Negation and Uncertainty Detection - Group 1

Marino Oliveros Blanco NIU: 1668563
Andreu Gascón Marzo NIU: 1670919
Pere Mayol Carbonell NIU: 1669503
Judith Zaragoza López NIU: 1634071

Github repository link:
https://github.com/marinocom/NLP-Detection-of-Negation-and-Uncertainty-Project-24

Index

Rule-Based Method Report

Index Rule-Based

References

Here is a brief introduction to the references considered on our implementation. We were invested in using a NegEx[1], RegEx focused on the Negation and Uncertainty detection system for our task in combination with an UMLS, we found difficulties finding a UMLS in Spanish or Catalan, so we conducted extra research stumbling upon Cutext[2], Construction of medical terminological resources for Spanish. The github repository lacks an UMLS in itself, so we decided to create our own. More on this is in the model implementation tab.

[1] Sanamaría, "NegEx-MES."
[2] Santamaría Martínez and Krallinger, "Construcción de Recursos Terminológicos Médicos Para El Español: El Sistema de Extracción de Términos CUTEXT y Los Repositorios de Términos Biomédicos."

Data Pre-Processing

Our preprocessing implementation is aimed at ensuring repeatability through all methods. To assure this, the pre-processing pipeline is executed through a caller function with the ability to turn on or off certain functions. We will provide evaluation results with different functions being called or not to see the effect on our results. Following the mantra of an ablation study. The data was loaded using the JSON library. It is important to clarify that due to how our model evaluation is set, we first have to tokenize and, afterward, remove patient information.

Our pipeline is divided in four:

1. Removing patient information
2. Removing punctuation
3. Spell checking
4. Tokenization

Removing patient information

Here, the pipeline removes important patient information, ensuring the privacy of those patients and reducing the possible number of false positives, using regex. To be exact, we remove all redacted entries by removing all asterisks, (*) as well as removing sentences that begin with ‘nº historia clinica:’ and end with ‘motiu d\'ingres’ and all sentences that begin with ‘nhc’ and end with ‘lopd’. Those data scopes correspond to sensitive information referring to patients.

Removing punctuation

This part of the pipeline can be turned on/off to compare how it affects results and tokenizing itself, as maybe ‘all’ punctuation should not be removed, this is a topic that we will explore in the following entries. Our function utilizes Python's str.maketrans() method to create a translation table that maps each punctuation character to None. Then, it applies this translation table to the input text using the translate() method. After some consideration for the evaluation part, we decided that in order to match the scopes with the JSON object, not all punctuation could actually be removed.

Spell checking and lemmatization

This method was implemented to first detect which language the word was in, then correct it based on a spell checker that changes from ‘es’ to ‘cat’ based on the language detected. Then it tokenizes the input text into the appropriate language model. Each token is then checked for any punctuation or whitespace; if it's a valid token, it gets spell checked and lemmatized, then reconstructed back into a string.

The problem that arises with this function is its long compute time; this could be due to a bad implementation or the size of the dataset. That is why, for the results we provide in this report due on April 29th, the function was turned OFF during the preprocessing.

Tokenization

First implementation

To tokenize the text and also provide the coordinates, we used RegEx. The function iterates over every token detected in the input text by making use of re.finditer(r'\S+', text). This regular expression \S+ matches one or more non-whitespace characters, effectively tokenizing the text. For each token found, the function extracts its text using token.group(0). The end position is then computed by adding the length of the token text to the starting position (token_start + len(token_text)).

To preserve the coordinates, the function stores the token text and its start (token_start), and end positions (token_end) in a tuple (token_text, token_start, token_end) in the list tokens_with_coordinates. After processing each token, the function updates the token_start variable to the end position of the current token (token_end). This ensures that the next token's start position is correctly calculated relative to the end position of the current token. Obtaining a nested list of all files and their tokenization with coordinates.

Definitive implementation:
At the end to match with the JSON object coordinates we ended up using spacy tokens, for the tokenization of the text.

Additional preprocessing actions
Removal of empty tokens was performed also the change from accent mark letters ‘á’ to letters without them ‘a’, this was performed using unidecode.

Target Variables & JSON Object Handling
The JSON object contains the ‘groundtruth’/’target’ that will be used for the supervised learning, the data and its labels. To extract this from the text, we handled the JSON object by processing the 254 documents extracting the data for each prediction in the item: obtaining the labels, start index, end index, and text segment of the annotation. If the label contains "NEG", it adds the text segment to the negations list. It then tries to find the scope of the negation by looking at adjacent annotations marked as scope ("NSCO") before or after the current negation. If found, it adds the start and end indices of the scope to the negation_scopes list. Similarly, for annotations containing "UNC", it adds the text segment to the uncertainties list and tries to find the scope of uncertainty and appends the start and end indices to the uncertainty_scopes list. After processing all annotations, duplicates are removed from the lists of annotations and their scopes. Obtaining a list of the ‘target’ variables and their coordinates.

