Finding a Needle in a Haystack: Named Entity Recognition for Large Research Papers

Akash Jagannathan

013856531

Karanbir Bains

009835800

Sudhanshu Shekhar

014535079

*Abstract*—In this project we are trying to extract references to the data source used in the scientific publications. Generally, these citations, to data source used, are hidden in plain-text and is very difficult to find manually. Knowing the source of the data, used for any research, can provide a sense reliability on the findings of the research. This can help government agencies make evidence-based decision based on best data and science available. This project aims at automating the process of extraction of source of data. This can make it easy to find out how public data is used in science. To automate the extraction of data source from publications we will be using Natural Language Processing [NLP]. NLP is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data. After researching about the techniques, we decided on using NER, Topic Modeling. NER is used to extract named entities from unstructured text, Topic Modeling is used to discover the sematic structure in a text body. The dataset used for training was provided on Kaggle for the "Coleridge Initiative - Show US the Data" competition. Dataset consisted of full text of 14.3K publications and also labeled data for each publication. We used this data and the techniques, NER with spaCy library, NER with sklearn library, and Topic Modeling to train our models. Best results were provided by the model trained using NER with spaCy. We were able to achieve 0.399 on Kaggle leaderboard. We were really excited with the results and would continue working on the model to improve our results in the competition. Going forward we would try using combination of models to see if we improve our results.

# Introduction

The Coleridge Initiative is a not-for-profit organization, originally established at New York University. It believes that data should be the ultimate factor used to make decisions in public policy, economics, and more. The Coleridge Initiative works with governments to help them make sense of data and how to interpret it. They are the organizers of the, “Show US the Data” Kaggle competition. That was the motivation for this project. The objective of this competition is to use the provided data set of research papers and return the scientific data set referenced in the publications. By automating the retrieval of data from papers, governments will have easier access to evidence for making decisions based on data from these research papers.

# System design & implementation details

The Kaggle competition description suggested that the optimal solution would use Natural Language Processing (NLP) techniques. None of the authors of this paper had experience using these techniques before. To begin, we did a general survey of NLP libraries that we could use to implement a solution [1]. After some deeper exploration of the tools NLP has to offer, we settled on two main approaches: Named Entity Recognition (NER) and topic modeling.

## Named Entity Recognition (NER)

Named Entity Recognition is an information extraction technique in which key words or phrases pertaining to a particular pre-defined topic are identified within a body of text [2]. Common entities that are identified using this technique include people, organizations, times, locations, works of art, etc. The fundamental process itself is quite complex, but in summary it involves using “a deep neural network based on a Convolutional Neural Network (CNN) with a few tweaks” [3]. Readers interested in the detailed neural architecture underlying NER are requested to refer to [4]. The key to any successful NER model is a large, labeled set of training data so that the statistical underpinnings of the model can learn to identify the locations of entities within text. We were delighted to find out about this technique as it seemed particularly suited to the task at hand, namely, to identify the dataset label within a large research paper.

Traditional NER models use predefined entities that are already trained on an available corpus of texts. For this project, we required a way to define a custom entity peculiar to the task at hand. Thankfully, there are certain NER implementations that allow for such custom entity creation. One such library that offers this is the Python library SpaCy [5].

We chose the SpaCy library to implement NER on our data for custom dataset labels. The technology involved in terms of programming was surprisingly simple. We were asked to define a blank SpaCy model and then add an ‘ner’ pipe to the model. The main parameters we had to decide when training the model were a) the batch size for each iteration and b) the number of iterations to train the model and re-calculate losses. After each iteration, losses were calculated based on the difference between the predicted dataset labels and the actual dataset labels in the texts provided.

## Workflow

The workflow we followed for training our SpaCy model is shown in Fig 1. We first took the data from Kaggle and Diagram

Description automatically generatedFig 1. *Workflow Diagram for the SpaCy NER model*

preprocessed it appropriately (more details in the next section). Then, we fed a portion of the preprocessed data to our SpaCy NER model to train on. The other portion of the data was held aside for evaluation purposes. After training the SpaCy NER model on part of the entire dataset, we performed extensive internal evaluation to determine how well our model performed on unseen data. We then adjusted the model as necessary and retrained until we were satisfied with our model’s performance. Finally, we trained a SpaCy NER model on the entire training data available, made predictions on the test data, and submitted the results to Kaggle for evaluation.

