., Stamler, J., Elliott, P., Lindon, J. C., Holmes, E., & Nicholson, J. K. (2020). Identifying unknown metabolites using NMR-based metabolic profiling techniques. *Nature Protocols*, *15*(8), 2538–2567. https://doi.org/10.1038/s41596-020-0343-3
9. Ginsburg, B., Castonguay, P., Hrinchuk, O., Kuchaiev, O., Lavrukhin, V., Leary, R., Li, J., Nguyen, H., Zhang, Y., & Cohen, J. M. (2020). Stochastic Gradient Methods with Layer-wise Adaptive Moments for Training of Deep Networks. *ArXiv:1905.11286 [Cs, Stat]*. http://arxiv.org/abs/1905.11286
10. Goodwin, S., McPherson, J. D., & McCombie, W. R. (2016). Coming of age: Ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, *17*(6), 333–351. https://doi.org/10.1038/nrg.2016.49
11. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
12. Honnibal, M., & Montani, I. (2017). SpaCy · Industrial-strength Natural Language Processing in Python. *SpaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks, and Incremental Parsing.* https://spacy.io/
13. Hu, Y., Sun, S., Rowlands, T., Beck, T., & Posma, J. M. (2021). Auto-CORPus: Automated and Consistent Outputs from Research Publications. *BioRxiv*, 2021.01.08.425887. https://doi.org/10.1101/2021.01.08.425887
14. Jaccard, P. (1901). Etude de la distribution florale dans une portion des Alpes et du Jura. *Bulletin de La Societe Vaudoise Des Sciences Naturelles*, *37*, 547–579. https://doi.org/10.5169/seals-266450

15. Kiss, T., & Strunk, J. (2006). Unsupervised Multilingual Sentence Boundary Detection. *Computational Linguistics*, *32*(4), 485–525. https://doi.org/10.1162/coli.2006.32.4.485
16. Kitaev, N., Kaiser, Ł., & Levskaya, A. (2020). Reformer: The Efficient Transformer. *ArXiv:2001.04451 [Cs, Stat]*. http://arxiv.org/abs/2001.04451
17. Kleene, S. (1956). Stephen Kleene. *S. C. Kleene, "Representation of Events in Nerve Nets and Finite Automata", in C. E. Shannon and J. McCarthy, Eds., Automata Studies, Annals of Mathematics Studies No. 34, Princeton University Press, 1956, Pp. 3-42.* https://student.cs.uwaterloo.ca/~cs462/Hall/kleene.html
18. Knuthf, D. E., Morris, J. H., Jr. :l, & Pratt, V. R. (1974). *Fast Pattern Matching in Strings**.
19. Kwak, H., Lee, M., Yoon, S., Chang, J., Park, S., & Jung, K. (2020). Drug-disease Graph: Predicting Adverse Drug Reaction Signals via Graph Neural Network with Clinical Data. *ArXiv:2004.00407 [Cs, Stat]*. http://arxiv.org/abs/2004.00407
20. Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., & Kang, J. (2020). BioBERT: A pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, *36*(4), 1234–1240. https://doi.org/10.1093/bioinformatics/btz682
21. Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2019). *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. https://arxiv.org/abs/1910.13461v1
22. Liu, P., Qiu, X., & Huang, X. (2016). Recurrent Neural Network for Text Classification with Multi-Task Learning. *ArXiv:1605.05101 [Cs]*. http://arxiv.org/abs/1605.05101
23. 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:1907.11692 [Cs]*. http://arxiv.org/abs/1907.11692
24. *Long Short-Term Memory | Neural Computation | MIT Press Journals*. (n.d.). Retrieved March 4, 2021, from https://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735
25. Lovis, C., & Baud, R. H. (2000). Fast Exact String Pattern-matching Algorithms Adapted to the Characteristics of the Medical Language. *Journal of the American Medical Informatics Association*, *7*(4), 378–391. https://doi.org/10.1136/jamia.2000.0070378
26. Lu, W., Su, X., Klein, M. S., Lewis, I. A., Fiehn, O., & Rabinowitz, J. D. (2017). Metabolite Measurement: Pitfalls to Avoid and Practices to Follow. *Annual Review of Biochemistry*, *86*, 277–304. https://doi.org/10.1146/annurev-biochem-061516-044952
27. McClanahan, C. (2010). *History and Evolution of GPU Architecture*. 7.
28. McCloskey, M., & Cohen, N. J. (1989). Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. In G. H. Bower (Ed.), *Psychology of Learning and Motivation* (Vol. 24, pp. 109–165). Academic Press. https://doi.org/10.1016/S0079-7421(08)60536-8
29. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *ArXiv:1301.3781 [Cs]*. http://arxiv.org/abs/1301.3781
30. Neumann, M., King, D., Beltagy, I., & Ammar, W. (2019). ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing. *Proceedings of the 18th BioNLP Workshop and Shared Task*, 319–327. https://doi.org/10.18653/v1/W19-5034

