# Clinical Information Extraction and Classification From Free Text

Final Report for CSM9060 Dissertation

Author: Aidan Hogden (aih16@aber.ac.uk)
Supervisor: Dr. Chuan Lu (cul@aber.ac.uk)

24th September 2021
Version 1.0 (Release)

This report is submitted as partial fulfilment of a MSc degree in Computer Science (G790)

# Contents

# List of Figures



# List of Tables

**Declaration of originality**

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work, I understand and agree to abide by the University's regulations governing these issues.

Name Aidan Hogden

Date 24.09.2021

**Consent to share this work**

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name .....

Date .....

Abstract

"Can NLP systems use narrative medical records to identify which patients meet selection criteria for clinical trials?"[1].

This question was asked in the 2018 National NLP Clinical Challenges (n2c2).

The goal of this project was to develop a machine learning system that could potentially answer the above question. This was done by looking at the all the given medical records provided by a cohort of patients from real clinical trials that had been tagged by a medical professional, and by using this combination of records and tags, develop a system that could determine whether the patient meets or does not meet a set of selection criteria.

Python, NLTK, NumPy and SciKit-Learn were used to process the data for this project.

The result was 4 different machine learning models each with varying degrees of success, with the best model being Logistic Regression at 85.32% average accuracy.

# 1 Background, Analysis and Process

## 1.1 Background

### 1.1.1 Machine Learning

Machine learning is a branch of artificial intelligence and allows the automation of analytical model building[2]. The core concept of machine learning is that it is possible to create a system or model on a machine that allows it to learn from patterns of data until it is able to make "decisions", or what we would consider to be a decision from a human point of view, with minimal human intervention[3].

When it comes to machine learning it can be broken down into 3 types depending on the data available, supervised, unsupervised and semi-supervised learning. In this project I used supervised learning as I was provided the patient records, which would be the input variables (X) as well as the tags that went along with the data, these would be the output or target variables (Y). This allows the model to train and learn by using data that is labelled, it can they be given testing data that is also labelled and predict the outcome of the test data and then compare the predict outcome with the correct answers, thus giving a way to measure the accuracy of the model. This requires all training and testing data to be understood and marked appropriately before the model is trained.

### 1.1.2 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a combined field of computer science and linguistics, also known as computational linguistics, and is primarily concerned with the interactions between computers and human language.

Linguistics itself is the "scientific study of language"[4]. It is a rather broad topic and can cover many areas including syntax, semantics, phonology and phonetics to name just a few. Classically linguists would devise and evaluate the rules of a language and generally good progress has been in this field. The same cannot be said however for converting natural language to some form of mathematical linguistics that computers can understand.

At the most basic fundamental level computers use numbers, these being 0 and 1. A computer can understand numbers and can work with numbers, it's how they are designed. However what they are not designed to deal with is words. Words for starters are not numbers and do not work the same way. They also have meanings which are constantly changing, numbers and mathematics by comparison is set in stone. One will always be One, it is the mathematical term for a single entity and is unlikely to change. Language however is fluid, words can have meaning and that meaning is subject to change over time, not only this but they can have different meanings depending on context in which they are said or the way in which they are said.

Take the phrase "Time flies" for example. To a person this is a very simple sentence, it comprises two words neither of which are overly complex. Time is a concept devised by humans to understand the passage of, well, time and flies is a verb, already a type on syntax within language, that describes a process or action. To a human we can understand this, it's a common and simple metaphor. We understand the concept of time and understand the concept of a verb, we know that time itself is not literally flying. Whilst this may seem simple to a human it is in fact more complex than one may initially think. A computer however is not a conscious being and so cannot learn the concept of time or flies. It can however be taught to take this combination of words and apply meaning to them, if it technically it does not understand it.

Another example would be the word read. To further prove my point, which version of the word read do you think I am referring to? Read in the context of I will read a book, or read in the context of I have read that book. Given the context this can be easy to workout but we have years of learning and experience to apply, but in reality it isn't as simple as we think. Sarcasm, idioms, metaphors, synonyms, all these and more make it more and more difficult

to write software that can understand language.

NLP looks at how to program a computer so that it can process, analyze and understand natural language and the contextual nuances within them. There are several tasks within NLP that help break down language in ways that allow the computer to make sense of what data it is being given such as Sentiment Analysis which allows it to understand the sentiment behind text, if a sentence is positive, negative, sarcastic, confused, etc.

### 1.1.3 Similar Projects

Prior to starting this project I was already aware of what this type of project would entail and also similar projects that had come before it. I myself have done a similar project previously in which I used NLP to determine the sentiment of tweets made by Donald Trump and whether or not those negative or positive sentiments would cause a rise or fall in the value of the US dollar.

I had also read two articles that looked using data to find the angriest song by the band Death Grips[5] and to also find the saddest or most depressing song by the band Radiohead[6]. From these alone I could see how interesting it is to take a mathematical approach to try and find a "right" answer to something so subjective, but also how depending on how the data is handled you can be given wildly different answers.

For example in the given articles regarding the band Death Grips, the author found that the song "This Is Violence Now" was 66.7 percent angry even when stop words had been excluded. If the author stopped here then that would be the angriest song in their discography. However this is because 12 of the 18 remaining words are "violence" repeated again and again. It can be argued that this isn't a good measure of what makes a song angry based purely on words alone, especially if the words are just repeated again and again. Further into the article once the author has gone through various methods of ranking the data based on words, sound, usage etc they find that the song "This Is Violence Now" doesn't even make it into the top 10 in terms of angry songs. This shows how big the variance in answers can be depending on how the data is treated.

The main project that is similar however is the 2018 National NLP Clinical Challenges. This was attempted by multiple different teams from various Universities and is also not the only challenge of this type done, there have been others done in a similar vein when it comes to analysing text based clinical data.

## 1.2 Analysis

Having already done a similar project that involved feature extraction I found that using NLTK and SciKit Learn would be the best step. Many of the classifiers within SciKit Learn are already very popular when it comes to NLP such as Naive Bayes and Support Vector Machines. Due to this I felt it was not necessary to reinvent the wheel and develop my own machine learning model but instead to use one a number of the pre-made classifiers and change the parameters such as the numbers of features or the way features are extracted.

## 1.3 Process

The overall process for the project was not set in stone however would follow a general idea and structure:

Set up - Originally I had intended on setting up a Github repository and using PyCharm as my IDE, however I had somewhat recently found out about Google Colab, a free Jupyter notebook environment that lets you run your program on the cloud. I decided to use this as it would mean I would not have to deal with Python versions, I would have no issues downloading any needed libraries, it could run calculations faster than my own computer should the workload become heavy and I could work on it from anywhere provided I had an internet connection.

Research - The research phase was not as big as I initially thought, having never used clinical data or medical data before I assumed this would be a rather large roadblock I would have to overcome but after some research I discovered that whether the next was medical based, review based, article based, it didn't really matter as the whole point of the machine learning system is to learn based on the text it was given. One major upside of using computers for NLP is that whilst I don't understand what phrases such as "Normocephalic" or "auscultation" mean in the context of a patients medical records, I didn't need to and it made no different to the machine learning model.

Data gathering - This phase was essentially done for me given the nature of the 2018 task, anyone attempting the task was given the same training and testing data as everyone else, the only issue for myself was gaining access to the data which was eventually sorted.

Feature extraction - This involved removing non-essential words, characters, numbers and general text from the medical records, such as stop words and punctuation.

Build model - Build the model that will predict the outcome of a patient based on their medical records and will see if they do or not meet the criteria for a given classification such as drug abuse or diabetes.

Extra tasks - Improve the feature extraction techniques by preprocessing the data better and maybe looking for connection between words.

## 1.4 Planning

I had an initial overall plan for the project and for the most part this didn't change during my time actually attempting to complete the project. What did change however was the methods I would use. I knew this was a possibility and so went with an agile approach when it came to how I would approach the project, what I initially envisioned as a way of complete a task might not actually be viable once tested and maybe a better result could have been found. This would allow me a good degree of flexibility which meant I wasn't locked into a certain method of accomplishing my task.

The project was split into sprints with each week or two, depending on the sprint at hand, being dedicated to completing one task. As mentioned above this was open to change and often did as I would go back to change core components of my code once I found out that the current method of my approach was not viable or could be done better.