Implementation Strategy

The strategy to implement the rule base method is based on finding the scopes of the different tags in the texts, that means to look for the initial and final indexes of each word or phrase that is predicted as a NEG or UNC entity. The same with the respective tokens they affect (NSCO and USCO). We use these indexes to evaluate the model with the training and test set.

Predefined Data

This rule based method uses predefined phrases and words about medicine and diagnoses to be able to detect them and to know if they are denied or not.

In the similar way, the negation and uncertainty words are predefined, but with a modification: here we distinguish between post/pre sentences. Post negation/uncertainty phrases are those that deny the following tokens, while pre negation/uncertainty are those that deny the previous tokens.

To choose the medical words, we have used several methods, both GitHub repositories with predefined words, and generative AI text.

The negation and uncertainty phrases have been extracted from the training set itself, taking into account the tags and indexes of the different texts. In addition to the training set, we have manually added more sentences that seemed interesting for a possible test set with data not seen in the training set.

Functions and usage

The two main functions (negation detection and uncertainty detection) have a similar operation, although one deals with negations and the other with uncertainties. We will focus on the negation function since they perform almost equally.

The first function iterates over the tokenized texts and checks with the phrase matching function (which we will explain later) if any negation of the total list (pos+pre) has been found.

If yes, it sees if it is a pro or post negation.

In case of a post denial, we look at the next 5 tokens and check if we find any medical word. In case of finding a medical word, it means that this pathology is denied, so we will add to the lists the indexes of the denial and the following tokens respectively.

If it is the case of a pre-negation, we will do the same process but checking the 5 previous tokens.

Secondary Function (phrase matching)

This function is used by the two functions to decide whether a negation or uncertainty phrase has been found in the text.

To make sure that we do not return the indices of the first (or shortest) negation word of a negation phrase (e.g., return 'no' instead of 'does not appear') we make sure to look at the longest phrases first by using sort by sentence length).

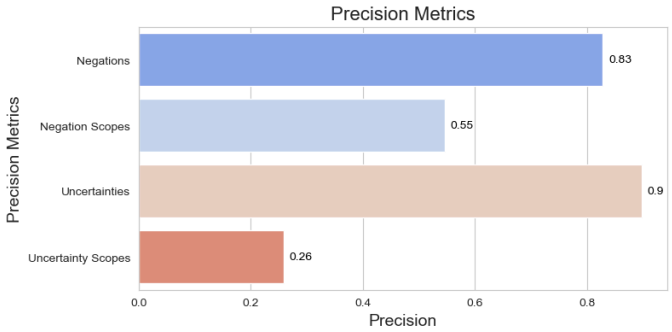
When it finds a negation or uncertainty phrase, the function return (yield) True, the negation found, and the start and end indices that will later be used to find the scope of the negation.

Model Evaluation

In the model evaluation, a comparison was made between the scopes extracted from the JSON object file and the output of the model.

To make sure the model works out, various scoring metrics were implemented. Firstly, to check accuracy we will use two separate lists, one with the predicted tags for a sequence of words and then a list for the

words we consider fall inside the scope of what has to be detected. This code simply iterates through the words in the prediction and their tags and checks if the word exists in the second list.
The problem with scopes in general is that sometimes the predicted scope falls outside the ground truth by a small error, even though it has been predicted properly. We can fix that by allowing the code to not only look for specific scopes we know are good, but also using a broader scope for them so they are detected correctly as well. By doing this, our performance improves considerably.



The next approach has been to compute the Recall, this has shown that for the Scopes in general, is very low compared to the Precision obtained.

```
RECALL:
Recall for Negations:  0.8483754512635379
-------------------------------------------
Recall for Negation scopes:  0.0003933019111326466
-------------------------------------------
Reall for Uncertanties:  0.9516129032258065
-------------------------------------------
Recall for Uncertanties scopes:  0.00025467732003436243
```

To finish the analysis of the Rule-Based model, the computation of the F1-Score has provided the remaining information. This way of analysis, as it makes use of both previous ones, shows the decompensation between Precision and Recall previously mentioned. The F1, as it uses both of them, reflects that for the negations and uncertainty words are pretty high, while on the scopes, is much lower due to the Recall.

```
F1 SCORE:
F1 score for Negations:  0.8483754512635379
-------------------------------------------
F1 score for Negation scopes:  0.0007861499250799122
-------------------------------------------
F1 score for Uncertanties:  0.9516129032258065
-------------------------------------------
F1 score for Uncertanties scopes:  0.0005080780620234398
```

Conclusion: Insights for the future

In conclusion, although the model can classify most of the individual words, it has issues on the identification and classification of the whole scopes in the text.