31. Neves, M., & Ševa, J. (2021). An extensive review of tools for manual annotation of documents. *Briefings in Bioinformatics*, *22*(1), 146–163. https://doi.org/10.1093/bib/bbz130
32. Petroni, F., Rocktäschel, T., Lewis, P., Bakhtin, A., Wu, Y., Miller, A. H., & Riedel, S. (2019). *Language Models as Knowledge Bases?* https://arxiv.org/abs/1909.01066v2
33. *Prodigy · An annotation tool for AI, Machine Learning & NLP*. (n.d.). Prodigy. Retrieved February 22, 2021, from https://prodi.gy
34. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (n.d.). *Language Models are Unsupervised Multitask Learners*. 24.
35. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *ArXiv:1910.10683 [Cs, Stat]*. http://arxiv.org/abs/1910.10683
36. Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2020). DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *ArXiv:1910.01108 [Cs]*. http://arxiv.org/abs/1910.01108
37. Schick, T., & Schütze, H. (2020). *Exploiting Cloze Questions for Few Shot Text Classification and Natural Language Inference*. https://arxiv.org/abs/2001.07676v3
38. Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673–2681. https://doi.org/10.1109/78.650093
39. Serrà, J., & Karatzoglou, A. (2017). Getting deep recommenders fit: Bloom embeddings for sparse binary input/output networks. *ArXiv:1706.03993 [Cs]*. http://arxiv.org/abs/1706.03993
40. *Streamlit Documentation*. (n.d.). 150.
41. Thompson, K. (1968). Programming Techniques: Regular expression search algorithm. *Communications of the ACM*, *11*(6), 419–422. https://doi.org/10.1145/363347.363387
42. Tikhomirov, M., Loukachevitch, N., Sirotina, A., & Dobrov, B. (2020). Using BERT and Augmentation in Named Entity Recognition for Cybersecurity Domain. In E. Métais, F. Meziane, H. Horacek, & P. Cimiano (Eds.), *Natural Language Processing and Information Systems* (pp. 16–24). Springer International Publishing. https://doi.org/10.1007/978-3-030-51310-8_2
43. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. *ArXiv:1706.03762 [Cs]*. http://arxiv.org/abs/1706.03762
44. Weischedel, Ralph, Palmer, Martha, Marcus, Mitchell, Hovy, Eduard, Pradhan, Sameer, Ramshaw, Lance, Xue, Nianwen, Taylor, Ann, Kaufman, Jeff, Franchini, Michelle, El-Bachouti, Mohammed, Belvin, Robert, & Houston, Ann. (n.d.). *OntoNotes Release 5.0* [Data set]. Linguistic Data Consortium. https://doi.org/10.35111/XMHB-2B84
45. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. V. (2020). XLNet: Generalized Autoregressive Pretraining for Language Understanding. *ArXiv:1906.08237 [Cs]*. http://arxiv.org/abs/1906.08237
46. Yin, W., Hay, J., & Roth, D. (2019). Benchmarking Zero-shot Text Classification: Datasets, Evaluation and Entailment Approach. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3914–3923. https://doi.org/10.18653/v1/D19-1404