Upon reaching the end of this workflow and submitting the results to Kaggle, we realized that Kaggle did not accept a csv file for submission. Rather, they required a notebook. We realized during training that this workflow was a lengthy process; therefore, we uploaded our trained model to Kaggle and created a notebook with the script we used to make entity predictions for test set papers.

## Topic Modeling

Topic modeling is technique that is used to abstract “topics” or themes and ideas from a corpus. Hopefully, topic modeling would be able to abstract out a topic that would contain data or research. In this way the model would be able to distinguish the data/ research portion of the paper from the other analysis, introduction, and conclusion.

To do this we used the nltk, re, pickle, pandas, gensim, and sklearn libraries. These python libraries allowed us to take the provided data set and clean it and work with it. The pandas library allows us to use data frames and manage data easily. The re library is the python regular expression library, It allows us to clean the data. Nltk library allows us to tokenize the words. Pickle allows us to save data frames and import them for later use. Gensim is an open-source library for unsupervised topic modeling and natural language processing, using modern statistical machine learning. Sklearn is a foundational package used for data science. It provides many relevant tools needed for analysis.

# experiments & internal evaluation

## Dataset

The data used for this project was provided by the Kaggle competition. The training dataset consisted of over 14,000 publicly available scientific publications in JSON format that combine to be about 681.1MB. In addition to the papers, there was a csv file containing over 19,000 rows. Each row contained an ID corresponding to the JSON filename of a paper, the name of the paper, and the name of the dataset embedded within the paper (this is what we were trying to train our model to predict). The mismatch of ~19,000 rows in the csv file and only ~14,000 papers provided can be explained by the fact that some papers had multiple datasets within them. As such, these papers were split into multiple rows in the csv file, each corresponding to a single dataset mentioned within a particular paper. Each paper contains on average 7145 words, 578 sentences, and 18 sections.

Kaggle provided us with data that contained no missing pieces. Indeed, dealing with raw data was not the challenge in this project.

The test data provided by Kaggle included four additional publications in JSON format, without dataset labels. There was an additional 8,000 publications in a hidden test set not available to competitors, 12% of which was used to determine the public leaderboard.

## Preprocessing

### Topic Modleing: In order to use this data for topic modeling, a couple of pre-processing steps need to be taken. We start with turning the text of the research paper to just be one large string rather than a JSON object that separates the sections. This would result in a simple pandas dataframe with two columns; the first being the title of the research paper and the other being the entire text of the paper. Luckily, the data given is already relatively clean. However, we can further clean the data by getting rid of punctuation, numbers, and lowercasing all of the letters. Then we tokenize the text by making each word a token. Before placing the tokens in the matrix, we can also get rid of stop words. The sklearn CountVectorizer can be used for creating the document term matrix. The matrix rows contain the document title and the columns contain the words that are within the document. Each of the corresponding cells within the matrix contain the word count of the token words. With the matrix we can use the gensim library to take in the document term matrix and output different topics. The genism provides the input amount of topics, it does so using black box machine learning techniques. Inputting the document term matrix and a number of the topics needed will return groups of words equivalent to the number of topics desired. These groups of words are what the model sorted words into and deemed to be a topic.

### SpaCy NER: To be able to train a SpaCy NER model to locate a dataset label within a full text publication, we first had to preprocess the data to convert it to a format compatible with the SpaCy framework. The format required was an array of tuples [6]. Each tuple would have to contain a) a string of text and b) an “entities” dictionary containing the starting and ending index locations of the dataset labels (i.e. entities) within that that string. If a particular entity was located multiple times within the string, all locations would have to be provided in the “entities” dictionary for that particular string.

In terms of preprocessing, our initial approach was to feed our model all of the publication text available. We did not want to perform common NLP preprocessing techniques such as stopword removal or lemmatization because we thought that all this additional textual “noise” included in the scientific papers in our training set would also be present in the real-world test examples that our model would eventually have to face. Ignoring the “noise” and simultaneously extracting a very minute yet key phrase from a large body of text was the ultimate goal of our model. Thus, we felt that any cleaning of the text would defeat our purpose.