To improve the whole model performance, we have to improve the ambiguity support of the model, as in some cases, a negation or uncertainty can be used as pre-uncertainty or post-uncertainty, this may cause the incorrect classification of the scopes. We have found out that due to a lack of depth on the UMLS can limit the capacity of the model, to recognize and classify the relevant words or scopes associated. This leads us to a misclassification due to a lack of comprehension on specific terms in medical context. The last point found that causes the main issues on the model and that lowers the performance of it is the JSON file tagging. These files can be a limitation on the model performance to capture in a precise and complete way the scopes. If the tagging system is not correctly optimized to reflect the medical text and its structure, the model can have difficulties to learn.

Who did what Rule Based Approach?

Code:

7. Data Pre-Processing: Marino
8. Model Implementation: Andreu
9. Model Evaluation: Pere & Judith

Report: Marino, Andreu, Pere, Judith

Machine Learning Method Report

Index Machine-Learning

Introduction

Our next approach would be that of using Machine Learning to detect the negation and uncertainty queues as well as their scopes. To achieve this, we conducted thorough research to devise an effective method. To describe our approach in the most straightforward way, we first have to take into account that we are looking to achieve Cue identification and Scope detection. We can describe it as a process composed of different parts:

1. Data loading & Preprocessing
    a. *IMPORTANT NOTE*: The data is pre-processed using the same preprocessing function as in the Rule-Based method.
2. Data annotation (label creation)
3. Feature Extraction
4. Model training
5. Model evaluation
    a. Validation set & K-Fold Cross-Validation
    b. Scoring

In the 'Negtool' paper[3] we studied the possibility of performing Cue detection using Support vector machines (SVMs) and although this idea seemed to work we decided to go with a Conditional Random Fields (CRF) approach as CRF could also be used for the Scope detection as stated both on the UiO2[4] and the Negation Cues Detection using CRF on Spanish Product Review Texts[5] papers.

[3] Enger, Velldal, and Øvrelid, "An Open-Source Tool for Negation Detection: A Maximum-Margin Approach."
[4] Lapponi et al., "UiO 2: Sequence-Labeling Negation Using Dependency Features."
[5] Loharja, Padró, and Borras, "Negation Cues Detection Using CRF on Spanish Product Review Texts."

So why use CRF?

For Cue Detection, CRFs can be trained by using various features extracted from the text. These features can include lexical, syntactic, and contextual information such as word embeddings, part-of-speech tags, and dependency parse trees. This will be explored further in the Feature Extraction section.
In negation and uncertainty cue detection, the task is often to label each word in a sentence as either belonging to a cue or not. CRFs perform well in learning the sequential dependencies between words in a sentence, allowing them to model the relationships between adjacent words and infer the presence of cues based on the context. This information leads us to annotate our data using a version of BIO tagging that we will dive further later.

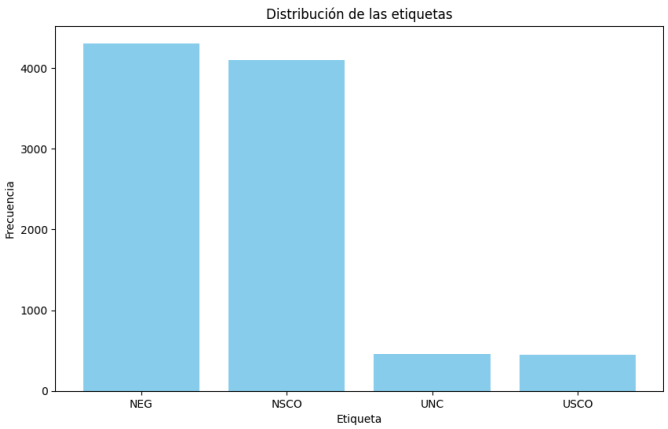
For scope detection, CRFs can model the dependencies between labels in a sequence. In the case of scope detection, the task is to determine the extent or scope of the negation or uncertainty triggered by a cue. CRFs can learn to predict the boundaries of these scopes by considering the sequential relationships between words and their corresponding labels.
Also, CRFs have the ability to estimate the transition probabilities between labels, allowing them to capture patterns in how negation or uncertainty scopes typically unfold in text. For example, a negation cue often marks the beginning of a scope, and CRFs can learn to recognize this pattern and adjust the scope boundaries accordingly.

