12-2021

Using Custom NER Models to Extract DOD Specific Entities from Contracts

Kayla P. Haberstich

USING CUSTOM NER MODELS TO EXTRACT DOD SPECIFIC ENTITIES FROM CONTRACTS

THESIS

Kayla P. Haberstich, Contractor, USAF

AFIT-ENS-MS-21-D-030

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENS-MS-21-D-030

USING CUSTOM NER MODELS TO EXTRACT DOD SPECIFIC ENTITIES FROM CONTRACTS

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Kayla P. Haberstich, BS

Contractor, USAF

December 2021

DISTRIBUTION STATEMENT A.
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT-ENS-MS-21-D-030

USING CUSTOM NER MODELS TO EXTRACT DOD SPECIFIC ENTITIES FROM CONTRACTS

Kayla P. Haberstich, BS
Contractor, USAF

Committee Membership:

Dr. Jeffery D. Weir
Chair

LTC Phillip LaCasse
Reader

AFIT-ENS-MS-21-D-030

Abstract

The Air Force has executed millions of contracts over the years. These contracts have a plethora of information in them, but that information needs to be extracted from each contract into a useable form to perform further analysis. This thesis focuses on being able to extract information from those contracts in a quick and repeatable way through the use of regular expression, commonly known as Regex, or standard named entity recognition (NER) models for the baseline analysis. For the exploratory analysis, custom NER models are used and then the results are compared to the output from the baseline analysis. This paper focuses on four specific entities to be extracted including National Stock Number (NSN), Part Number, Commercial and Government Entity (CAGE) Code, and Supplier Name but the methods in this paper can be extended for other entities in these contracts.

Results show that NSNs are extracted similarly by both the Regex and custom NER models. The consistent pattern of NSNs is what Regex thrives on so the uses of custom NERs does not improve the ability to extract correct NSNs from the contracts. NER did improve the model's ability to pick up NSNs that were inputted slightly wrong, thus adding to the model's extraction power. The F1 score of the part number entity improved from 3.2% with the Regex model to 95% with the use of custom NER while the supplier name entity F1 score improved from 5.9% using the Regex approach to 63.6% with the used of custom NER over the standard SpaCy NER "org" entity. An additional set of suppliers was collected through the use of the CAGE Code. There is no standard NER model to capture these as it is specific to the DoD and simply being 5 alpha-numeric characters Regex cannot distinguish CAGE Codes in free form text. When combining the CAGE Codes and supplier names extracted using custom NER models, the F1 score increases to 86.6%. The accuracy of the models should continue to increase as annotations are added.

This newly extracted information will allow the Air Force to identify what parts are supplied by which vendors. This information along with historical pricing for the vendor specific part number can give decision makers the ability to negotiate pricing based on historical data and competitor pricing. In addition to just pricing, part numbers can be aligned with maintenance data to make informed decisions on which vendor to go with by analyzing life cycle costs of a part.

Acknowledgements

I would like to express my sincere appreciation to my faculty advisor, Dr. Jeffery Weir, for his guidance throughout the course of this effort. I would also like to thank coworker Bradley Guthrie for his knowledge and assistance through this effort.

Kayla Haberstich

Table of Contents

List of Figures

List of Tables

USING CUSTOM NER MODELS TO EXTRACT DOD SPECIFIC ENTITIES FROM CONTRACTS

I. Introduction

Background

The amount of data available is ever increasing and most of it is captured in an unstructured textual format. To glean the most information out of this data that is available analysts must structure the data into a format that is more easily useable. This is where text mining comes into place. Text mining deals with unstructured data found in documents, emails, social media, and the web. Thus, the difference between regular data mining and text mining is that in text mining the patterns are extracted from natural language text rather than from structured databases of facts (Hassani, Beneki, Unger, Mazinani, & Yeganegi, 2020). To extract these patterns from natural language, a process called natural language processing (NLP) is used.

Natural language processing is a subset of text mining that is a set of methods for making human language accessible to computers. These methods help people every day with things such as automatic machine translation, classifying text in emails that flags it as spam or lets it through to our inbox, a higher sophisticated capability of search engines, and many more. These diverse NLP applications all have a common set of ideas, drawing on algorithms, linguistics, logic, statistics and more (Eisenstein, 2019). Searching for a specific entity within a natural language landscape gets into a subset of NLP called named entity recognition (NER).

NER is a fundamental task in natural language processing due to the fact that the named entities often convey the key information of the text. Entities are the subject of interest in a NER model that are targeted to be extracted (Lample, Ballesteros, Subramanian, Kawakami, & Dyer, 2016). This can be done by either a standard NER model or a custom NER model. See Chapter II: Literature Review for a more in depth explanation of standard vs. custom NER models and how these techniques can be leveraged in the Department of Defense (DoD) as well as all of the other methods described above.

Problem Statement

There are hundreds of thousands of contracts executed each year by the Air Force. There is no standardized format for these contracts; there are six different front form formats that are consistently used throughout the contracts in the data set but the rest of the contract has no standard form. Thus, the extraction of useful information from this unstructured data can be a difficult and time-consuming process.

Contract front forms contain a designated location for the supplier name and associated CAGE (Commercial and Government Entity) code where national stock number (NSN) and part number usually resides in the contract line item number (CLIN) text or in other areas throughout the contract such as rights assertion (RAT) tables and government furnished property (GFP) tables. These two types of tables are outside the scope of this analysis but are crucial in understanding the full picture of the parts data. They will be discussed more in Chapter V with how parts can be extracted from these tables in future efforts.

The supplier name and CAGE code can also reside in the CLIN text. The CLIN text contains a free text field where the user can write in additional comments about the specific line item. When the desired information resides in this field, additional extraction techniques are needed to pull it out of the form and make sure the information is still captured accurately.

A lot of the data in these contracts is specific to the DoD, thus making standard models a less reliable approach. Information such as the supplying organization can be very specific. A commercial supplying organization, such as Boeing, would be captured in the standard NER model while a DoD specific supplying organization, such as 88th Air Base Wing, would not get recognized by the Wikipedia trained model. Thus, a custom NER model is needed to accurately capture the entities of interest. In addition to supplying organization, part numbers, CAGE codes, and national stock numbers (NSNs) are specific to the DoD as well.

The national stock number is the official label applied to an item of supply that is repeatedly procured, stocked, stored, issued, and used throughout the federal supply system. It is a unique, item-identifying series of numbers separated by dashes. NSNs are an essential part of the military's logistics supply chain used in managing, moving, storing, and disposing of material (Defense Logistics Agency, 2019). The NSNs are the same regardless of the supplier. Each supplier that makes a specific part will have different part numbers that correspond to the same NSN. Thus, there can be an NSN with multiple part numbers associated with it, but not a part number that corresponds to multiple NSNs. Once these three entities are extracted from the contracts, there is potential for further analysis that will be described in the Recommendations section of Chapter V.

In this paper a custom knowledge base for the supplying organization is created using a custom NER model and improvement is quantified as compared to the standard supplier list. In addition, custom NER models are implemented to be able to extract the Cage Code, NSN and part number from these contracts and see how much better it performs at capturing these entities than when using the Regex approach.

Research Questions

1. Can more Suppliers successfully be extracted using a custom NER model as opposed to the standard SpaCy "org" entity model?
2. Can more NSNs successfully be extracted using a custom NER model as opposed to a regex model?
3. Can more Part Numbers successfully be extracted using a custom NER model as opposed to a regex model?

Scope and Limitations

This research seeks to extract information such as the supplying organization, CAGE Code, national stock number (NSN), and part number out of parts contracts. As such, the format and information available in these contracts could be significantly different from other types of contracts in the corporate world as well as other areas of the DoD. This could make the reliability of methods discussed in this paper non-applicable to other types of contracts depending on the format and what is to be extracted. That being said, the methods of this paper may still be leveraged for different entities on different types of contracts, but the success of the information extraction may vary by contract type and entity being extracted.

Thesis Overview

In the next section, Chapter 2, relevant methods of data extraction are reviewed as well as previous work that has been done in the world of contracts data extraction and relevant machine learning algorithms. Chapter 3 outlines the data and methodology of the research while chapter 4 contains results of the analysis as well as significant findings. The last chapter, Chapter

5, summarizes the preceding chapters, states the relevance of the findings, and presents potential ideas for future research in this area.

II. Literature Review

Chapter Overview

This section explores the overarching field of text mining and its components, different machine learning algorithms used in natural language processing, and how this can be beneficial when applied to contracts data. First, text mining and its components are explored that will be of interest later in this paper such as natural language processing (NLP) and named entity recognition (NER). Both standard NER and custom NER will be discussed. Next, some Python libraries are reviewed, such as SpaCy, Carrot2, NLPKT, Prodigy, and PyLighter, that can assist in building a NER model. Then, an overview of machine learning and some specific ML algorithms that will be of interest later in this paper such as transformers, convolutional neural networks (CNN), and bloom embeddings will be discussed. This section finishes off with popular evaluation metrics for the NER model such as accuracy, precision, recall, and F1 score.

Text Mining

According to Statista, the amount of data created, captured, copied, and consumed globally reached an all-time high of 64.2 zettabytes in 2020. This is up from 2 zettabytes just 10 years prior in 2010. It is expected that over the next five years, the amount of data consumed in the world will grow to over 180 zettabytes yearly by the year 2025 (Statista, 2021). As the amount of data collected is ever increasing, analysts need a way to turn these numbers and text into something meaningful, and text mining is the way to do that. Text mining is the process of transforming unstructured and semi-structured text into a structured format to identify meaningful patterns and new insights. It is typically used in instances where there is a need to process large volumes of text-based data for insights but would otherwise be too resource and time-intensive to be analyzed manually by humans (Chen, 2020).

Text mining is a broad, overarching technique that encompasses many different areas of analytics such as document classification, document clustering, information extraction, natural language processing (NLP) concept extraction, web mining, information retrieval, and others. Figure 1 shows the relationship between text mining and other related fields.

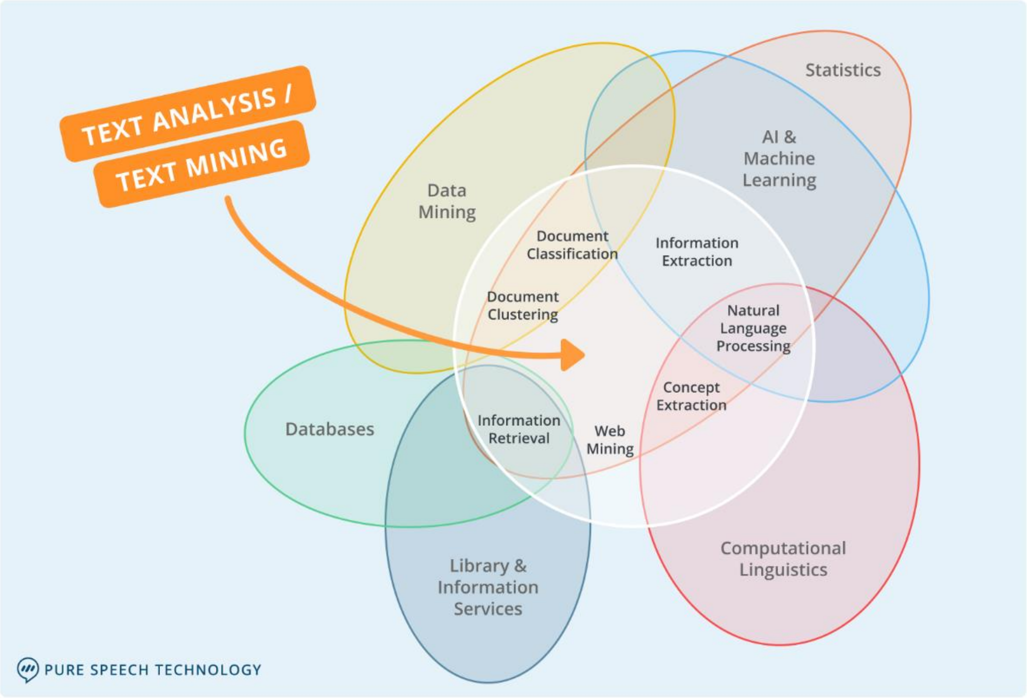


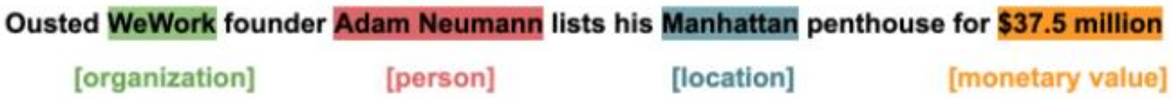
Figure 1: Text Mining Venn Diagram (Chen, 2020)

Natural Language Processing

Within text mining, there is a component called Natural Language Processing (NLP). NLP is a field of computer science and engineering that has developed from the study of language and computational linguistics within the field of Artificial Intelligence (AI). The goals of NLP are to design and build applications that facilitate human interaction with machines and other devices through the use of natural language. Some of the major areas of NLP include question answering systems, summarization, machine translation, speech recognition, and document classification (Stubbs & Pustejovsky, 2013). The NLP algorithms need a consistent knowledge base to be able to execute the classification of words and phrases in the text. There are predefined knowledge bases out there for use in a standard NER model, or a knowledge base can be built for a more specific purpose for use in a custom NER model.