However, some preprocessing was required to convert the full publications into a the SpaCy-friendly format described above. To do this, a script was written to combine each publication into an exceedingly long, single string, and then to use a string search algorithm to find the location(s) of the dataset label within that publication. One can realize just how long these single strings for each publication were by imagining this entire paper to be one long string! Additionally, imagine finding the exact start and stop locations of the word “Kaggle” within this text! As one can see, this was a lengthy process, especially because we were running through this process for ~19,000 papers. Due to the length of this process, we decided to save our SpaCy-friendly preprocessed data to a text file to be read in for future use. That text file turned out to be over 1 GB size and contain over one billion characters!

After preprocessing the data in this way, we split the data into a training set (70%) and an evaluation/test set (30%) to get an idea of how our model performed on unseen data. We performed this split randomly. We decided not to implement cross-validation or multiple subsampling because the SpaCy NER model has a notoriously slow training time [7], even when we trained on computationally-rich systems. Furthermore, the holdout set method we were using is typically employed when the dataset is large [8].

We then proceeded to train the SpaCy NER model on this data. Keep in mind that we were using data in which each tuple contained a single string which contained an entire research paper. We ran into issues training the model on this shape of the data. The RAM being used was over what we had available and our program repeatedly crashed. Even after significantly increasing our RAM available, we still faced the same issue.

After struggling with this for some time, we decided to try an alternative strategy of preprocessing the data. Instead of fitting an entire publication into one string, we decided to break up each publication into a series of sentences. We created a dataset of tuples containing a sentence-long string and the dataset entities (if any) contained within that string. When we passed this reformatted data to the training phase of the SpaCy model, we found that our training was taking much more time than we had available. We realized we needed to further trim the data we were feeding to our model.

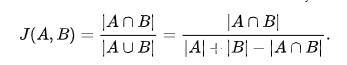
We decided that instead of including every sentence in each publication along with the entities in that sentence, we would include only three sentences from each publication. Two sentences would be ones containing no entities, and one would contain the dataset entity found in that paper. We decided to keep this ratio to more accurately reflect the actual ratio of sentences containing dataset labels and sentences not containing them, while at the same time limiting our data to a reasonable size so our model would be able to train in a reasonable amount of time. However, the ratio of two sentences without entities to one sentence with entities was far more biased towards finding sentences with entities than a real-world distribution would be. Nonetheless, we stuck with this ratio.

Upon splitting our publications into sentences and then passing it to our SpaCy NER model, we were encountering an error due to there being empty strings in the training data. Thus, we wrote a script to remove all blank strings from the training data.

After completing these preprocessing steps, we passed our SpaCy-friendly data to our model. We ran ten iterations with each iteration processing batches of 500 tuples at a time. Each iteration consisted of 67 such sized batches. The entire time required to run ten iterations on this data was approximately three hours. Due to this length of time required, we found a way to save our trained model locally so that we would not have to re-run this process every time.

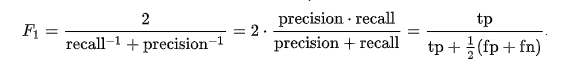
## Evaluation and Results

The objective of the model is to identify the mention of datasets within scientific publications. Jaccard-based FBeta score is used to evaluate the predictions against the ground truth. "The Jaccard index, also known as the Jaccard similarity coefficient, is a statistic used for gauging the similarity and diversity of sample sets [9]" Formula for Jaccard is, where A and B are two sets with discrete values. Jaccard is the intersection of the values of A and B.



(1)

F-score is the harmonic mean of the precision and recall. F-beta is more generic score which applies a positive real factor beta, valuing one of precision or recall more than the other [10]. For our evaluation we use a value of 0.5 for beta.



(2)



(3)

Precision is the ration of correctly classified positive examples to that of total examples classified as positive. Recall is the ratio correctly classified positive examples to that of total number of actual positive examples in the test set. For evaluation of our model, in case of multiple predictions all prediction are delineated using a pipe (|) character and then sorted alphabetically. For each publication's set of predications, a token-based Jaccard score is calculated for each potential prediction / ground truth pair. The prediction with the highest score for a given ground truth is matched with that ground truth.

Predicted strings for each publication are sorted alphabetically and processed in that order. Any scoring ties are resolved on the basis of that sort. Any matched predictions where the Jaccard score meets or exceeds the threshold of 0.5 are counted as true positives (TP), the remainder as false positives (FP). Any unmatched predictions are counted as false positives (FP). Any ground truths with no nearest predictions are counted as false negatives (FN). All TP, FP and FN across all samples are used to calculate a final micro F0.5 score

We can see the performance of the model over several iterations of training through loss graph given in Fig 2. As we can see in the graph there is a big improvement in loss after first iteration. After first iteration there is gradual decrease in loss over each iteration. We used 10 iterations to train the model as the loss curve eventually flattens and there is no significant improvement in results with the increased number of iterations.