Data analysis

In order to carry out a deep analysis of the data that has been provided to us, we wanted to make a first approach to data analysis. With this, we have been able to improve performance and know what type of data we are handling.

First of all, we have carried out the analysis with the training file. The main reason for this is that we were going to train our CRF model with it, so we had to know how we were going to treat the data. We have counted how many labels we had for each of them, and we have found a total of:

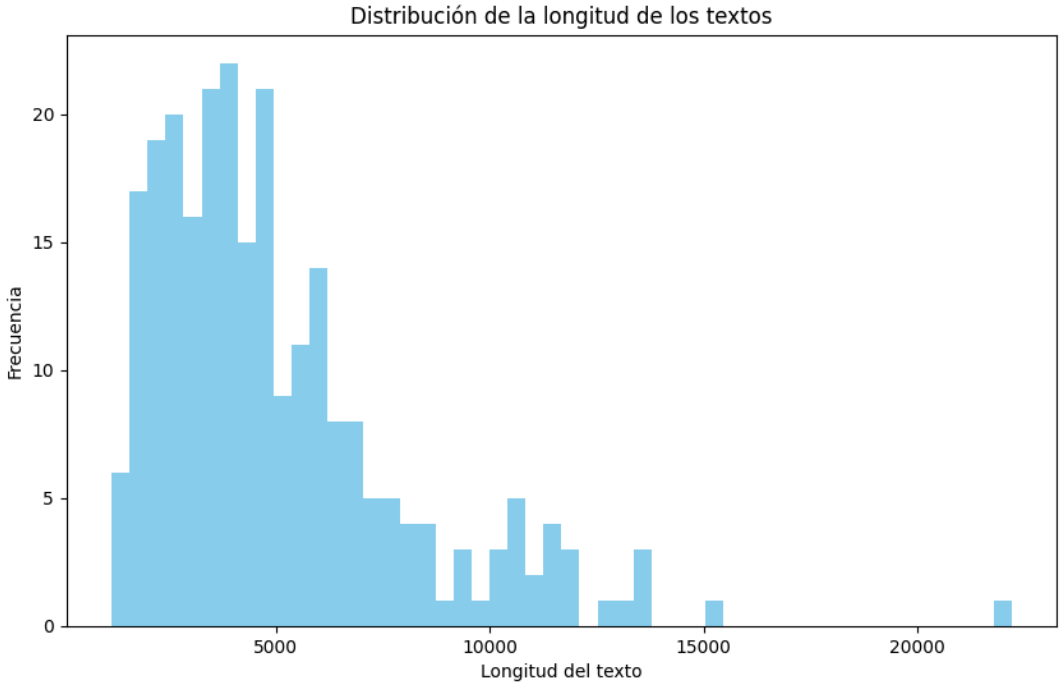
	Label	count
0	NEG	4307
1	NSCO	4103
2	UNC	458
3	USCO	451



After that we asked ourselves what type of data we had in our DataFrame, since later we will have to manage everything, so we extracted a small summary of it with the result of:

```
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   data         254 non-null    object
 1   annotations  254 non-null    object
 2   predictions  254 non-null    object
dtypes: object(3)
```

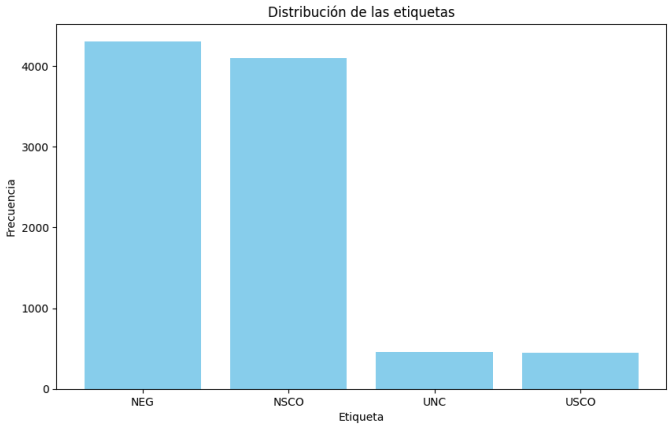
This tells us that we have a length of 254 data with 3 types of objects within each of them. In this data, none of the values are empty, giving us a complete data frame of information to work with. Since we had non-empty values, we checked the length of the texts they gave us. In this way, with all the information we will be able to carry out a better analysis of each of the texts and, with this, train the model better.



