

EXTRACTING DATA FROM INSURANCE DOCUMENTS WITH NATURAL LANGUAGE
PROCESSING AND MACHINE LEARNING

JAMES MACKENZIE

This dissertation was submitted in part fulfilment of requirements for the degree of MSc
Software Development

DEPT. OF COMPUTER AND INFORMATION SCIENCES
UNIVERSITY OF STRATHCLYDE

AUGUST 2019

DECLARATION

This dissertation is submitted in part fulfilment of the requirements for the degree of MSc of the University of Strathclyde.

I declare that this dissertation embodies the results of my own work and that it has been composed by myself. Following normal academic conventions, I have made due acknowledgement to the work of others.

I declare that I have sought, and received, ethics approval via the Departmental Ethics Committee as appropriate to my research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to provide copies of the dissertation, at cost, to those who may in the future request a copy of the dissertation for private study or research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to place a copy of the dissertation in a publicly available archive.
(please tick) Yes [✓] No []

I declare that the word count for this dissertation (excluding title page, declaration, abstract, acknowledgements, table of contents, list of illustrations, references and appendices) is 18,602.

I confirm that I wish this to be assessed as a Type 1 2 3 4 5 Dissertation

Signature: 

Date: 19th August 2019

ABSTRACT

The purpose of this study is to develop a piece of software that will automatically extract information from insurance documents to save time and reduce errors. The insurance industry is ripe for disruption due to legacy systems and heterogeneous data sources and is still operating in a similar manner to almost 100 years ago. There is a need for major innovation to protect companies' market position from outside forces, and this can be achieved by embracing modern technology.

In the recent past the issue of extracting certain information from PDFs and other semi or unstructured documents has been dealt with by humans in most cases, due to the complexity of the task and the need for an understanding of the content. There have been attempts in the past to automate this process, which involved developing rule-based systems to extract the necessary information. This way of working does do the job, to an extent, but what about when the document is updated and something new is added, or a new structure is adopted? Well this is where machine learning comes in. Machine learning gives us the ability to feed a computer annotated examples of the types of data we are trying to extract, and the computer will then try to make predictions when shown data which is not annotated. During the course of this project I have explored multiple avenues while trying to solve the problem and, in the end, I came back to the idea of training a custom model for this specific task. Due to the extraordinary amount of training data that would need to be created and annotated, and the limited timescale, I decided to solve the problem on a smaller scale by focusing on one type of document from a specific insurer.

During this dissertation I explored the current market for extracting information from PDF documents and came to the conclusion that the best suited method for the purpose of this project was through the use of machine learning and natural language processing. Through working closely with the insurance company, I was able to identify the type of information that was required to be extracted and build a piece of software around this. The software was built using the Python programming language and the natural language processing and machine learning was handled by the spaCy NLP library. The first iteration of the software led to 82% accuracy when extracting information from a test document. This first build of the data extraction software has served as a successful proof of concept for a larger piece of work. I will now be working with the insurance company for the foreseeable future to further develop the data extraction software and build a web application component to integrate with their online broker system.

Acknowledgements

I would first like to thank my dissertation advisor Billy Wallace, who is a Knowledge Exchange Fellow at the University of Strathclyde. His continuing support and guidance throughout have helped me complete my MSc dissertation.

Finally, I would also like to thank CCRS Insurance brokers for allowing me to work so closely with them on the project and providing me with the resources I required.

Table of Contents

Chapter 1.....	1
Introduction.....	1
Chapter 2.....	3
Background and The Current Market	3
2.1 The Current Market for Data Extraction tools.....	3
2.2 Natural Language Processing, Information Extraction and spaCy.....	7
Chapter 3.....	10
Methodology	10
3.1 The Iterative Model.....	11
3.2 Version Control.....	13
Chapter 4.....	14
Analysis	14
4.1 Requirements	15
4.2 Design	16
4.3 Construction	23
4.4 Testing	31
4.5 Deploy	32
4.6 Evaluation.....	33
Chapter 5.....	46
Conclusion and Future Work	46
5.1 Conclusion	46
5.2 Future Work	47
Appendix A.....	48
Appendix B	48
References.....	49
Figure 1: Where Chisel fit in AI (Source: Chisel.ai)	5
Figure 2: The AI workflow (Source: Chisel.ai)	6
Figure 3: Types of named entities (Source: Chisel.ai)	6
Figure 4: SpaCy NLP pipeline (Source: spacy.io)	9
Figure 5: Visualisation of dependencies (Source: spacy.io)	9
Figure 6: Visualisation of named entities (Source: spacy.io)	10
Figure 7: Sample use case diagram from an ATM system	12
Figure 8: How git works	14
Figure 9: Visualisation of the dependency tree (Source: spacy.io)	21
Figure 10: Class diagrams.....	48

Chapter 1

Introduction

Given the large amount of manual data entry tasks still associated with the insurance industry, it is nearing a stage where the development and/or adaptation of new technology could be the key to getting a step ahead of your competitors. As part of my dissertation project I was tasked with researching and developing a piece of software to automate the extraction of key information from various insurance PDF documents. The last few months of work on this project has been a successful proof of concept which has led to further work in the future. The future work will involve refining the software that is described in this dissertation and building a web application that will be integrated with the company's existing digital broker software. As part of this dissertation I will also go into some detail on the future plans for building the web application component using the Django framework. The company I am working with are clear on their vision to adopt the latest technology and push for further success in their sector.

At present, copies of client's existing policy schedules are received and read in order to manually extract some of the following information:

- Sums insured
- Limits of Indemnity/policy limits
- Estimates such as wage roll, turnover, fee income, exports/imports
- Vehicle schedules
- Policy extensions and inner limits

This information is then pulled through in a report format and they issue this to an underwriter who uses the information to provide a quote. The underwriter provides their terms in a quotation document. They then manually compare this quotation document with the information that was keyed in to ensure the underwriter's terms are on the correct basis and meets with the client's requirements.

This process not only results in admin errors which can have significant consequences for clients (underinsurance) and the company (errors & omissions claims) but is also time consuming. This process is repeated for every policy a client has for which they provide a quote and for each quote they receive. Depending on the nature of the marketing exercise, they may approach over 10 different underwriters for a single policy.

The research question I will focus on answering throughout this dissertation is as follows:

RQ1. What type of tools can I build to extract the required information with a high level of accuracy and minimal maintenance?

Early on in this project it was determined that the most effective way to tackle the problem was to first extract the text from the PDF documents and process this text through the use of natural language processing and machine learning. Originally my approach was to first

identify the monetary values that needed to be extracted, and then to determine how much context would have to be extracted along with these values. Extracting text in a structured manner from a PDF document that was not created by a machine proved to be a difficult task and in the end it was determined that the most effective way to extract the text was to first convert the PDFs into images and then apply optical character recognition (OCR) on the images. This provided me with the text that I was now able to start working on using SpaCy, an open source natural language processing and machine learning library written in Python.

Natural language processing (NLP) is a subfield of computer science and artificial intelligence which has been around since the 1950s. In computer science NLP is considered a difficult problem due to the unstructured and sometimes unpredictable nature of human language. The manner in which people speak to each other can vary significantly, for example, it might be difficult for a computer to pick up sarcasm being used in a sentence. During the literature review section of this dissertation I will discuss some of the main aspects involved in NLP, namely Point of Speech (PoS) Tagging, Named Entity Recognition and Tokenisation

I will be discussing Information Extraction, a subtask of natural language processing, which is the ability to extract meaningful information from texts in order to create some sort of knowledge base, specifically for this task, which is the extraction of key insurance data in order to populate a database which is then used by the company for various other activities. During the course of this project I looked into many different ways of extracting data from PDF documents, but in the end, I wanted to train a model using machine learning, in order to create a solution which would require little supervision and maintenance, and has the potential to improve over time as the available dataset becomes larger and we are able to label a significant amount of training data. Machine learning allows us to train a statistical model to perform tasks without explicit instructions at each stage of the task, this is achieved by making use of training data in order to train the model. The model is fed examples of sentences which are of the same context as the insurance documents, the only difference at this stage is that the sentences are labelled to provide the model with the correct answers to learn from. A certain amount of this training data is held back and not used during training; this is then used for testing the trained model for accuracy. I have focused on training the Named Entity Recogniser (NER) in SpaCy after achieving good results with a few thousand annotated training examples.

During the development of the data extraction software I explored multiple methods of extraction including looking at the relationships between words in a sentence and being able to extract data by iterating over these relationships until I have gathered enough context from the sentence to extract along with the monetary amounts. I also looked at using a rule based matching approach for data extraction, where I would write rules to find certain patterns in text and then extract these patterns from each document. I came across some interesting issues when training a machine learning model, like the catastrophic forgetting problem and the need to provide context in training data and not just the values you want the model to learn. During the course of this dissertation I will be discussing the issues above in more detail.

I will also be discussing the state of the current market for data extraction tools of a similar nature, one of which is an insurance industry specific data extraction tool, produced by a Canadian company named Chisel.ai, which uses NLP and machine learning.

Chapter 2

Background and The Current Market

Natural language processing is a subject that has been around since the 1950s and is being used in conjunction machine learning, and together they provide the basis for a plethora of possibilities in the field of computing science. Since this dissertation is based on the development of a piece of software which applies NLP and machine learning, in the chapter, I am going to first focus on the current market for tools that can extract data from PDF documents, ranging from tools that are “smart” (meaning they apply techniques which require less supervision) to tools which are based more on rules and extracting data from specific locations on a page, and therefore require more input, maintenance and supervision from the end user. I will then focus on the concepts of NLP and information extraction.

2.1 The Current Market for Data Extraction tools

At the moment there are a few options available if you are looking to extract information from PDF documents. I have chosen some examples of software which I think could achieve the result we require from the data extraction software, I was only able to find one example which is very closely matched to the route I had in mind with this project, the other examples can extract the required information, but would require a lot of maintenance and would be fairly restricted to only working on certain document types.

2.1.1 Docparser

Docparser is a web-based software that allows you to convert PDF documents into easy to handle structured data by letting you create data extraction templates based on certain document types. In their own words “Build a customized document capture and data extraction solution within minutes”^[1]. When you are creating a data extraction template you are first required to provide multiple examples of the same document type and from there you create parsing rules for each piece of data you would like to extract, for example, extracting text from a certain position on a page, or parsing check box inputs from PDF forms. You then have to go through one of the documents and place a box around each piece of data you want to extract. Each time you highlight a piece of data, you can then add a parsing rule, which will let you identify it, using the parsing rule you have created, in a general manner in the future. There are some pre-set types of parsing rules which will automatically detect the type of data you have highlighted and give it a predefined parsing rule and in some cases format it correctly, for example, in the case of a date. After processing a document and adding in all the required parsing rules, you will then have a

complete template for extracting the required information from that specific document type.

Docparser would be sufficient in performing the required data extraction from the PDF documents given the fact it uses the specific position on a page to extract the required data, so I could extract and categorise each entity and that would be enough information for me to fill a database with accurate information. Docparser also supports Optical Character Recognition (OCR) which is required for this project since the insurance PDF documents are not machine made. Some of the concerns I would have with recommending a product like docparser would be as follows:

- Each document type would require an employee to use docparser to identify all the entities on a document and categorise them correctly, which has a risk of human error associated with it and could be time consuming depending on the size of the document and the amount of information that is required to be extracted.
- Each time a client sends through a PDF document, it would have to be checked thoroughly to make sure it does not have any layout changes at all, because if any piece of data is in a slightly different position on the page, this would potentially cause a risk of docparser extracting incorrect information.

2.1.2 Adeptia

Adeptia is a company based in Chicago that provide various solutions for extracting data from multiple sources. One of their solutions is a tool for extracting data from unstructured files such as a PDF. “Adeptia provides an easy way for customers to extract unstructured data from PDF files. We not only extract the unstructured data and make it usable, but also make it easy enough to be used by non-IT users. Rules governing data extraction from a PDF can be defined through a simple graphical user interface that can be easily used by business users.” [2]. Part of the solution they offer is a user interface that allows you to upload a PDF file and then go through the file and select key and value pairs, while the program uses this information to create an XML file containing these user defined key and value pairs. For each key and value pair you create, you can then right click on the value in the PDF and create a definition that will be part of the XML structure, and will let you group multiple key and pair values into one overall key, for example, you might have 5 key and pair values that make up the “Property Damage” section in an insurance schedule. Adeptia also has a tool for dealing with table data in a pdf. They allow you to select the table headers and then define the first and last columns in a table, so that it knows what to look for when parsing this type of file. Finally, they allow you to select fixed sections of the PDF such as the company name, which you can select and add a tag which will identify the selected string as whatever you choose to name it. After creating the final XML there is a tool to map each key to the relevant data field in a database, which will mean that you can be sure that your data is going to end up in the correct place in the end.

I feel that Adeptia’s data extraction tool is a reasonable solution to the problem, however I think that there could be issues with using this technique. My concerns would be as follow:

- This software requires a user to go through each type of PDF, select key and value pairs, then select table headers and finally fixed strings. This would mean a lot of manual work setting up each template and the constant need to make sure that neither the wording nor structure has been changed within the templates.
- The initial output, after the user had defined all the fields, is in XML format which your average user has probably not seen before unless they are a technical person. This would require employing a technical person to do this job, or train staff to understand the output
- The level of oversight and maintenance required might not be worth the time saved by implementing this software.