The approach I used was similar to my previous project called Scrumban which is a combination of Scrum and Kanban. This would let me split the work into sprints and make consistent progress even if the progress was relatively small. This would also mean I could look back at certain sprints and weight up whether or not I was happy with the work I had completed there or if I decided I should go back and rework it or at the very least optimise it.

# 2 Literature Review

The literature review I conducted was mostly centered around the 2018 challenge itself as well as various other NLP challenges. Numerous teams took part in the 2018 challenge as well as other challenges around the world and many of them published an paper or article on the research and results they had formulated whilst attempting these challenges. Whilst my project was also centered around medical data, the fact it was medical data was not all too important to the overall project, you could swap out the medical data for something more analytical and the results and processes would be fairly similar.

As mentioned earlier, one of the major issues with NLP is that computers are not built to understand it and so one of the first things to look up was the major problems facing NLP today and how other have overcome them. Below are some major problems as highlighted by an expert survey and panel discussion at the Deep Learning Indaba in 2018.

## 2.1 Main Issues Facing NLP

Firstly is Natural Language Understanding. "we should develop systems that read and understand text the way a person does"[7]. This quote is attributed to Kevin Gimple, a computer linguist from the Toyota Technological Institute at Chicago. According to him, until we are able to develop systems that read and understand text in the same way we do we are doing nothing more than improving a systems ability to match patterns. At the Deep Learning Indaba, many experts agreed with the above statement and argued that the problem of natural language understanding is central as it a prerequisite for many other tasks in the NLP sphere, such as language generation. The consensus was that no current model exhibit a "real" understanding of language and they are, as Kevin Gimple says in his quote, simply matching patterns.

One question that can be considered key is what biases should be built into a model on purpose. All models are built with biases as they are built by humans who are inherently biased. According to Karen Hao at the MIT Technology Review[8], there are three stages at which bias in a model can occur, this being when framing the problem, when collecting data and when preparing data.

For two of these stages I am not really in control and so cannot influence them in this project, however someone did set the challenge and someone did collect the data, so even though my biases may not be part of them they are still inherently biased. Firstly when a computer scientist sets out what they want to achieve, what problem they want to fix. For example with the 2018 challenge, the target was to set out to develop a system that could identify which patients meet selection criteria for clinical trials. But why were certain criteria picked over others? Could this be down to a bias in whoever decided the challenge, or was it due to what data they had available?

The second stage is collecting the data. Here there are two main ways in which bias can show in the data, first one being that the data is not representative of reality. It could be argued that the data provided in the 2018 challenge was not representative of the average person, this could be to cut back on useless records that would not meet any of the criteria but it could also be due to an existing prejudice or bias. A good example of an inherent prejudice is when Amazon created an internal recruiting tool that was dismissing female candidates because the model has been trained on previous hiring decisions which favoured men[9]. Another interesting paper article was put forward by ProPublica[10] on COMPAS, an artificial intelligence system that was designed to predict whether or not someone is likely to commit a crime and it had an implicit bias against African Americans and would double the amount of false positives than it would for Caucasians.

The final stage is the actual preparation of the data and is the one that would be mostly affected by me and my biases. Deciding what attributes are and are not "worth" using in a model. For example further into this report I

state how I used parameters that removed features if they were used in over 70% of records. This can be considered a bias in the system, as I personally decided that those words are not worth feeding into the models.

It's hard to know how to combat bias in machine learning models, especially in NLP. One reason is the unknown unknowns. You don't know what you don't know. Choices made at the start of project may only have an unintended affect much further down the line when the connection between the two seems irrelevant. In the case of Amazons internal hiring tool, the programmers didn't explicitly create the program with the intention of ignoring female workers, but by basing it off previous hiring decisions the model started to pick up on implicitly gendered words even once the system had been told to ignore the explicitly gendered words.

In the earlier mention expert panel, many argued that when developing a NLP system you should build what you know into the model. The difference of what should be learned and what should be argued was further debated between Yann LeCun and Christopher Manning[11].

Stephan Gouws, a senior research scientist at DeepMind argued in this panel that we should used information from available sources such as wikidata. He noted that humans learn through experience and interaction, similarly I mentioned earlier that humans have a wealth of knowledge and learning to call upon when deciding the meaning behind sentences and words, a computer does not. Whilst this could work it should be noted that the computing resources for such an idea would be enourmous. AlphaGo, the first computer program to defeat a professional human Go player[12] took a huge amount of infrastructure to win in what is seen as a well defined board game.

Another issue that I first thought would not apply to my project is how NLP will handle colloquialisms and slang. Whilst this is not going to be commonly used in medical data, sorted phrases or common slang within the field may well be. Instead of writing "25 year old male" this can be sorted to "25yo male". Whilst this may not seem like a huge different at first, having this happen multiple times in records spanning many years could be an issue. Another problem raised with this is that colloquialisms and slang are very personal and specific to a persons upbringing and culture. Where it might be common for one doctor to write "25yo male" as their shortened version of the word, another doctor may just write "male, 25" with the assumption that a person reading the data can work this out for themselves. This is a very simple example of the problem[13][14].

Finally another issue facing NLP that I did not even attempt to tackle and could be vitally important is the importance of temporal extraction and allowing a model to understand past, current and future clinical events. Understanding the connection between events such as the medication of a drug and the onset of a side effect or the clearing of a symptom is of major importance when it comes to NLP and clinical data. A particularly interesting study was posted in the Journal of Biomedical Semantics that looked at temporal information extraction and using this attempted to determine how long patients had been undergoing a form of psychosis without and treatment[15].

Looking at mental health from an NLP perspective may be even more difficult that looking at normal medical records. Normal medical records can be, for want of a better word, clinical in their nature, containing nothing but issues a patient has felt, exactly where the issue is affecting the patient such as the "pulmonary artery" and what was done to fix it. The same cannot be applied to mental health where a patient may describe their feelings or thoughts which to the average person may not make much sense to begin with.

## 2.2 Overcoming The Main Issues

For many of these issues there are no catch all solutions, or even any solutions at this point. The more I looked into the issue the more I saw that there is still a long way to go with many aspects of NLP. Some I would simply have to put more time and thought into, such as making sure that bias was not present in my models and if it was, how can I reduce the amount of bias within it[16]. This can be done at multiple stages as mentioned earlier by Karen

Hao, but again there is no method that is set in stone for this. One of the best ways is to look at the results of the data and go back and look at the system or the data provided to see what type of bias might cause these results and then make changes accordingly.

One of the key issues that can be overcome in NLP so far is how to handle data. The first is the preprocessing of data, standardization, lemmatization, tokenization etc[17], especially in regards to big unstructured data. Getting information from unstructured data can be difficult but this one problem that whilst not completely solved, is not a key issue anymore. This is highly relevant to my project as nearly all NLP will have to do some form of preprocessing but also because there is no standard way to write a patients medical records. With the data I was given it was simply split into TEXT and TAGS with no other structure to the data. Being able to work with this data into a format that could be more easily manipulated and understand would be key to this project.

As mentioned earlier, in many cases a computer might not understand exactly what the meaning behind words or phrases are, but outwardly they may appear to do so. This can be seen as a way of overcoming one of NLPs major issues. If the computer does or does not truly understand the meaning behind words, does it really matter if the end result is the same? Some neuroscientists don't believe they ever will truly understand[18]. In the case of this project, does it really matter if a computer knows what a heart it, does it matter if they know what diabetes is? So long as they can take the given information and provide a reasonable output that is helpful to the patient, that's what matters.

## 2.3 Literature Conclusion

Having looked at the literature available, I came to the conclusion that making any active gains in the field of NLP is far beyond the scope of this project. It does however provide insight into how truly complex the understanding of language is and how difficult it can be to create a system that truly understand natural language and also how easy it is to create one that seems like it understand natural language when it fact is it simply using patterns and numerical recognition.

Looking at existing articles on how to preprocess data, how to implement machine learning models, what solutions work well, which ones work badly etc has provided a large insight into what goes into NLP on the technical side and how it would be implemented and the issues it might face, but also whether true understand is truly needed for NLP. The Turing test for example shows how a computer can trick a person into thinking the computer is another person. Sociopaths who feel no emotion can trick people into thinking they do. Looking at what I did in the project attests to this, I managed to create a model that can match words with an output without truly understanding the words themselves or how they work or join together.