Named Entity Recognition (NER)

Named entity recognition (NER) – also called entity identification or entity extraction – is a NLP technique that automatically identifies named entities in a text and classifies them into predefined categories. Entities can be names of people, organizations, locations, times, quantities, monetary values, percentages, and more (Roldos, 2020). Depending on the data at hand a standard knowledge base may be appropriate to capture the entities you are looking for in the text. See Figure 2 below for an example of standard NER entities.



Ousted WeWork founder Adam Neumann lists his Manhattan penthouse for $37.5 million

Figure 2: Standard Entities for NER (Roldos, 2020)

If there are special cases of the entities of interest, a custom knowledge base could be more successful at correctly extracting the information needed. The next two sections explore the differences between standard and custom NER models and gives examples of when each would be most relevant to use.

Standard Named Entity Recognition Model

Standard NER models seek to capture information that already has a known Wikipedia trained knowledge base. It is the problem of finding the members of various predetermined classes, such as person, organization, location, date/time, quantities, numbers, etc. (Goyal, Gupta, & Kumar, 2018). From the example in Figure 2 above, it is seen that the standard NER model can pick up that “WeWork” is an organization, “Adam Neumann” is a person, “Manhattan” is a location, and “$37.5 million” is a monetary value.

For many purposes, a standard model can accurately capture the entities as they are intended. It is still important to note that unique variations on these entities might not be picked up by a standard NER model. The unique supplier names in DoD are a prime example of when a standard model is not sufficient in picking up the intended entity with sufficient accuracy. When the standard model does not accurately pick up the entities of interest well, a custom NER model should be explored.

Custom Named Entity Recognition Model

One of the most limiting factors of NER is a fixed number of classes of entities that are currently available (Stepanyan, 2020). Custom NER models are used mainly when a standard NER model is not available or not sufficient at extracting the entities of interest accurately enough. This can be due to special cases of common entities or needing to extract entities that are not present in a standard model. However, developing a corpus of custom named entities (CNE) is a cumbersome task requiring an annotated dataset (Stepanyan, 2020). In the sections below, NER packages that can help with speeding up the annotation process will be explored.

Once the annotations are complete, the NER model then gets trained on the annotated corpus of data.

For example, dates used in the DoD vary from how the rest of the world formats dates. This special case of a common entity can make it hard for the NER model to pick up that information. As well as dates, organizations are not standard in the DoD either. Some supplying organizations, such as Boeing, a standard NER model would be able to recognize but the unique suppliers to the DoD, such as 88th Air Base Wing, would not be recognizable as an organization to the model. In addition to special cases of common entities, there are some entities that standard models just do not have. Information such as NSN and Part Number, which are entities of interest in this analysis, do not exist in standard models as they are specific to the DoD. Cases such as these are where it is most beneficial to have a custom NER model help more accurately capture some entities that are not as mainstream as the ones included in a standard NER model.

NER Tools

There are many tools available to help analysts build named entity recognition models. Some of the more popular open source, code-based packages to perform NER are StanfordNER, OpenNLP, GATE, Spacy, and NLTK. StanfordNER, OpenNLP, and GATE are available for use in Java while Spacy and NLTK are available in Python. For this analysis, packages available in Python are the focus because the contracts data needs to be kept on the HPC system for privacy reasons and Python is readily available for use on the HPC system. For more information on the Java packages see (Vychegzhanin & Kotelnikov, 2019) and (Schmitt, Kubler, Robert, Papadakis, & LeTraon, 2019).

In addition to these code-based approaches, there are some newer tools available to allow analysts to annotate data in a graphical user interface (GUI) format. A lot of these tools require you to download the software to your computer, thus giving up the privacy of your data. Two packages that were found that allow the analyst to use the GUI format of annotating data as well as keep the data secure are Prodigy (a subset of Spacy) and PyLighter. These tools will be discussed in more detail below.

SpaCy

SpaCy is an Explosion AI product that is an open-source library for NLP in Python. It comprehends and delineates the text (either small or large) by processing the same. Moreover, SpaCy provides a wide range of in-built features which makes it an efficient tool for text processing and language modeling. SpaCy selects the best algorithm on its own, using an object-oriented approach, saving the analyst time in not having to assess each algorithm by hand. This is a main benefit over the competing NLTK where you have to manually select the best algorithm (Jugran, Kumar, Tyagi, & Anand, 2021).

SpaCy has been used in recent DoD contract extraction efforts. Butcher 2021 uses SpaCy to extract the organization fulfilling the contract, the effective date of the contract, and the expiration date of the contract. These were found by using the “organization” and “date” tags that come standard in the SpaCy package. 40.5% of the supplying organizations were successfully extracted while 60.1% of the start date and 31.9% of the end dates were extracted successfully from the contracts (Butcher, 2021). He suggests that building a custom NER model and combining it with classification models to produce more accurate results in less time. Butcher points out that this labeling of thousands of contracts is very time consuming, thus two libraries are explored below (Prodigy and PyLighter) that are in a GUI format that can help decrease the time needed to label a corpus.

NLKT

Natural Language Toolkit (NLTK) is a Python library that is used for processing text string by string. The input and output using NLTK is the sequence of characters i.e, string. Providing several options for various algorithms for a particular problem is one of the specialties of this tool, but it sometimes tends to be tedious and time consuming to select and work accordingly (Jugran, Kumar, Tyagi, & Anand, 2021). Below in Table 1, a comparison between the features that are available in NLTK and SpaCy are shown. From the features that Jugran et. al. explored; it is shown that SpaCy has double the capabilities than NLTK.

Table 1: Comparison of SpaCy and NLTK (Jugran, Kumar, Tyagi, & Anand, 2021)

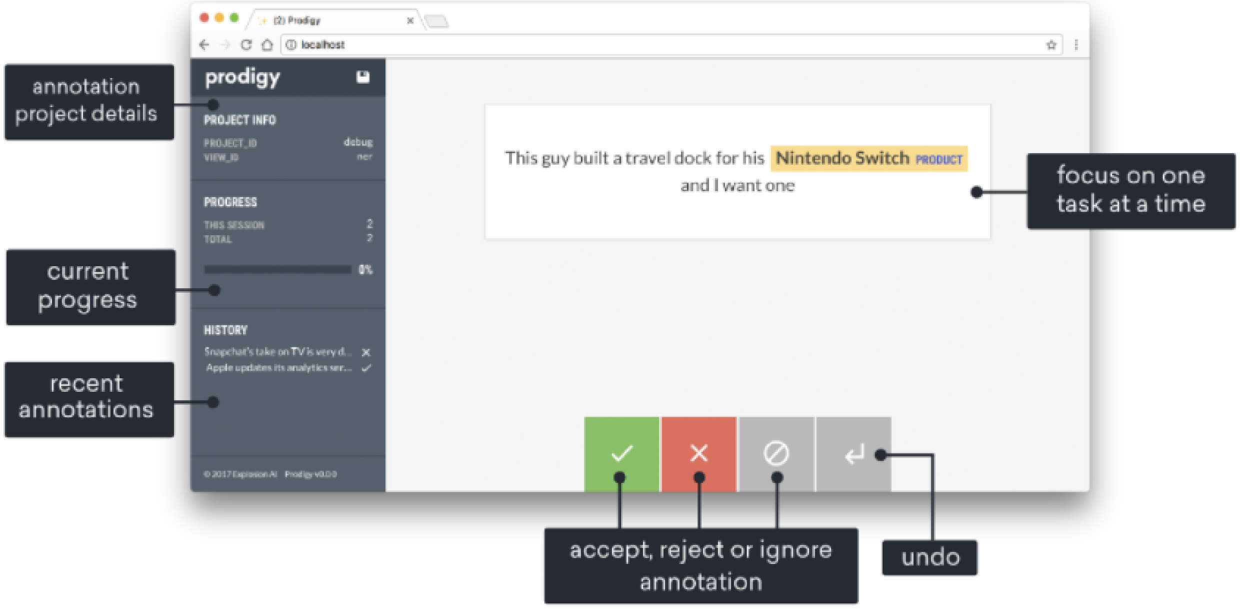
Features/Packages	**NLTK**	**SpaCy**
Classifier	Yes	Yes
Topic Modeling	No	Yes
Vectorization	No	Yes
Tokenization	Yes	Yes
Parsing	Yes	Yes
TF-IDF	No	Yes

Prodigy

Prodigy is under the same parent company as SpaCy, Explosion AI. Although Prodigy is not free, it is an annotation tool so efficient that data scientists can do the annotation themselves, enabling a new level of rapid iteration (ExplosionAI, 2020). The software automatically labels entities based on input you give it from the model initially. It only asks the analyst to annotate examples that the model is uncertain about, saving you time not having to label entities the software is confident about having labeled correctly (ExplosionAI, 2020). When a new task pops up, the user simply hits the accept, reject, or ignore buttons seen below in Figure 3. This tells the model the annotation it gave was correct, wrong, or skipped. The model then learns from the input it receives to classify the entities more accurately in the future.

In addition to named entity recognition, prodigy can be used for text classification, image classification, and free form text analysis like translations from one language to another. These use cases for Prodigy also come in a nice GUI format that is easy to use and allows the analyst to quickly accept, reject, or adjust the text or image classification that Prodigy initially gives to the data.

Prodigy was initially the option that was chosen to go ahead with for building a custom NER model of NSN, Part Number, and Supplier. However, the software was unable to be purchased so other options needed to be explored. This led to some additional research to other alternatives for Prodigy that could be used on the HPC. In the next section PyLighter is discussed as a further option. PyLighter is another GUI based annotation tool that can be used on the HPC and was the alternative that was chosen for this analysis.



(2) Prodigy
localhost
prodigy
PROJECT INFO
PROJECT_ID debug
VIEW_ID ner
PROGRESS
THIS SESSION 2
TOTAL 2
0%
HISTORY
Snapchat's take on TV is very d... ✗
Apple updates its analytics ser... ✓
© 2017 Explosion AI Prodigy v0.0.0
annotation project details
current progress
recent annotations
focus on one task at a time
accept, reject or ignore annotation
undo

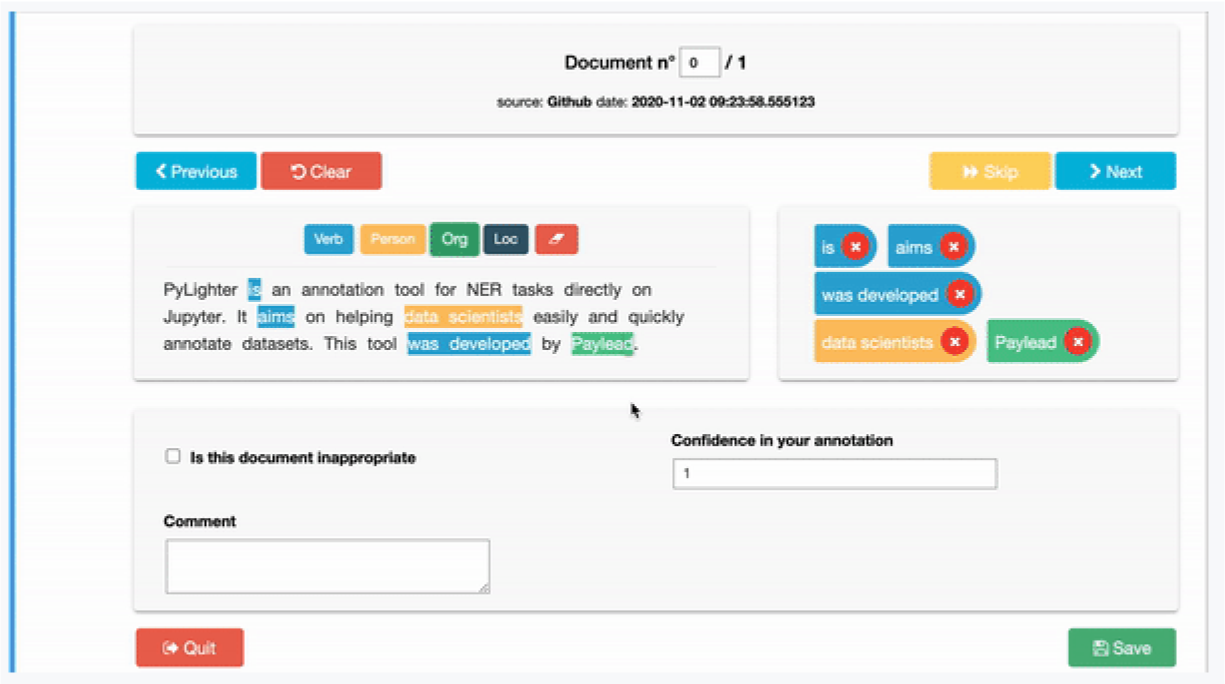
This guy built a travel dock for his Nintendo Switch PRODUCT and I want one