2.1.2 Chisel.ai Data Extraction

The solution developed by the Canadian company Chisel.ai is the closest match I could find to what I had in mind for this project. Chisel.ai have developed a purpose build Natural Language Processing and Artificial Intelligence solution specifically for the insurance industry, which is able to “extract, interpret, classify and analyze unstructured data in policies, quotes, submissions, applications, binders and endorsements 400 times faster than a human can—with significantly greater accuracy” [3]. The picture below gives an idea of where Chisel place themselves within the field of AI, which includes NLP and Machine Learning:

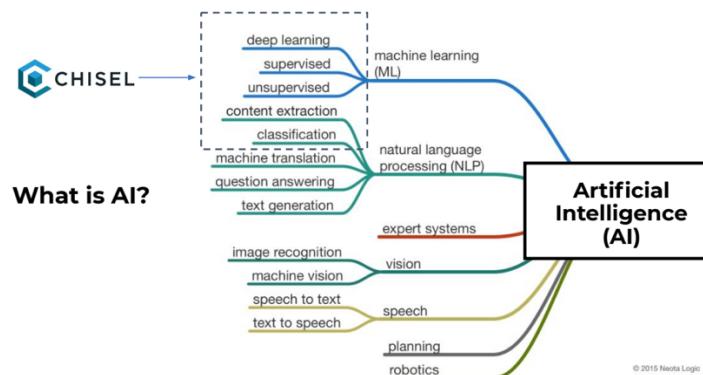


Figure 1: Where Chisel fit in AI (Source: Chisel.ai)

They have been working closely with multiple insurance companies over the past few years to develop this system and with the help of industry expertise they have been able to use both supervised and unsupervised machine learning to train their model to extract the correct information from various documents.

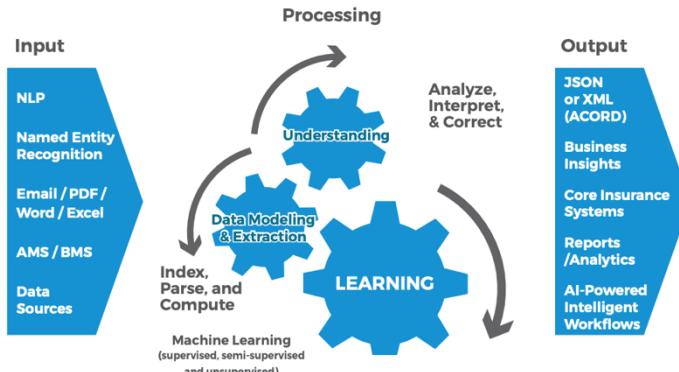


Figure 2: The AI workflow (Source: Chisel.ai)

From my research it is clear that they have come across some of the same issues that I encountered, for example they emphasise the point that the results are more accurate when you are initially extracting text from machine made PDF documents, because there is no need for optical character recognition and you are able to extract the text directly from the PDF document. They also mention the importance of having a large data set for each document type in order to create enough training data to train the model to an acceptable level of accuracy, which involves labelling thousands of examples of named entities. Their software currently recognises over 500 named entities and once fully trained, it is able to categorise these entities automatically when extracted from a document. The process of labelling data is very time consuming, and for that reason they have a team dedicated to this process.

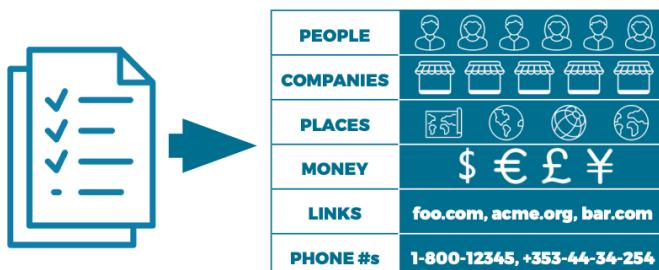


Figure 3: Types of named entities (Source: Chisel.ai)

Implementing Chisel's data extraction system takes roughly 4 months in total and this involves gathering data, training the model, performing user acceptance testing and finally QA and random document spot checks, however, the training is on-going because each time an employee catches an error and corrects it, this correction is recorded and added to the training data for the next round of training, therefore it can take up to 12 months for the client to be completely satisfied with the level of accuracy.

Chisel's software would definitely be an acceptable solution to the data extraction task I am working on and I am confident it would eventually achieve the results we are looking for, however, I would be concerned with the initial and ongoing costs and the fact that the company are based in Canada, it would be both difficult and expensive if time was required on-site during the setup, training and maintenance periods.

2.2 Natural Language Processing, Information Extraction and spaCy

NLP is a subfield of computing science and artificial intelligence that has been around since the 1950's and is the task of essentially giving a computer the ability to process and analyse large amounts of human (natural) language and “by “natural language” we mean a language that is used for everyday communication by humans; languages such as English, Hindi, or Portuguese. In contrast to artificial languages such as programming languages and mathematical notations, natural languages have evolved as they pass from generation to generation and are hard to pin down with explicit rules”^[4].

NLP is applied in many of the applications we use on a daily basis, some examples of these applications are as follows:

- Personal assistants in smartphones like Siri, Google Assistant, Alexa and Cortana.
- Google Translate for processing text in one language and providing an output in another language.
- The automated voices we sometimes have to speak with when we initially call our bank.
- Word processors like Microsoft word use NLP to check grammar and spelling while we are typing.

NLP has multiple applications, some of which include tokenisation, named entity recognition, part of speech (PoS) tagging, automatic text generation and information extraction. For extracting data from insurance documents, I have focussed on two of the above topics, named entity recognition and information extraction. By creating informative named entities and training a model to recognise them, this becomes a form of information extraction through the use of named entity recognition.

If information is in a structured format like a table for example, then extracting information from that table is extremely easy, however if data is unstructured, such as within a sentence, the task of extracting information becomes very difficult. To make the information in a sentence useful we need to find a way to convert the sentence into structured data, then we can extract information using powerful tools like SQL. Information extraction “has many applications, including business intelligence, resume harvesting, media analysis, sentiment detection, patent search, and email scanning.”^[5]. The end product of this project is having all the required information in a database and information extraction is key to this. The process of information extraction involves the use of multiple concepts from NLP including the following:

Sentence segmentation: The process of breaking a group of text into sentences based on rules around structure and punctuation.

Tokenisation: Given a sentence or any other sequence of characters, tokenisation is the process of breaking this sequence into smaller pieces, usually words, and sometimes removing tokens like punctuation and stop words when they are not deemed necessary for the NLP task at hand.

Part of Speech Tagging: The process of marking up a word from text, based on its definition and context, as corresponding to a particular part of speech, these tags usually state if the word is a noun, verb, adjective, adverb etc. This was a laborious process that was originally done by hand and is now performed by computers using computational linguistics and algorithms which are either rule-based or stochastic.

Named Entity Recognition: NER is the process of classifying certain entities within text into pre-defined categories, these categories could essentially be anything you would like them to be, for example, we could have an entity type for people's names and call it PEOPLE, or one for cities and call it CITY.

Relation Recognition: This process identifies the likely relations between entities in a text. This is usually in the form of a tuple, for example a sentence like "the insurance schedule was sent from the AXA offices in London" might produce a relationship like ([ORG], 'AXA', 'in' ,[LOCATION, 'London']) which indicates that the company AXA is located in London, which is extremely useful information that has been extracted from a sentence without the need of human interpretation. This enables us to extract some sort of context from the data, rather than just a list of entities without any way of linking them.

Not all named entities are one token long, so we are required to use a process called chunking to allow us to recognise a named entity which might span 2 or more tokens, for example 'Heathrow Airport' might be a named entity in the 'AIRPORTS' category and we need to be able to pick it up as one named entity rather than two, "Chunking or shallow parsing is the task of identifying and segmenting the text into syntactically correlated word groups. It is considered as an intermediate step towards full parsing." [6]. Chunks can be identified through the use of regular expressions, the process involves looking for certain patterns in the part of speech tags in a sentence, like a sequence of NNP tags, which is the tag given to a proper noun, or looking for a sequence of determiner(DT)/possessive(PP), adjectives(JJ) and nouns(NN), this could have a regular expression that may look like "DT|PP\\$\>?<JJ>*<NN".

In the initial stages of this project I performed some research on the various options for which library to use for natural language processing and in the end, I landed on the spaCy library. One of the main reason I went with spaCy was down to the detail and quality of their documentation and the examples that were provided, I found it was quick for me to learn how to use, whereas with some of the other libraries I looked into, like the Natural Language Toolkit and StanfordCoreNLP, were developed up to 18 years ago and the documentation was not as intuitive and easy to follow. The ease of learning and using spaCy saved me a lot of time that I would have otherwise wasted, which allowed me to focus more on the task in hand. One of the main attractions to spaCy was how easy they make it to train one of their statistical models, which are powered by their machine learning library called Thinc. Thinc implements models using the "Embed, code, attend predict"

architecture, it is optimised for CPU usage, NLP and deep learning with text. In their own words, “spaCy v2.0 features new neural models for **tagging**, **parsing** and **entity recognition**. The models have been designed and implemented from scratch specifically for spaCy” [7] and “The parser and NER use an imitation learning objective to deliver accuracy in-line with the latest research systems, even when evaluated from raw text. With these innovations, spaCy v2.0’s models are 10x smaller, 20% more accurate, and even cheaper to run than the previous generation.” [8]. The ability to train these models with ease meant I was able to work on creating training data without having to worry about too many of the details around training a machine learning model. The way spaCy is designed is such that you feed it some text, and this is then processed through a pipeline of components that you can choose, like a tokenizer, tagger, parser, named entity recogniser and text categoriser, you can also add your own custom pipeline components.

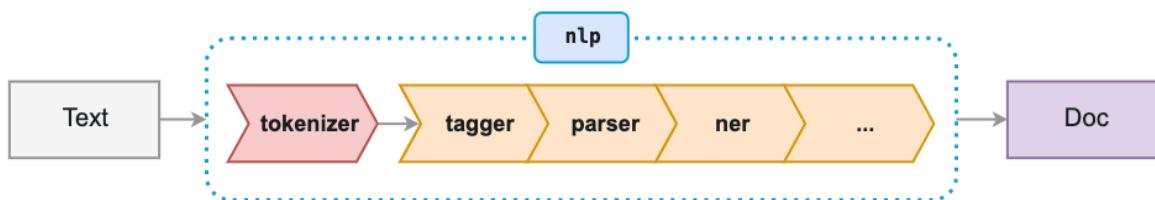


Figure 4: SpaCy NLP pipeline (Source: spacy.io)

Once your text has been processed using the various pipeline components spaCy returns a Doc object which you are then able to perform various methods on including extracting, tokens, named entities, sentences and PoS tags. The syntax used in spaCy is very straightforward, for example, to process a sentence and return a Doc object you would use the following code:

```
doc = nlp(u"This is a sample text")
```

Spacy also allows you to isolate and train each component without having any effect on the others, this was particularly useful in the case of this project, as I was able to isolate the named entity recogniser and train it on the entities that I wanted it to find. With the use of displaCy you are able to easily visualise dependencies in a sentence and named entities within a paragraph of text, below are a couple of examples:

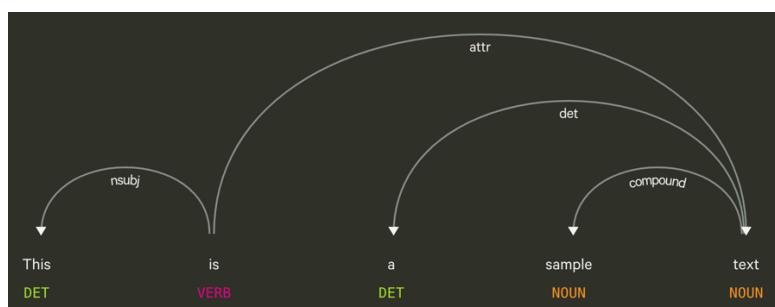


Figure 5: Visualisation of dependencies (Source: spacy.io)

When **Sebastian Thrun** PERSON started working on self-driving cars at **Google** ORG in **2007** DATE, few people outside of the company took him seriously. “I can tell you very senior CEOs of major **American** NORP car companies would shake my hand and turn away because I wasn’t worth talking to,” said **Thrun** PERSON, now the co-founder and CEO of online higher education startup Udacity, in an interview with **Recode** ORG earlier this week DATE.

Figure 6: Visualisation of named entities (Source: spacy.io)

spaCy currently provides pre-trained models for 8 different languages and you have the option of downloading small medium or large versions of these models. The small model does not include word vectors, which are useful for determining similarity between words, for example, the words “banana” and “apple” will have a similar word vector, whereas the words “tree” and “computer” will not have similar word vectors. You are also able to calculate an overall vector value for sentences, paragraphs and whole documents, which enables you to determine if you are looking at similar types of documents in a very efficient manner with a high probability of being correct.

In the end it was this ease of use, vast functionality, informative and comprehensive documentation and the ability to train machine learning models with relative ease which led me to choose spaCy over NLTK, StanfordCoreNLP and AllenNLP.