Overall the research has shown just how far we have to go in term of NLP and having true language understanding, what the end goals are for NLP and to some degree, how we can reach them. If someone where to expand upon this research I would recommend looking into novel approaches that are looking to properly understand natural language instead of just finding patterns between words.

# 3 Design

## 3.1 Overall Design

### 3.1.1 Language

For a project such as this there are a number of different languages and libraries available however the general consensus seems to be to use Python for most machine learning projects, it's very simple to learn and write, it has some of the best machine learning support in the form of libraries like NLTK and SciKit-Learn. It also has the added benefit of being the language used in Google Colab and is the language most used when looking up articles or tutorials on how to accomplish a machine learning task, by default they are almost always done in Python and you have to go out of your way to find NLP tutorials in other languages such as Java or R.

Having used both Java, R and Python in the past technically all three of them were viable for me to use however in the context of machine learning I had only every truly used Python which again pushed me more in that direction. Java is one of the most popular languages out there and so there would be heaps of information available to me and R is also commonly used for NLP, it plays a crucial role when it comes to investigating big data. Java however is more complex from a syntax point of view and would require more time and effort being dedicated to actually writing the program itself rather than focusing on the overall idea of how I would achieve my goal. I wanted to be to just write the program and have it work and not have to worry. R whilst being a common language in NLP is still not that common of a language, routinely falling behind both Python and Java in terms of real world usage[19].

Ultimately I decided to use Python due to my familiarity with language and the main libraries needed and also the sheer amount of online information that would be available to me.

### 3.1.2 Libraries

When it came to libraries I looked mainly at NLTK, SciKit-Learn and Pandas. NLTK is one of the most, if the the most used library in Python for language processing and is widely considered to be essential when it comes to supporting classification and tokenization in Python. It provides a corpus of stop words to remove from the text to help narrow down the number of features present.

When it comes to actually classifying the data given and training it then one of the best libraries by far is SciKit-Learn. It provides a wide array of algorithms for building an actual machine learning system with smaller helpful imports such as the train-test-split method, and it is well documented and is used in most tutorials or articles relating to NLP with Python. One downside however is that it does not use neural networks for text processing and so it is not advised to use it for more complex preprocessing tasks.

Pandas was also essential for handling the data and placing it in easy to use structures. It would allow me to read in CSVs as Data Frames which was the main method of reading in the data to a machine learning model. Like with NLTK and SciKit-Learn, Pandas is commonly used hand in hand and is also used in most examples and articles due to it's easy of use.

Beautiful Soup was not a library I had used previous to this project but came in very handy when it came to pulling the data from the training data files I had been given. Bs4 as it is also known is Python package used for parsing XML and HTMl documents. The data I was given was sorted solely in XML files and so was used in place of the XML ElementTree when it came to parsing the XML data.

There were a number of other libraries and imports used that were not considered essential to this task but did make the experience easier. NumPy was used to small degree within the project when it came to saving the data at certain points. Smaller imports were used such as os and re to name two, these again while not essential but

did help with grabbing information from files within directories or with preprocessing and the removing of certain characters and strings.

#### 3.1.3 Storage

As mentioned above all data was provided in an XML format initially however it was easier to convert this into CSV files with each patients records being a single row in the CSV file and the words used being separated into columns or features. There was no need to make this overly complicated as it was simple enough to read in and write using Python and Pandas once in a CSV format. All the data was stored on my Google Drive account as it could be mounted to Google Colab for easy access. This does not caused any security issues as the medical records are already freely available and not confidential but have also been edited so preserve the privacy of the original patient with names, dates, times and places all changed. Any results in terms of accuracy of the model were also stored on my Google Drive in simple text files.

### 3.2 Detailed Design

The project is split between 3 main sections, these being the main program and all parts it contains and the data sets themselves with the main program split into two parts, these being the preprocessing section and the training and testing section.

#### 3.2.1 Preprocessing

Training Data - The folder that contained the XML files with all the training data patient records.

Test Data - The folder that contained the XML files with all the testing data patient records.

Preprocessing - The method that would prepare the above data by combining them into one data set and cleaning that data up e.g. by removing stop words.

XList - A number of CSVs that would contain the tags from the patient records according to their criteria e.g. a list of all the tags from every patient record that provided whether or not the patient met the criteria for alcohol abuse would be found in the alcoholList CSV file.

#### 3.2.2 Training and Testing

countVectorizer - A method of extracting features from the trainingData using the CountVectorizer.

tfidfVectorizer - A method of extracting features from the trainingData using the TfidfVectorizer.

featureNaming - A section of the code that would go through a name each feature numerically from 0 to X, able to scale with any number of features.

fileAmending - A section of code that would amend the target tag file to the end of the trainingData file for use in training and testing.

trainDF - A Dataset that contained just the X feature variables and Y out variables from the trainingData once training and testing had been separated.

testDF - A Dataset that contained just the X feature variables and Y out variables from the testData once training and testing had been separated.

GNB - A machine learning model that used the Gaussian Naive Bayes classification model to train the classifier and then test said classifier against the test data.

MNB - A machine learning model that used the Multinomial Naive Bayes classification model to train the classifier and then test said classifier against the test data.

LR - A machine learning model that used the Logistic Regression classification model to train the classifier and then test said classifier against the test data.

SVM - A machine learning model that used the Support Vector Machine classification model to train the classifier and then test said classifier against the test data.

#### 3.2.3 Data Sets

trainingData - A CSV file that contains all the data needed to train the model, including both the data from the trainingData folder and the testData folder.

countFeatures - A CSV file that contains all the features extracts from the trainingData once ran through the CountVectorizer.

tfidfFeatures - A CSV file that contains all the features extracts from the trainingData once ran through the TfidfVectorizer.

Results - A folder containing all the text files with the accuracy results of each model and how well they performed on a given criteria.

### 3.3 Tools Used

Having mentioned earlier on, I choose to use Google Colab for this project which already comes with it's own version control. This meant I would not need to use an form of online repositories such as Github or Gitlab. I wouldn't say I have used Python extensively so having access to an easy to use and set up IDE was of great use to me during this project and was much easier to naviagate and use as opposed to something like PyCharm.

The automatic cloud saving and version control also helped put my mind at ease as it meant I could make changes to my program without having to worry about saving every 5 minutes for fear of losing any progress I had made, and the in-built version control provided my with an easy way to roll back the project if I had made a change I believed was detrimental or simply was not going to work.

Related to that I also used Google Drive as the storage for my permanent files such as the trainingData and Results, as this was easy to access via Google Colab allowing me to mount my Google Drive. It also meant I had easy access to all my data should I run into any issues with my computer breaking or even not having access to the internet. I could just go up to the University campus and access my work from any computer without having to go through the setup that would normally come with cloning a project and setting up an IDE from scratch.

# 4 Implementation

## 4.1 Data Gathering

The first step of my project was to gather together all the data. In it's current form the data was split into training data which contained 202 annotated files and testing data which contained 86 annotated files. From here I had two options, as I wasn't actively taking part in the 2018 challenge I did not have to strictly use the training data and testing data separately. I could combined it all into one data set that I preprocess, extract features from and then train and test on. Since all the data including that of the testing data was labelled with the Y output variable the results would have largely been the same. Combining it all and then doing some form of train-test-split would have been viable. The other option was fairly similar as it would involve combining the data together however before training I would then split the data up again into training and testing so that they were both used separately, as would need to be done if I was taking part in the 2018 challenge for real.

I decided to go with the second option as following the method used by the other teams in the challenge would make for more accurate and fairer comparisons when comparing my end results. I did however have to make sure that I split the data back into it's training and testing sets before I train the data and not after. If I train the model using both the training and testing data when they are combined into one data set then I am giving the model a peek at what will be inside the testing data, which then gives it an advantage when it comes to predicting. In theory the model should not know what data it is going to be tested against and so by giving it that peek it gives it an undue advantage which makes the model seem more accurate than it otherwise is when it comes to evaluation. This is known as data leakage, which is when information from outside the training set is used to train and create the model[20].