After knowing the distribution of the training data, we needed to know if the test data would be suitable for testing our model. If there are some extra tags or none of the training tags match, we will not be able to adapt our CRF model in any way if we do not modify the base JSON file.

Therefore, we also perform a general data analysis with this other file. We found that for tags we had less information for each of them, but overall it was proportional to the amount of data we had. This told us that the data split was done correctly, since it will be appropriate to train a model with those proportions.

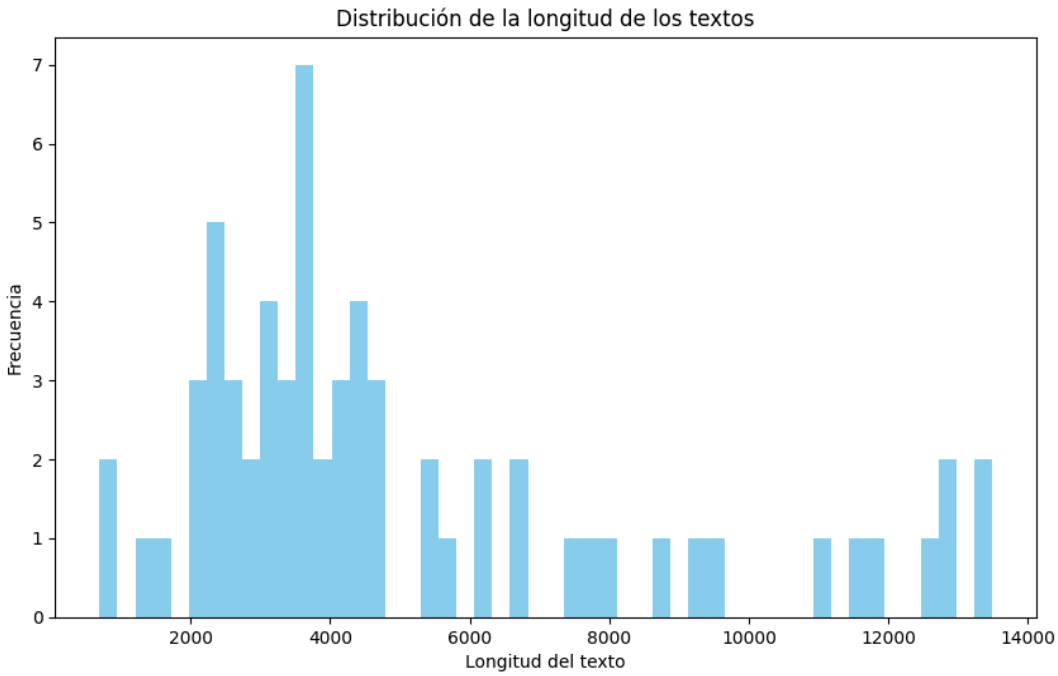
	Label	count
0	NEG	1132
1	NSCO	1074
2	UNC	131
3	USCO	129



Regarding the length, we found that there are obviously fewer entries in the test JSON file. In the test file we found a total of 64 entries compared to the 254 we mentioned we had in the other file. With that data we had to verify that the objects and all the other information were of the same type as in the other file, and as we had verified everything was configured correctly, since they were objects of the same data type.

```
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   data         64 non-null     object
 1   annotations  64 non-null     object
 2   predictions  64 non-null     object
dtypes: object(3)
```

Regarding the text extension, it is true that we find the main difference between the files. We found that in the training file the texts were longer than in the test file. Furthermore, we found that the distribution was greater in this second file, the texts we had were more dispersed in length, with shorter text samples and more texts of 4,000 words, while in the training file the largest amount of text was around 5,000 words.



The data analysis performed has provided a deep understanding of the nature and characteristics of the training and testing data sets. Firstly, by examining the distribution of labels in the training set, we have observed an imbalanced proportion between the different labels, which indicates that the data set is not highly representative in terms of the categories to be predicted. This information is crucial for developing machine learning models, as an imbalance in labels could bias model performance toward dominant classes as we will see in further steps.

Furthermore, when analyzing the length of the texts in both data sets, a notable difference has been revealed. While training texts tend to be longer and focus on around 5,000 words, test texts show a more dispersed distribution, with significant variability in their length. This finding suggests that the model must be able to effectively handle texts of different lengths during training and testing to ensure its robustness and generalization to unseen data.

Finally, consistency in data structure and format between training and test sets is critical to successful model development. Consistency in data organization facilitates the implementation of model training and preprocessing processes, leading to greater efficiency and effectiveness in model development. Together, this comprehensive analysis provides a solid foundation for creating machine learning models that can take full advantage of the information available in training and testing data sets to make accurate and generalizable predictions.