47. Zhang, J., Zhao, Y., Saleh, M., & Liu, P. J. (2020). PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization. *ArXiv:1912.08777 [Cs]*. http://arxiv.org/abs/1912.08777
48. Zhou, Z., Siddiquee, M. M. R., Tajbakhsh, N., & Liang, J. (2020). UNet++: Redesigning Skip Connections to Exploit Multiscale Features in Image Segmentation. *IEEE Transactions on Medical Imaging*, *39*(6), 1856–1867. https://doi.org/10.1109/TMI.2019.2959609
49. Zhu, W., & Razavian, N. (2021). Variationally Regularized Graph-based Representation Learning for Electronic Health Records. *ArXiv:1912.03761 [Cs, Stat]*. http://arxiv.org/abs/1912.03761

Appendix A

The first mention of regular expressions is in a paper by Stephen Kleene (Kleene, 1956). The intended aim was to create a software notation that would describe the work of deterministic finite-state automata (DFA). They were used for string matching. The steps of how DFA describes text strings are illustrated in figure 2.1.1.

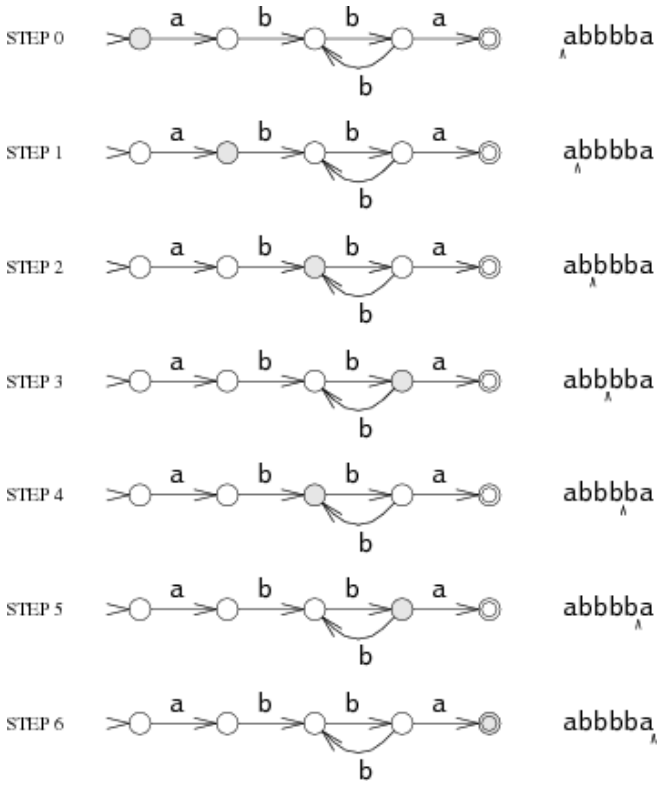


Fig 2.1.1. - Overview of DFA for string matching, Source: Russ Cox, 2007

Afterward, a lot of different implementations were made, most notable were the ones in Unix systems (Deutsch & Lampson, 1967) and also a multi-state non-deterministic finite automata (NFA) formulation of regular expressions, which are used in many practical implementations today (Thompson, 1968). In figure 2.1.2 The equivalence between regular expressions and finite state automata is illustrated.