## 4.2 Data Processing

### 4.2.1 TEXT Processing

Once I had decided what I wanted to in regards to the data, I then had to process it. This was the first major step and is also one of the most import steps when making a machine learning model. Firstly the program would iterate through the given directory and would get each XML file and extract all the info from said XML. Once converted using Bs4, the record is split into two main components, TEXT and TAGS. TEXT is all the information within the actual record itself and TAGS are the tags given to the data letting the user know if this patients records fulfil the needed criteria for certain medical issues such as diabetes.

```
]]></TEXT>
<TAGS>
<ABDOMINAL met="not met" />
<ADVANCED-CAD met="met" />
<ALCOHOL-ABUSE met="not met" />
<ASP-FOR-MI met="met" />
<CREATININE met="not met" />
<DIETSUPP-2MOS met="met" />
<DRUG-ABUSE met="not met" />
<ENGLISH met="met" />
<HBA1C met="not met" />
<KETO-1YR met="not met" />
<MAJOR-DIABETES met="met" />
<MAKES-DECISIONS met="met" />
<MI-6MOS met="met" />
</TAGS>
</PatientMatching>
```

Figure 1: Example of TAGS included with XML records

Once the TEXT from the XML file had been put into a string the next step was to then go through that string of text and clean it to make it ready for feature extraction and training later.

Some of the more common steps are[21]:

- Removing URLs from online text based data
- Removing all forms of punctuation
- Removing stop words
- Converting everything to lower case
- Tokenization of the words
- Stemming
- Lemmatization

These steps can for the most part be done in any order, but generally it works best if they are done in a certain order for example removing any form of URL first is better than removing to punctuation in the URL to only then be left with a string of text that out of context doesn't make as much sense. Due to the nature of my data being medical records and being in an XML format I didn't have to worry about this step. To prepare my data I would remove all special characters (e.g. { }, %, #, @, !, etc), then I would remove all numbers from the text. After this I would removed all the single characters from the text as the removal of special characters would turn words such as "it's" into "it s". Then I would removed any multiple spaces and replace them with a single space and convert all text to lower case. Finally a tokenizer would be run over the string that would divide the string into sub-string by splitting them on the specified string, in this case wherever there was a space between words. Each word is now it's own separate string.

Once this is done the next step is to go through each word and determine if it is a stop word or not. Stop words are a set of commonly used words in language[22], regardless of what the topic is, for example in the English language the most commonly used stop words are "a", "the", "is" and "are". Whether someone is writing a technical manual,

report, medical record, review or newspaper articles, these words are likely to appear. Due to their prevalence in language they don't provide any sort of information or clue about what the body of text is about. If you had to try and guess what a body of text is about and you saw the words "review", "character", "writing" and "plot", you could probably hazard a guess and say it's about a film or book without the use of stop words, ultimately they hold little lexical content and "their presence is a text fails to distinguish it from other texts"[23]. By keeping these words in it just pollutes the data and by removing it we can focus on the more important information. For my program I used the NLTK library that already came with a corpus of stopwords labelled as "English". This removes stopwords specifically found in the English language which is what all my patient records are written in.

My next step was lemmatization. lemmatization is similar to stemming but with the ability to make sure that the word does not lose meaning. When stemming words like "Programmer", "Programming" and "Program", they will all be stemmed down to "Program". Whilst this can lower the amount of words in a body of text is it not a full proof method as some words such as "available" might be stemmed down to "avail" and so can possibly lose its meaning. With lemmatization this is avoided by stemming the words and then grouping similar words together and changing them all into the same similar word. For example "Goose and "Geese" are stemmed down to "Goos" and "Gees" but after lemmatization are both turned into "Goose". This helps reduce the number of words even further without losing the meaning of said words. This alone reduced the amount of features extracted in a later process from 21,365 down to 19,898, a change of 1467 features or around 6.8%.

After this the final step was to write every tokenized word in the list to a CSV file with each XML file being a row and each word in said record being a column. This process is done on every XML file in the directory until all the files have been processed and each file is subsequently amended to the end of the CSV file until it looks something like this:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| text | record | date | mercy | care | center | mercy | internal |
| text | record | date | campbell | orthopedic | associate | madera | circle |
| text | record | date | personal | data | overall | health | patient |
| text | record | date | huntington | emergency | dept | visit | thomas |
| text | record | date | jennifer | booker | lc | unit | nashua |
| text | record | date | august | dr | mabel | duvall | medicine |
| text | record | date | northwest | iowa | health | center | date |
| text | record | date | edvisit | root | percy | carr | rachel |
| text | record | date | jensen | clinic | greenfield | drive | here |
| text | record | date | mr | villegas | seen | today | seen |
| text | record | date | dameron | emergency | dept | visit | valenzuela |
| text | record | date | silver | ridge | emergency | dept | visit |
| text | record | date | personal | data | overall | health | year |
| text | record | date | bch | emergency | dept | visit | hess |
| text | record | date | ah | emergency | dept | visit | branch |
| text | record | date | melissa | cummings | hgh | unit | saint |
| text | record | date | ocampo | nathan | oliver | hdh | internal |
| text | record | date | july | jacob | bautista | eldorado | circle |
| text | record | date | problem | diabetes | mellitus | hypertension | psoriasis |

Figure 2: Excel spreadsheet showing the tokenized words

This is then done to the testing data after the training data so that the testing data is all amended to the end of the file and thus grouped together for later use.

#### 4.2.2 TAG Processing

One the text from the records has been successfully processed and stored I had a CSV file containing all the data that would be used as the X variable for training the model. What came next was getting the Y variable or the output/target variable that the model would be trying to predict. At the bottom of each XML file were the TAGS as seen earlier in picture X. These needed to be sorted so that they could be used as the Y value when training the model but this depending on how I wanted to train the model. I will cover how I decided I wanted to train the model in the next section but ultimately I decided to grab each individual tag from each record and amend them to their respective list and CSV. So first I would look at the first XML file in the directory and grab only the TAGS using Bs4. These TAGS would then be sorted into lists. For example if the first set of TAGS for the a record are as follows:

- ABDOMINAL met="not met"
- ADVANCED-CAD met="met"
- ALCOHOL-ABUSE met="not met"
- ASP-FOR-MI met="met"
- CREATININE met="not met"
- DIETSUPP-2MOS met="met"
- DRUG-ABUSE met="not met"
- ENGLISH met="met"
- HBA1C met="not met"
- KETO-1YR met="not met"
- MAJOR-DIABETES met="met"
- MAKES-DECISIONS met="met"
- MI-6MOS met="met"

These would then be sorted into lists such as: abList = ["not met"], adcadList = ["met"], alcoholList = ["not met"] etc.

If the next XML file instead had ABDOMINAL met="met" then it would add this to the abList which in turn would now look like this, abList = ["not met", "met"]. This would happen to every XML file in both training and test data until I was provided with 13 lists of data consisting of "met" and "not met" in the same order in which the records had been read in. These lists were then turned written into CSVs with each attribute in the list being it's own row so they would correspond with the TEXT data that has been preprocessed and added to the TrainingData CSV.

### 4.3 Building the Model

Instead of training one singular model I decided it would be better to train multiple models and compare them, different models work differently depending on the data they are provided and some are better for NLP than others. One key issue however is how I wanted to go about training the models. Due to how the data was provided, I decided to tackle this project from the point of view of it being a binary classification. Each patient has multiple criteria they could fulfil which one could argue would make it a multi-classification issue but I'm not trying to figure

out if the patient falls into one of these criteria but if they do or not fulfil every criteria which themselves only had 2 classifications, these being "met" and "not met".

I decided I would train each one a number of times with varying levels of features ranging from 100 to 5000 to see how the features affected the accuracy of each model. This meant I would only solve one criteria at a time for each patient, but I could run through all 86 testing records and determine if each individual one did or did not meet the criteria for say abdominal issues. This means that whilst I couldn't run through a single patient and classify each individual criteria I could run through multiple patients classifying one of the criteria and then simply repeat this process for every criteria I wanted to test.