Data annotation

This along with the model section is our most ablation-esque entries, as we encountered many different ways to do it.

To create the labels for our 'y_train' we have to ensure every word gets tagged. At first, we devised a technique to tag every word that is not NEG or UNC cue with 'O'; something that was maintained over all our approaches. Then we took our lists of cues from the Rule-Based Approach, tagging each word that matched as NEG-PRE, NEG-POS, UNC-PRE and UNC-POS. After seeing no increase in our models' performance, we settled for a simpler NEG, UNC cue tagging.

BIO tagging was used, at first, for the scope handling as the B could stand for B_NEG or B_UNC depending on whether it was negation or uncertainty. Then we marked the words from the beginning of the scope to the end of it, and O for words that were none of the aforementioned classes. After some thought, we devised the technique to mark them as NSCO, USCO directly.

Example:
"No se descarta la posibilidad de una infección."
Previous approach:
['NEG-PRE', 'O', 'NEG-PRE', 'O', 'O', 'O', 'O', 'O']
Current approach:
['NEG', 'O', 'NEG', 'O', 'O', 'O', 'O', 'O']

"No se descarta la posibilidad de una infección. Insulina 7mg"
Previous approach:
['B_NEG', 'I', 'I', 'I', 'I', 'I', 'I', 'I', 'O', 'O', 'O']
Current approach:
['NEG', 'NSCO', 'NSCO', 'NSCO', 'NSCO', 'NSCO', 'NSCO', 'NSCO', 'O', 'O', 'O']

Feature Extraction

This idea was based on the 3.3 Learning The Model for Negation Cue Detection using CRF section of the Negation Cues Paper[6]. To use the NLTK CRF tagger as our model we need to train it using featured data. We extracted 16 different features ranging from the most basic (the vocabulary of the word) to POS tagging and bigrams. We encountered multiple problems on the after and before POS tagging functions as the POS tagging had mix ups because of the language changes we tried to use the spell checker but after removing it it worked just fine, we also encountered some scope issues that by doing printouts of the code we ended up fixing.

In total, we have 16 Feature functions which include:
WORD: the vocabulary of words. "paciente" (string)
POS: the information of part of speech of the word "NOUN" (string)
INIT CAP: word starts with capitalization. (True or False)
ALPHANUM: A word consists of alphanumeric characters. (True or False)

[6] Loharja, Padró, and Borras.

HAS NUM: word contains a number. (True or False)
HAS CAP: word contains a capitalized letter. (True or False)
HAS DASH: word contains dash (-). (True or False)
HAS US: word contains underscore (_). (True or False)
PUNCTUATION: word contains punctuation. (True or False)
SUFn: suffixes in the n character length ranged from two to four. (‘paciente’ = 'te', 'nte', 'ente')
PREFn: prefixes in the n character length ranged from two to four. ('paciente' = 'pa', 'pac', 'paci')
2GRAMBEFORE: bigram of the word itself and the previous one ('paciente' = ' el paciente')
2GRAMAFTER: bigram of the word itself and the next one ('paciente' = 'paciente presenta')
BEFOREPOS: the information of part of speech of the previously observed word. ('paciente', 'DET')
AFTERPOS: the information of part of speech of the word after the observed word. ('paciente', 'VERB')
SPECIAL: word is one of the special words in the special dictionary. (True or False)

The extraction functions create a dictionary of features for each tokenized word, stored in a list with the rest of the token features in the given text. Finally, this lists (one for each text) is appended to another list that fit to the CRF model.
The input to the model in summary is a nested list of dictionaries.

for the start and end of the text we created two new features [‘BOS’] and [‘EOS’] standing for the beginning and end of the sentence. In this way, we ensure no eros when computing before and after features like bigrams and part of speech

Model

Feeding data into the model

The model is trained with X_train (features) and y_train (labels) from the training set. And evaluated at first using our Validation set. The functions used to train are train_crf from the python_crfsuite library.

Model structure & Fine Tuning

We are using the CRF function from the python_crfsuite library. This model contains the following parameters to be fine-tuned:

1. *Algorithm*: the optimization method used to train the CRF model. - **‘lbfgs’** (the best option for us as it is suited for small-medium sized datasets, like ours)
2. *C1*: controls the L1 ‘lasso’ regularization strength. - small value for a simpler model like ours, **0.1**
3. *C2*: controls the L2 ridge regularization strength. - **0.1**
4. *Maximum number of iterations*: specifies the maximum number of iterations allowed during training. Training stops when either this limit is reached or convergence criteria are met. - **20** Although the training time is increased, the model’s score is also increased.
5. *All possible transitions*: determines whether all possible label transitions are allowed or whether specific transitions are constrained. - **True**

6. Verbose: parameter used to print detailed information about the training progress to the standard output - **True**

The fine-tuning was done by hand tuning looking at the K-fold results. As we had a lot of data imbalance with the UNC and USCO tags, the model underestimated these labels and misclassified them, that is why we decided to lower both L1 and L2 and increase the number of iterations to see if the model improved and it did although it still classifies negations much better than uncertainty in general as can be seen in the confusion matrix of the validation set in the model evaluation part. in the end, the changed hyperparameters are the following with respect to the initial parameters: c1 = 0.01 , c2 = 0.01, max_iterations = 50.