Figure 3: Prodigy GUI (Explosion AI, 2020)

PyLighter

PyLighter is an open-source annotation tool that can be installed and ran right from Python on the HPC system. It brings an annotation tool for NER tasks to the most used platform in data science: Jupyter (PayLead, 2020). It allows data scientists to annotate any corpus of documents with ease in a customizable fashion. PyLighter does not require any setup or installation of any kind other than just installing the package into the designated user space. Moreover, data scientists don't need to set up a pipeline. PyLighter is built for use in Jupyter, giving data scientists the ability to freely manage their data and quickly use their freshly annotated data to train their machine learning models (PayLead, 2020). This is ideal for the data that is being worked with in this analysis because it must be secured on the HPC and not launched from an outside program.

PyLighter is slightly different from Prodigy in that the entities are not initially highlighted. The entities of interest are shown at the top of the text in question in their respective colors. This can be seen by looking at the Verb, Person, Org, and Loc buttons below in Figure 4. You can highlight an entity in the text by clicking at the start of the entity and clicking again at the end of the entity you wish to highlight. This brings the highlighted word or phrase over to the box on the right that stores the entities you confirm for the given text. Once you are ready to move on to the next set of text, click the next button in the upper right corner. After all the documents are annotated, you save the model by clicking the save button in the lower right-hand corner.



Document n° 0 / 1

source: Github date: 2020-11-02 09:23:58.555123

PyLighter is an annotation tool for NER tasks directly on Jupyter. It aims on helping data scientists easily and quickly annotate datasets. This tool was developed by Paylead.

☐ Is this document inappropriate

Confidence in your annotation: 1

Comment:

Figure 4: PyLighter GUI (PayLead, 2020)

Once the annotations are complete, PyLighter saves them to a .csv file in your user space. The annotations are in the format of the CLIN text in the first column and the labels in the second column. Labels are done by character with an "O" indicating the character is not part of an entity, a "B-Entity" for the beginning of the entity, and "I-Entity" for a character that is associated with the entity but not first character of the entity. SpaCy does not accept the annotations in this format, so some post processing must be done to turn the annotations into something SpaCy can ingest. The end result is a .json file that includes the CLIN text along with the starting position and ending position of the entity along with the entity name. This file is used to train the custom NER model using machine learning techniques that will be discussed below.

Machine Learning Algorithms

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy (IBM Cloud Education, IBM Cloud Learn Hub, 2020). In machine learning, the aim is to construct a program that fits the given data. A learning program is different from an ordinary computer program in that it is a general template with modifiable parameters, and by assigning different values to these parameters the program can do different things. The learning algorithm adjusts the parameters of the template—which we call a model—by optimizing a performance criterion defined on the data (Alpaydin, 2016).

There are many of different types of machine learning out there, but largely there are three major recognized categories: supervised learning, unsupervised learning, and reinforcement learning (Heidenreich, 2018). Supervised machine learning is defined by its use of labeled datasets to train algorithms that classify data or predict outcomes accurately. As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately, which occurs as part of the cross-validation process (IBM Cloud Education, Supervised Learning, 2020). This varies from unsupervised machine learning as unsupervised learning does not require a known response variable. Unsupervised learning instead uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention (IBM Cloud Education, Unsupervised Learning, 2020). The last major type of machine learning is reinforcement learning. Reinforcement learning is the training of machine learning models to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment. In reinforcement learning, an artificial intelligence faces a game-like situation. The computer employs trial and error to come up with a solution to the problem. To get the machine to do what the programmer wants, the artificial intelligence gets either rewards or penalties for the actions it performs. Its goal is to maximize the total reward (Osinski & Budek, 2018). See Figure 5 below for a graphical representation and quick synopsis of what each ML technique is.

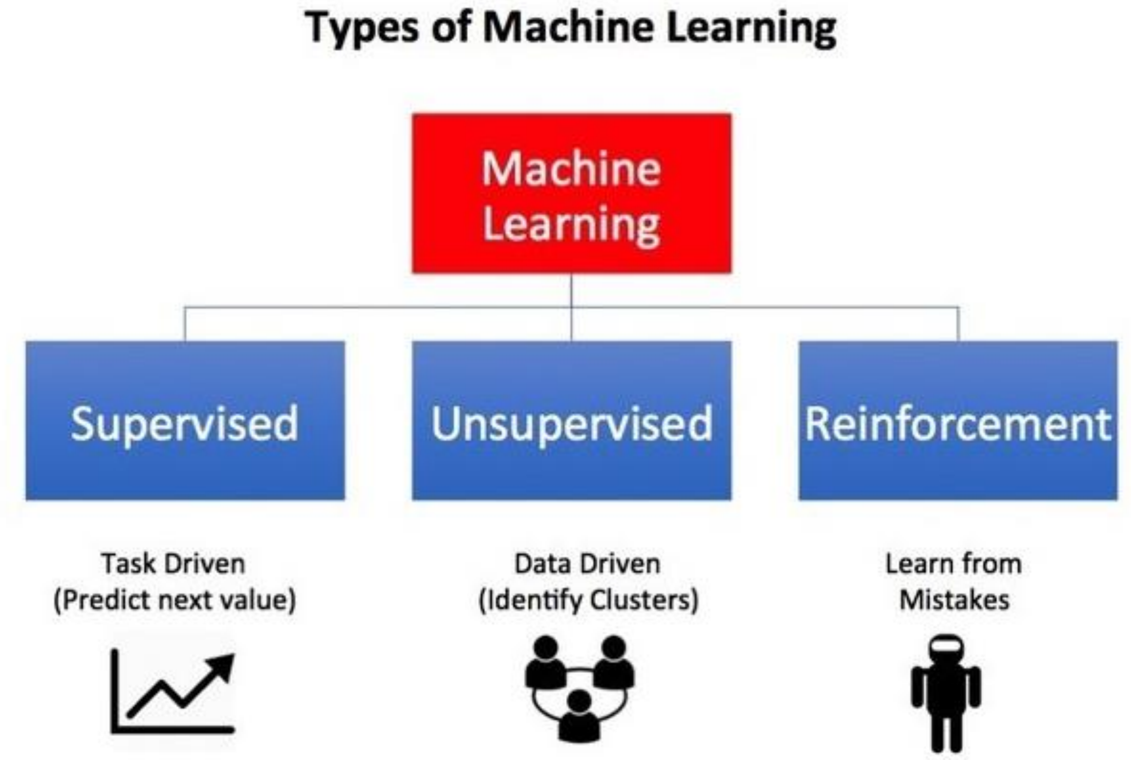

Types of Machine Learning
Machine Learning
Supervised
Unsupervised
Reinforcement
Task Driven (Predict next value)
Data Driven (Identify Clusters)
Learn from Mistakes

Figure 5: Machine Learning Types (Heidenreich, 2018)

SpaCy uses various different ML algorithms, all unsupervised and semi-unsupervised in nature, depending on the version of SpaCy that is being used. SpaCy 2.0 uses incremental parsing with Bloom embeddings and residual Convolutional Neural Networks (CNNs) (Honnibal, SpaCy, 2017) while SpaCy 3.0 uses transformer-based pipelines (Honnibal, Montani, Van Landeghem, & Boyd, 2021). These ML techniques will be discussed in further detail in the sections below in regard to what they bring to SpaCy as well as a general overview of the technique themselves.

Bloom Embeddings

SpaCy v2.0's Named Entity Recognition system features a sophisticated word embedding strategy using subword features and "Bloom" embeddings, a deep convolutional neural network with residual connections, and a novel transition-based approach to named entity parsing. The system is designed to give a good balance of efficiency, accuracy and adaptability (Honnibal, SpaCy, 2017).

At a high level, bloom embeddings are a compression technique that can be applied to the input and output of neural network models. They are computationally efficient, reducing training and prediction times as well as help save space (Serra & Karatzoglou, 2017). This is done by reducing the number of vectors necessary to store the vocabulary that the model needs. The normal table gets hashed down to a smaller number of rows and the words get mapped into an arbitrary integer – called a “hash value” (Honnibal, Chapter 4: Compact word vectors with Bloom Embeddings, 2018). In addition to Bloom Embeddings, SpaCy 2.0 incorporates CNNs into the models as well. This ML technique will be discussed below as well as how SpaCy combines them for use in NER models.

Convolutional Neural Networks

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, to provide an unmatched balance of speed, size, and accuracy. The new models are 10× smaller, 20% more accurate, and even cheaper to run than the previous generation (Explosion AI, 2017).

At a high level, CNNs are neural networks that make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network (Li, Krishna, & Xu, 2021). The difference between a regular neural network (left) and a convolutional neural network (right) can be seen below in Figure 6.

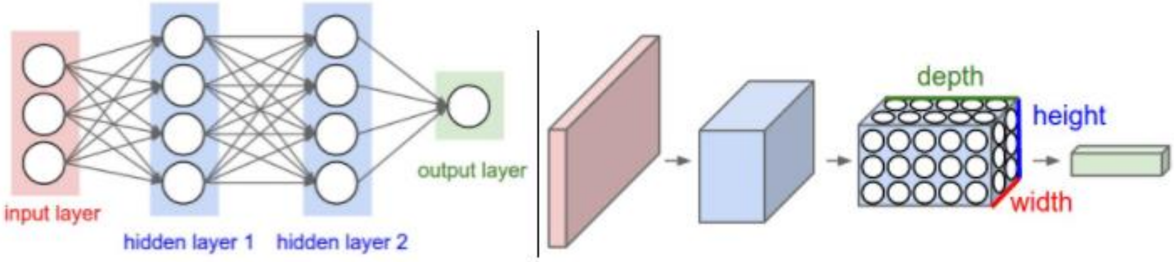


Figure 6: Regular vs. Convolutional Neural Networks (Li, Krishna, & Xu, 2021)

CNNs are combined with recurrent neural networks (RNNs), two of the most common neural network architectures used in deep learning, to create transformers (Lawton, 2021). Transformers are what SpaCy 3.0 uses (the version used for this analysis) and will be discussed in greater detail in the next section.

Transformers

SpaCy v3.0 features all new transformer-based pipelines that bring spaCy's accuracy right up to the current state-of-the-art, and a new workflow system to help you take projects from prototype to production. It's much easier to configure and train a pipeline, and there are lots of new and improved integrations with the rest of the NLP ecosystem (Honnibal, Montani, Van Landeghem, & Boyd, 2021). Two reasons why transformers perform significantly better are looked at below: combining CNN and RNN, as well as using attention.

As stated above, transformers combine the two most prominent neural nets used by researchers. Transformers look at all the elements (such as all the words in a sequence) at one time, while also paying closer attention to the most important elements in the sequence. Previous approaches could do one or the other, but not both. This gives transformers two key advantage over other models. First, they can be more accurate because they can understand the relationship between sequential elements that are far from each other. Second, they are fast at processing a sequence since they pay more attention to its most important parts (Lawton, 2021).

In addition to combining two of the most used models, the key to the transformer's ground-breaking performance is its use of attention. While processing a word, attention enables the model to focus on other words in the input that are closely related to that word. The Transformer architecture uses self-attention by relating every word in the input sequence to every other word (Doshi, 2020). In Figure 7 below it is seen that with the use of attention, the meaning of "it" can be differentiated in this sentence.