## 4.4 Feature Extraction

### 4.4.1 Count Vectorization

Now that the data is ready and I have chosen how I was going to implement the model, the next step was to turn the data into something the computer can understand and work it. Initially the computer cannot handle text data, only numerical. To transform it into numerical data that can be understood is called featurization. What this does it turns each word in the trainingData CSV into a feature. For example if we were to look at the sentence "The man types on his keyboard" we would have a total of 6 features like this:

Table 1: Feature Example

| **THE** | **MAN** | **TYPES** | **ON** | **HIS** | **KEYBOARD** |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 |

We can see that each word is a feature and is currently present in the given text. But if we had another sentence such as "The boy types on his phone" we would end up with a total of 8 features like this:

Table 2: Feature Example 2

| **THE** | **MAN** | **BOY** | **TYPES** | **ON** | **HIS** | **KEYBOARD** | **PHONE** |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |

Here we can see that the first sentence contains the features "THE", "MAN", "TYPES", "ON", "HIS", "KEYBOARD" but not the features "BOY" and "PHONE". If a sentence were to have multiple of the same feature or word then it would appear as a 2 or more in the above table. Because text is not easily understood by computer it must be turned into numerical data like in the above example, this particular example is one called Count Vectorization. What Count Vectorization does is it reads through a body of text, in this case the trainingData and finds how many individual and unique features there are in it, in this case it would read through the data and find every unique word and then list how many times it appears in a given record.

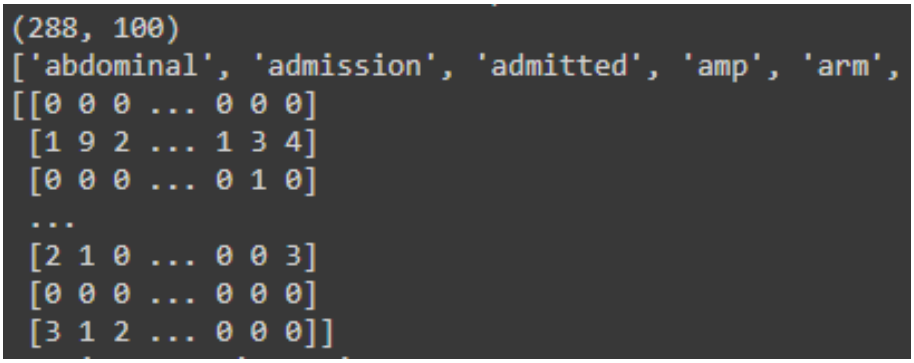


Figure 3: CountVector extracting 100 features

In the above example the Count Vectorizer is extracting only 100 features from a potential 19,000+. We can see that in the very first row of the trainingData CSV that the first three features it reads, these being "abdominal", "admission" and "admitted" are not present, but in the very last row of the CSV file they are present 3 times, one time and two times. So not only can we increase and decrease the amount of features we want to extract from the trainingData to use when actually training the model, CountVector also has a number of parameters that can be used to change how it grabs the desired features. The first one used is the max_features parameter that simply determines the maximum amount of features the CountVector will grab from a body of text. What this does is it will grab features or words that occur the most throughout the entire text. So by setting the max features to 100 in the above example we know that abdominal is one of the 100 most used words through the entire body of text.

Next we have max_df. This has a similar use to remove stopwords and so helps reduce the number of features even more. If we set max_df to 0.7, as used in my actual project, it will look for any feature that appears in over 70% of the documents or rows in the data frame. If a feature does appear in over 70% then that feature is ignored. The same can be set with min_df where a feature must appear in a minimum number of documents before it can be chosen as a feature. If the user had not done any form of preprocessing this would be useful for remove words with high frequency such as "the", but since this was already done then it allows us to remove even more useless features. A good example in my project was the start of every record in the trainingData CSV start with the words "Text, record, date" due to every XML files containing those words. They're not considered stopwords and so were not removed in the preprocessing stage but they also shouldn't be considered relevant to the data and so can be easily ignored using the max_df parameter.

The final parameter I used was the ngram parameter. One issue with the tokenization method is that it splits every phrase and sentence into individual words. This means that phrases may lose their meaning once split into individual words. For example the phrase "good food" means more when the words are together and less when the words are then split up and observed independently from one another such as "good" and "food". ngrams allows a user to get around this by allowing combinations of words to stay together as a single feature so that they might carry more meaning. Before this was used, the Count Vector would only grab singular words from the CSV as seen in figure 1 above. But with the addition of the ngram parameter we can some words are put together to create a more meaningful phrase.


Figure 4: CountVector with ngrams

Here we can see that coronary is used in both a singular term and also in the phrase "coronary artery". This gives us more meaning with that phrase as we now know that the wording "coronary artery" is used enough to be significant and specifically points to the heart whereas before that we could only really guess that the separate words of "artery" and "coronary" might have been connected.

#### 4.4.2 TF-IDF Vectorization

Another method of feature extraction is using the TF-IDF Vectorizor or Term Frequency - Inverse Document Frequency. This works in much the same way the CountVector method, I even set the same parameters so it would grab the features, the only key difference however is that instead of listing the features present in a body of text and how many times they are present as an integer, it instead looks at how often a word appears within a document and compares it against all the documents. So a word that appears many times in a single patients medical records but don't appear in any other records will be given a higher score whereas words that appear frequently across all documents will be given lower scores.

Table 3: IF-IDF Example Output

| VOCABULARY | IDFS | VECTORS |
|---|---|---|
| This | 1.91629073 | 0 |
| is | 1.22314355 | 0.43877674 |
| the | 1.51082562 | 0.54197657 |
| first | 1.22314355 | 0.43877674 |
| document | 1.91629073 | 0 |
| second | 1.91629073 | 0 |
| and | 1.0 | 0.35872874 |
| third | 1.91629073 | 0 |
| one | 1.22314355 | 0.43877674 |

Given the phrase "This is the first document. This is the second second document. And the third one. Is this the first document?", we can see how the TF-IFD vectorizer would score these words. And is scored at 1.0, the lowest score it can be given so that no word is completely ignored meanwhile this, document, second and third are some of the highest scoring and are deemed the most relevant.

```
Printing Count Vectorizer Info:
(288, 100)
['abdominal', 'admission', 'admitted', 'amp', 'arm', 'artery', 'asa', 'atenolol', 'bilateral', 'bruits', 'cad', 'cc', 'chronic', 'coronary', 'coronary artery',
[[0 0 0 ... 0 0 0]
 [1 9 2 ... 2 1 3]
 [0 0 0 ... 2 0 1]
 ...
 [2 1 0 ... 5 0 0]
 [0 0 0 ... 3 0 0]
 [3 1 2 ... 2 0 0]]

Printing tfidf Vectorizer Info:
(288, 100)
['abdominal', 'admission', 'admitted', 'amp', 'arm', 'artery', 'asa', 'atenolol', 'bilateral', 'bruits', 'cad', 'cc', 'chronic', 'coronary', 'coronary artery',
[[0.         0.         0.         ... 0.         0.         0.        ]
 [0.02366336 0.19835206 0.0439162  ... 0.04473486 0.02557581 0.06587429]
 [0.         0.         0.         ... 0.06956408 0.         0.03414552]
 ...
 [0.01542584 0.00718351 0.         ... 0.03645259 0.         0.        ]
 [0.         0.         0.         ... 0.0580644  0.         0.        ]
 [0.11436647 0.03550546 0.07074988 ... 0.07206876 0.         0.        ]]
```

Figure 5: Comparison of CountVector and TF-IDF extracting the same 100 features but with different scores

Whilst the max_df may get rid of a lot of features that are prevalent in most documents, using TF-IDF helps to further narrow down which of the remaining features is deemed more interesting and could potentially train a better model.

## 4.5 Splitting The Data

A very small but none the less important section of my project was splitting the data back into the training and testing data. In this section the program would ask what CSV feature file the user wants to use, either CountVector or TF-IDF. Once this is chosen the program then grabs that file, reads it in as a CSV and replaces all NaN values with 0 should there be any in the data. It then takes the chosen "target" list file e.g. abList.csv or dietsuppList.csv which contains the "met" or "not met" tags from earlier and appends as the last column in the CSV and converts them into 0s and 1s with 0 being "not met" or negative and 1 being "met" or positive. The chosen feature file now has all the needed features and the outcomes associated with them. From here the last 86 rows of the CSV or data frame are dropped and added to the test data frame. We now have 2 separate data frames, one for training the machine learning model and the other for testing it, each one containing all the needed X feature variables and Y output variables. I then set the X features and their Y outputs or "Targets" as the X_train and Y_train variables respectively. I do the same for the X_test and Y_test with the testing data. The data is now fully prepared to be loaded into the model.