We can see performance results of the models in Fig 3 and Fig 5. Fig 3 shows the confusion matrix for the predication results. This shows our model has high precision as most of the true positive identification was correct. On the other hand the recall is not that great for the model as it half of the identifies labels were wrong.

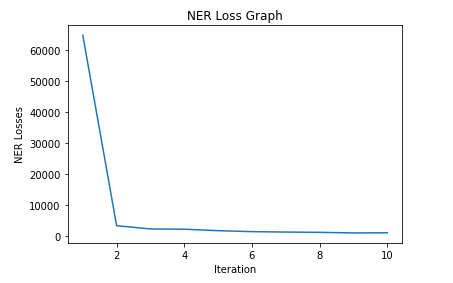


Fig 2. *Model training loss graph*

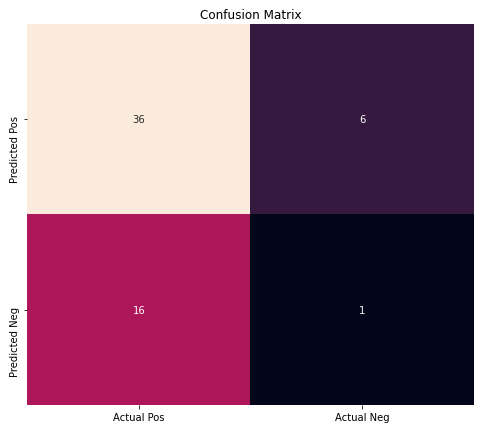


Fig 3. Model Evaluation Confusion Matrix

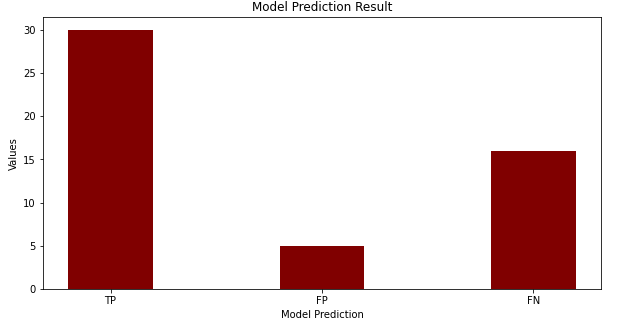


Fig 4. Model Evaluation Result

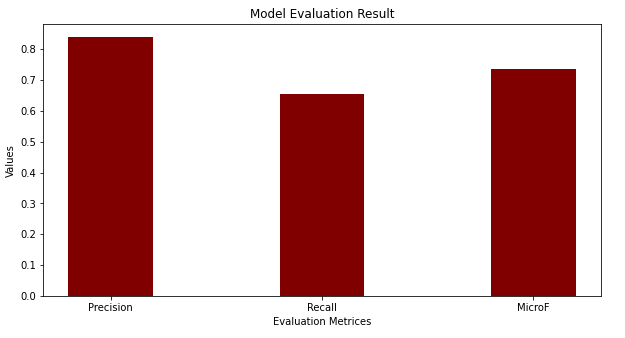


Fig 5. Evaluation Metrices

## Analysis of Results

### Topic Modeling: Upon analysis of these groups, it was clear that they did not seem to follow a clear theme or topic. Due to the black box nature of the gensim library, we could only try to tune some of the variables to try to get grouped data. However, because each of the research papers are on a diverse set of topics, the groups of topics are very limited and contain words that seem to have little to nothing in common. This task of interpreting the results is the hardest part of data science, after analysing the results it was clear that topic modeling was not a good fit for this problem and would not be the solution.

### SpaCy NER: A major limitation in the SpaCy NER training phase was the time it took to train the model. We had to adjust many preprocessing steps and limit the number of epochs to account for this factor. However, as we can see from the graph of model losses over training epochs that even with a limited epoch size of 10, we were able to achieve some type of convergence for the losses at a value of around 1,000. SpaCy does not explicity define the loss function it uses for NER [11], however they mention that a relatively high loss value does not necessarily indicate a poorly trained model due to the way in which they calculate the loss [12]. They suggest using other metrics such as an F-score to more accurately determine the performance of the model.