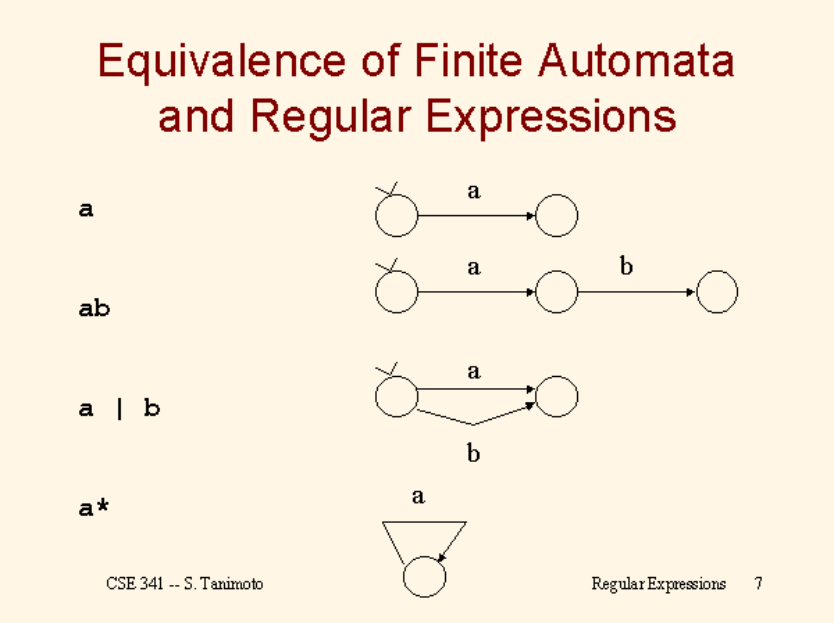


Fig 2.1.2 - Regular expressions equivalence to finite state automata,

source: CSE341-S.Tanimoto

The following is the list of assays compiled by experts and used in this project:

1H NMR
H1 NMR
(1)H NMR
H(1) NMR
NMR
1H-NMR
1H-NMR
Proton NMR
Nuclear Magnetic Resonance spectroscopy
NMR spectroscopy
600 MHz NMR
500 MHz NMR
750 MHz NMR
1D NMR
1D-NMR
diffusion-edited NMR
J-resolved NMR spectroscopy
J-resolved
1H-1H J-resolved
JRes
J-Res
Jres
13C NMR
13C NMR
Carbon NMR
13-carbon NMR
Correlation spectroscopy
1H-1H correlation spectroscopy
COSY
Total correlation spectroscopy
1H-1H total correlation spectroscopy
TOCSY
2D 1H-13C heteronuclear single quantum coherence
1H-13C heteronuclear single quantum coherence
heteronuclear single quantum coherence spectroscopy
HSQC
2D 1H-13C heteronuclear multiple-bond quantum coherence
1H-13C heteronuclear multiple-bond quantum coherence
heteronuclear multiple-bond correlation
hetero-nuclear multiple bond correlation
hetero-nuclear multiple-bond correlation spectroscopy
HMBC
LC-NMR-MS
2D-NMR
1D proton NMR
Homonuclear NMR
Homo-nuclear NMR
13C Distortionless Enhancement by Polarization Transfer
DEPT
HPLC-NMR
HPLC-NMR-MS
LC-SPE-NMR
CPMG
Carr-Purcell-Meiboom-Gill
Mass spectrometry
mass spectrometer
MS
tandem mass spectrometry