## 4.6 Training the Model

From here the data is loaded into the model in a loop, each time with a varying amount of features based on a list set by the user, in this case 100, 250, 500, 1000, 2500 and 5000 to start with. Originally each model was trained on the given features 100 times and every time the model was trained it was then tested and it's prediction given an accuracy score. This would then be added up and the overall accuracy was then divided by number of run times to provide an average accuracy score for that model. This however was due to me using the train-test-split method which would take the training data and split it 80/20 with 80% of the data used to train the model and then the other 20% used to test it. This meant that the data used to train and model and the data used to test it would be different each time depending on what data was chosen.

Once the rest of the program had been sorted including splitting the training data and testing data back into separate data frames, each model only needed to be ran once and would produce the same results each time, thus cutting down on training time by a significant margin. Some runs could take as much as 10 minutes given the number of features and run times but now has been cut down to sub 1 minute, with an average of 40 seconds, this

includes extracting a different number of features each time the program is run.

### 4.6.1 Gaussian Naive Bayes

"Naive Bayes is a statistical classification technique based on Bayes Theorem".[24] Gaussian Naive Bayes or GNB, was one of the first machine learning methods I implement due to its simple classification technique. GNB assume that features follow a normal distribution[25], also visualised as a bell curve mean data near the mean being more frequent than data far from the mean.

Whilst being a very simple classification method, it isn't great for NLP. One reason being that the NB classifier assumes that the effect of one feature is unrelated and independent of other features. So if two different words or features tend to show up together across the data set, NB will assume that there is no connection between the two. This simplification of the features is where it gets the name Naive from. Whilst it may be possible for a set of features to be entirely independent of one another, in reality this is essentially never the case and certainly not the case with the data I am working with.

It does however have some advantages which are the reason I chose it as a baseline to compare the other three models against as they would be the main focus of the system. For starters it is a fast and still relatively accurate method for prediction, it can work well with large data sets and has low computational cost.

### 4.6.2 Multinomial Naive Bayes

Multinomial Naive Bayes or MNB was my next choice to compare against Gaussian, they both share many of the same advantages however Multinomial considers a feature vector where a given term represents the numbers of times it appears in comparison to Gaussian's continuous distribution. It is commonly used in text classification as it works well for data that can be easily turned into counts, such as word counts in text. For this reason MNB was a natural second choice.

### 4.6.3 Logistic Regression

Next was a classification known as Logistic Regression or LR and is another relatively simple to understand classification. Regression itself is a type of supervised learning but generally has a different output to classification. In regression the output variable is a numerical value that exists on a continuous scale, usually represented as a floating point. This means that two outputs could be completely different floating point numbers but still technically be in the same classification.

LR is used when the data in question has a binary output, in this case whether or not the output belongs in the "met" or "not met" class, and so was a natural pick for one of the classifiers to test. The different with LR is that there is a threshold at which point an output variable will move from one class to another. For example a threshold of 0.5 would mean that any output with a 50% chance or greater of being considered "met" will be placed in the "met" class and anything below that will be placed in the "not met" class. These threshold are not always set to 50% however. This allows LR to classify outputs but with varying degrees of certainty or probability.

### 4.6.4 Support Vector Machine

Finally there is Support Vector Machine or SVM. SVM is another widely used classification technique although it can also be used for regression as well. The main goal of an SVM is to find a hyper-plane in an N-dimensional space (where N is the number of features)[26] that lets it distinctly classify data points. Support Vectors themselves are the data points that are near to the hyper-plane. These are crucial as removing them would alter the position of the dividing hyper-plane which would in turn alter where the borders for the classes change. A hyper-plane itself

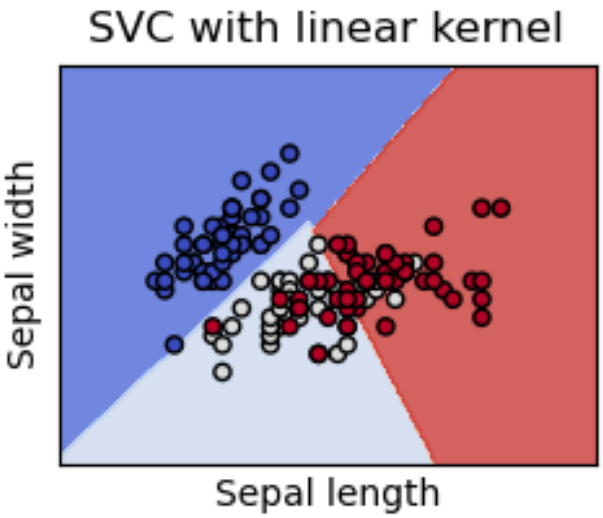


Figure 6: SVM classifying some data points using the popular Iris dataset [27]

in the context of my project that only has two classes, is a line that separates and classifies the data. The further away from the line a data point is, the more confident the SVM is about saying that that specific data point is part of that class.

The distance between the hyper-plane itself and the two nearest data point (or Support Vectors) is known as the margin. The greater this margin, the more confident we are in saying that the classes are separate from each other. The end goal of the SVM is to choose a hyper-plane that has the greatest possible margin between classes.

SVM has different kernels to choose from depending on the situation and can be chosen accordingly, such as Linear, RBF or Polynomial.

Looking at the example above, we can see for the most part the hyper-plane that separates the blue data points from the red data points is fairly clear, same with the blue from the white colour. The data points in the blue class are relatively bunched up apart from one or two outliers. The same cannot be said for the line between red and white however, the data points are overlapping and they're all bunched up as though they were one big data point. From this we can see that the blue class is easy to distinguish from the others, however the SVM had trouble distinguishing what made one data point fall into the red class and what made another fall into the white class.

## 4.7 Results

At first I generated lists of results from varying levels of features, ranging from 100 to 5000 and using both Count Vectorizer and TF-IDF as the feature extracting methods. I eventually realised however that this would be tricky to collate and would be an overload of unnecessary information and so instead decided pick TF-IDF as my main method for feature extraction with the same parameters mentioned earlier such as max_df = 0.7 and ngram = (1,2). One crucial change I made however was I also added a min_df that means it would only pick features if they appeared in over 10% of the records but under 70% of them. The reason this was crucial is because if I did not add the min_df then the TF-IDF vectorizer would extract around 262,694 features from the text and this was far too many and increased the machine models learning time to above 30 minutes at which point I cancelled the run and decided it wasn't worth the time. Having the min_df parameter in place set the number of features to a much more reasonable 2785 and could complete the training and testing in around 10 seconds.

Table 4: F1 Scores for GNB

| | F1 Scores |
|---|---|
| ABDOMINAL | 0.70 |
| ADVANCED-CAD | 0.61 |
| ALCOHOL-ABUSE | 0.95 |
| ASP-FOR-MI | 0.74 |
| CREATININE | 0.70 |
| DIETSUPP-2MOS | 0.62 |
| DRUG-ABUSE | 0.95 |
| ENGLISH | 0.78 |
| HBA1C | 0.58 |
| KETO-1YR | 1.00 |
| MAJOR-DIABETES | 0.78 |
| MAKES-DECISIONS | 0.95 |
| MI-6MOS | 0.86 |
| AVG | 0.78 |

Table 5: F1 Scores for MNB

| | F1 Scores |
|---|---|
| ABDOMINAL | 0.53 |
| ADVANCED-CAD | 0.48 |
| ALCOHOL-ABUSE | 0.95 |
| ASP-FOR-MI | 0.70 |
| CREATININE | 0.71 |
| DIETSUPP-2MOS | 0.56 |
| DRUG-ABUSE | 0.95 |
| ENGLISH | 0.78 |
| HBA1C | 0.48 |
| KETO-1YR | 1.00 |
| MAJOR-DIABETES | 0.60 |
| MAKES-DECISIONS | 0.95 |
| MI-6MOS | 0.86 |
| AVG | 0.73 |

Table 6: F1 Scores for LR

| | F1 Scores |
|---|---|
| ABDOMINAL | 0.57 |
| ADVANCED-CAD | 0.47 |
| ALCOHOL-ABUSE | 0.95 |
| ASP-FOR-MI | 0.70 |
| CREATININE | 0.79 |
| DIETSUPP-2MOS | 0.56 |
| DRUG-ABUSE | 0.95 |
| ENGLISH | 0.78 |
| HBA1C | 0.49 |
| KETO-1YR | 1.00 |
| MAJOR-DIABETES | 0.68 |
| MAKES-DECISIONS | 0.95 |
| MI-6MOS | 0.86 |
| AVG | 0.75 |

Table 7: F1 Scores for SVM

| | F1 Scores |
|---|---|
| ABDOMINAL | 0.56 |
| ADVANCED-CAD | 0.64 |
| ALCOHOL-ABUSE | 0.95 |
| ASP-FOR-MI | 0.70 |
| CREATININE | 0.81 |
| DIETSUPP-2MOS | 0.51 |
| DRUG-ABUSE | 0.95 |
| ENGLISH | 0.78 |
| HBA1C | 0.47 |
| KETO-1YR | 1.00 |
| MAJOR-DIABETES | 0.75 |
| MAKES-DECISIONS | 0.95 |
| MI-6MOS | 0.86 |
| AVG | 0.76 |

#### 4.7.1 Overall Results

Below is a combined table showing all the results from all 4 of the machine learning models, each machine learning models average and how on average they did at predicting certain criteria.