Model evaluation

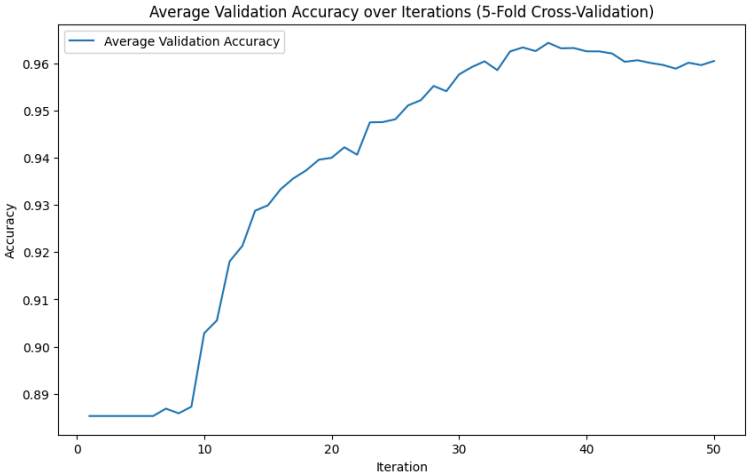
Validation set & K-Fold Cross-Validation

As our data quantity is rather low, we decided to apply certain measures to counter this. We started by creating a Validation Set of 80:20 (training-validation) split, this is done so we can tune hyperparameters and assess model performance during training. The validation set helps prevent overfitting by providing an independent dataset for evaluation.

On another note, we indulged in Cross-Validation; a technique used to assess how well a model will generalize to an independent dataset. Instead of having a single train/validation split, cross-validation involves splitting the dataset into multiple folds, training the model on different subsets of the data, and then averaging the performance metrics across these different folds. So in turn, we could see a better estimate of the model's performance and reduce the variance.

To deal with our limited data quantity, we used K-fold. K-fold cross-validation is a specific type of cross-validation where the dataset is divided into k equal-sized folds. The model is trained k times, each time using k-1 folds for training and the remaining fold for validation. This process is repeated k times, with each fold being used exactly once as the validation data. The performance metrics are then averaged across all k iterations. K-fold cross-validation is particularly useful when the dataset is small or when you want a more reliable estimate of the model's performance.

Here we can see the average validation accuracy using a 5-fold cross-validation:



We see that after some iterations the accuracy starts decreasing possibly due to overfitting, so 50 iterations seems a good option for a future test set prediction.

In the next image we can see the training loss, which also confirms a good performance in predicting around 40 and 50 iterations.

Test Set

Once the hyperparameters have been corrected we will fit the test set to see the real predictions in unseen data. For this we will use the confusion matrix as well as different metrics provided by it such as the f1 score.

We can see although we have a good accuracy on the test set (0.963), it is mostly influenced by O tags, as well as NEG and NSCO. Since UNC and USCO are underrepresented in the train and test set, the model has a harder time predicting them correctly.

From this table we can see that the most affected classes are the most underrepresented (UNC) and (USCO) with very low recall and f1 score values indicating a large number of false negatives in the prediction.

	precision	recall	f1-score
NEG	0.980591	0.925022	0.951996
NSCO	0.952035	0.725430	0.823427
O	0.963926	0.996930	0.980150
UNC	0.941176	0.489796	0.644295
USCO	0.821782	0.168016	0.278992

Conclusion & Insights for the future

After revising and improving our previous report we started working on our Machine Learning approach. Although our model works we think it could be improved further by applying better annotation of our text as well as better overall preprocessing. If we would have had more time we would also have tried Grid Search for parameter optimization and finetuning of our CRF model.

The poor results for UNC and USCO were to be expected since we had very little data for these classes and the model has found it difficult to generalize without overfitting the data.

We started gazing at some future approaches for the Deep Learning Model. These include Bidirectional Long Short-Term Memory (BiLSTM) and CNN models.

Who did what Machine Learning Approach?