MS/MS
MS-MS
Gas chromatography
Gas chromatography mass spectrometry
Gas chromatography coupled mass spectrometry
Gas chromatography-coupled mass spectrometry
GCMS
GS-MS
GC/MS
GC-EI-TOF
Electron ionization
EI
flame ionization detector
FID
GCxGC
Liquid chromatography
Liquid chromatography mass spectrometry
LC-MS
LC/MS
Ultra-Performance Liquid Chromatography Mass Spectrometry
UPLC-MS
HPLC
High-pressure liquid chromatography
High performance liquid chromatography
High-performance liquid chromatography
Ultra-Performance Liquid Chromatography
Ultra Performance LC
ultra-fast liquid chromatography
UHPLC-QTOF-MS
reverse-phase HPLC
reversed-phase HPLC
RP
Hydrophilic interaction chromatography
hydrophilic interaction liquid chromatography
HILIC
LC-MS/MS
LC-ESI-QQ
Tandem Quadrupole
QQ
Single Quadrupole
Electrospray ionization
Electrospray ionisation
Electro-spray ionization
Electro-spray ionisation
ESI
quadrupole
TOF
Time-of-flight
Time of flight
QTOF
Quadrupole Time-of-flight
QQQ
QqQ
Triple quadrupole
TQMS
quadrupole mass analyser
QMS
Quadrupole mass filter
Secondary ion mass spectrometry
SIMS
secondary electrospray ionization
SESI
matrix-assisted laser desorption/ionization
MALDI
direct analysis in real time
DART
Nanospray desorption electrospray ionization
desorption ionization
DI
Ion mobility spectrometry
IMS
Ion mobility spectrometry-mass spectrometry
IMS/MS
IMMS
Surface-enhanced laser desorption/ionization
SELDI
Direct-injection electrospray ionization
Direct Flow Injection Mass Spectrometry
Desorption electrospray ionization
DESI
DIMS
DI-MS
Fourier transform ion cyclotron resonance
FTICR
FT-ICR
Orbitrap
Ion trap
quadrupole ion trap
Inductively coupled plasma mass spectrometry
ICP-MS
IEC
Ion exchange chromatography
Ion-exchange chromatography
Solid Phase Extraction
SPE
Lipidomics
Lipidomics profiling
Electrophoresis
Capillary zone electrophoresis
Capillary electrophoresis
Biocrates
Nightingale
Metabolon
Thermo
Waters
Agilent
Thermo Fisher
Bruker
Bruker BioSpin
JEOL
Shimadzu
Varian
SCIEX
Perkin Elmer
LC-Mass spectrometry
(LC)
gas chromatography
chromatography
spectroscopy
spectrometry
gas chromatography
gas chromatography-mass spectrometry
gas chromatography coupled mass spectrometry
gas chromatography-coupled mass spectrometry
liquid chromatography
chromatography
spectroscopy
1H NMR
ionization
electrophoresis
sequencing
ionisation
spectroscopic
chromatographic
spectrometer
quadrupole time-of-flight
Illumina
Perlegen
Affymetrix
PCR
Polymerase chain reaction

Appendix B

In the following table, all the packages and versions used are listed. It is important to note that not all packages are part of the final solution, some have been used for testing, prototyping, and debugging. Also, many of the packages can be substituted with others. The requirements.txt file for running the website is available as a separate file on the Github repository.

Package	Version
beautifulsoup4	4.6.3
en-core-web-sm	2.2.5
gensim	3.6.0
google	2.0.3
google-api-core	1.16.0
google-api-python-client	1.7.12
google-auth	1.27.0
google-auth-httplib2	0.0.4
google-auth-oauthlib	0.4.2
google-cloud-core	1.0.3
google-cloud-translate	1.5.0
google-colab	1.0.0
ipykernel	4.10.1
ipython	5.5.0
ipython-genutils	0.2.0
ipython-sql	0.3.9
ipywidgets	7.6.3
jsonschema	2.6.0
jupyter	1.0.0
jupyter-client	5.3.5
jupyter-console	5.2.0
jupyter-core	4.7.1
jupyterlab-pygments	0.1.2
jupyterlab-widgets	1.0.0
nltk	3.2.5
notebook	5.3.1
numpy	1.19.5
pandas	1.1.5
pathlib	1.0.1
Pillow	7.0.0
pip	19.3.1
pip-tools	4.5.1
plotly	4.4.1
PyDrive	1.3.1
python-utils	2.5.6
regex	2019.12.20
scikit-learn	0.22.2.post1
scipy	1.4.1
sklearn-pandas	1.8.0
spacy	2.2.4
sympy	1.7.1
thinc	7.4.0