Table 8: All accuracy scores

| | GNB | MNB | LR | SVM | AVG |
|---|---|---|---|---|---|
| ABDOMINAL | 69.76% | 59.30% | 65.11% | 61.62% | 63.94% |
| ADVANCED-CAD | 62.7% | 54.65% | 53.48% | 65.11% | 58.98% |
| ALCOHOL-ABUSE | 96.51% | 96.51% | 96.51% | 96.51% | 96.51% |
| ASP-FOR-MI | 80.23% | 79.06% | 79.06% | 79.06% | 79.35% |
| CREATININE | 69.76% | 75.58% | 81.39% | 81.39% | 77.03% |
| DIETSUPP-2MOS | 61.62% | 55.81% | 55.81% | 55.16% | 64.60% |
| DRUG-ABUSE | 96.51% | 96.51% | 96.51% | 96.51% | 96.51% |
| ENGLISH | 84.88% | 84.88% | 84.88% | 84.88% | 84.88% |
| HBA1C | 59.30% | 59.30% | 59.30% | 58.13% | 59.00% |
| KETO-1YR | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% |
| MAJOR-DIABETES | 77.90% | 63.95% | 69.76% | 75.58% | 71.79% |
| MAKES-DECISIONS | 96.51% | 96.51% | 96.51% | 96.51% | 96.51% |
| MI-6MOS | 89.53% | 90.69% | 90.69% | 90.69% | 90.40% |
| AVG | 80.40% | 77.90% | 85.32% | 80.08% | 79.96% |

Considering GNB was only chosen to help establish a baseline, I was surprised by how well it fared in this project. It managed an average F1 score of 0.78, the highest F1 score of the 4 models. It's lowest score was 0.58 on the HBA1C criteria however this was still higher than all of the other machine learning models F1 score for this criteria by a significant margin. It also attained the second highest average accuracy score, slightly beating SVM.

MNB fared the worst by a significant margin, falling 3% behind the next highest average accuracy and a whole 8% behind GNB. It managed an F1 score of 0.73, which was the lowest F1 score put forward by all the models. It's lowest F1 score was 0.48 getting this on both the Advanced-Cad and English.

LR also did relatively well, getting the highest average accuracy at a total of 85.32% and getting a middle of the road average F1 score of 0.75. It's lowest F1 score was 0.47 on the Advanced-Cad criteria.

Finally is SVM, which I personally expected to fair the best out of all 4 of the models. It did average getting a middle of the road score for both it's average accuracy and it's average F1 score at 80.08% and 0.76 respectively.

SVM managed to get the lowest F1 score of the group getting 0.47 on the HBA1C criteria.

A few things worth nothing, every model managed a 100% accuracy rate on the Keto-1YR criteria. This is due to the Keto tags all being "not met" in both the training and testing data, so any combination of features was always going to result in a 100% accuracy rate. Due to this the Keto results can be safely ignored as the model would likely always provide a "not met" output. A few of the models also obtained the exact same accuracy for the same criteria, such as all models getting an accuracy rating of 96.51% for ALCOHOL-ABUSE, DRUG-ABUSE and MAKES-DECISIONS. This shows that alcohol and drug abuse are fairly easy to recognize in a body of text, this is likely to doctor records simply using the terms alcohol or drugs when referring to a patients past.

## 4.8 Further Implementation

Had I had more time for both research and implementation, and had I not changed my overall method and goal on numerous occasions, there are a number of things I would have liked to have implemented. One of the first ones would be reducing the amount of data initially gathered from the patients records. All the patient records were dated and so it could be possible to only use recent data when determining if a patient fulfils a criteria. If a patient had a history of drug abuse but it has been 10 years since any records mentioned it, chances are in the most recent record the patient does not fulfil the criteria for drug abuse and so should not be placed in the drug abuse class.

There were a few things I would like to implement in terms of simplifying the who process and getting it running smoother. For example in the current form I had to run the program every time I wanted to train the model and test it using a given list of tags such as the abList CSV. Whilst the actual training and testing was done relatively quickly it was just inefficient and unnecessarily time consuming. Putting it all into some form of loop that could run once and load in each CSV in order would help save some time.

Another implementation would have been using Maximum Likelihood Estimation or MLE along with the Logistic Regression. Logistic Regression like most models has a number of parameters it can use, and one of the best ways of figuring out these parameters is the MLE framework. What this does is it estimates the parameters of the model by trying "to maximize the conditional probability of observing data given a specific probability distribution and its parameters".[28] Given that the implemented version of LR got the best accuracy and second best F1 score, it would be interesting to see if this would have increased or decreased the accuracy by any significant margin.

Finally there is Word2Vec. In the current form, apart from finding ngrams within the text there is no real way to find word associations within a body of text. Word2Vec would seek to solve this. Word2Vec uses a neural network to learn word associations within the given body of text. Once this has been trained it can then detect synonymous words. However I didn't feel this would be massively important to the overall system, as finding synonyms is not going to be common within medical data, a doctor is unlikely to use medical terms when referring to things such as arteries or abdomen. Finding the associations within words could prove to be useful though as certain words are inevitably going to be associated with certain criteria.

# 5 Critical Evaluation

## 5.1 Evaluation

The main aim of this project was relatively open ended, there was no goal set in stone, however I set myself a task of trying to create a machine learning system that can take a patients medical records and provide an output that would be considered reasonably accurate using the same data provided by the 2018 challenge. I would say that ultimately this goal was achieved, when tested against the provided testing data it was rare for any of the trained models to fail to correctly predict at least 50% of the records with the average overall prediction across the board being 79.96%.

I believe Python was the correct choice for the project, it felt easy to use and at no point did I ever feel held back due to the language I was using or due to a lack of skill or familiarity with that language. The libraries are a similar case, at no point did I ever feel like the library was the blocking factor nor did I feel like the libraries were not supported well through either documentation or online tutorials.

I could have expanded my horizons more and looked beyond NLKT and Sci-Kit Learn and maybe looked into trying deep learning methods instead and I could have then compared the best classifier within Sci-Kit Learn against a deep learning model I developed, regardless of whether the deep learning method worked well or not. I could have also used other libraries such as PyTorch, however given my complete inexperience with libraries such as that I do feel like it would have been detrimental to try this project on a completely new and unused library.

I personally am not happy with how the project has gone, I believe I could have done a much better job and the system in it's current form is overly simplistic when it comes to the actual training of the model. I think the way I cleaned and preprocessed the data and handled the data was done well, whilst there is always room for improvement I would say I am happy with that aspect of the project.

One critique I do have that was out of my control was the data that was provided in the 2018 challenge, however I suspect some of this was intentional. I personally do not feel like enough data was provided, with only 202 patients and many of these patients not meeting more than a few criteria. With 13 criteria that was an average of 15 patient per criteria which even with multiple records per patient would not have been a good amount to try and train a model on. In some cases there was only one patient who matched a certain criteria. The best example of this was the Keto criteria, with no patients the training fulfilling this criteria and so it could not accurately be trained, or it could be accurately trained by was most likely over-fitted.

## 5.2 Improvements

Ultimately I believe the ground for this project laid out and the objective has been met, however I am aware that there is plenty of room for improvement such as with the further implementation section mentioned above. Some improvements were not attempted due to lack of time and others were not attempted due to lack of knowledge in how to accomplish it, such as my mention of using the PyTorch library above.