Chapter 3

Methodology

When developing software, it is important to have a clearly defined methodology to guide the development. Software can be extremely large and complex and the process of creating it therefore needs to be structured, planned and controlled. There are many methodologies used in software development, from the traditional waterfall method, to the more modern approach of agile development, each has their own advantages, however with the size of some modern projects and the need to reduce risk, traditional methods like waterfall are not the way to go in most cases. Agile software development allows for the requirements of a project to evolve through the collaboration of self-organising and cross-functional teams and the end users/stakeholders. The real advantage of agile is the ability to adapt to change, which is a powerful thing in modern software development. According to the agile manifesto:

- **Individuals and Interactions** over processes and tools
- **Working Software** over comprehensive documentation
- **Customer Collaboration** over contract negotiation
- **Responding to Change** over following a plan

This does not mean the items on the right are not important, it is just that the items on the left are valued more. There are some common misconceptions with agile working, such as a lack of planning, processes and documentation. Agile is not a methodology in itself, but rather more of an umbrella term for several ways of working that share common values and priorities, some of which are as follows:

- Sharing of knowledge and control
- Finding value in delivering an end product
- Accepting and anticipating change
- Working concurrently rather than separately
- Minimising waste and overheads
- The ability to focus on quality from an early stage and throughout

During the development of the data extraction software I have adopted the iterative model due to the fact that the requirements were constantly evolving as I learned more about the development process. The project will take at least a year to complete and during this time I will be working with the company on the project and the iterative model has allowed us to develop a small version of the software which acts as a proof of concept for the project. During this chapter I will describe the iterative model and how it has applied to this project and I will layout my reasons for choosing it, along with some of the pros and cons of using this methodology.

3.1 The Iterative Model

Unlike a model like the classic Waterfall Model, which involves setting out a detailed list of requirements and the steps to complete them at the start of a project, the iterative model is an agile development model which starts with developing a simplified version of the software which increases in complexity with each iteration. Iterative methods have been used to great effect in the past in not only software development, but in the development of aircraft and products in other industries. After an initial stage of planning there are a set of stages that are repeated throughout the process improving the software each time. This process allows for additional functionality to be added each time, which is extremely useful in a project like this one because I am constantly learning throughout the process and realising the need for additional pieces of functionality. The usual stages involved in the development of a piece of software are as follows:

Requirements: This is a fundamental part of any software development project; this stage usually involves meeting with stakeholders to discuss the required functionality of the software. It is at this stage where the project is defined, and it is important to make sure everyone is in agreement and has the same end goal in mind. The process of gathering requirements can take place over multiple meetings and the outcome of this is usually a specification document that can be used for reference during the project. There following are some useful tools for helping gather requirement effectively:

- User stories – These describe how each type of user would interact with the application e.g. “As a user I sign in to application and I can upload a document.”. If a user story is too long or complicated it can usually be broken into multiple stories.

- Use case diagrams – This is where you visually map out the interaction between the different users, other applications and the system you are developing. Below is a simple example of a use case diagram:

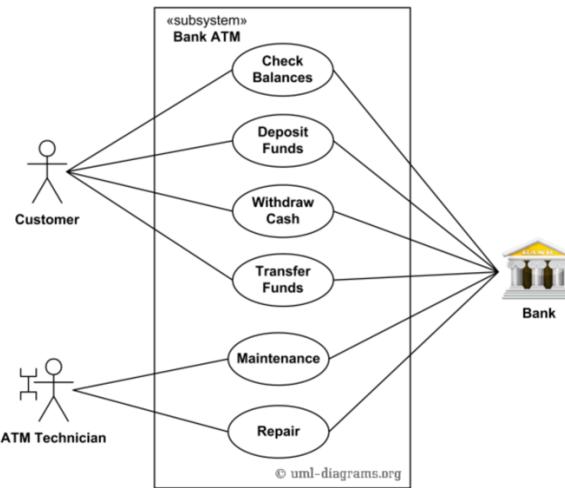


Figure 7: Sample use case diagram from an ATM system

Both of these tools for gathering requirements involve putting yourself in the position of the user and looking at things from their perspective, which is an extremely useful skill to have as a software developer.

Design: Involves the identification of various required components like database schemas, business logic and the technical requirements for using these components, like programming languages and frameworks. It is at this stage you need to break the problem down into smaller parts and figure out exactly how each part will be built and what will be required to complete the task, rather than just starting to code without thinking about it, because this could lead to issues further down the line which are not easy to foresee at first glance and could cost a lot of time trying to resolve. Some of the issues that need to be resolved are issues regarding security, data persistence, error handling, system architecture, design patterns and user interface design.

Build/Implementation: After gathering requirements and completing the detailed design, the initial iteration of the application can now be coded at this stage

Test: After the current iteration is implemented the application can now be tested so you are able to identify any bugs, this can involve various methods for testing like unit testing and user acceptance testing.

Deploy: The application is officially deployed to the end user in the applicable manner, depending on whether it is being deployed internally within a company or on some other platform like the Google Play Store for example.

Evaluation: At this stage you are not testing the correctness of the code anymore, but rather things like the opinion of end users are collected using methods such as usability studies etc.

It is at this stage in the iterative model where we loop back to the requirements stage at the beginning and, with all the information and feedback from the previous iteration, we start on the next iteration and build in any additional required functionality and improvements we feel are necessary. Given the flexibility of the iterative model and the fact that feedback is gathered after each iteration and incorporated into the requirements for the next iteration, it made this method the best fit for this project because I will be working closely with the stakeholders at each stage and it enables me to involve them in the development process as much as possible. As with adopting any method, there are advantages and disadvantages to the iterative model:

3.1.1 Advantages of the Iterative Model

Due to the nature of the iterative model it is easily adaptable to changes in the needs of the client, which are a pretty common occurrence. During the development of the data extraction software this is a clear advantage because the new nature of the work meant it was difficult to know the exact requirements at the start of the project and the iterative method allows for the evolving shape of the software as I learned more about the possible functionality.

Since the iterations can be quite short it makes it easier to find flaws in the design of the software early on in the project and then adjust the design as required.

It allows for there to be less time spent on documenting the process and more time spent on designing and coding.

After each iteration you are able to get feedback from users which can then be taken into account quickly in the next iteration. This removes some of the risk of developing a large piece of software only to find out at the end that it doesn't suit the needs of the users.

3.1.2 Disadvantages of the Iterative Model

Each iteration must be complete before the next one can start, so there is no overlap which makes it quite a rigid method.

It can be difficult to estimate an exact end date for the project since the software is evolving with each iteration and the feedback from the users could result in there being more iterations than originally planned.

There are possibilities of having expensive system architecture problems due to there being a lack of a full requirements specification for the system, from the initial stages of the development process.

3.2 Version Control

During the development of the data extraction software it was important to be able to keep track of any changes in the source code, especially when multiple people are working on the same piece of software. To make this process easier I used Git, which is a distributed

version control system for tracking changes in source code. This fits particularly well into an agile development environment, where there are constant changes to the code and multiple iteration of the same project. Below is a simple illustration of how git works:

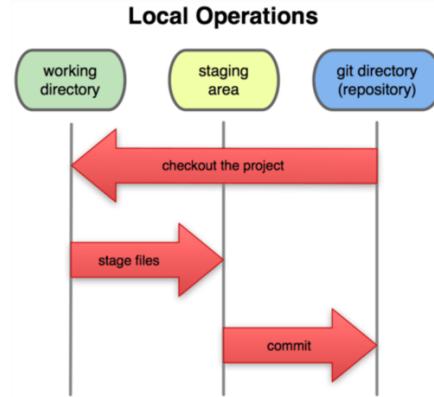


Figure 8: How git works

Git has three main states for your files: committed, modified and staged. Committed means that the changes you have made to the files have been stored, modified means you have made changes to some files and they have not yet been committed, and staged means the files you have changed have been marked as modified and will be saved on your next commit. When using git you have a working directory and a git directory, which is a directory containing a single version of your files that you are currently working on, and the git directory contains all the required metadata and a store of all the objects required for your document. When you clone a git repository in order to start working on the project in a team, for example, you are cloning the latest version of the git repository to your local storage, the fact that most of the data required for git resides on your local storage, means that you can access any piece of a project at speed and not have to deal with the issues associated with working over a network.

As the data extraction software becomes larger and I have developed the web application component, git is going to play an important role in handling version control for the project, as more people are involved in the development, git will allow us to work almost seamlessly together on the same project. Git is particularly useful in situations where you have been working on the same code as another member of your team and have both tried to merge when changes have been made to the same part of the code, within two separate branches. In this type of situation git will mark the files as conflicting, it is then the responsibility of the developers to do some research and figure out why there is a conflict and what the correct file should look like.

Chapter 4

Analysis

To date, this application is a proof of concept which was used to demonstrate that it is feasible to build an application for the purposes of extracting information from insurance

documents. Currently, there are very few products out there that meet the requirements of this project. Some of the types of technology that currently exist include the following:

- Docparser
- Chisel.ai
- Adeptia

Most of the current technology is targeted at companies that are looking for solutions in order to fix short term problems, however, these solutions require a lot of ongoing maintenance. In the following sections I will cover the specific steps that were taken to develop the data extraction software to the point it is at today. Currently I have completed one iteration of the process and I am in the process of planning the second iteration.

4.1 Requirements

The most important aspect of gathering requirements is communication with the various stakeholders, as this is the part of the process where you get a better understanding of how the product is going to work and what it is going to do. It is at this stage where we need to be able to connect the technical language of a software engineer, with the most likely non-technical language of a stakeholder, this is important to enable there to be the same understanding from both parties involved, because sometimes people interpret things differently to others, and it's pivotal at this stage that there is a general consensus.

I currently have an initial set of requirements which were gathered from our discovery meeting, however, the requirements gathering process was an on-going aspect of this project since I chose to adopt the iterative, instead of using waterfall, where I would gather requirements in the beginning and they would be consistent throughout the project. I feel using agile will enabled me to build a better, more relevant final product and therefore optimise the time spent on the project. The source of the requirements for this project will come from the client and my research into the subject.

During the initial meeting the reasons for doing the project were discussed and it became apparent that there are products out there, but they don't fulfil all needs or were unfeasible. It is this lack of a suitable product which has driven this project and will hopefully produce a solution that will fulfil all the necessary requirements.

So far, from my meetings with the stakeholders and the research I have performed, the user stories at this point are as follows:

User:

- As a user I should be able to upload a document.
- As a user I should be able to view the entities that have been extracted.
- As a user I should be able to view the output in Excel form.

The end goal of this project is to produce an application that alleviates much of the need for manual keying in of information into the company's online broker system. As things stand the PDFs are read over and each piece of required data is entered into a specific section of the online broker system. The online broker system is a piece of software which has recently been developed in-house, it is a web-based application which makes use of the Django framework. As part of a future iteration of the data extraction project I will be writing a Django based web application and incorporating the data extraction software into it, this will enable us to streamline the various pieces of the current in-house software with the new software.

The above user stories will embody the requirements and will evolve throughout the project as I work through each iteration of the project. It was at this point that I felt that I had a good set of initial requirements which now enable me to start the design process, and the process of learning the technology involved to write the software.

4.2 Design

I am going to start with describing the architecture (high-level design) of the application as it is a key stage in the design process. There are various types of architecture I could have chosen to adopt in this case, but I chose to go with Model View Controller (MVC) since its good when designing a GUI because it gives you a structure to work with that keeps your model code separate from the code that is used to control/animate your GUI.

Taking a closer look at MVC, it breaks down into the following categories:

Model - this is the data access layer where the applications data is stored (some data will be stored in a database for data persistence requirements across the application) and manipulated

View - this is the GUI layer where the nice-looking application is displayed to the user and they can interact with it

Controller - this acts as the intermediate between the model and view, for example if a user clicks a button which alters a piece of data, the controller will register this button click and update the model accordingly

The Django framework really embraces the MVC architecture, however when using the Django framework, the parts of the program that handle the model, view and controller are structured/named a little differently, as described below:

Model – This handles the data side of things and is used to create the structure of the underlying database. You first have to provide the username and password for your database in the Django settings and then when you write a model class in Python, you just have to migrate this using the command “python manage.py migrate” and Django will then produce the necessary SQL to create your database tables modelled around the structure of the model class you have just created.

Template – This is similar to the view in MVC since it acts as a layer that handles the presentation logic and controls how things are displayed to the user. This is where your HTML, CSS and JavaScript will be written.

View – This is essentially the controller in MVC since it handles the necessary business logic and acts as a bridge between the model and the template. This will take the input from the user in the template (like a post or get request for example) and perform some sort of method and then use the output from this method to write/read information from the underlying database via the model.

In the case of the data extraction software, the web application will serve as a user interface where the user can upload the PDF document and from there, they will then see the list of extracted entities which have been added to the online broker database.

I will now look at the design in more detail (low-level design) by going through the classes that have been developed for the purpose of data extraction and give some detail on the reasoning for them and some of the issues faced during development.

During the design of the data extraction software I identified four key classes that serve an overall purpose of purely extracting the required data from the PDF insurance documents. These classes do not include any of the classes that will be required for the web application component which will be developed in future iterations of this software. These key classes are as follows (The associated class diagrams are in Appendix B):

ConvertPDFText – This class holds the methods which are required to convert a PDF into an image file, which is then read with OCR and converted into a text file.

CreateDoc – This class is responsible for taking the text output and returning a doc object which is required for use with spaCy. This class is dependent on ConvertPDFText.

SpacyFunctions – This is a large class that depends on CreateDoc and contains methods which perform the following spaCy functions:

- Return a list of tokens
- Return tokens along with their lemmatisation
- A method to open a DisplaCy visualisation of names entities or dependencies
- Return a list of all the sentences contained in a body of text
- A method that returns a pandas dataframe along with an extra column containing the expected monetary amount that should be extracted
- Methods to extract currency relations
- A method that uses regular expressions to add pound values to the list of named entities

TrainingSpacyModel – This class depends on having a substantial amount of training data within TrainingData.py and contains all the methods necessary for training the named entity recogniser.