This addition is important in allowing the model to associate words/entities that are far apart from each other. There are multiple cases where the part numbers do not come directly after the phrase “Part number”, “P/N”, etc. and instead they come further along in the sentence. The use of transformers should help the model in associating the entities that are further apart from the key word. For example, some common CLIN text is “part numbers and quantities… (Part QTY) (Part QTY) (Part QTY).” The Regex based approach was not able to understand the part numbers came further after the phrase “part numbers.” The results of the NER model using transformers (SpaCy 3.0) significantly increases the performance of the model.

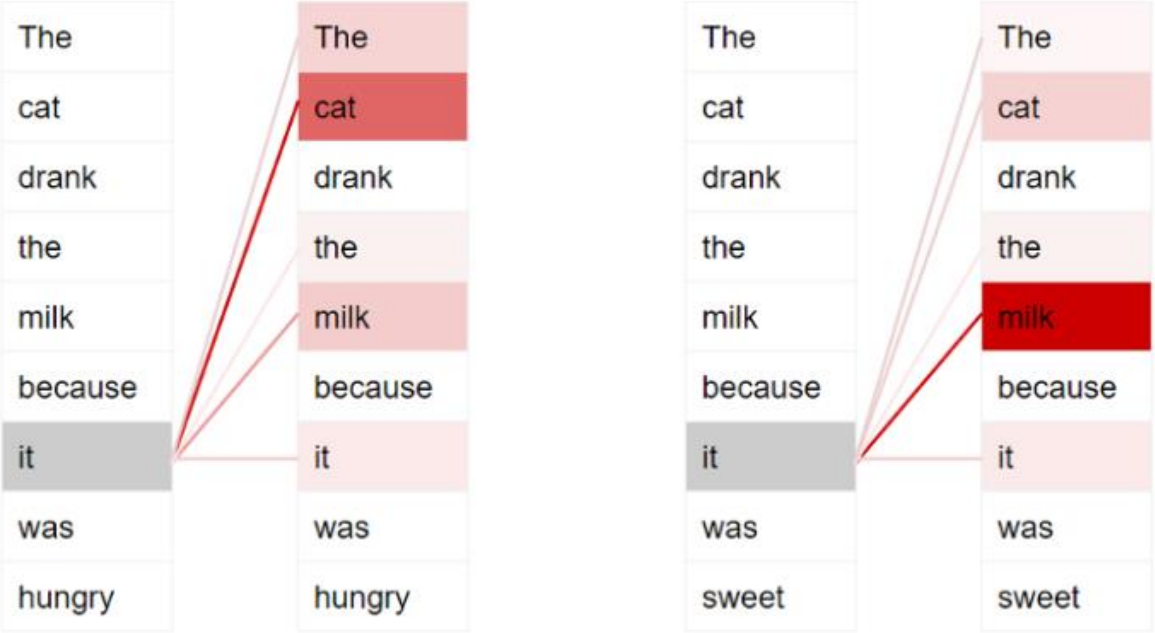


Figure 7:Attention Example (Doshi, 2020)

Once SpaCy trains the model with these ML algorithms that were discussed above, the performance of the output is evaluated. The evaluation metrics that will be used to assess the performance of the NER models will be discussed below.

NER Evaluation Metrics

Once the NER model is trained, it needs to be evaluated for performance. To evaluate the performance of the models, accuracy, precision, recall, and F1 metrics are looked at. These metrics are the most commonly used in the NLP community (Riggio, 2019). To understand the calculations behind precision and recall, it is important to first look at the metrics that are used to calculate them. These include true positives, true negatives, false positives, and false negatives.

True positives (TP) occur when the predicted and actual results are both positives, while true negatives (TN) occur when the predicted and actual results are both negatives. These are both the ideal situation: the model correctly predicts the classification. False Negatives (FN) occur when the model predicts the results as negatives, when they should have been positive. False Positives (FP) occur when the model predicts the results as positive, when they should have been negative (Riggio, 2019). This can be shown in tabular form below with the row representing an instance in the predicted class while the column represents the instance of the actual class.

Table 2: Classification Confusion Matrix

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Accuracy

Accuracy is the most widely know of the metrics, and in terms of machine learning models we usually talk about classification accuracy (Mishra, 2018). Classification accuracy simply captures the ratio of correct predictions to the total number of predictions. It works best when there is a similar number of samples belonging to each class.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions\ Made} = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision

Precision is a measure of how much information returned by the system is actually correct (Aryoyudanta, Adji, & Hidayah, 2016). This metric is most useful by itself when the costs of false positives are high. An example of this for our contract entity extraction would be the number of correct Suppliers the model picks up compared to the total Suppliers that are found.

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

Recall

Recall is a measure of how much relevant information has been extracted from the text (Aryoyudanta, Adji, & Hidayah, 2016). This metric is most useful by itself when the cost of false negatives is high. An example of this for our contract entity extraction would be the number of correct Suppliers the model picks up compared to the amount of contracts that contained Suppliers.

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

F1 Score

The F1 score takes both precision and recall into account to ultimately measure the accuracy of the model. It tries to consider the fact that false positives and false negatives can be absolutely crucial to the study while true negatives are often less important. Thus, the F1 score gives more weight to false negatives and false positives while not letting large numbers of true negatives influence the score (Riggio, 2019). It takes both a high precision and high recall to receive a high F1 score, making the F1 score the most valuable metric.

$$F1\,Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Chapter Summary

Named entity recognition is a powerful tool for data extraction and there are multiple different packages available to help analysts build NER models. This section looked at previous work that has been done in the NER realm and what tools they have used. A very similar contract text extraction effort and some of the recommended ways forward gleaned from that paper were also explored. The goal is to improve upon previous studies and successfully extract more information accurately with a custom NER model as opposed to using the Regex approach or standard NER models. The next chapter will discuss in greater detail the data that is being used for this research as well as the methodological approach to this analysis.

III. Methodology

Chapter Overview

This chapter goes into detailed information about the data and methodologies used to analyze the data. First, the source of the data, its characteristics, and the subset used for this analysis are looked at. Next, the Regex and NER methods are explained in greater detail with how they are applied to the contracts data for use in extracting NSN, Part Number, CAGE Code, and Supplying Organization. This section finishes off with the statistical methods that will be used to compare the Regex model to the NER model to see if better results are achieved with NER than with Regex, where applicable.

Data

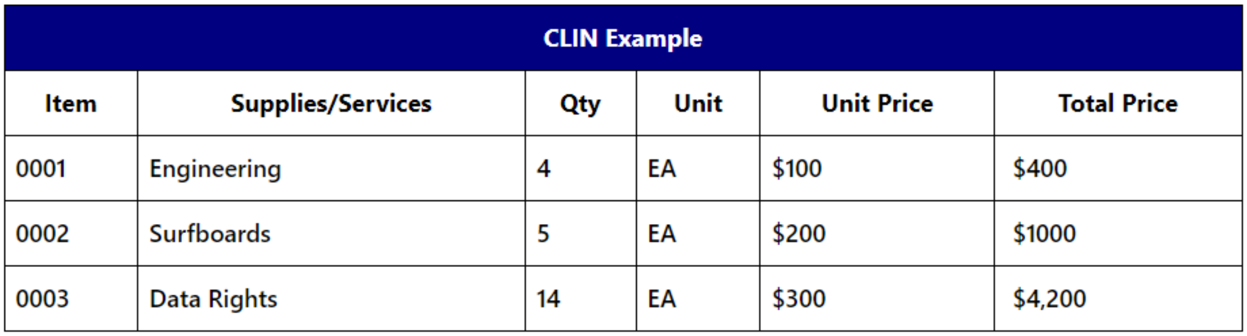
The Data Analytics Resource Team (DART) has access to approximately 3.7 million contracts with starting dates from the beginning of FY 04 to January of 2020. These contracts initially come in the form of pdf files, but the team has extracted the info from the files and converted them to text documents using optimal character recognition (OCR). There are more contracts available that are currently going through the process to be converted to text files that have starting dates in FY 21. The methods in this paper can be leveraged and rerun on the new contracts once they become available, and for future contracts that the team gains access to. The text version of the contracts is the format that is used for this analysis.

It is important to note that the NER model can only perform as accurately as the information that is extracted from the text. There are cases where the number 0 gets recorded as the letter O and some other instances where characters get misread by OCR. The NER model will still return the phrase as the intended entity, but this can cause issues later on when the entity is extracted and there are no matches for the given entity.

Within each contract there are contract line item numbers (CLINs). The purpose of CLINs within the contracts is to break the contract down by the commodities being procured and provide for traceable accounting classification citations (AcqNotes, 2021). An example of a CLIN can be seen below in Figure 8 and characteristics of a CLIN include:

- Single unit price/extended amount
  - Separately identifiable
  - Hardware: no more than one NSN, item description, mfg part #
- Services: no more than one scope or work/description of services
- Separate delivery schedule, a period of performance, or completion date
- Single accounting classification citation

Each CLIN within a contract has different information in it that makes up the entire contract, thus this analysis seeks to extract the information for this analysis on the CLIN level.



CLIN Example

Item	Supplies/Services	Qty	Unit	Unit Price	Total Price
0001	Engineering	4	EA	$100	$400
0002	Surfboards	5	EA	$200	$1000
0003	Data Rights	14	EA	$300	$4,200

Figure 8: CLIN Example (AcqNotes, 2021)

With there being 3.7 million contracts and 7.4 million CLINs available, the data must be scaled down to a useable subset. The subset of data that is used for this analysis is CLINs that

that are associated with Air Force Life Cycle Management Center (AFLCMC) work. This was accomplished by obtaining a list of Department of Defense Activity Address Codes (DoDAACs) that are connected to AFLCMC and taking a subset of the complete dataset that had one of the DoDAACs on the list. After breaking down the entire data set to the AFLCMC contracts, there are 44,450 contracts and 1,622,840 CLINs in the data. This set was then further subsetted into smaller parts for training and testing purposes.

Within the AFLCMC contract set, the training/testing set was narrowed down even further. CLINs were selected to try and get a good overall representation of the contracts in both the training and testing sets. The testing set was comprised of 117 CLINs while the training set included 518 CLINs over 3 iterations. The breakdown of number of entities in the testing set along with the entities in each iteration can be seen in Table 3: Training/Test Set BreakdownTable 3.

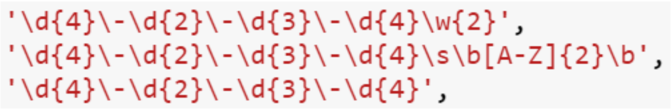
Regular Expression (Regex)

Regular expression (regex) is a sequence of characters that are used to find patterns in text. Each of the parameters has its own regex. Each regex is created by referring to the public forum and can be customized to suit the system and programming language used. Regex is used to detect any text data that matches any of the parameter. If the regex found any, the matched data will be fetched and inserted to the output (Pakhari, Jamil, Rusli, & Rahim, 2020).

The regex approach has the highest success rate when there are consistent patterns in the data. This works well for the NSN which follows a consistent pattern with an optional two characters tacked on to the end. Part numbers, on the other hand, are very inconsistent and can take on many different patterns. Multiple regex patterns must be put into place and look for groups of numbers/characters that come after certain phrases that would indicate the part number is to follow. With CAGE Codes being 5 alpha-numeric characters, the regex set had an extremely large amount of false positives to the point where it was not feasible to go through them by hand. Thus, trying to quantify how many CAGE Codes Regex actually picked up because of its lack of accuracy is useless. Below the approaches used for NSNs and Part Numbers are looked at in greater detail.

Regex Approach for NSN

National stock numbers follow a pattern of 4-2-3-4 numeric characters (0000-00-000-0000). Sometimes they have two upper-cased characters after the last digit and these were kept in case they are needed for further analysis, but can also be removed easily if just the base NSN is needed. This consistent pattern works very well for regex. For the regex method of extracting NSNs from contracts, there were three patterns used to identify them. The patterns are shown below in Figure 9. The first pattern looks for an NSN with 2 characters on the end, the second representing a NSN followed by a space and two upper-case characters, and the last being the standard NSN format.