Our partially trained model achieved an average of 36 true positives (TP), 5 false positives (FP), and 16 false negatives (FN) when making entity predictions on 60 unseen papers. This indicates that about half of the entity predictions our model makes are correct. We feel we could improve this aspect of our model’s performance if we are able to access more computational power to be able to train on more sentences from each paper.

From these confusion matrix values, it follows that our partially trained model achieved a precision score of 0.840, a recall score of 0.655, and a F-score of 0.736. The high precision score indicates that our model does not often make erroneous predictions. When it predicts a dataset label to be present, it is usually present [13]. The lower recall value indicates that the major weakness of our model is that it sometimes misses dataset labels that actually exist in the papers analyzed. These values indicate that our model will predict dataset labels that most likely exist but might miss some that are there.

Based on the current Kaggle competition public leaderboard standings, our F-score of 0.736 would place us in the top 10. In reality, when we trained our model fully and submitted predictions on 12% of the hidden test data, we achieved an F-score of 0.399 on the public leaderboard. We are not sure what caused this huge difference between our internal evaluation results and the public leaderboard results. Public leaderboard results can often be misleading, so we will wait to see the score we receive on the private leaderboard and see if it tallies more with what we achieved in internal evaluation.

# Discussion and conclusions

## Decisions Made/Things That Worked

### Topic Modeling: We decided to use SpaCy over topic modeling as NER was better suited for the task. Additionally, the difficulty of analyzing the results of the topic modeling algorithm left us with little to no choice. SpaCy was not only able to get a result, but it also was able to get a F-score of .73.

### SpaCy NER:The first major decision we had to make regarding NER is which library to use. There are a few NLP libraries capable of performing NER such as NLTK and Stanford NER. We ended up settling on SpaCy because it allowed us to define a custom entity to be extracted by the model, which is exactly what we needed in this project.

A major challenge we faced in training our SpaCy model was determining how to feed each publication into the model. After some trial and error with individual string sizes that were fed into the model, we finally settled on extracting exactly three sentences from each paper: two without a dataset entity and one containing such an entity. In retrospect, we believe this process worked in our favor. SpaCy NER seems to work well when processing smaller rather than larger strings. Furthermore, due to the way we extracted sentences from each paper, our “placebo” sentences (sentences containing no dataset labels) ended up being chosen close in proximity to the dataset-containing sentences (usually the sentences before or after). This, we believe, made our model more robust because it was seeing sentences semantically similar but learning to recognize specific labels from some sentences and not others.

Another aspect of our pipeline that worked magnificently was the SpaCy NER model itself. We found the ability of the model to learn to find “a needle in a haystack” (the dataset label withing a sentence) truly remarkable. The power of deep learning and CNNs were manifested in the power and simplicity experienced when using this model. Furthermore, making predictions on the test data turned out to be a relatively trivial programming exercise due to the well-designed architecture of this library.

We also feel we picked the correct epoch size for model training. At 10 epochs, our loss seemed to converge to a much lower number than it was initially.

We were very happy with the precision score we received in internal evaluation, which suggests that when our model predicts, it generally predicts correctly. Even the recall score we achieved was not totally abysmal. Additionally, we were very pleased the F-score we achieved in our evaluation. This was our first foray into NLP, not to mention NER, for all of us. To achieve an internal evaluation F-score that would place in the top-10 on the Kaggle public leaderboard is something we found truly remarkable.

## Difficulties Faced/Things That Didn’t Work Well

### SpaCy NER: The main difficulties we faced were related to lack of computational power, both in terms of processing speed and memory available. The SpaCy NER model is a state-of-the-art deep learning process. As such, it is reasonable to expect it to require massive amounts of processing power to execute optimally.

The first stumbling block we faced occurred in the preprocessing phase. As mentioned in Section II, we were required to convert the raw publications from JSON format to a SpaCy-friendly format, which was an array of tuples. We initially converted each publication into one large string. This resulted in our training phase causing our RAM to exceed capacity and our program to crash. Ideally, we would like our model to look at entire papers and extract entities from that large chunk of text, because this is the main purpose of this competition. However, we had to settle to breaking up each publication into sentences.