TrainingData.py - This is a large file containing all the new entity labels, a list of the expected output from the tested insurance document and large lists of training data that are required for training the spaCy model.

4.2.1 Issues Faced During Development

1. Extracting text from PDFs

When using natural language processing to process information from a document, the first step is getting the information in that document into a format that can be used with the chosen NLP library, which is text format. My first task was to find a way to pull the text data out of the PDFs and keep it in a structure that still makes sense. The order of words and money amounts is pivotal, because if these values are in the wrong place after extraction, this will lead to wrong numbers being extracted and the whole process would be a waste of time. Initially I tried using tools like PDFMiner, which is a tool written in Python for the purpose of extracting text data from PDF documents. After reading through the documentation for PDFMiner, it sounded as though it was well suited to the task, however after testing it on a few of the insurance documents, the text output was not useable for the following reasons:

- Losing its original structure
- Monetary values were being placed in amongst totally unrelated wording

Other tools I tested ended up with similar results to PDFMiner and after some reading into the matter I realised that this issue is common when the PDF documents are created and edited by a human, rather than automatically generated by a machine. Since most machine generated PDFs follow a set structure and contain the correct metadata, they are easier to extract text from in a structured manner, however the PDFs I am using in this project have been edited manually to make them aesthetically pleasing, however, in the process of doing this, they don't have a complete underlying structure. It was at this point I decided to look into using Optical Character Recognition (OCR) in order to extract the text data in a useable manner. The OCR engine I chose to use is Tesseract OCR which is an open source engine developed originally by Hewlett Packard, later sponsored by Google and considered one of the most accurate OCR engines. In order to use Tesseract, I first had to convert the PDF files into images, and for this task I used an ImageMagick binding for Python called Wand. Wand allowed me to select the resolution and file type of the image after conversion. After converting the PDF to images I was then able to extract the text data from the images using Tesseract. Tesseract allows a vast choice of different page segmentation modes (PSM), as listed below:

Page segmentation modes:

- 0 Orientation and script detection (OSD) only.
- 1 Automatic page segmentation with OSD.
- 2 Automatic page segmentation, but no OSD, or OCR.
- 3 Fully automatic page segmentation, but no OSD. (Default)
- 4 Assume a single column of text of variable sizes.
- 5 Assume a single uniform block of vertically aligned text.

- 6 Assume a single uniform block of text.
- 7 Treat the image as a single text line.
- 8 Treat the image as a single word.
- 9 Treat the image as a single word in a circle.
- 10 Treat the image as a single character.
- 11 Sparse text. Find as much text as possible in no particular order.
- 12 Sparse text with OSD.
- 13 Raw line. Treat the image as a single text line,
bypassing hacks that are Tesseract-specific.

After testing the output of each PSM, I found that PSM 6 was most suitable for the data extraction software because it was able to extract all the text and keep it in an order that I was able to work with using NLP.

4.2.2 Data Extraction Approaches Explored

During the initial stages of the design process I spent some time exploring various methods for extracting relevant information from the documents using NLP:

1. Extracting monetary amounts using spaCy's PhraseMatcher, Matcher and EntityRuler

The phraseMatcher in spaCy is a pipeline component that allows you to search for specific sequences of tokens within a spaCy doc object (which is a list of tokens that have been created from input text). Using the phrase matcher, I was able to create specific rules for each document type which allowed me to extract all the monetary amounts from an insurance document along with the applicable descriptive text. Since each token will have a part of speech tag and dependency label, it is also possible to match on these attributes, rather than just the specific text of the token. The PhraseMatcher must first have access to the vocabulary of the specific nlp object used when creating the doc object. According to the spaCy documentation “The Vocab object provides a lookup table that allows you to access Lexeme objects, as well as the StringStore. It also owns underlying C-data that is shared between Doc objects.”. Below is a sample of making use of the PhraseMatcher for matching a sequence of tokens:

```
from spacy import PhraseMatcher
matcher = PhraseMatcher(nlp.vocab)
matcher.add("Money – Stored in safe overnight", None, nlp(u"Stored in safe on
premises at night"))
doc = nlp("The text output from a converted insurance document would be placed
here")
matches = matcher(doc)
```

I would then just add patterns for each and every piece of information I want to find within a document. The phrase matcher returns a list of (match_id, start, end) where the start is the start position of the match and the end is the end position on the match in terms of where it sits within the character count of the document. Once I have the start and end points of the match, I can then extend the end point of the match to extract the monetary

value associated with the matched sequence of tokens. The second parameter of the PhraseMatcher's add function is an optional callBack function which you could write an on_match function for, which will confirm you have found a match and then print the match on the screen.

Unlike the PhraseMatcher which accepts patterns in the form of doc objects, the Matcher accepts patterns in the form of lists of token descriptions. The Matcher also accepts patterns in the form of regular expressions. This was particularly useful, as when I first started using spaCy it was unable to pick up pounds as a MONEY named entity and was only working for US dollars. The Matcher gave me the functionality I needed to search for pound amounts and extract them from a document with relative ease. The issue I had with the Matcher was, when I tried to add the pound amounts to the MONEY named entities, I would get an error because the part of the pound amount, not including the “£”, was already a CARDINAL named entity. Below is the code I wrote to try and label the pound values as MONEY named entities:

```
def on_match(self, matcher, doc, i, matches):
    # Get the current match and create tuple of entity label, start and end.
    # Append entity to the doc's entity. (Don't overwrite doc.ents!)
    match_id, start, end = matches[i]
    entity = Span(doc, start, end, label="MONEY")
    self.doc.ents += (entity,)
    print(entity.text)

def add_entity_for_pounds(self):
    pattern = [{"LOWER": "£"}, {"TEXT": {"REGEX": "\^\\ ?[0-9]+(\\,\\ ?[0-9]{3})?\\.\\ [0-9]+\\$"}]}
    matcher.add("POUNDS", self.on_match, pattern)
    matches = matcher(self.doc)
    for match_id, start, end in matches:
        string_id = nlp.vocab.strings[match_id] # Get string representation
        span = self.doc[start:end] # The matched span
        print(match_id, string_id, start, end, span.text)
```

To overcome this issue and enable me to add pound amounts to the MONEY named entities I had to use another spaCy pipeline component called the EntityRuler. The EntityRuler allows you to add spans, which are the start and end points of a sequence of tokens, to the doc.ents (the list of named entities within a doc object). Using this method, I was able to automatically add all pound amounts as “MONEY” named entities using a regex patter as in the following code:

```
ruler = EntityRuler(nlp)

def add_pounds_to_money(self):
    patterns = [{"label": "MONEY",
                 "pattern": [{"LOWER": "£"}, {"REGEX": "\^\\ ?[0-9]+(\\,\\ ?[0-9]{3})?\\.\\ [0-9]+\\$"}]}
    ruler.add_patterns(patterns)
    nlp.add_pipe(ruler)
    nlp.to_disk("test_model")

    self.identify_entities()
```

Using the EntityRuler I would be able to identify all of the pound values and extract them, however in order to extract the context I would have to extract a span from around the monetary values and then find a way of categorising the information based on the context I have extracted.

The methods mentioned above, of extracting the necessary data would work, however it would be very rigid and would need to be constantly modified and updated to make sure any slight wording changes in the insurance documents were captured. The need to write patterns for every piece of information that needs to be extracted would a very labour-intensive task, hence this method was quickly ruled out.

2. Using the dependency tree to find the noun phrases associated with “MONEY” entities

Using the spaCy dependency parser, you can use various methods to navigate the dependency tree. To iterate over the dependency tree spaCy uses the terms “head” and “child” to describe the words connected by a single arc in the dependency tree, and the type of syntactic relationship is identified by using the term “dep”. Below is a visualisation of the dependency tree:

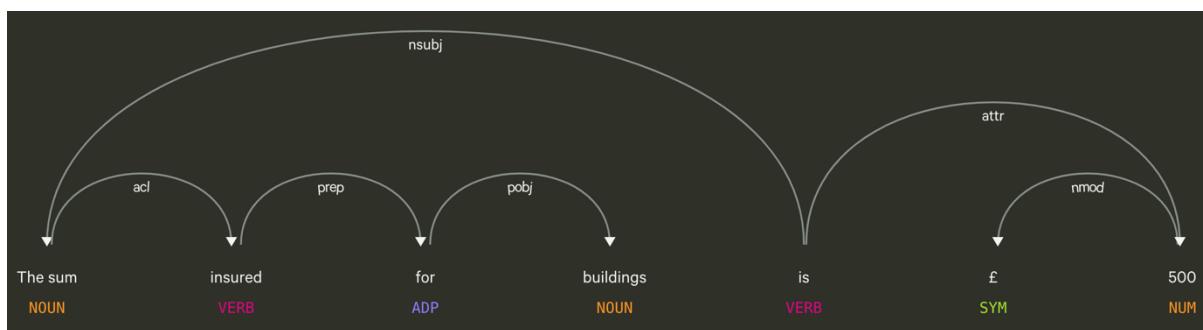


Figure 9: Visualisation of the dependency tree (Source: spacy.io)

As you can see each word is related in some manner and connected by arcs which have labels like “nsubj, prep, pobj, attr and nmod”, a full list of the types of labels, and their associated description, which can appear in the dependency tree are listed in Appendix A.

The idea behind this method was to find “MONEY” entities and then extract the associated noun chunks and other context by making use of the ability to iterate through the words using the dependency tree. Below is some code I used to try and complete this task:

```

def filter_spans(spans):
    # Filter a sequence of spans so they don't contain overlaps
    get_sort_key = lambda span: (span.end - span.start, span.start)
    sorted_spans = sorted(spans, key=get_sort_key, reverse=True)
    result = []
    seen_tokens = set()
    for span in sorted_spans:
        if span.start not in seen_tokens and span.end - 1 not in seen_tokens:
            result.append(span)
            seen_tokens.update(range(span.start, span.end))
    return result
  
```

```

def extract_currency_relations(doc):
    # Merge entities and noun chunks into one token
    spans = list(doc.ents) + list(doc.noun_chunks)
    spans = filter_spans(spans)
    with doc.retokenize() as retokenizer:
        for span in spans:
            retokenizer.merge(span)

    relations = []
    for money in filter(lambda w: w.ent_type_ == "MONEY", doc):
        if money.dep_ in ("attr", "dobj"):
            subject = [w for w in money.head.lefts if w.dep_ == "nsubj"]
            if subject:
                subject = subject[0]
                relations.append((subject, money))
        elif money.dep_ == "pobj" and money.head.dep_ == "prep":
            relations.append((money.head.head, money))
    return relations

```

The above code performs the following tasks:

- Retokenize the default spaCy spans (which are just single tokens by default) into named entities and then noun chunks, this means I am able to group these as effectively single tokens made up from a sequence of tokens.
- Filter to look for “MONEY” entities only, and then if the associated dependencies were attribute(attr) or a direct object(dobj) then I would look for the associated token on the dependency tree using head.left and if the associated dependency arc label was nominal subject (nsubj) then I added the subject and the “MONEY” entity to the relations list.
- If the “MONEY” entity dependency was not “attr” or “dobj” and it was object of preposition (pobj) and the dependency of the head of the “MONEY” entity was prepositional modifier(prep) I would then find the head of the head of the money entity and add that noun chunk and the “MONEY” entity to the list of relations.

This method of extracting data worked in some cases but was not reliable enough to use as the basis of the data extraction software because of the way the documents are worded and structured. The wording in the insurance documents don’t follow a natural way of using sentences, instead in the text data there would be a lot of sentences like “Sum insured – Buildings - £500” which does not create a very reliable dependency tree from which I could iterate over and extract the data in an accurate manner.

3. Incorrect Method of Creating Training Data for the NER Model

In the early stages of training the named entity recogniser, during the process of creating the required training data, I started off by providing sentences from the insurance documents which I felt would be most useful to extract as a whole named entity containing multiple tokens. To better explain my point here, I will provide a sample of the early training data and explain it:

```
TRAIN_DATA = [  
    ("Total turnover excluding exports to North America £3,700,000", {"entities": [(0, 60,  
        LABEL10)]}),  
]
```

In the above training data, the whole sentence has been tagged as the named entity associated with LABEL10. After training the named entity recogniser model using this type of data, I was receiving poor results of around 10% accuracy, which was not acceptable at all. I found that the issue with training data like the above, is that the model is not being provided with any context because the whole sentence is being tagged as a named entity. If the model is not able to generalise and find new examples of a new named entity, then it will always produce poor results. If I were only looking for exact matches to the sentence I provided in training, I would be better using something like a pattern matcher or regular expressions (via spaCy's Matcher or PhraseMatcher) to extract the exact matches. Another method to more generally match named entities would be through the use of word vectors, which will make use of the vector representations of each word in a sentence and calculate an overall vector representation, then find sentences with a similar vector representation. Using word vectors would mean you can match sentences that are similar rather than exactly the same as what you are searching for. In order to obtain better results while training I decided to start only tagging the pound values within the sentences in my training data, like in the example below:

```
TRAIN_DATA = [  
    ("Total turnover excluding exports to North America £3,700,000", {"entities": [(50, 60,  
        LABEL10)]}),  
]
```

Once this change was made to the training data, the accuracy of the model increased to around 70-80% in some of the models that I eventually trained.