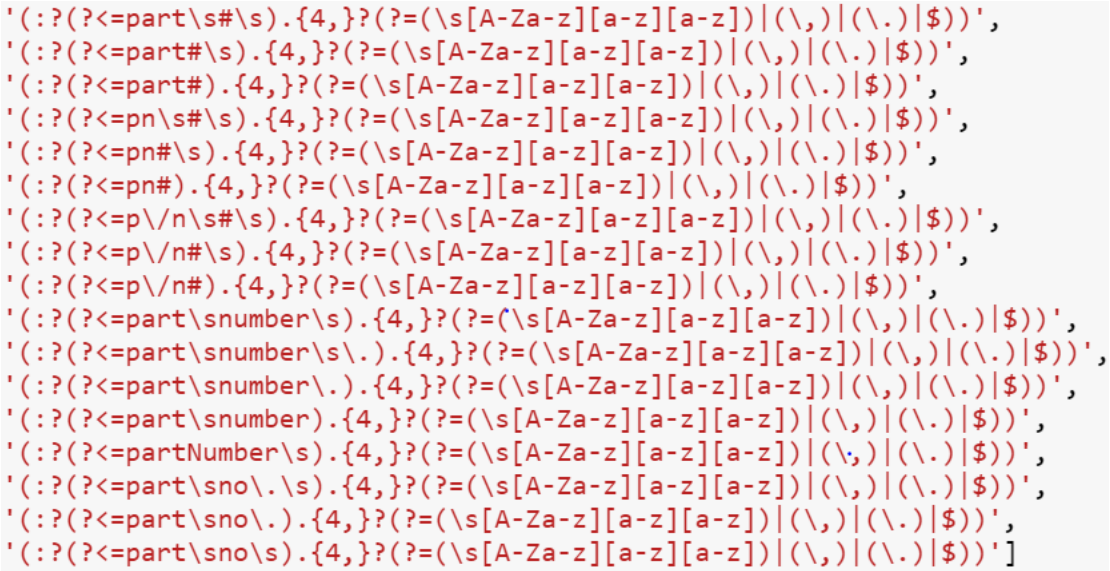


```
'\d{4}\-\d{2}\-\d{3}\-\d{4}\w{2}',
'\d{4}\-\d{2}\-\d{3}\-\d{4}\s\b[A-Z]{2}\b',
'\d{4}\-\d{2}\-\d{3}\-\d{4}',
```

Figure 9: Regex Patterns for NSN

Regex Approach for Part Number

Part numbers come in many different lengths, patterns, formats, etc. This makes it much tougher for the regex approach to pick up part numbers in the contracts. Instead of trying to generate a list of all the possible patterns, the regex patterns used look for phrases that would indicate a part number is to follow. Phrases such as “Part Number”, ”Part #”, “PN”, etc. are used to try and identify where the part number is in the contract, and then take the characters after it and give that the entity of part number. A lot more Regex patterns were used for the search of part number given the non-standardized format, and these patterns are shown below in Figure 10.



```
'(:?(?<=part\s#\s).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))',
'(:?(?<=part#\s).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))',
'(:?(?<=part#).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))',
'(:?(?<=pn\s#\s).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))',
'(:?(?<=pn#\s).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))',
'(:?(?<=pn#).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))',
'(:?(?<=p\/n\s#\s).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))',
'(:?(?<=p\/n#\s).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))',
'(:?(?<=p\/n#).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))',
'(:?(?<=part\snumber\s).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))',
'(:?(?<=part\snumber\s\.).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))',
'(:?(?<=part\snumber\.).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))',
'(:?(?<=part\snumber).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))',
'(:?(?<=partNumber\s).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))',
'(:?(?<=part\sno\.\s).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))',
'(:?(?<=part\sno\.).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))',
'(:?(?<=part\sno\s).{4,}?(?=(\s[A-Za-z][a-z][a-z])|(\,)|(\.)|$))']
```

Figure 10: Regex Patterns for Part Number

Named Entity Recognition

Named entity recognition is the process of extracting information by locating and classifying named entities. These entities are then grouped into pre-defined categories such as the names of persons, organizations, locations, dates, monetary values, percentages, etc. (Li S. , 2018). The named entity recognition approach uses a corpus of data to try and identify the entities in the text as opposed to regex where patterns are the most important. With the four entities of interest in this analysis not being standard entities, a custom knowledge base for the NER model will be needed in identifying NSNs, Part Numbers, CAGE Codes, and Suppliers in the text. Once the annotations are complete, they are fed to the NER model to be used as the corpus for the model to be trained on.

Annotating the Knowledge Base

Data annotation plays a crucial role in building a representative set and ensuring NER models are trained with the right information. Producing the necessary annotation from any asset at scale can be a challenge due to the complexity and time involved with annotation (Zeng, et al., 2021). In addition to being complex, the process can be lengthy especially when the text originated from a .pdf file where text from different boxes gets out of order and care must be taken to make sure the right entity is captured.

The annotations were initially done on 117 CLINs for the test set and 500 CLINs for the first iteration of the training set. The test set was chosen from CLINs in the AFLCMC subset that were suspected to have a part number by the regex output. It is important to make sure there was a good representation of all the entities that are being explored in the test set. In addition, a manual process of looking for CLINs that had Suppliers/CAGE codes in them was used to make sure there were a sufficient amount in the testing set. There were several CLINs with very similar information, so it was important to pick CLINs with the most variation in information as well as format to be able to scope the model to various CLIN text formats.

The initial training set methods were very similar. Again, CLINs were annotated that were suspected to have part numbers from the regex results as well as CLINs that had a good representation of CAGE codes and supplier names from the manual examination of the CLINs. This set was much larger than the testing set so the strong representation of formats of the data occurred more naturally when annotating a larger set.

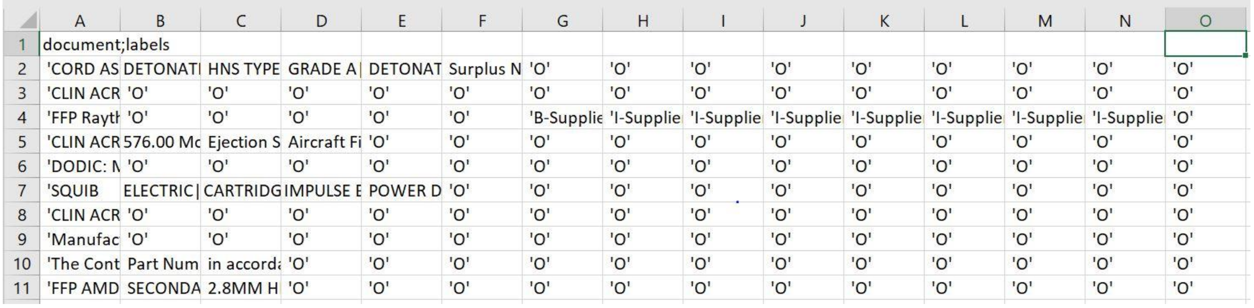
Additional iterations of the training set annotations were added until the performance of the model was sufficient. The breakdown of entities in the training (by iteration) and testing set can be seen below in Table 1

Table 3.

Table 3: Training/Test Set Breakdown

	Test Set	**Iteration 1**	**Iteration 2**	**Iteration 3**
Number of CLINs Annotated	**117**	**500**	**114**	**104**
Suppliers	43	56	25	88
CAGE Codes	61	265	32	39
Part Numbers	126	514	233	149
NSNs	61	338	32	35

Once the corpus was annotated, the PyLighter output needed to be adjusted into a format SpaCy can recognize. The output that is exported from PyLighter is a row for each CLIN that was annotated. This can be seen below in Figure 11 where the first column is the CLIN text, and if its over a certain number of characters the text goes into the next column(s) as well. The columns following the CLIN text contain the classification of the text. The text labels were done by character and the output is an 'O' if the character was not a part of an entity, a 'B-Entity' with the entity name filled in respectively if that character represented the beginning of an entity, and 'I-Entity' with the entity name filled in respectively if that character was part of the entity but not the first character in the entity. We can see at the beginning of row 4 “Raytheon” is the Supplier, starting at the 6th character in the CLIN text.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	document;labels														
2	'CORD AS	DETONATI	HNS TYPE	GRADE A|	DETONAT	Surplus N	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'
3	'CLIN ACR	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'
4	'FFP Rayth	'O'	'O'	'O'	'O'	'O'	'B-Supplie	'I-Supplie	'I-Supplie	'I-Supplie	'I-Supplie	'I-Supplie	'I-Supplie	'I-Supplie	'O'
5	'CLIN ACR	576.00 Mc	Ejection S	Aircraft Fi	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'
6	'DODIC: N	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'
7	'SQUIB	ELECTRIC|	CARTRIDG	IMPULSE E	POWER D	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'
8	'CLIN ACR	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'
9	'Manufac	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'
10	'The Cont	Part Num	in accord:	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'
11	'FFP AMD	SECONDA	2.8MM H	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'	'O'

Figure 11: PyLighter Output

SpaCy does not accept the annotations in this format, so some post processing was done to turn the annotations into something SpaCy can ingest and use for the custom NER model. The PyLighter .csv file was turned into a .json file that includes the CLIN text along with the starting position and ending position of the entity along with the entity name.

Training the Model

The annotations that were explained in the previous section are the inputs used for training the model. The annotations in .json format are read in by SpaCy and iterated over a specified number of times until the model is sufficiently trained. The train_spacy function was used from the SpaCy documentation. The number of iterations can be varied to produce different results of the model. Each entity is trained separately with its own model, this way if there is lack of an entity in a CLIN the model has an easier time picking up each entity individually.

Within the normal train_spacy function, there are different customizations options that can be used for tuning the model to better suit the data at hand. Tokenization allows for changing where the tokens are split. The infix, prefix, and suffix settings within tokenization allows for specification on where to add splits in the text. This parameter is important in splitting tokens where there might not normally be splits in the English language.

For example, some of the CLIN text had part numbers that did not have a space between the word and the part number itself (part#394-4859-21). This whole character string was being returned as one token instead of multiple. By adding “#” as an infix, this allows the token to be split up into three separate tokens: “part”, “#”, and “394-4859-21.”

There was also a common occurrence of the last word of the description being followed with a “(“ and then the supplier name (e.g. “cord(Lcom Part #”). Since there was no space in between the word and the “(“ it was picking up “cord(Lcom” as a single token. To fix this problem, “(“ was added as an infix to the NLP function. By adding “(“ as an infix, this allows the token to be split up into three separate tokens: “cord”, “(“, and “Lcom.”

Comparing Results

In answering the research questions of how much better does the custom NER model perform than the Regex approach or standard NER approach at extracting the entities of interest, the baseline results of the test set are compared to the test set performance of the custom NER model. Baseline results for NSN and Part Number using regex as well as baseline results for supplying organization using the standard SpaCy “org” entity can be found in Table 4.

To compare the results, the improvement of four evaluation metrics are used: accuracy, precision, recall, and F1 score. The fact that the standard SpaCy model does not pick up any of the CAGE codes as suppliers as those are specific to the DoD must also be accounted for. This additional information gleaned in the custom NER model will be discussed more in the results section of this paper.

Chapter Summary

In this chapter the contracts data that is available was looked at as well as the subset of the overall data that will be used in this analysis. Next, the methodological approach to applying regex and a custom NER model to the contracts data was explained. Last, the performance measures were set on how to evaluate the custom NER model in comparison to the previous methods used. The following chapter will provide a detailed look at the results of both the regex and NER methods as well as a comparison of the results.

IV. Results and Analysis

Chapter Overview

This chapter presents the results from applying the methodology laid out in the Methodology section above. The results are shown from applying Regex and standard SpaCy "org" entity to the data set. It is shown how many Part Numbers, NSNs, and suppliers are extracted from the both the test set of contracts and the AFLCMC contracts. Next the results are shown from applying the custom NER model that was trained to the data set to see how many Part Numbers, NSNs, Suppliers, and Cage Codes are extracted from the same sets of contracts. Finally, the results from the two models are compared to see if there is any improvement from building a custom NER model as opposed to using the Regex approach to extract the entities of interest from contracts.

Baseline Results

This section explores at the results of applying the Regex approach to the CLIN text. It is first seen how this technique performs in extracting Part Numbers and NSNs on the entire AFLCMC subset and then scope it down to see how it performs on the testing set. In addition, the SpaCy "org" entity is used on the testing subset to get baseline results on how the standard NER model performs on the data.

AFLCMC Subset

Applying the Regex patterns to the full set of AFLCMC contracts produced the results shown below in Table 4. The NSNs that were outputted from the model looked very good from the perspective of there being no false negatives that showed up in the results. However, there was some clean up involved in getting rid of false positive Part Numbers in the output. These were manually removed from the data and not included in the found Part Numbers results in Table 4. The list of false positive Part Numbers (sorted by highest number of occurrences) that were removed from the data can be seen below in

Table 5. It is important to note that this is not an all-inclusive list of false positives in the data. These are just the ones that were caught by eye when looking through the Part Numbers that Regex returned.