We initially wanted to use all the sentences in each paper. This is because our model would get an accurate idea of how sparse a dataset label is within a large publication. When we attempted to feed all the sentences from each paper to our training phase, the process of training ended up being extremely slow. It was not at all practical to continue in this fashion. As such, we had to further settle by extracting only three sentences from each publication. As mentioned in Section II, this was a heavily optimistically biased ratio of dataset labels to non-dataset labels in the training text.

While the SpaCy NER model was extremely powerful in predicting entities in unseen text, it was quite a time-consuming process to train it. Other users of this library online have noted that this model is notoriously slow in its training phase [7]. This speed limitation prevented us from doing extensive hyperparameter tuning and cross-validation. Doing so might have significantly increased the performance of our model.

Another difficulty we faced is the time required to use our trained model to predict on unseen, but labeled, “test” data. We kept aside approximately 6,000 publications to evaluate our model internally. This was approximately 30% of the entire training data available to us. However, when making predictions, we were required to break up the test publications into sentences and make entity predictions for each sentence, since this was the type of data our model was trained on. As you may recall, we were forced to only extract three sentences from each paper in the training phase due to the exorbitant amount of time the SpaCy NER model took to train. When conducting internal evaluation, we did not have the same luxury because we wanted to see how our model performed on real-world data. As it turned out, our model took too long to predict entities for 6,000 papers broken into sentences. Indeed, just predicting entities for 60 papers (1% of the test data) took approximately 10 minutes. We settled on using only a random 1% of the test data to evaluate our model’s performance. We sampled a random 1% (60 full publications) five times and then took the average over all samples to get our internal evaluation results.

Our Kaggle public leaderboard F-score did not reflect what we achieved in internal evaluation. We will wait and see whether the private leaderboard reflects a similar mismatch.

## Conclusion

Text data are often very bulky and unstructured. They need a lot of preprocessing before they can be used to train models. In our case we found the NER technique was best suited for the task at hand and spaCy library provides built in model for NER along with the flexibility to define custom labels for the entities. Given our inexperience in the field of NLP we were really excited the results we were able to obtain. Our trained model was comfortably able to cross the baseline score of the competition. Going forward we are planning to try combination of models and also try other NLP techniques improve the training data structure. It would be interesting to see how far we can go in this competition and where do we stand at the end of it.

# Project Plan/Task Distribution

| Task Descriptions & Assignments | | |
| --- | --- | --- |
| Task Description | Assigned To | Completed By |
| Shared ideas with each other about our individual research in how to extract dataset labels from a publication using NLP. Akash proposed and researched SpaCy. Karanbir proposed and researched Topic Modeling. | All | All |
| Created GitHub repository and linked it to the Google Drive with data | Akash | Akash |
| explored using the NLP technique of topic modeling as a solution | Karanbir | Karanbir |
| Dataset exploration to find relevant anomalies and statistics | Karanbir | Karanbir |
| (Preprocessing for SpaCy) Wrote script to convert raw data to SpaCy friendly format and saved to text file | Akash | Akash |
| (Preprocessing for SpaCy) After encountering RAM issues due to data size, wrote a script to compress and data so that key sentences were extracted out of entire papers. Further preprocessed sentences by removing blanks. Confirmed that SpaCy was able to process this data | Akash | Akash |
| (SpaCy model) Added comments on preprocessing and SpaCy model training scripts | Akash | Akash |
| created a document matrix of the text and its content to run into the model | Karanbir | Karanbir |
| Trained SpaCy Named Entity Recognition (NER) model on 70% of the training data to identify dataset names in sentences containing datasets (also used sentences without datasets to balance the class distributions). Due to lengthy time of training the model, model was saved to our shared Google Drive folder | Akash | Akash |
| (SpaCy Internal Eval.) Used 70% trained NER model to predict dataset labels for test data | Sudanshu | Sudanshu |
| SpaCy Internal Evaluation Confusion Matrix Graph | Sudanshu | Sudanshu |
| SpaCy training losses per iteration graph | Sudanshu | Sudanshu |
| SpaCy Averages graph | Sudanshu | Sudanshu |
| (SpaCy Internal Eval.) Extracted a Jaccard-based FBeta (Beta = 0.5) confusion matrix of the SpaCy model’s predictions on unseen “test” set data. From this, got the final micro F0.5 score used for internal evaluation purposes (this is how we are evaluated on Kaggle) | Sudanshu | Sudanshu |
| Determined that the different topics were too vague and topic modeling was not a viable solution to the problem. | Karanbir | Karanbir |
| Restored data and started training a model on full training data, wrote scripts for outputting Kaggle-friendly csv submissions with the model’s predictions | Akash | Akash |
| Created Kaggle submission notebook and submitted partially trained model as first valid submission (0.399 micro F0.5 score, 549th out of 596 on Kaggle competition leaderboard based on 12% of overall test data) | Sudanshu | Sudanshu |
| Report Abstract | Sudanshu | Sudanshu |
| Report Introduction | Karanbir | Karanbir |
| Report: System Design and Implementation – SpaCy | Akash | Akash |
| Report: System Design and Implementation – Topic Modeling | Karanbir | Karanbir |
| Report Dataset description and Stats | Akash & Karanbir | Akash & Karanbir (Akash did overall description, Karanbir did exploration/stats) |
| Report Preprocessing Decisions | Akash & Karanbir | Akash (StaCy) and Karanbir (Topic Modeling) |
| Report: Evaluation Methodology | Sudanshu | Sudanshu |
| Report: Analysis of Results | Akash (SpaCy) & Karanbir (Topic Modeling) | Akash (SpaCy) & Karanbir (Topic Modeling) |
| Report: Things that worked/didn’t work | Akash (SpaCy) & Karanbir (Topic Modeling) | Akash (SpaCy) & Karanbir (Topic Modeling) |
| Report: Conclusions | Sudanshu | Sudanshu |