4.3 Construction

During the construction of the data extraction software, as with writing any piece of software, it was important to consider that the code is going to be read many more times than it was written, so making code legible and well-structured is a key aspect of the construction process, this involved using the correct naming conventions, commenting and making sure the structure of the code is in line with the intended design and architecture. The use of commenting and system documentation will ensure the software is easy to follow and run. During the construction of this software I was coding in Python and using the PyCharm IDE, which helped me keep my code accurate and structured due to the various tools build in to PyCharm to sense check code as you write. This project was my first time working with natural language processing and I feel that the spaCy library and it's easy

to follow documentation and ability to train machine learning models with relative ease, made the process of learning feel quick and efficient.

As explained in the design section of this chapter, the first iteration of the data extraction software accomplishes the task in a basic manner. Most of the development time was spent learning the technologies and trying different approaches. As part of the construction section I will go through the 4 main classes in more detail, while only covering the most important methods.

ConvertPDFText:

```
class ConvertPDFText:

    def __init__(self):
        self.root = tk.Tk()
        self.dir_name = filedialog.askdirectory()

    def select_file(self):
        self.root.withdraw()
        file_name = filedialog.askopenfilename()
        return file_name

    def select_dir(self):
        self.root.withdraw()
        dir_name = filedialog.askdirectory()
        return dir_name

    def convert_pdf_to_image(self, file_path):
        pdf = wi(filename=str(file_path), resolution=500)
        print("Select a folder to save the images in")
        save_path = self.select_dir()
        pdfImage = pdf.convert("png")

        i = 1
        for img in pdfImage.sequence:
            page = wi(image=img)
            page.save(filename=save_path + "/" + str(i) + ".png")
            i += 1

        print("Images are now saved")

    @staticmethod
    def convert_multiple_image_to_text(img_dir):
        img_dir = Path(img_dir)
        text = ''
        for img_file in img_dir.iterdir():
            file_obj = Image.open(img_file)
            reader = pytesseract.image_to_string(file_obj, config='--psm 6')
            text += reader

        return text

    @staticmethod
    def convert_image_to_text(file_path):
        return pytesseract.image_to_string(Image.open(str(file_path)), config='--psm 6')
```

This class contains methods which allow me to select a directory or a file using a simple user interface created via Tkinter. The method called “convert_pdf_image” uses wand to convert each page of a PDF document into an image file of which I can choose the resolution and file type, these images are then saved into a directory of my choosing. A pivotal part of this project is being able to process text, hence the method “convert_image_to_text” plays a key role by using tesseract OCR to read the image files and provide the text required for natural language processing.

CreateDoc:

```
class CreateDoc:

    def __init__(self):
        self.doc = None

    def set_doc(self, multiple_images):
        convert = ConvertPDFText()
        if not multiple_images:
            self.doc =
nlp(str((convert.convert_image_to_text(convert.select_file()))))
        else:
            self.doc =
nlp(str(convert.convert_multiple_image_to_text(convert.select_dir())))
```

This class allows me to take the text produced by ConvertPDFText and create a doc object using spaCy.

SpacyFunctions:

```
class SpacyFunctions:

    def __init__(self, doc):
        self.myarr = []
        self.doc = doc

    @staticmethod
    def identify_entities(doc):
        for entity in doc.ents:
            print(entity.text + ' - ' + entity.label_)

    @staticmethod
    def extract_entities_to_pandas_df(doc, excel_path):
        d = []
        for entity in doc.ents:
            d.append(
                (entity.text, entity.label_,
                 list({val for key, val in ENTITY_VALUE_LIST.items() if key ==
entity.label_})[0]))

        df = pd.DataFrame(d, columns=['Extracted_Entity', 'Entity_Description',
'Expected_Value'])
        with pd.option_context('display.max_rows', None, 'display.max_columns',
None):
            df['Match_Check'] = df['Extracted_Entity'] == df['Expected_Value']
            print(df)
            df.to_excel(excel_writer=excel_path)

    @staticmethod
    def extract_entities_sentence_list():
        for doc1 in newDoc.set_sentence_doc():
```

```

        print(doc1.ents[0].text + ' - ' + doc1.ents[0].label_)

    for ent in self.doc.ents:
        for child in ent.text:
            triples.append((ent.text, child.text))

    return triples

    def on_match(self, matcher, doc, i, matches):
        # Get the current match and create tuple of entity label, start and end.
        # Append entity to the doc's entity.
        match_id, start, end = matches[i]
        entity = Span(doc, start, end, label="MONEY")
        self.doc.ents += (entity,)
        print(entity.text)

    def add_entity_for_pounds(self):
        pattern = [{"LOWER": "£"}, {"TEXT": {"REGEX": "\^\\ ?[0-9]+(\\,\\ ?[0-9]{1,3})?(\\.\\ [0-9]+)?$"}]}
        matcher.add("POUNDS", self.on_match, pattern)
        matches = matcher(self.doc)
        for match_id, start, end in matches:
            string_id = nlp.vocab.strings[match_id] # Get string representation
            span = self.doc[start:end] # The matched span
            print(match_id, string_id, start, end, span.text)

    def add_pounds_to_money(self):
        patterns = [{"label": "MONEY",
                     "pattern": [{"LOWER": "£"}, {"REGEX": "\^\\ ?[0-9]+(\\,\\ ?[0-9]{1,3})?(\\.\\ [0-9]+)?$"}]}]
        ruler.add_patterns(patterns)
        nlp.add_pipe(ruler)
        nlp.to_disk("test_model")

        self.identify_entities()

```

The SpacyFunctions class contains various methods for performing NLP tasks. Above I have provided a few of the most important methods in terms of the core task of extracting entities and displaying the output to the user. The method “extract_entities_to_pandas_df” makes use of the Pandas data analysis package for Python and allows me to look for all the entities within a doc object, examine the entity tag and value, then compare it to a list of expected values and display this information to the user. This method also exports the results to an Excel spreadsheet.

TrainingSpacyModel:

```

class TrainingSpacyModel:

    def __init__(self, model, new_model_name, output_dir, n_iter):
        self.model = model
        self.new_model_name = new_model_name
        self.output_dir = output_dir
        self.n_iter = n_iter
        self.test_text = "Public and products liability ~ insured £550.00 adviser."
        if self.model is not None:
            self.nlp = spacy.load(self.model) # load existing spaCy model

```

```

        print("Loaded model '%s'" % self.model)
    else:
        self.nlp = spacy.blank("en") # create blank Language class
        print("Created blank 'en' model")
    # Add entity recognizer to model if it's not in the pipeline
    # nlp.create_pipe works for built-ins that are registered with spaCy
    if "ner" not in self.nlp.pipe_names:
        self.ner = self.nlp.create_pipe("ner")
        self.nlp.add_pipe(self.ner)
    # otherwise, get it, so we can add labels to it
    else:
        self.ner = self.nlp.get_pipe("ner")

    def get_model(self):
        return self.model

    def get_new_model_name(self):
        return self.new_model_name

    def get_output_dir(self):
        return self.output_dir

    def get_number_if_iterations(self):
        return self.n_iter

    def get_test_text(self):
        return self.test_text

    def set_model(self, model):
        self.model = model

    def set_new_model_name(self, new_model_name):
        self.new_model_name = new_model_name

    def set_n_iter(self, n_iter):
        self.n_iter = n_iter

    def set_test_text(self, test_text):
        self.test_text = test_text

    def set_up_training(self):
        random.seed(0)

        for lbl in ENTITY_LIST:
            self.ner.add_label(lbl)

    def train_model(self):
        if self.model is None:
            optimizer = self.nlp.begin_training()
        else:
            optimizer = self.nlp.resume_training()
        # get names of other pipes to disable them during training
        other_pipes = [pipe for pipe in self.nlp.pipe_names if pipe != "ner"]
        with self.nlp.disable_pipes(*other_pipes): # only train NER
            sizes = compounding(1.0, 4.0, 1.001)
            # batch up the examples using spaCy's minibatch
            i = 0
            for itn in range(self.n_iter):
                random.shuffle(AXA_TRAIN_DATA)
                batches = minibatch(AXA_TRAIN_DATA, size=sizes)
                losses = {}
                for batch in batches:
                    texts, annotations = zip(*batch)
                    self.nlp.update(texts, annotations, sgd=optimizer, drop=0.35,
losses=losses)
                i += 1

```

```

        print("Iteration " + str(i) + " Losses", losses)

def test_trained_model(self):
    # test the trained model
    doc = self.nlp(self.test_text)
    print("Entities in '%s'" % self.test_text)
    for ent in doc.ents:
        print(ent.label_, ent.text)

def save_trained_model(self):
    # save model to output directory
    if self.output_dir is not None:
        output_dir = Path(self.output_dir)
        if not output_dir.exists():
            output_dir.mkdir()
    self.nlp.meta["name"] = self.new_model_name # rename model
    self.nlp.to_disk(output_dir)
    print("Saved model to", output_dir)

def test_saved_model(self):
    # test the saved model
    move_names = list(self.ner.move_names)
    print("Loading from", self.output_dir)
    nlp2 = spacy.load(self.output_dir)
    # Check the classes have loaded back consistently
    assert nlp2.get_pipe("ner").move_names == move_names
    doc2 = nlp2(self.test_text)
    for ent in doc2.ents:
        print(ent.label_, ent.text)

```

This class allows the user to train a spacy model. It allows the user to choose if they want to train an existing or blank model, choose where to save the model and other settings such as the number of iterations during training and the batch sizes. Depending on the setting the user selects, the training process can be time consuming. The models I trained during this dissertation took between 15 mins and 1 hour to complete training. Once the model is trained there are methods in this class to test the model and save it to a directory of your choice.

I have left out some of the code in the above examples, the complete code will be submitted along with this dissertation.

Next, I will give some detail on the development tools and languages I was using during the process:

4.3.1 Python – Programming language

Python is a high-level, interpreted language which was first released back in 1991 and is intended to make code more readable. It's a multi-paradigm language which supports procedural programming, object-oriented programming and functional programming (to name a few) and is useful for writing code for small- and large-scale projects. After some initial research I found that Python was the best language to go with for the purposes of this project, due to the vast number of libraries available for natural language processing and extracting text from documents. Python is also very popular for use in data analytics, which makes use of libraries such as pandas.

4.3.2 PyCharm – IDE

PyCharm is an integrated development environment (IDE) developed by a Czech company called JetBrains (the same people that developed IntelliJ for using when coding in Java) which is specifically developed for coding in Python. PyCharm provides a graphical debugger, code analysis, integrated unit tester, version control, and it also supports web application development in Django, which will be extremely useful in the future iterations of the data extraction software.

4.3.3 SpaCy – Natural Language Processing Library

SpaCy is an open source NLP library with a vast amount of functionality that I was able to learn about via their easy to follow documentation and code examples. SpaCy also has easily trainable machine learning pipeline components that allowed me to develop the basic functionality of the data extraction software with relatively little code, as long as I provided a substantial amount of tagged training data for the supervised training of the machine learning model.

4.3.4 Pandas – Python Library

Pandas is a Python library used for data analytics and manipulation. It provided various features such as a dataframe, which is similar to a spreadsheet, that you can fill with data easily from the output of a python method, this data is then easily manipulated in the dataframe and various calculations can be performed. The data within a dataframe can then easily be exported to other formats like Excel or CSV. This library was particularly useful for making it easy to visualise the output from the data extraction software. I was able to extract all the matched entities and their associated description to a dataframe and then add columns to show the expected output along with a check on whether the extracted entities were correct or not. I was then able to extract this data to an Excel spreadsheet, which is the most common used format within the company where I was working on the project.

4.3.5 Tesseract OCR – Optical Character Recognition Software

This is the optical character recognition software I used for extracting text from image files. It was initially developed by Hewlett Packard and now sponsored by Google and is known as one of the best tools to use for OCR tasks given the number of different settings available which are useful depending on the specific task being performed.

4.3.6 ImageMagick – Convert a PDF to an Image

ImageMagick allows me to convert the PDF files to image files, with the ability to choose the format and resolution of the output image.

4.3.7 Tkinter – Python Graphical User Interface Package

Tkinter is the most commonly used GUI package for Python. Tkinter stands for Tk Interface because it is the standard Python interface to the Tk GUI toolkit. It enables you to easily

create GUI components like windows that contain widgets such as buttons, labels, menus, messages and also entry widgets that accept input from a user. As part of the first iteration of the data extraction tool I used Tkinter's filedialog module to enable me to display a GUI for the user to select a file from their machine to be processed via the data extraction software. This functionality will most likely be performed by the Django web application in future iterations of this software.

The various tools mentioned above were crucial in the development of the data extraction software and worked very well together on the whole, other than some issues during installation. I first started the project on a Windows 10 machine and was having some issues installing spaCy, at this point I didn't want to waste more time and switched to a MacBook running MacOS Mojave and everything installed smoothly at my first attempt.