It is also important to note that Regex picked up a ton of CAGE codes as part numbers. One common theme of the CLINs was the phrase "Manufacturer Part Number" followed by the CAGE code, then the part number. Regex is picking up what comes directly after part number, which in a lot of cases is the CAGE code. This affects the validity of these Regex Part numbers even if by eye it can't be determined it is not a part number. The next section shows how a custom NER model helps with distinguishing between Part Number and CAGE code.

There were also some special characters that are not part of the part number that the Regex model returned that needed to be cleaned up as well. Characters such as colons, periods, pound sign, quotation marks, and spaces showed up at the beginning or end of some of the part numbers. These leading or trailing characters were removed by using the gsub function to replace the character with nothing. This list of leading characters is all inclusive for the AFLCMC contracts, but might need to be expanded when applied to other sets of contracts. This did not affect the count of Part Numbers that Regex captured but is still an important step in the process of making sure the parts are extracted accurately.

Table 4: Regex Results on full AFLCMC Subset

	Contracts Searched	CLINs Searched	CLINs with Found Entities	Percentage of CLINs with entity	Max. entities found in one CLIN	Avg. entities found in one CLIN*
NSN	44,450	1,622,840	92,497	5.70%	20	1.07
Part Number	44,450	1,622,840	129,924	8.01%	63	1.17

*average value is calculated based off of the CLINs with found entities, not the entire set of CLINs that were searched

Table 5: False Positive Part Numbers - Regex

False Positive Part Number	**Number of Occurrences Removed**
s and	457
s for	226
s, quantities	47
Part	41
Quantity	36
Off Contract	17
Or NSN	12
QTY Handling	8
QTY Drying	6
NOUNQUANTITY	5
QTY Bomb	5

, Quantity	4
FOB:	4
QUANTITY	3
NOUNQty	3
QTY	3
From:	3
Includes	2
Changes as	2
OF GFE TO BE	1
s that	1
Qty.Total w/o	1

Testing Set

The regex and SpaCy “org” entity results on the testing set are shown below. False positives here include entities that were extracted but include additional information along with the correct entity. For example, some of the CLINs have part numbers and CAGE codes next to them and the Regex code picks up the entire sequence as the part number. Even though the correct part number is in there, the entity as a whole is incorrect as it includes the CAGE code (or other unnecessary information) as well.

The results are color coded based on their score percentage where green indicates a score of 95% or better, light green represents 90% to 94.9%, yellow represents scores in the 80s, orange represents scores in the 70s, red represents scores in the 60s, and dark red represents scores less than 60%.

Table 6: Baseline Results on Test Set

	Part Number (Regex)	**NSN (Regex)**	**Supplier (SpaCy Standard NER)**
True Positives	4	60	10
True Negatives	0	56	7
False Positives	118	0	281
False Negatives	122	1	33
Total Entities in text	**126**	**61**	**43**
Total CLINs Searched	**117**	**117**	**117**
Accuracy	1.6%	99.1%	5.1%
Precision	3.3%	100%	3.4%
Recall	3.2%	98.4%	23.3%
F1	3.2%	99.2%	5.9%

Just like anticipated, the regex method performed extremely well for identifying NSNs in the CLIN text. The consistent pattern of an NSN is what regex thrives on so it is not surprising that the model has an accuracy of 99.1% as well as an F1 score of 99.2%. Part Numbers, on the other hand, performed horribly with regex. The accuracy of 1.6% is not even close to sufficient along with the 3.2% F1 score. A lot of the poor performance of the model has to do with the regex model picking up additional information around the part number itself that should not be in the entity. This causes a false positive and false negative in the same entity which is why the model performs very poorly. This is due to the fact that there is no uniform pattern to part numbers, so regex is just picking up whatever text comes after the word part number. In a lot of cases, the manufacturer (CAGE Code) and part number are in the format MFG Part Number 12345 123-1234-12345 where what comes directly after the phrase "Part Number" is actually the CAGE code. There is no way for regex to know this as it is not a machine learning technique.

Extracting suppliers with the SpaCy "org" entity did not go well either. There were tons of false positives in the data, which would be easy to weed out but the model still only extracted 10 of 43 (23%) of the suppliers in the testing set data. In the next section it will be seen how applying a custom NER model significantly improves the performance of extracting Part Numbers and Suppliers from the CLIN text.

NER Results

This section looks at the results of applying a custom NER model to the CLIN text of the contracts. It is first shown how this technique performs in extracting Suppliers, CAGE Codes, Part Numbers, and NSNs on the entire AFLCMC subset and then scoped down to see how it performs on the testing set.

AFLCMC Subset

Applying the custom NER models to the full set of AFLCMC contracts produced the results shown below in Table 8. The NSNs that were outputted from the model looked very good from the perspective of there being no false negatives that showed up in the results. There were some false positives where "Not Applicable" got picked up by the model, but these can easily removed. Another case of false positives happens when in the CLIN text, they have NSN: XXXX. Where the spot the NSN usually is, the Federal Supply Class (FSC) is shown without the National Item Identification Number (NIIN) tacked on to the end. The model still picks this up as the NSN but again these can easily be removed.

The need for part number clean up decreased significantly although there were some cases of false positives that needed to be taken care of. These were manually removed from the data and not included in the found Part Numbers results in

Table 7. The list of false positive Part Numbers (sorted by highest number of occurrences) that were removed from the data can be seen below in Table 8. It is important to note that this is not an all-inclusive list of false positives. There are some that include the part number with additional information after it that are one-offs. The one offs are not captured in the table, its meant to be what the model is consistently getting wrong. The ones shown in the table are just the ones that were caught by eye when looking through the Part Numbers that the NER model returned.

There were also some special characters that are not part of the part number that the NER model returned that needed to be cleaned up as well. Again, these were decreased significantly from the amount that needed to be cleansed in the Regex results, but still a necessary step. Characters such as colons, and dashes showed up at the beginning or end of some of the part numbers. These leading or trailing characters were removed and replaced with nothing. This list of leading characters is all inclusive for the AFLCMC contracts, but might need to be expanded when applied to other sets of contracts.

Table 7: NER Results on Full AFLCMC Subset

	Contracts Searched	CLINs Searched	CLINs with Found Entities	Percentage of CLINs with entity	Max # entities found in one CLIN	Avg. # entities found in one CLIN*
NSN	44,450	1,622,840	124,013	7.64%	17	1.05
Part	44,450	1,622,840	97,461	6.01%	39	1.08
Supplier	44,450	1,622,840	48,264	2.97%	101	1.30
CAGE Code	44,450	1,622,840	57,929	3.57%	4	1.00

*average value is calculated based off of the CLINs with found entities, not the entire set of CLINs that were searched

Table 8: False Positive Part Numbers - NER

False Positive Part Number	**Number of Occurrences Removed**
:	192
-	12
NameQuantitUnit	4
ITEM NAMEQUANTITY	2
0 - -	2

Testing Set

The NER results on the testing set are shown below by iteration. There were 3 iterations in the process to come to the final model that was proven the best (or equal) to the previous two iterations. False positives here include entities that were extracted incorrectly. For example, some of the part numbers have spaces between dashes. In some cases, the model picks up part of the part number. Even though the model correctly picked out the location of the entity, the entity as a whole is incorrect as it does not give the full part number, thus there is a false negative that goes along with it as well. The same thing was present in the Supplier entity as well. Some of the Supplier names were not picked up in full, just part of the name was extracted and these were labeled as a false positives along with a false negative.

In the first iteration of the model, 500 CLINs were annotated. This set was comprised of 56 Suppliers, 265 CAGE Codes, 514 Part Numbers, and 338 NSNs. The results that this model produced on the testing set can be seen below in Table 9. Since the CAGE Code and Supplier Name are giving us the same information (Supplier) these entities are combined into a single column, Supplier Overall, to see how the model does in picking up the Supplier entity as a whole. The results are color coded based on their score percentage where green indicates a score of 95% or better, light green represents 90% to 94.9%, yellow represents scores in the 80s, orange represents scores in the 70s, red represents scores in the 60s, and dark red represents scores less than 60%.

Table 9: NER Results on Testing Set (Iteration 1)

	CAGE Code	**Supplier Name**	**Supplier (Overall)**	**Part Number**	**NSN**
True Positives	54	14	68	103	61
True Negatives	56	65	121	0	53
False Positives	0	13	13	4	6
False Negatives	7	29	36	23	0
Total Entities in text	**61**	**43**	**104**	**126**	**61**
Total CLINs Searched	**117**	**117**	**117**	**117**	**117**
Accuracy	94%	65.3%	79.4%	79.2%	95%
Precision	100%	51.9%	84%	96.3%	91%
Recall	88.5%	32.6%	65.4%	81.7%	100%
F1	93.9%	40%	73.5%	88.4%	95.3%

The second iteration of the NER model had an additional 114 CLIN annotations added to the model. This set was comprised of 25 Suppliers, 32 CAGE Codes, 233 Part Numbers, and 32 NSNs. F1 scores for three of the entities increased (CAGE Code, Part Number, and NSN) while the fourth (Supplier) decreased. Although the number of false positives decreased, so did the number of true positives in the data which caused the lower score. Full results for the second iteration of the model produced on the testing set can be seen below in Table 10.

Table 10: NER Results on Testing Set (Iteration 2)

	CAGE Code	**Supplier Name**	**Supplier (Overall)**	**Part Number**	**NSN**
True Positives	59	7	66	106	61
True Negatives	56	70	126	0	54
False Positives	0	6	6	5	2
False Negatives	2	36	38	20	0
Total Entities in text	**61**	**43**	**104**	**126**	**61**
Total CLINs Searched	**117**	**117**	**117**	**117**	**117**
Accuracy	98.3%	64.7%	81.4%	80.9%	98.3%
Precision	100%	53.8%	91.7%	95.5%	96.8%
Recall	96.7%	16.3%	63.5%	84.1%	100%
F1	98.3%	25%	75%	89.4%	98.4%

The third iteration of the NER model uses an additional 104 CLIN annotations to be added to the new model. This set was comprised of 88 Suppliers, 39 CAGE Codes, 149 Part Numbers, and 35 NSNs.

CLINs were targeted that had a good representation of Suppliers as this was the entity lacking success. When training the model, there were multiple warnings of Supplier annotations not being used. After some exploration, it was found that spacing was not well preserved around some of the Supplier names. The last word of the description was followed with a “(“ and then the supplier name (E.X. “cord(Lcom Part #”). Since there was no space in between the word and the “(“ it was picking up “cord(Lcom” as a single token. To fix this problem, “(“ was added as an infix to the NLP function. All warnings went away after this was added. CLINs with multiple part numbers were also targeted as this is the main case of false negative results.

The targeted annotations worked as the Supplier and Part Number results both improved. The CAGE code results improved slightly as well while the NSN results were exactly the same as the previous iteration. Full results for the third iteration of the model produced on the testing set can be seen below in Table 11.

Table 11: NER Results on Testing Set (Iteration 3)

	CAGE Code	**Supplier Name**	**Supplier (Overall)**	**Part Number**	**NSN**
True Positives	60	21	81	115	61
True Negatives	56	72	128	0	54
False Positives	0	2	2	1	2
False Negatives	1	22	23	11	0
Total Entities in text	**61**	**43**	**104**	**126**	**61**
Total CLINs Searched	**117**	**117**	**117**	**117**	**117**
Accuracy	99.1%	79.5%	89.3%	90.6%	98.3%
Precision	100%	91.3%	97.6%	99.1%	96.8%
Recall	98.4%	48.8%	77.9%	91.3%	100%
F1	99.2%	63.6%	86.6%	95%	98.4%

The entities extracted are all important, thus it is imperative to understand how many of the CLINs have all the entities extracted correctly for use in further analysis. The CAGE Code and Supplier Name are grouped together as a single entity (shown as Supplier in Table 12) since they are both looking to extract the same piece of information. The Supplier Name is looked up in the CAGE Code database, and if there is a single match it is considered as a found entity. In cases where there are multiple CAGE Codes for the given Supplier Name, it is not considered a found entity. 59 of the 117 CLINs in the test set had all 3 entities available to extract. Table 12 below shows a breakdown of the CLINs that had the potential for all three to be extracted and the success rate by entities.