##### References

1. “5 Heroic Python NLP Libraries,” *EliteDataScience*, 09-Jun-2020. [Online]. Available: https://elitedatascience.com/python-nlp-libraries. [Accessed: 26-Apr-2021].
2. C. Marshall, “What is named entity recognition (NER) and how can I use it?,” *Medium*, 02-Jun-2020. [Online]. Available: https://medium.com/mysuperai/what-is-named-entity-recognition-ner-and-how-can-i-use-it-2b68cf6f545d. [Accessed: 26-Apr-2021].
3. “Which Deep Learning Algorithm does Spacy uses when we train Custom model?,” *Stack Overflow*, [Online]. Available: https://stackoverflow.com/questions/60381170/which-deep-learning-algorithm-does-spacy-uses-when-we-train-custom-model. [Accessed: 26-Apr-2021].
4. G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, “Neural Architectures for Named Entity Recognition,” Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2016.
5. “spaCy · Industrial-strength Natural Language Processing in Python,” *Industrial-strength Natural Language Processing in Python*. [Online]. Available: https://spacy.io/. [Accessed: 26-Apr-2021].
6. K. Jaiswal, “Custom Named Entity Recognition Using spaCy,” *Medium*, 18-Apr-2019. [Online]. Available: https://towardsdatascience.com/custom-named-entity-recognition-using-spacy-7140ebbb3718. [Accessed: 26-Apr-2021].
7. “Speed up Spacy Named Entity Recognition,” *Stack Overflow*, [Online]. Available: https://stackoverflow.com/questions/49702372/speed-up-spacy-named-entity-recognition. [Accessed: 26-Apr-2021].
8. M. Eirinaki, “CMPE 256\_3\_Evaluation methods (part 2)” Magdalini Eirinaki, San Jose.
9. “Jaccard index,” *Wikipedia*, 24-Apr-2021. [Online]. Available: https://en.wikipedia.org/wiki/Jaccard\_index. [Accessed: 03-May-2021].
10. “F-score,” *Wikipedia*, 24-Mar-2021. [Online]. Available: https://en.wikipedia.org/wiki/F-score. [Accessed: 03-May-2021].
11. “How to understand 'losses' in Spacy's custom NER training engine?,” *Artificial Intelligence Stack Exchange*, 1AD. [Online]. Available: https://ai.stackexchange.com/questions/25627/how-to-understand-losses-in-spacys-custom-ner-training-engine. [Accessed: 01-May-2021].
12. “How is the Loss function calculated in spacy NER?? · Issue #5392 · explosion/spaCy,” *GitHub*. [Online]. Available: https://github.com/explosion/spaCy/issues/5392. [Accessed: 01-May-2021].
13. M. Eirinaki, “CMPE 256\_3\_Evaluation methods (part 1)” Magdalini Eirinaki, San Jose.