Given that the first iteration of this project was a proof of concept, there was an initial focus on trying to show that the basic concept of extracting data from PDFs using NLP, was possible, hence the reason there is not a huge amount of code. Most of the time was spent trying various approaches which I have explained above. During the process of writing the code I came across a few interesting issues, which are as follows:

1. The Catastrophic Forgetting Problem

When you decide to train an existing spaCy model to add a new named entity label, there is a risk that you could cause the model to forget everything it has already learned. This is due to the fact you are trying to optimise more than one learning problem at the same time while using the weights from the first problem to initialise the weights of the second problem. In spacy this has to do with the optimisation algorithm being overly sensitive to initialisation. This isn't a problem that is specific to spaCy, it's been an issue that has been around in neural networks for a while. One of the ways to avoid this problem when training a pre-existing spaCy model, is to make sure you include examples in your training data that the model already knows, along with the new examples of the entities you are trying to add to the model. I first came across this issue as I was developing the data extraction software, when I spend some time putting together a large amount of training data and tried to train a pre-existing spaCy model. After waiting one and a half hours for the training to complete, the resulting model was not tagging any entities at all. After this I decided to do some research, and this is when I discovered the catastrophic forgetting problem and was able to adjust my training data to avoid the issue in the future.

2. Isolating Pipeline Components During Training

When I first started attempting to train the named entity recogniser, I was getting very bad results no matter how good the training data was or how many times I iterated over the training data. For some reason the text was not being parsed correctly anymore, the tokenizer was not tokenizing things and the named entity recognised did not seem to be learning anything from the training. I was able to find that other people had experienced the same issue and this is when I came across the fact that you are able to disable certain pipeline components during the training process, which enabled me to isolate the named

entity recogniser and train it without messing up the other components in the process.

Below is the code that was required to isolate the NER:

```
other_pipes = [pipe for pipe in self.nlp.pipe_names if pipe != "ner"]
```

```
with self.nlp.disable_pipes(*other_pipes): # only train NER
```

4.4 Testing

Testing is a vital stage of the software development lifecycle, because a complex piece of software will more than likely have a few bugs even when it is released. Testing is a way of uncovering these bugs and then fixing them and re-testing. The process of testing needs to be focused on the bugs that could cause major disruption, then working your way down to the more minor bugs, if you ever get to them.

Testing doesn't only uncover bugs, but also makes us think about whether or not the software is doing what it was intended to do, i.e. have I met the requirements? There are multiple types of testing, some of which are listed below:

1. Unit Testing

- This is completed by the developers and it tests each method within the classes, to make sure that they are doing what they are supposed to do. In unit testing you usually know the expected output and test to make sure it matches, if you set them up correctly you can test automatically after altering code, this ensures that you aren't causing bugs when you think you are improving a piece of code

2. Integration Testing

- This is usually performed by integration testers and they test the interaction between classes, modules, interfaces, to make sure they are behaving as they were intended to. This stage doesn't usually require the tester to be a developer, therefore the tester doesn't need to see the actual code. This is useful when you need to pull together a lot of code written by different developers or at different times

3. System Testing

- This can be carried out by the end user to make sure the system is doing what they want it to do. They are usually asked to test inputs and outputs only, to see if they can find any hidden bugs that have been missed in the earlier stages of testing

4. Acceptance Testing

- This is the final stages of testing and is similar to system testing except the users are usually testing the whole system and checking for hidden bugs while performing day to day tasks. Sometimes the users are provided with scripts to use that outline a process they can follow, then they will report on their experience for each test. This makes sure that the system is doing what they wanted it to do in the first place.

As part of this project I have started unit testing the software and will be performing the other types of testing at a later stage when I have a product that has a working web application component. Due to the fact that this software is going to form part of the pre-existing online broker system, integration testing will play an important role in the process. We will need to test how well the data extraction software's components are interacting with the components of the online broker system, a few examples of the things that need to be tested are as follows:

- Is the extracted data being saved correctly in the database?
- Is it intuitive for the user to navigate to and use the data extraction component?

When it comes to time for system testing, I will be asking a couple of the employees that currently deal with the manual keying of the insurance data to test the software for me and provide me with feedback on their experience. I will be asking them to use the GUI to upload an Insurance document and then sense check the output against what they would expect it to be. This will also enable the user to uncover any bugs that may exist and that will give me time to make the necessary changes before moving onto acceptance testing. At the stage of acceptance testing I will provide the users with a script of actions to perform on the software and I will score each action to make sure they were able to complete it within a reasonable time with minimal errors. I may also add in some steps which involve using the current online broker system, so I am able to assess if they work together in a seamless manner.

4.5 Deploy

Since the web application component has not yet been developed, I recommend creating a virtual environment and opening up all the code in an IDE like PyCharm. In order to run the data extraction software, you must first install all the required dependencies, which are as follows:

- Python
- SpaCy
- Wand
- Pytesseract
- Pandas
- Tkinter

As part of the code that was submitted, I have included a model that I have trained. Once the dependencies above are installed you will then need to make sure the model is being loaded correctly, by saving the model in a folder and then pointing the code to that directory, as follows:

```
model = 'place the link to the directory here'
```

Due to the confidentiality of the documents from which I was extracting data during development, I was unable to send any examples of them, instead I have provided a few sample sentences to run the software on, which will enable the user to see the entities being extracted and placed in a pandas dataframe that will be printed in the console for

easy viewing. I have included some code at the bottom of the file that will run the software on the sample sentences, so at this point the user can now run the code. If the user wants to try training an NER model themselves, the following steps should be performed:

1. Make sure the training data is available and imported into the module
2. Use the following code to set up the training to your requirements, train, test and save the new trained model:
 - a. `trainer = TrainingSpacyModel(None, 'Name the new model here', 'add the directory you want to save it in here', choose a number of iterations)`
 - b. `trainer.set_up_training()`
 - c. `trainer.train_model()`
 - d. `trainer.test_trained_model()`
 - e. `trainer.save_trained_model()`
 - f. `trainer.test_saved_model()`

Depending on the number of iterations the user chooses to perform, the training can take a substantial amount of time in some cases. The user will end up with a new trained NER model which they can then test as normal.

4.6 Evaluation

In order to test the true effectiveness and accuracy of the first iteration of the data extraction software I trained the spaCy named entity recogniser using various different settings. In this section I will go through each of the models I ended up with as a result of the training and look at the level of accuracy and the coverage of the data that was extracted. In order to train a spaCy model, you first need a relatively large amount of training data. Training data is in the following format:

```
TRAIN_DATA = [  
    (  
        "£2,300 is the amount that was paid",  
        {"entities": [(0, 6, "MONEY")]},  
    ),  
]
```

A sentence is provided, along with the start and one after the end point of the entity you are trying to label, and the label you would like to give that entity. In the example above I have labelled “£2,300” as a “MONEY” named entity. To create training data for the data extraction software, I went through the documents and used sentences from the documents and tagged the entities I wanted to name, I also duplicated some of the examples to better train the model, which resulted in a higher level of accuracy. The developers of spaCy also recommend adding in example sentences which have no entities you would like to tag, because this allows the model to learn to ignore certain parts of the document that we are not interested in. SpaCy allows you to train one of their existing models, but I chose to train a blank model because I was not interested in any of the current entity types provided by spaCy. I created the following list of new entity types to train the model to recognise:

LABEL1 = "Money - on premises during business hours"
 LABEL2 = "Money - on premises outside business hours not in safe"
 LABEL3 = "Money - on premises outside business hours in locked safe"
 LABEL4 = "Money - in bank night safe"
 LABEL5 = "Money - in transit"
 LABEL6 = "Money - at home of director or authorized employee"
 LABEL7 = "Money - Non-negotiable money"
 LABEL8 = "Money - excess"
 LABEL9 = "Property damage - Premium excluding IPT"
 LABEL10 = "Business interruption - Premium excluding IPT"
 LABEL11 = "Money and personal accident assault - Premium excluding IPT"
 LABEL12 = "Public and products liability - Premium excluding IPT"
 LABEL13 = "Employers liability - Premium excluding IPT"
 LABEL14 = "Legal protection plan - Premium excluding IPT"
 LABEL15 = "Goods in transit - Premium excluding IPT"
 LABEL16 = "Property damage - Stock(sum insured)"
 LABEL17 = "Property damage - Other(sum insured)"
 LABEL18 = "Property damage - Subsidence Excess"
 LABEL19 = "Business interruption - Gross profit"
 LABEL20 = "Business interruption - Gross revenue"
 LABEL21 = "Business interruption - Increased Cost of Working Only"
 LABEL22 = "Business interruption - Gross Rentals"
 LABEL23 = "Business interruption - Additional Increased Cost of Working"
 LABEL24 = "Goods in transit - Limit per vehicle"
 LABEL25 = "Goods in transit - Annual value in transit"
 LABEL26 = "Goods in transit - Excess"
 LABEL27 = "Legal expenses - Excess"

The new entity types correspond to the various sections of the commercial combined insurance documents I was working on and the idea is to extract the corresponding monetary amount for each one. To enable me to test the accuracy of the data extracted, I went through the document I was testing and created a list of the expected output for each of the entity types, and in the output (which is in the form of a pandas dataframe) I added the expected amount to each entry and a Boolean test to make it easily distinguishable at first glance.

When training a model there are a number of parameters to consider and tweak to find the optimal setting for the specific task. During the training of the data extraction software I varied the following parameters:

Batch size: I used a function called compounding to vary the batch size every time the generator is called, as in the following code:

```
Batch_size = compounding(1.0, 10.0, 1.5)
```

```
Batches = minibatch(TRAIN_DATA, Batch_size)
```

In the above example the batch size would start at 1.0 and the next batch size is the previous multiplied by the 1.5, which is the compounding factor, until the batch size reaches a maximum of 10.0. These iterating batch sizes are then iterated over using another spaCy function called minibatch, which allows you to process all the training data in chunks of varying sizes of data.

Dropout Rate: The dropout rate is the percentage of training examples you would like to hold back during training, these examples are then used by spaCy to test the trained model.

Number of iterations: The number of iterations is the number of times spaCy runs through the training data before training is complete.

In each of the following section I used the same set of training data containing around 7,000 tagged samples and tested each model by extracting entities from a selected PDF insurance document. The sample document I was testing with had a maximum of 22 pieces of data that should be extracted. In order to measure the success of each model I exported the extracted data to an Excel spreadsheet where I was able to calculate the overall percentage of accuracy. The Excel output has been included for each model to provide some detail of the output when testing each new trained model.

4.6.1 Trained Model 1

The following setting were used during training:

Batch size: Batch_size = compounding(1.0, 10.0, 1.5)

Dropout rate: 10%

Number of iterations: 30

Table 1: Model 1 Excel output

Extracted_Entity	Entity_Description	Expected_Value	Match_Check
£2,536.26	Property damage - Premium excluding IPT	£2,536.26	TRUE
£1,338.12	Business interruption - Premium excluding IPT	£1,338.12	TRUE
£100.00	Money and personal accident assault - Premium excluding IPT	£100.00	TRUE
£550.00	Public and products liability - Premium excluding IPT	£550.00	TRUE
£1,408.76	Employers liability - Premium excluding IPT	£1,408.76	TRUE
£261.80	Legal protection plan - Premium excluding IPT	£261.80	TRUE
£100.00	Goods in transit - Premium excluding IPT	£100.00	TRUE
£212,000.00	Property damage - Stock(sum insured)	£212,000	TRUE
£4,000,000.00	Property damage - Other(sum insured)	£4,000,000	TRUE
£1,000.00	Property damage - Subsidence Excess	£1,000	TRUE
£3,375,000.00	Business interruption - Gross profit	£3,375,000	TRUE
£100,000	Business interruption - Additional Increased Cost of Working	£100,000	TRUE
£500.00	Money - on premises outside business hours not in safe	£500	TRUE
£500.00	Money - on premises outside business hours not in safe	£500	TRUE
£500.00	Money - at home of director or authorized employee	£500	TRUE
£250,000.00	Money - Non-negotiable money	£250,000	TRUE
£250.00	Money - excess	£250	TRUE
	Money - on premises during business hours	£500	FALSE
	Money - on premises outside business hours in locked safe	£1,000	FALSE
	Money - in transit	£2,500	FALSE
£10,000.00	Goods in transit - Limit per vehicle	£10,000	TRUE
£0.00	Goods in transit - Annual value in transit	£0	TRUE
£100.00	Goods in transit - Excess	£100	TRUE
£1,000,000.00	Legal expenses - Excess	£250	FALSE
Overall Accuracy			83%

Out of the possible 24 entities to be extracted this model was able to extract 20 of them correctly. Out of the 4 that were not extracted correctly, 3 were not detected at all and the incorrect value was extracted for the last one.

Accuracy level attained: 82%

4.6.1 Trained Model 2

The following setting were used during training:

Batch size: Batch_size = compounding(4.0, 32.0, 1.001)

Dropout rate: 50%

Number of iterations: 30

Table 2: Model 2 Excel output

Extracted_Entity	Entity_Description	Expected_Value	Match_Check
	Property damage - Premium excluding IPT		FALSE
£1,338.12	Business interruption - Premium excluding IPT	£1,338.12	TRUE
£100.00	Money and personal accident assault - Premium excluding IPT	£100.00	TRUE
£550.00	Public and products liability - Premium excluding IPT	£550.00	TRUE
£100.00	Business interruption - Premium excluding IPT	£1,338.12	FALSE
	Employers liability - Premium excluding IPT	£1,408.76	FALSE
£1,408.76	Employers liability - Premium excluding IPT	£1,408.76	TRUE
	Legal protection plan - Premium excluding IPT	£261.80	FALSE
	Goods in transit - Premium excluding IPT	£100.00	FALSE
	Property damage - Stock(sum insured)	£212,000	FALSE
£4,000,000	Property damage - Other(sum insured)	£4,000,000	TRUE
	Property damage - Subsidence Excess	£1,000	FALSE
£3,375,000	Business interruption - Gross profit	£3,375,000	TRUE
£500	Money - on premises outside business hours not in safe	£500	TRUE
£500	Money - at home of director or authorized employee	£500	TRUE
	Money - Non-negotiable money	£250,000	FALSE
£250	Money - excess	£250	TRUE
	Money - on premises during business hours	£500	FALSE
£1,000	Money - on premises outside business hours in locked safe	£1,000	TRUE
	Money - in transit	£2,500	FALSE
	Goods in transit - Limit per vehicle	£10,000	FALSE
£0	Goods in transit - Annual value in transit	£0	TRUE
	Goods in transit - Excess	£100	FALSE
	Legal expenses - Excess	£250	FALSE
Overall Accuracy		46%	

During the training of this model I decided to increase the dropout rate, to hold back 50% of the training data, which has had a detrimental effect on the accuracy level. I also increased the maximum batch size to 32 and decreased the compounding factor to 1.001. After 30 iterations of training, out of the possible 24 entities to be extracted this model was able to

extract 11 of them correctly. Out of the 13 that were not extracted correctly, 9 were not detected at all and the wrong value was extracted for the remaining four.