Table 12: Fully Correct CLINs

Entities Correct	Number of CLINs	Percentage of CLINs
Part	0	0%
NSN	0	0%
Supplier	0	0%
Part & NSN	0	0%
Part & Supplier	0	0%
NSN & Supplier	4	6.8%
Part, NSN & Supplier	55	93.2%

Comparison Results

Of the three entities of interest that have baseline results, two performed significantly better with a custom NER model (Part Number, Supplier) while the third entity (NSN) showed similar results between the Regex model and the NER model. Figure 12 below shows the number of each entity successfully extracted from the baseline model (blue) and NER (orange) with the grey representing the total number of entities in the test set. The next sections will dive deeper into the results of each of the entities.

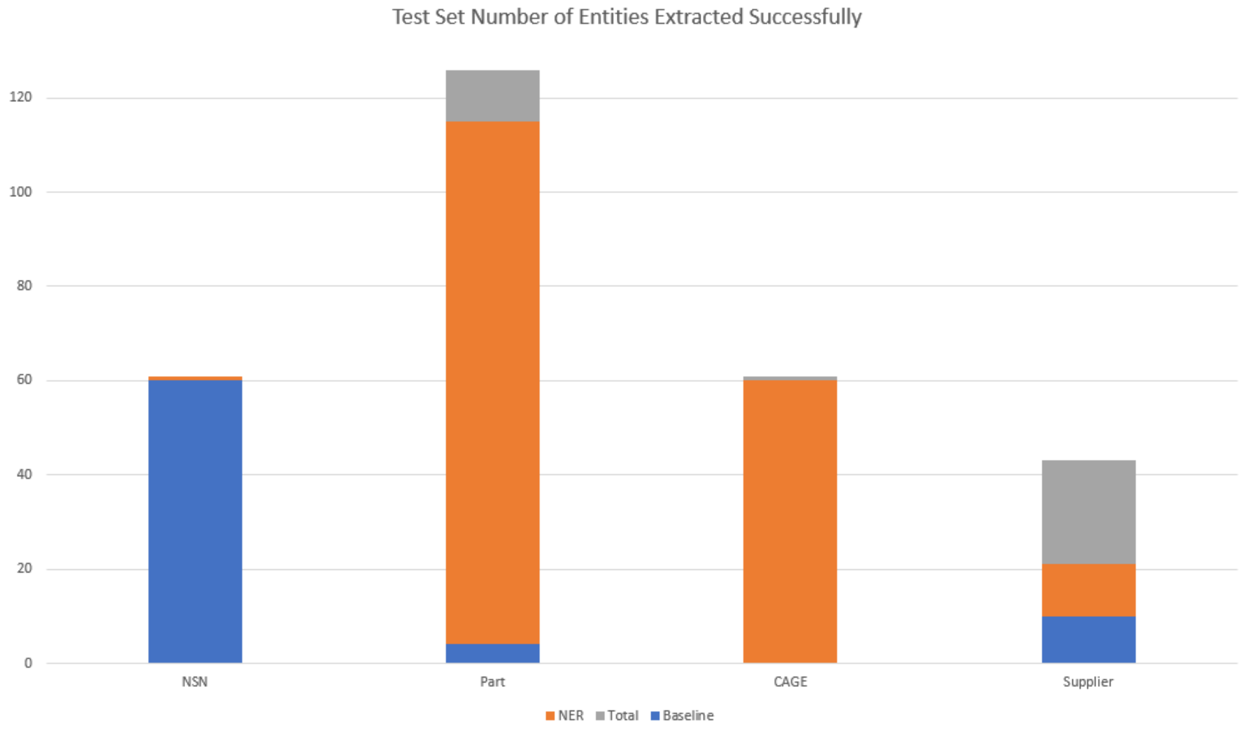


Figure 12: Comparison Results for Test Set

Looking at the baseline results, it was found that NSN performed very well with Regex. The accuracy of the model was 99.1% while the F1 score was 99.2%. When compared to the NER results, Regex performed slightly better. The accuracy of the NER model was 98.3% (down 0.8%) with an F1 score of 98.4% down (0.8%). This is due to an additional false negative result in the NER model. Overall, both models are sufficient in capturing the NSN. The NER model will give some more false positives but also pick up NSNs that do not include the dashes or are in other odd formats.

Part numbers were extracted with much greater success in the custom NER model. 115 of the 126 part numbers were extracted correctly with the custom NER model as opposed to the 4 of 126 that were extracted correctly with the Regex approach. The regex approach was close in a lot of the part numbers, but added additional information or cut off part of the whole part number. Accuracy increased from 1.6% to 90.6% when comparing the regex results to the NER results while the F1 score increased from 3.2% to 95%.

The last entity, Supplier, also saw a huge improvement when switching to a custom NER model as opposed to the standard SpaCy “org” entity. When comparing the SpaCy “org” entity to the Supplier Name custom NER model, the accuracy improves from 5.1% to 79.5% while the F1 score improves from 5.9% to 63.6%. A big advantage of the custom NER model as opposed to just the standard SpaCy “org” entity is the ability for the model to catch CAGE Codes as well. The custom model performed extremely well on CAGE Codes, picking up 60 of the 61 that were in the testing set. The accuracy of the CAGE Codes was 99.1% while the F1 score was 99.2% with the custom NER model. The Supplier Name along with location can be found from this CAGE code, giving us more information for further analysis than just having the Supplier Name itself. Combining the custom NER for Supplier Name and CAGE Code proves to see even greater improvements as compared to just using the standard SpaCy “org” entity.

Chapter Summary

In this chapter the results that the baseline model produced using regex (Part Number and NSN) and standard SpaCy entities (Supplier Name) were presented along with the results from the custom NER models that were built for Part Number, NSN, Supplier Name, and CAGE Code. It was shown how the models performed on a testing set of 117 CLINs as well as performance when the model was scaled up to the entire AFLCMC subset. Lastly, the results were compared between the baseline analysis results and the custom NER model. In the next section the results will be further discussed and conclusions drawn from this research will be provided.

V. Conclusions

Chapter Overview

This chapter utilizes the results from the previous chapter to answer the initial research questions proposed in Chapter I. The relevance of those findings is then discussed to see how they will benefit the DoD in future analyses. Limitations of the analysis follow, along with recommendations for future work in the contracts analysis space. The paper wraps up with a summary of the final thoughts.

Findings

In looking at the results from the previous section, the research questions that were posed in the introduction are evaluated. The three questions that were posed along with their results will be discussed below.

The first question that was posed was can more Suppliers successfully be extracted by using a custom NER model as opposed to a standard SpaCy "org" entity model? The supplier entity saw an improvement in true positives of over 100% when comparing the SpaCy "org" entity model results to the custom NER model results. 21 supplier names were successfully extracted with the custom NER as compared to the 10 with standard NER model. The F1 score also had a significant improvement from 5.9% with the standard SpaCy "org" entity to 63.6% with the custom NER model. The big enhancement in extracting the supplier from the contract comes in being able to extract the CAGE code as well. If CLINs do not contain the Supplier name, sometimes they instead contain the CAGE Code. This 5-digit code has great extraction performance with a custom NER model. 60 of 61 CAGE Codes were successfully extracted from the testing set. Combining these two models in extracting the supplier from the CLINs, there is an overall true positive rate of 81/104 with an F1 score of 86.6%. There were an additional 13 CLINs that did not have a Supplier name or CAGE Code in it. There will be some methods proposed in the Recommendations section below to associate a supplier name with the CLINs that are missing the information.

The second question that was posed was can more NSNs be successfully extracted using a custom NER model as opposed to a regex model? The improvement that was seen in NSN performance was negligible. The custom NER model was able to pick up one additional NSN as compared to the Regex model. This specific NSN was chosen intentionally because it was missing the third "-" in the sequence, and the NER model was able to still pick it up as an NSN. There were a couple false positives in the NER model where Regex did not have any false positives. These are easy to pick out as NSNs are very standardized, and can be post-processed to take out the false positives in the results fairly easily. Even though the custom NER model did not perform significantly better than Regex, it is still recommended so that the one-off cases (like missing a dash) can be picked up.

Finally, it was questioned if more Part Numbers can successfully be extracted using a custom NER model as opposed to a regex model. The part numbers saw the largest improvement of the three entities. Using Regex, the model was only able to successfully extract 4 of 126 part numbers correctly. By iteration 3 of the custom NER model, 115 of 126 part numbers were extracted correctly. The F1 score of the Regex approach was 3.2% while it increased to 95% with the custom NER model. The use of a custom NER model resulted in huge improvements on the extraction of this entity without any standard patterns, thus it is recommended to use a custom NER model for extracting part numbers and any other entity that might come up that does not have a standard format that Regex can pick up well.

Relevance

The information gleaned from extracting the entities from the contracts is just the first step in providing useful analysis. Having the Part Numbers, Suppliers (from the Supplier Name and CAGE Code entities combined) and NSNs will allow for further analysis to be done on the contracts. Specifically, this will allow for the part/vendor relationships to be analyzed along with adding in pricing data that is available.

Figure 13 below shows one example NSN with its associated Part Numbers, CAGE Codes, and CLIN unit prices (where applicable). The prices are sensitive information so for this example the dollar amounts are scaled into three categories with "$" being the least expensive unit price, followed by "$$" being a medium unit price, and "$$$" being the most expensive unit price. The vendor supplying part number 167C2209 has three different CAGE Codes depending on what location the part is purchased from. We can see that vendor 64609 supplies four different part numbers (R0980861578009, R0980861578005, R0980861578018, R0980861578019) for NSN 4933-01-068-8007. For each of their four parts, the CLIN unit price is in the lowest category ($). The vendor that supplies part number 167C2209 at three different locations has varying unit prices based on the vendor location. The CLIN that had CAGE Code 52477 did not have data on average unit price, thus it was left as a "?". The other two locations supplying part umber 167C2209 had multiple CLINs each with unit pricing information. CAGE Code 1BAM3 had CLIN unit prices in the middle category ($$) while CAGE Code 94117 had the most amount of CLINs with this CAGE Code/Part Number pairing. All of their CLIN unit prices fell into the highest category ($$$). Thus, it would be recommended to purchase this NSN from the vendor with CAGE Code 64609 since all four of the parts they offer are significantly cheaper than the vendor offering part number 167C2209.

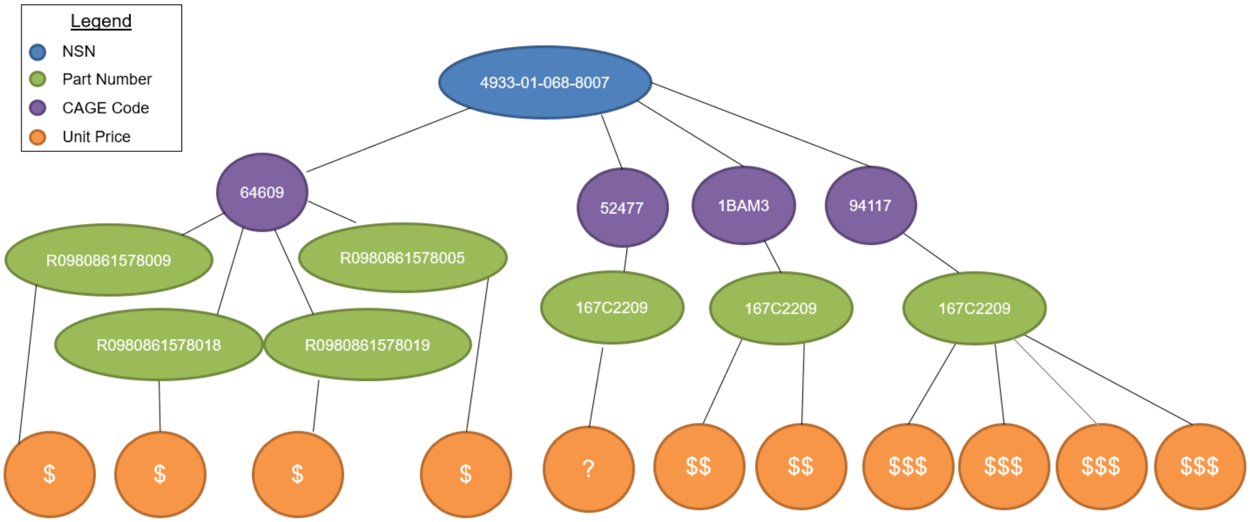


Figure 13: Part to Vendor Relationship

Joining this data with pricing data from the contracts, it can be determined if there is a price difference between buying a part from one vendor over another. It can also be noted if there are multiple contracts within an organization that are buying the same part from the same vendor, there is the potential to buy in bulk and take advantage of economies of scale to get better pricing. This can also be done across different organizations to see if there can be any collaborating in buying parts.