There is not a lot of room to expand the program in its current form, however with a small amount of tweaking I think it could be viable to use this model for other medical data regardless of the tags. It should be noted that it would all have to be supervised learning, the model is not designed to handle untagged data. If a deep learning model was used then the model could potentially handle unsupervised learning and therefore would be able to take any medical data provided and still give a reasonable output. The issue however would be getting hold of medical data to test this on as medical data is highly confidential and is not handed out lightly. Even in this challenge where the patients handed out their data willingly it was still edited to remove all mentions that could tie the records to the real life person without affecting the validity of the data.

Another improvement could have been the use of some more visual output such as with matplotlib. Whilst this is my no means a necessity it would provide various way to look at and interpret the data given by the system.

A small improvement that could be made is on something I took for granted. Medical records are written and people and so are subject to spelling mistakes and errors. As the data was provided I did not perform any kind of spell check on said data and assumed that for the most part everything would be spelled correctly. Considering the amount of preprocessing and feature reduction that was done in this specific project I don't believe having a few words misspelt here and there would make a huge difference, but ultimately the end goal in a project such as this is to provide a doctor with a faster method of identify whether or not patients meet selection criteria for clinical trials[1] and so getting the absolute most accuracy is imperative when it could be affecting someones life or well being.

A very time consuming improvement that could be implemented in the future is creating a more medically annotated corpus. In the current system the corpus is simply a list of every word that is contained in the records and this is slowly filled down to get a more accurate model. Creating a corpus for exclusively medical data that was updated as more records were provided could prove to be a viable strategy for anyone who seeks to use a system like this in the real world over a long term basis.

# A References


[1] Stubbs, A., Filannino, M., Soysal, E., Henry, S. and Uzuner, Ö., 2019. Cohort selection for clinical trials: n2c2 2018 shared task track 1. [online] https://academic.oup.com/. Available at: ¡https://academic.oup.com/jamia/article/26/11/116 [Accessed 24 September 2021].

[2] Machine Learning. SAS 2020. [Online]. Available: https://www.sas.com/engb/insights/analytics/machine-learning.html [Accessed 24 September 2021].

[3] Brownlee, J., 2021. Supervised and Unsupervised Machine Learning Algorithms. [online] Machine Learning Mastery. Available at: ¡https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/¿ [Accessed 24 September 2021].

[4] Halliday, M. and Webster, J., 2007. On language and linguistics. Beijing: Peking University Press.

[5] Medium. 2021. Using Data to Find the Angriest Death Grips Song. [online] Available at: ¡https://medium.com/@evanopp/ang death-grips-data-anger-502168c1c2f0¿ [Accessed 24 September 2021].

[6] Rcharlie.com. 2021. FitteR HappieR. [online] Available at: ¡https://www.rcharlie.com/blog/fitter-happier/¿ [Accessed 24 September 2021].

[7] RUDER, S., 2021. The 4 Biggest Open Problems in NLP. [online] Sebastian Ruder. Available at: ¡https://ruder.io/4-biggest-open-problems-in-nlp/¿ [Accessed 24 September 2021].

[8] MIT Technology Review. 2021. This is how AI bias really happens—and why it's so hard to fix. [online] Available at: ¡https://www.technologyreview.com/2019/02/04/137602/this-is-how-ai-bias-really-happensand-why-its-so-hard-to-fix/¿ [Accessed 24 September 2021].

[9] Dastin, J., 2021. Amazon scraps secret AI recruiting tool that showed bias against women. [online] U.S. Available at: ¡https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G¿ [Accessed 24 September 2021].

[10] ProPublica. 2021. Machine Bias. [online] Available at: ¡https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing¿ [Accessed 24 September 2021].

[11] Abigailsee.com. 2021. Deep Learning, Structure and Innate Priors — Abigail See. [online] Available at: ¡https://www.abigailsee.com/2018/02/21/deep-learning-structure-and-innate-priors.html¿ [Accessed 24 September 2021].

[12] Deepmind. 2021. AlphaGo: The story so far. [online] Available at: ¡https://deepmind.com/research/case-studies/alphago-the-story-so-far¿ [Accessed 24 September 2021].

[13] Myers-Stanford, A., 2021. How artificial intelligence can teach itself slang - Futurity. [online] Futurity. Available at: ¡https://www.futurity.org/deep-learning-language-1452822-2/¿ [Accessed 24 September 2021].

[14] Wilson, S., Magdy, W., McGillivray, B., Garimella, K. and Tyson, G., 2021. Urban Dictionary Embeddings for Slang NLP Applications. [online] ACL Anthology. Available at: ¡https://aclanthology.org/2020.lrec-1.586/¿ [Accessed 24 September 2021].

[15] Viani, N., Kam, J., Yin, L., Bittar, A., Dutta, R., Patel, R., Stewart, R. and Velupillai, S., 2020. Temporal information extraction from mental health records to identify duration of untreated psychosis. Journal of Biomedical Semantics, 11(1).

[16] Medium. 2021. Bias in Natural Language Processing (NLP): A Dangerous But Fixable Problem. [online] Available at: ¡https://towardsdatascience.com/bias-in-natural-language-processing-nlp-a-dangerous-but-fixable-problem-7d01a12cf0f7¿ [Accessed 24 September 2021].

[17] Sisense. 2021. Tips for Overcoming Natural Language Processing Challenges — Sisense. [online] Available at: ¡https://www.sisense.com/blog/overcoming-common-challenges-in-natural-language-processing/¿ [Accessed 24 September 2021].

[18] Sanders, R., 2021. Will computers ever truly understand what we're saying?. [online] Berkeley News. Available at: ¡https://news.berkeley.edu/2016/01/11/will-computers-ever-truly-understand-what-were-saying/¿ [Accessed 24 September 2021].

[19] Statisticstimes.com. 2021. Top Computer Languages 2021 - StatisticsTimes.com. [online] Available at: ¡https://statisticstimes.com/tech/top-computer-languages.php¿ [Accessed 24 September 2021].

[20] Brownlee, J., 2021. Data Leakage in Machine Learning. [online] Machine Learning Mastery. Available at: ¡https://machinelearningmastery.com/data-leakage-machine-learning/¿ [Accessed 24 September 2021].

[21] Analytics Vidhya. 2021. Text Preprocessing NLP — Text Preprocessing in NLP with Python codes. [online] Available at: ¡https://www.analyticsvidhya.com/blog/2021/06/text-preprocessing-in-nlp-with-python-codes/¿ [Accessed 24 September 2021].

[22] Kavita Ganesan, Ph.D. 2021. What are Stop Words? - Kavita Ganesan, Ph.D. [online] Available at: ¡https://kavita-ganesan.com/what-are-stop-words/¿ [Accessed 24 September 2021].

[23] Nltk.org. 2021. 2. Accessing Text Corpora and Lexical Resources. [online] Available at: ¡https://www.nltk.org/book/ch02.ht [Accessed 24 September 2021].

[24] Navlani, A., 2018. Naive Bayes Classification using Scikit-learn. [online] https://www.datacamp.com/. Available at: ¡https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn¿ [Accessed 24 September 2021].

[25] Analytics Vidhya. 2021. Learn Naive Bayes Algorithm — Naive Bayes Classifier Examples. [online] Available at: ¡https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/¿ [Accessed 24 September 2021].

[26] Medium. 2021. Support Vector Machine — Introduction to Machine Learning Algorithms. [online] Available at: ¡https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47¿ [Accessed 24 September 2021].

[27] Scikit-learn.org. 2021. 1.4. Support Vector Machines — scikit-learn 0.24.2 documentation. [online] Available at: ¡https://scikit-learn.org/stable/modules/svm.html¿ [Accessed 24 September 2021].

[28] Brownlee, J., 2021. A Gentle Introduction to Logistic Regression With Maximum Likelihood Estimation. [online] Machine Learning Mastery. Available at: ¡https://machinelearningmastery.com/logistic-regression-with-maximum-likelihood-estimation/¿ [Accessed 24 September 2021].

# B Third Part Code and Libraries

NLTK - Natural Language Toolkit

SciKit-Learn - A set of python modules for machine learning and data mining

NumPy - NumPy is the fundamental package for array computing with Python

Pandas - Powerful data structures for data analysis, time series and statistics

Beautiful Soup 4 - Python package for parsing HTML and XML documents

All libraries used without modification