Accuracy level attained: 46%

4.6.1 Trained Model 3

The following setting were used during training:

Batch size: Batch_size = compounding(1.0, 10.0, 1.001)

Dropout rate: 50%

Number of iterations: 30

Table 3: Model 3 Excel output

Extracted_Entity	Entity_Description	Expected_Value	Match_Check
	Property damage - Premium excluding IPT	£2,536	FALSE
£1,338.12	Business interruption - Premium excluding IPT	£1,338.12	TRUE
£100.00	Money and personal accident assault - Premium excluding IPT	£100.00	TRUE
£550.00	Public and products liability - Premium excluding IPT	£550.00	TRUE
£1,408.76	Employers liability - Premium excluding IPT	£1,409	TRUE
	Legal protection plan - Premium excluding IPT	£262	FALSE
	Goods in transit - Premium excluding IPT	£100.00	FALSE
	Property damage - Stock(sum insured)	£212,000	FALSE
	Property damage - Other(sum insured)	£4,000,000	FALSE
	Property damage - Subsidence Excess	£1,000	FALSE
	Business interruption - Gross profit	£3,375,000	FALSE
£212,000	Business interruption - Additional Increased Cost of Working	£100,000	FALSE
	Money - on premises during business hours	£500	FALSE
£500	Money - on premises outside business hours not in safe	£500	TRUE
£500	Money - at home of director or authorized employee	£500	TRUE
	Money - Non-negotiable money	£250,000	FALSE
£250	Money - excess	£250	TRUE
	Money - on premises during business hours	£500	FALSE
£1,000	Money - on premises outside business hours in locked safe	£1,000	TRUE
	Money - in transit	£2,500	FALSE
£10,000	Goods in transit - Limit per vehicle	£10,000	TRUE
	Goods in transit - Annual value in transit	£0	FALSE
100	Goods in transit - Limit per vehicle	£100	TRUE
	Legal expenses - Excess	£250	FALSE
Overall Accuracy			42%

During the training of this model I decided to hold back 50% of the training data, with a max batch size of 10 and a compounding factor of 1.001. After 30 iterations of training, out of the possible 24 entities to be extracted this model was able to extract 10 of them correctly.

Accuracy level attained: 42%

4.6.1 Trained Model 4

The following setting were used during training:

Batch size: Batch_size = compounding(1.0, 10.0, 1.5)

Dropout rate: 10%

Number of iterations: 20

Table 4: Model 4 Excel output

Extracted_Entity	Entity_Description	Expected_Value	Match_Check
	Property damage - Premium excluding IPT	£2,536	FALSE
£1,338.12	Business interruption - Premium excluding IPT	£1,338.12	TRUE
£100.00	Money and personal accident assault - Premium excluding IPT	£100.00	TRUE
£550.00	Public and products liability - Premium excluding IPT	£550.00	TRUE
£1,408.76	Employers liability - Premium excluding IPT	£1,409	TRUE
	Legal protection plan - Premium excluding IPT	£262	FALSE
£100.00	Goods in transit - Premium excluding IPT	£100.00	TRUE
£212,000	Property damage - Stock(sum insured)	£212,000	TRUE
£4,000,000	Property damage - Other(sum insured)	£4,000,000	TRUE
	Property damage - Subsidence Excess	£1,000	FALSE
£3,375,000	Business interruption - Gross profit	£3,375,000	TRUE
£212,000	Business interruption - Additional Increased Cost of Working	£100,000	FALSE
£500	Money - on premises during business hours	£500	TRUE
£500	Money - on premises outside business hours not in safe	£500	TRUE
£500	Money - at home of director or authorized employee	£500	TRUE
	Money - Non-negotiable money	£250,000	FALSE
£250	Money - excess	£250	TRUE
	Money - on premises during business hours	£500	FALSE
£1,000	Money - on premises outside business hours in locked safe	£1,000	TRUE
£2,500	Money - in transit	£2,500	TRUE
£10,000	Goods in transit - Limit per vehicle	£10,000	TRUE
	Goods in transit - Annual value in transit	£0	FALSE
100	Goods in transit - Limit per vehicle	£100	TRUE
£250	Legal expenses - Excess	£250	TRUE
Overall Accuracy		71%	

During the training of this model I decided to hold back 10% of the training data, with a max batch size of 10 and a compounding factor of 1.5. After 20 iterations of training, out of the possible 24 entities to be extracted this model was able to extract 17 of them correctly.

Accuracy level attained: 71%

4.6.1 Trained Model 5

The following setting were used during training:

Batch size: Batch_size = compounding(1.0, 4.0, 1.001)

Dropout rate: 10%

Number of iterations: 30

Table 5: Model 5 Excel output

Extracted_Entity	Entity_Description	Expected_Value	Match_Check
	Property damage - Premium excluding IPT	£2,536	FALSE
£1,338.12	Business interruption - Premium excluding IPT	£1,338.12	TRUE
£100.00	Money and personal accident assault - Premium excluding IPT	£100.00	TRUE
£550.00	Public and products liability - Premium excluding IPT	£550.00	TRUE
£1,408.76	Employers liability - Premium excluding IPT	£1,409	TRUE
	Legal protection plan - Premium excluding IPT	£262	FALSE
£100,000.00	Goods in transit - Premium excluding IPT	£100.00	TRUE
£212,000	Property damage - Stock(sum insured)	£212,000	TRUE
£4,000,000	Property damage - Other(sum insured)	£4,000,000	TRUE
	Property damage - Subsidence Excess	£1,000	FALSE
£3,375,000	Business interruption - Gross profit	£3,375,000	TRUE
£212,000	Business interruption - Additional Increased Cost of Working	£100,000	FALSE
£500	Money - on premises during business hours	£500	TRUE
£500	Money - on premises outside business hours not in safe	£500	TRUE
£500	Money - at home of director or authorized employee	£500	TRUE
	Money - Non-negotiable money	£250,000	FALSE
	Money - excess	£250	FALSE
	Money - on premises during business hours	£500	FALSE
£1,000	Money - on premises outside business hours in locked safe	£1,000	TRUE
£2,500	Money - in transit	£2,500	TRUE
£10,000	Goods in transit - Limit per vehicle	£10,000	TRUE
	Goods in transit - Annual value in transit	£0	FALSE
100	Goods in transit - Limit per vehicle	£100	TRUE
£250	Legal expenses - Excess	£250	TRUE
Overall Accuracy			67%

During the training of this model I decided to hold back 10% of the training data, with a max batch size of 4 and a compounding factor of 1.001. After 30 iterations of training, out of the possible 24 entities to be extracted this model was able to extract 16 of them correctly.

Accuracy level attained: 67%

4.6.1 Trained Model 6

The following setting were used during training:

Batch size: Batch_size = compounding(1.0, 4.0, 1.001)

Dropout rate: 10%

Number of iterations: 5

Table 6: Model 6 Excel output

Extracted_Entity	Entity_Description	Expected_Value	Match_Check
	Property damage - Premium excluding IPT	£2,536	FALSE
£1,338.12	Business interruption - Premium excluding IPT	£1,338.12	TRUE
£100.00	Money and personal accident assault - Premium excluding IPT	£100.00	TRUE
£550.00	Public and products liability - Premium excluding IPT	£550.00	TRUE
£1,408.76	Employers liability - Premium excluding IPT	£1,409	TRUE
	Legal protection plan - Premium excluding IPT	£262	FALSE
£100,000.00	Goods in transit - Premium excluding IPT	£100.00	TRUE
£212,000	Property damage - Stock(sum insured)	£212,000	TRUE
£4,000,000	Property damage - Other(sum insured)	£4,000,000	TRUE
	Property damage - Subsidence Excess	£1,000	FALSE
£3,375,000	Business interruption - Gross profit	£3,375,000	TRUE
£212,000	Business interruption - Additional Increased Cost of Working	£100,000	FALSE
£500	Money - on premises during business hours	£500	TRUE
£500	Money - on premises outside business hours not in safe	£500	TRUE
£500	Money - at home of director or authorized employee	£500	TRUE
	Money - Non-negotiable money	£250,000	FALSE
	Money - excess	£250	FALSE
	Money - on premises during business hours	£500	FALSE
£1,000	Money - on premises outside business hours in locked safe	£1,000	TRUE
£2,500	Money - in transit	£2,500	TRUE
£10,000	Goods in transit - Limit per vehicle	£10,000	TRUE
	Goods in transit - Annual value in transit	£0	FALSE
100	Goods in transit - Limit per vehicle	£100	TRUE
£250	Legal expenses - Excess	£250	FALSE
Overall Accuracy		63%	

During the training of this model I decided to hold back 10% of the training data, with a max batch size of 4 and a compounding factor of 1.001. After 5 iterations of training, out of the possible 24 entities to be extracted this model was able to extract 15 of them correctly.

Accuracy level attained: 63%

4.6.1 Trained Model 7

The following setting were used during training:

Batch size: Batch_size = compounding(1.0, 4.0, 1.001)

Dropout rate: 35%

Number of iterations: 5

Table 7: Model 7 Excel output

Extracted_Entity	Entity_Description	Expected_Value	Match_Check
	Property damage - Premium excluding IPT	£2,536	FALSE
£1,338.12	Business interruption - Premium excluding IPT	£1,338.12	TRUE
£100.00	Money and personal accident assault - Premium excluding IPT	£100.00	TRUE
£550.00	Public and products liability - Premium excluding IPT	£550.00	TRUE
£1,408.76	Employers liability - Premium excluding IPT	£1,409	TRUE
	Legal protection plan - Premium excluding IPT	£262	FALSE
£100,000.00	Goods in transit - Premium excluding IPT	£100.00	TRUE
£212,000	Property damage - Stock(sum insured)	£212,000	TRUE
£4,000,000	Property damage - Other(sum insured)	£4,000,000	TRUE
	Property damage - Subsidence Excess	£1,000	FALSE
£3,375,000	Business interruption - Gross profit	£3,375,000	TRUE
£212,000	Business interruption - Additional Increased Cost of Working	£100,000	FALSE
£500	Money - on premises during business hours	£500	TRUE
£500	Money - on premises outside business hours not in safe	£500	TRUE
£500	Money - at home of director or authorized employee	£500	TRUE
	Money - Non-negotiable money	£250,000	FALSE
	Money - excess	£250	FALSE
	Money - on premises during business hours	£500	FALSE
£1,000	Money - on premises outside business hours in locked safe	£1,000	TRUE
£2,500	Money - in transit	£2,500	FALSE
£10,000	Goods in transit - Limit per vehicle	£10,000	TRUE
	Goods in transit - Annual value in transit	£0	FALSE
100	Goods in transit - Limit per vehicle	£100	TRUE
£250	Legal expenses - Excess	£250	FALSE
Overall Accuracy		58%	

During the training of this model I decided to hold back 35% of the training data, with a max batch size of 4 and a compounding factor of 1.001. After 5 iterations of training, out of the possible 24 entities to be extracted this model was able to extract 14 of them correctly.

Accuracy level attained: 58%

4.6.1 Trained Model 8

This model was trained by adding the new named entity labels to an existing spaCy named entity recognition model. The following setting were used (same as model 8):

Batch size: Batch_size = compounding(1.0, 4.0, 1.001)

Dropout rate: 35%

Number of iterations: 5

Table 8: Model 8 Excel output

Extracted_Entity	Entity_Description	Expected_Value	Match_Check
	Property damage - Premium excluding IPT	£2,536	FALSE
£1,338.12	Business interruption - Premium excluding IPT	£1,338.12	TRUE
£100.00	Money and personal accident assault - Premium excluding IPT	£100.00	TRUE
£550.00	Public and products liability - Premium excluding IPT	£550.00	TRUE
	Employers liability - Premium excluding IPT	£1,409	FALSE
	Legal protection plan - Premium excluding IPT	£262	FALSE
£100.00	Goods in transit - Premium excluding IPT	£100.00	TRUE
	Property damage - Stock(sum insured)	£212,000	FALSE
	Property damage - Other(sum insured)	£4,000,000	FALSE
	Property damage - Subsidence Excess	£1,000	FALSE
£3,375,000	Business interruption - Gross profit	£3,375,000	TRUE
£100,000	Business interruption - Additional Increased Cost of Working	£100,000	TRUE
£500	Money - on premises during business hours	£500	TRUE
£500	Money - on premises outside business hours not in safe	£500	TRUE
	Money - at home of director or authorized employee	£500	FALSE
£250,000	Money - Non-negotiable money	£250,000	TRUE
	Money - excess	£250	FALSE
	Money - on premises during business hours	£500	FALSE
£1,000	Money - on premises outside business hours in locked safe	£1,000	TRUE
	Money - in transit	£2,500	FALSE
	Goods in transit - Limit per vehicle	£10,000	FALSE
	Goods in transit - Annual value in transit	£0	FALSE
	Goods in transit - Excess	£100	FALSE
	Legal expenses - Excess	£250	FALSE
Overall Accuracy		42%	

During the training of this model I decided to hold back 35% of the training data, with a max batch size of 4 and a compounding factor of 1.001. After 20 iterations of training, out of the possible 24 entities to be extracted this model was able to extract 10 of them correctly.