In addition to pricing data, maintenance data can be brought in to see how the depot records are for a certain part. This would allow us to see if a certain vendor has more reliable parts than a competing vendor and if so, what is the cost benefit analysis on price to longevity of the part.

The process of making these graphs is currently manual and can be done for any NSN that is in the data. This is a time-consuming process and would take a significant amount of man-hours to complete for each NSN that is cataloged. Thus, with the help of a graph database software, graphs like these could be automated saving the analyst a significant amount of time.

Limitations

The results of this analysis are done on a subset of the overall contracts. Even though the test set was selected to be representative of all the contracts, there might be some formats that were missed when selecting the set. That being said, the scalability from a test set of 117 CLINs to over 7 million might not see the same results as there can be varying formats that were not seen in the test set or the annotation set. Another limitation is there are other areas of the contracts where information lies. Parts can come in government furnished property (GFP) tables along with rights assertion (RAT) tables while the Supplier Name and CAGE Code can also reside in the front forms. Recommendations will be given in the next section that could help with the limitations of the current analysis.

Recommendations

To help improve the first limitation mentioned above, the first recommendation is to keep increasing the annotation set and tuning parameters to get a model that can be more generalized to the entire 3.7 million contracts that are currently available and future contracts that become available. Time is a big limiter here and annotating large sets is time consuming although it is very important to improving model performance. The more annotations that are added to the model, the more the model can learn and generalize to better predict the entities. In addition to annotations, parameter tuning will need to continue to be done as more annotations are added. These parameters will need to be tuned as the model changes to best suit the new model output and data. This will give the model the best chance to be able to scale to the overall contracts set of data.

The second recommendation is to use front form cage codes as a secondary source for where the CLIN does not have a CAGE Code or Supplier Name associated with it. An assumption will have to be made that if there is no supplier in the CLIN, the supplier is the same as it is on the front of the contract. This will allow for a more comprehensive extraction of the Supplier Name and make doing further analysis more complete.

In addition to a more comprehensive extraction of suppliers, more information can also be gleaned on part numbers and NSNs by extracting from different locations. There are part numbers along with NSNs that can reside in the GFP tables and RAT tables in the contracts. Building additional NER models that look at the pages of contracts that contain a GFP or RAT table, respectively, will allow for more of these entities to be extracted from the contracts. Annotations will have to be done separately for these tables as they are in different formats to allow the models to be specific to each table format.

Finally, as discussed in the relevance section, analysis can start to be done with the information that was extracted from the contracts. Pairing this newly extracted information with pricing data and maintenance data can allow us to explore the relationships between part and vendor. Adding additional information like pricing data and maintenance data can be important in understanding which vendors will provide the best overall value for a certain part. This information can be used by program offices when deciding which vendor to go with for future buying opportunities to save the Air Force money, whether it be on the initial price of the part or in the long run of buying a more reliable part that will not need to be repaired as frequently.

Summary

A custom NER model can increase the extraction power of DoD specific entities significantly. As with any NER model, the annotation set is time consuming but vital to being able to scale the results to a larger set of data. Once these entities are extracted, the analysis that can be performed by merging these results with pricing and maintenance data can help the Air Force make more educated decisions on which vendor to use for certain parts as well as help negotiate better prices for said parts.

Glossary

AFLCMC – Air Force Life Cycle Management Center

AFSC – Air Force Sustainment Center

AI – Artificial Intelligence

CAGE Code – Commercial and Government Entity Code

CLIN – Contract Line Item Number

CNE – Custom Named Entity

CNN – Convolutional Neural Network

DART – Data Analytics Resource Team

DLA – Defense Logistics Agency

DoDAAC – Department of Defense Activity Address Code

FN – False Negative

FP – False Positive

FSC – Federal Supply Class

GUI – Graphical User Interface

ML – Machine Learning

OCR – Optimal Character Recognition

NER – Named Entity Recognition

NIIN – National Item Identification Number

NLP – Natural Language Processing

NSN – National Stock Number

RNN – Recurrent Neural Network

TN – True Negative

TP – True Positive

Regex – Regular Expression

SVM – Support Vector Machine

Bibliography

AcqNotes. (2021, June 7). *Contracts & Legal: Contract Line Item Number*. Retrieved from AcqNotes Program Management Tool for Aerospace: https://acqnotes.com/acqnote/careerfields/contract-line-item-number

Alpaydin, E. (2016). Machine Learning: The New AI. MIT Press.

Aryoyudanta, B., Adji, T. B., & Hidayah, I. (2016). Semi-Supervised Learning Approach for Indonesian Named Entity Recognition (NER) Using Co-Training Algorithm. *2016 International Seminar on Intelligent Technology and Its Application.*

Butcher, Z. (2021). *Contract Information Extraction Using Machine Learning.* Air Force Institute of Technology.

Chen, M. (2020, October). *A Guide: Text Analysis, Text Analytics, & Text Mining.* Retrieved from https://towardsdatascience.com/a-guide-text-analysis-text-analytics-text-mining-f62df7b78747

Defense Logistics Agency. (2019). *The National Stock Number The Gear The Keeps The Supply Chain Running*. Retrieved from Defense Logistics Agency: https://www.dla.mil/HQ/LogisticsOperations/Services/CustomerOutreach/Brochures/NSNBooklet.aspx

Doshi, K. (2020, 12 13). *Transformers Explained Visually (Part1): Overview of Functionality*. Retrieved from Towards Data Science: https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

Eisenstein, J. (2019). Introduction to Natural Language Processing. MIT Press.

Explosion AI. (2017). *SpaCy*. Retrieved from What's New in v2.0: https://spacy.io/usage/v2

ExplosionAI. (2020). *Radically efficient machine teaching, powered by active learning*. Retrieved from Prodigy: https://spacy.io/universe/project/prodigy

Goyal, A., Gupta, V., & Kumar, M. (2018, August). Recent Named Entity Recognition and Classification techniques: A systematic review. *Computer Science Review*, pp. 21-43.

Hassani, H., Beneki, C., Unger, S., Mazinani, M. T., & Yeganegi, M. R. (2020). Text Mining in Big Data Analytics. *Big Data and Cognitive Computing*.

Heidenreich, H. (2018, 12 4). *What are the types of Machine Learning?* Retrieved from Towards Data Science: https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f

Honnibal, M. (2017, 11 12). *SpaCy*. Retrieved from SpaCy's NER model: https://spacy.io/universe/project/video-spacys-ner-model

Honnibal, M. (2018, 6). *Chapter 4: Compact word vectors with Bloom Embeddings*. Retrieved from Prodigy Support: https://support.prodi.gy/t/can-you-explain-how-exactly-hashembed-works/564/2

Honnibal, M., Montani, I., Van Landeghem, S., & Boyd, A. (2021, 1 31). *SpaCy*. Retrieved from Introducing SpaCy v3.0: https://explosion.ai/blog/spacy-v3

IBM Cloud Education. (2020, 7 15). *IBM Cloud Learn Hub*. Retrieved from Machine Learning: https://www.ibm.com/cloud/learn/machine-learning

IBM Cloud Education. (2020, 8 19). *Supervised Learning*. Retrieved from IBM Cloud Learn Hub: https://www.ibm.com/cloud/learn/supervised-learning

IBM Cloud Education. (2020, 9 21). *Unsupervised Learning*. Retrieved from IBM Cloud Learn Hub: https://www.ibm.com/cloud/learn/unsupervised-learning

Jugran, S., Kumar, A., Tyagi, P. S., & Anand, V. (2021). Extractive Automatic Text Summarization using SpaCy in Python & NLP. *International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE).*

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., & Dyer, C. (2016). Neural Architectures for Named Entity Recognition. *arXiv preprint arXiv:1603.01360*.

Lawton, G. (2021, 4 5). *Transformer neural networks are shaking up AI*. Retrieved from TechTarget: https://searchenterpriseai.techtarget.com/feature/Transformer-neural-networks-are-shaking-up-AI

Li, F.-F., Krishna, R., & Xu, D. (2021). *Convolutional Neural Networks for Visual Recognition*. Retrieved from Stanford: https://cs231n.github.io/convolutional-networks/

Li, S. (2018, August 17). *Named Entity Recognition with NLTK and SpaCy*. Retrieved from Towards Data Science: https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da

Mishra, A. (2018, February 24). *Metrics to Evaluate your Machine Learning Algorithm*. Retrieved from Towards Data Science: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234

Osinski, B., & Budek, K. (2018, 7 5). *What is reinforcement learning? The complete guide*. Retrieved from DeepsenseAI: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/

Pakhari, M. H., Jamil, N., Rusli, M. E., & Rahim, A. A. (2020). Implementation of Token Parsing Technique for Regex Based Classification of Unstructured Data for Cyber Threat Analysis. *International Conterence on Information Technology and Multimedia (ICIMU).*

PayLead. (2020). *Meet PyLighter*. Retrieved from https://blog.paylead.fr/introducing-pylighter-for-jupyter-payleads-open-source-annotation-tool/

Riggio, C. (2019, November 1). *What's the deal with Accuracy, Precision, Recall, and f1?* Retrieved from Towards Data Science: https://towardsdatascience.com/whats-the-deal-with-accuracy-precision-recall-and-f1-f5d8b4db1021

Roldos, I. (2020, March 30). *Named Entity Recognition: Concept, Tools, and Tutorial*. Retrieved from https://monkeylearn.com/blog/named-entity-recognition/

Schmitt, X., Kubler, S., Robert, J., Papadakis, M., & LeTraon, Y. (2019). A Replicable Comparison Study of NER Software: StanfordNLP, NLTK, OpenNLP, SpaCy, Gate. *Sixth International Conference on Social Networks Analysis, Management and Security.*

Serra, J., & Karatzoglou, A. (2017). Getting Deep Recommenders Fit: Bloom Embeddings for Sparse Bianry Input/Output Networks. *arXiv*.

Song, F., & de la Clergerie, E. (2020). Clustering-based Automatic Construction of Legal Entity Knowledge Base from Contracts. *International Conference on Big Data.*

Statista. (2021, June). Retrieved from Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025: https://www.statista.com/statistics/871513/worldwide-data-created/

Stepanyan, L. (2020). Automated Custom Named Entity Recognition and Disambiguation. *International Journal of Signal Processing*.

Stubbs, A., & Pustejovsky, J. (2013). Natural Language Annotaton for Machine Learning: a Guide to Corpus-Building for Applications. O'Reilly.

Vychegzhanin, S., & Kotelnikov, E. (2019). Comparison of Named Entity Recognition Tools Applied to News Articles. *Ivannikov Ispras Open Conference.*

Zeng, Q., Yu, M., Yu, W., Jiang, T., Weninger, T., & Jiang, M. (2021). *Validating Label Consistency in NER Data Annotation.* Notre Dame, IN: arXiv.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)* 23-12-2021	**2. REPORT TYPE** Master’s Thesis	**3. DATES COVERED** *(From – To)* JAN 2020 – DEC 2021

TITLE AND SUBTITLE
Using Custom NER Models to Extract DOD Specific Entities from Contracts

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Haberstich, Kayla P, Contr. (AFLCMC/LZIA)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
Wright-Patterson AFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT-ENS-MS-21-D-030

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Life Cycle Management Center
1865 4th St
Wright-Patterson Air Force Base, OH 45433
POC: Mr. Philip Ball (Philip.ball@us.af.mil)

10. SPONSOR/MONITOR’S ACRONYM(S)
AFMC AFLCMC/LZIA

11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRUBTION STATEMENT A. APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT
The Air Force Sustainment Center collected 3.7 million contracts onto the Air Force Research Laboratory’s high power computers. They are in the format of a .pdf or scanned document, making them unstructured data. The Data Analytics Resource Team extracted the documents into a textual format for use in further analysis. This thesis looks to extract four DOD specific entities (NSN, Part Number, CAGE Code, and Supplier Name) from the contracts using custom NER models. This newly extracted information will allow the Air Force to identify what parts are supplied by which vendors. This information along with historical CLIN pricing for the vendor specific part number can give decision makers the ability to negotiate pricing based on historical data and competitor pricing. In addition to just pricing, part numbers can be aligned with maintenance data to make informed decisions on which vendor to go with by analyzing life cycle costs of a part.

15. SUBJECT TERMS
Machine Learning, Named Entity Recognition, Natural Language Processing, High Power Computing, Word Embedding

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. Jeffery Weir, AFIT/ENS
U	U	U	UU	73	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636 Ext 4523 (DSN 785) (Jeffery.Weir@afit.edu)

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18