Accuracy level attained: 42%

The above models had varying levels of success, with the model 1 and 4 showing the highest levels of success. Model 1 and 4 shared similar settings, the only difference being that model 1 went through 10 more iterations than model 4, which resulted in an accuracy increase of 11%. From the tests I have performed there tends to be better results from restricting the maximum size of batches, increasing the compounding rate and having a higher amount of iterations. I limited the maximum number of iterations to 30 due to time constraints, but I would expect the accuracy to increase with more iterations, up to a point when it will become detrimental to the level of accuracy. When testing the above models, I was testing on the text from a full insurance document and observing how accurately the model was able to identify each piece of relevant information. Later in the project I decided to test how well the models deal with smaller samples of text by testing each model on 5 sample sentences. Below are the 5 sample sentences used for the test, along with the results for each of the 8 models:

```
SENTENCE_LIST = {  
    "Business interruption Y insured £1,338.12 insured have not been",  
    "Goods in transit Y insured £100.00",  
    "Money and personal accident VW insured £100.00 would like to change your",  
    "2 Additional increased cost £100,000 - 18 months All Risks",  
    "Money from the premises out of business hours from any £1,000 unspecified safe"}
```

Please note: The sentences do not read like natural English, some of the pound amounts are not in the correct place and there are some letters within the sentences that do not make sense, this is because I have trained the models with the output from Tesseract OCR.

Model 1:

Output:

£1,000 - Money - on premises outside business hours in locked safe
£100,000 - Business interruption - Additional Increased Cost of Working
£100.00 - Money and personal accident assault - Premium excluding IPT
£100.00 - Goods in transit - Premium excluding IPT
£1,338.12 - Business interruption - Premium excluding IPT100%

Accuracy level attained: 100%

Model 2:

Output:

£1,000 - Money - on premises outside business hours in locked safe
£100,000 - Business interruption - Additional Increased Cost of Working
£100.00 - Money and personal accident assault - Premium excluding IPT
£100.00 - Goods in transit - Premium excluding IPT
£1,338.12 - Business interruption - Premium excluding IPT100%

Accuracy level attained: 100%

Model 3:

Output:

£1,000 - Money - on premises outside business hours in locked safe
£100,000 - Business interruption - Additional Increased Cost of Working
£100.00 - Money and personal accident assault - Premium excluding IPT
£100.00 - Goods in transit - Premium excluding IPT
£1,338.12 - Business interruption - Premium excluding IPT100%

Accuracy level attained: 100%

Model 4:

Output:

£1,000 - Money - on premises outside business hours in locked safe
£100,000 - Business interruption - Additional Increased Cost of Working
£100.00 - Money and personal accident assault - Premium excluding IPT
£100.00 - Goods in transit - Premium excluding IPT
£1,338.12 - Business interruption - Premium excluding IPT100%

Accuracy level attained: 100%

Model 5:

Output:

£1,000 - Money - on premises outside business hours in locked safe
£100,000 - Business interruption - Additional Increased Cost of Working
£100.00 - Money and personal accident assault - Premium excluding IPT
£100.00 - Goods in transit - Premium excluding IPT
£1,338.12 - Business interruption - Premium excluding IPT100%

Accuracy level attained: 100%

Model 6:

Output:

£1,000 - Money - on premises outside business hours in locked safe

£100,000 - Business interruption - Additional Increased Cost of Working
£100.00 - Money and personal accident assault - Premium excluding IPT
£100.00 - Goods in transit - Premium excluding IPT
£1,338.12 - Business interruption - Premium excluding IPT100%

Accuracy level attained: 100%

Model 7:

Output:

£1,000 - Money - on premises outside business hours in locked safe
£100,000 - Business interruption - Additional Increased Cost of Working
£100.00 - Money and personal accident assault - Premium excluding IPT
£100.00 - Goods in transit - Premium excluding IPT
£1,338.12 - Business interruption - Premium excluding IPT100%

Accuracy level attained: 100%

Model 8:

Output:

£1,000 - Money - on premises outside business hours in locked safe
£100,000 - Business interruption - Additional Increased Cost of Working
£100.00 - Money and personal accident assault - Premium excluding IPT
£100.00 - Goods in transit - Premium excluding IPT
£1,338.12 - Business interruption - Premium excluding IPT100%

Accuracy level attained: 100%

From the above results it is clear to see that the models deal much better when processing a small amount of text at a time. This was a discovery that was made late in the project and will be something I will investigate further in the coming months. My plan is to take the text I extract from each document and break it down into a list of sentences, from there I will write some code to iterate through the sentences and extract the entities from each sentence. So far, I have performed some small tests using this method, and the results look very promising, as illustrated above.

Chapter 5

Conclusion and Future Work

After completing the test on the various trained models and discussing the results I am now able to conclude that this dissertation has been successful in achieving the desired outcome. This view has been solidified by an offer of employment with the insurance company to continue working on the development of the data extraction software. I have pointed out during the dissertation that there will be future work to develop a web-based application to act as a user interface for the functionality of the data extraction software. In this section I will make my conclusions on the work covered in this dissertation and discuss future work in more detail.

5.1 Conclusion

The aim of this dissertation was to develop a piece of software to automate the extraction of key information from various insurance PDF documents through the use of natural language processing and machine learning. During the evaluation of the software I was able to show results ranging from 42% to 82% accuracy when extracting named entities from a full document using trained model with various settings, and 100% accuracy across all eight models when testing them on five sentences, one sentence at a time. I was able to show that training a spaCy named entity recogniser to identify the custom named entities proved to be the most effective way to extract the required information in a way that requires little maintenance and supervision. It was shown that the other methods which I explored, like using SpaCy's Matcher to match sequences of tokens, using regular expression and extracting spans from doc object, would be able to complete the task, however these methods would require a large amount of coding for each specific document type and there would be a requirement to check that the insurance documents format and wording were not changing, as this would result in the extraction of incorrect information. Given the timeframe to write this dissertation, there was not enough time to test the data extraction software using a much larger set of training data. I believe that the software could reach close to 100% accuracy, on full documents, when extracting information once I am able to provide it with a large dataset with around 50,000 examples. During the course of this dissertation I have explained the various problems I came across during development of the software, including the difficulty faced when trying to extract text from PDF documents which are not machine made, the catastrophic forgetting problem faced when training a pre-existing spaCy model and the early mistakes I made when creating training data. It was shown that there are companies out there with products that have the ability to extract the required information, however there was only one company I could find that are doing it in a similar manner, through the use of natural language processing and machine learning. I feel that this dissertation has answered my research question (RQ1) sufficiently, by showing the various methods that could be used to achieve the required functionality, and the levels of accuracy achieved during the evaluation.

5.2 Future Work

As a direct result of the work on this project I have been asked to continue working on the data extraction software on a full-time basis for the foreseeable future. At this moment in time I have a few key areas that I aim to start work on, which are as follows:

1. Expand the SpaCy Training Data

I will be spending some time putting together a comprehensive amount of training data to train a future model. This process will involve selecting key sentences from the insurance documents and tagging the named entities within each one. As things stand this is a time-consuming manual process, however there may be ways in which I will be able to automate the training data creation process, which I will be looking into throughout the coming weeks.

2. Research into Alternative Technology for PDF Text Extraction

Within the last few days we have been looking into ways of extracting the PDF text without the need of optical character recognition, in a way that will be more accurate and also maintain the structure of the text. There is a piece of software available at the moment called GhostScript which does exactly that, the txtwrite device within GhostScript will extract the text from a PDF document accurately and keeping the structure. This option requires more research as there may be licensing issues that need to be considered before deciding if this option is feasible.

3. Develop the Web Application in Django

I have touched on this point during this dissertation because it is a key phase in the development of this software and will enable the software to be integrated with existing systems. Django is a web app development framework which uses a mix of Python, JavaScript, CSS and HTML. It follows the Model, Template, View (MTV) architecture, which is very similar to Model, View, Controller (MVC). Django allows for rapid database driven web app development through its use of python throughout. Django also provides an administrative create, read, update and delete (CRUD) interface which is created very easily using the provided admin controls. This will enable me to easily create a user interface for the data extraction software and integrate it with the existing digital broker web application that was also created using Django.

Appendix A

The table below shows the descriptions for the various labels that can appear on the dependency tree.

Table 9: Dependency tree labels

Label	Description	Label	Description
acl	relative clause	expl	expletive
advcl	adverbial clause	foreign	foreign words
advmod	adverbial modifier	goeswith	typo
amod	adjectival modifier	iobj	indirect object
appos	apposition	list	list
aux	auxiliary	mark	subordinating conjunction
auxpass	passive auxiliary	mwe	multiword expression
case	adposition	name	multiword named entity
cc	coordinating conjunction	neg	negation word
ccomp	clausal complement	nmod	nominal modifier
compound	compound	nsubj	subject
conj	coordinated word	nsubjpass	passive subject
cop	copula	nummod	numeric modifier
csubj	clausal subject	parataxis	parataxis
csubjpass	clausal passive subject	punct	punct
dep	dependency	remnant	argument in ellipted sentences
det	determiner	reparandum	repairs
discourse	discourse element	root	root
dislocated	dislocated element	vocative	vocative
dobj	object	xcomp	open clausal complement

Appendix B

Below are the class diagrams for the four main classes of the data extraction software.

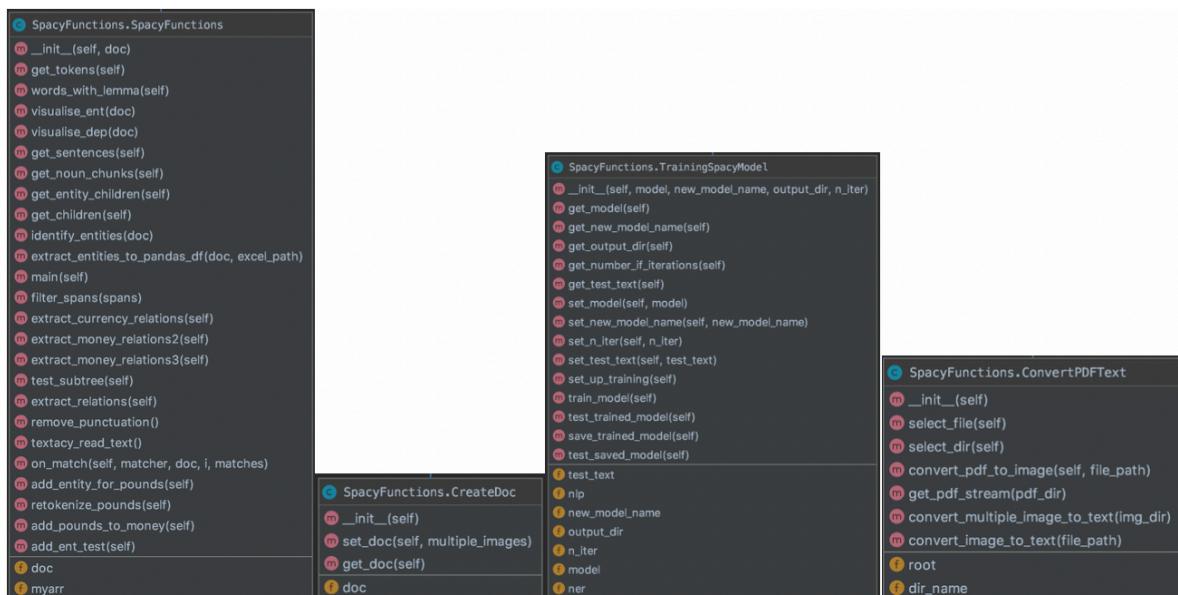


Figure 10: Class diagrams

References

- [1] Docparser 2019, Features, Viewed 18 Aug 2019, < <https://docparser.com/features/>>
- [2] Adeptia 2019, Data Extraction from Unstructured PDF Files, Viewed 18 Aug 2019, < <https://adeptia.com/blog/data-extraction-unstructured-pdf-files/>>
- [3] Chisel.ai 2019, Data Extraction, Viewed 18 Aug 2019, <<https://www.chisel.ai/data-extraction/>>
- [4] Natural Language Processing with Python (Bird, Klein and Loper, 2009, Page ix)
- [5] Natural Language Processing with Python (Bird, Klein and Loper, 2009, Page 262)
- [6] Chunking with Support Vector Machines (Kudo and Matsumoto, 2001, Page 1)
- [7] SpaCy 2019, Models, Viewed 18 Aug 2019, < <https://spacy.io/models/>>
- [8] SpaCy 2019, Models, Viewed 18 Aug 2019, < <https://spacy.io/models/>>