Code:

- Preprocessing: Marino
- Data Analysis: Judith
- Data annotation: Marino
- Feature extraction: Andreu & Pere (adapting data types: Marino)
- Model: Marino & Andreu
    - Feeding data into the model
    - Model structure & Fine Tuning
- Model evaluation: Marino & Andreu
- Model correction and improvements: Andreu

Report: Marino & Andreu, (Data Analysis: Judith)

Deep Learning Method Report

Index Deep Learning

Introduction

To affront this challenge we followed a Bidirectional LSTM (BiLSTM) implementation as stated on all followed papers such as Neural Networks For Negation Scope Detection[7], where a thorough explanation of why to use BiLSTMs for this task and the math behind the model. Negation and Uncertainty detection tasks require understanding the context in which words are used, and BiLSTMs are well-suited for capturing contextual information from both past and future words in a sentence. Here's a detailed explanation:

- Negation Detection: Identifying negation requires understanding the scope and impact of negation words (like "not", "never", "no") on the rest of the sentence. For example, in "I do not like apples," the word "not" changes the sentiment of "like". A BiLSTM can process the sentence in both forward and backward directions, ensuring that it understands the influence of "not" on "like".
- Uncertainty Detection: Similar to negation, detecting uncertainty involves identifying words or phrases that indicate doubt or lack of certainty (like "might", "could", "possibly"). The context around these words is crucial to accurately interpret the uncertainty. A BiLSTM's ability to capture information from both directions helps in better identifying these cues.

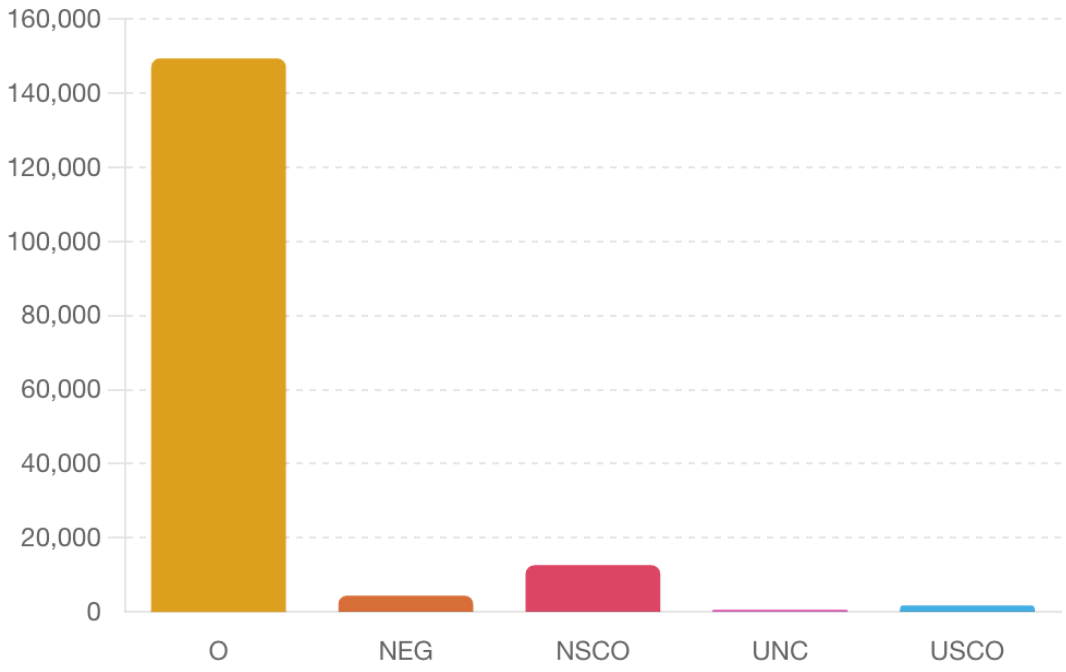
Traditional LSTMs process sequences in one direction, either from start to end (forward) or end to start (backward). This can limit their ability to capture dependencies that span across the sentence in both directions. BiLSTMs, on the other hand, consist of two LSTMs running in parallel, one processing the input sequence from the start to the end and the other from the end to the start. This allows the model to have a more comprehensive understanding of the entire sentence, considering both previous and upcoming words.

[7] Fancellu, Lopez, and Webber, "Neural Networks For Negation Scope Detection."

In summary, BiLSTMs are advantageous for negation and uncertainty detection because they provide a better understanding of the context by processing text in both forward and backward directions. This bidirectional approach allows the model to capture the nuanced impact of negation and uncertainty cues within the sentence, leading to more accurate and reliable detection.

Data Analysis, annotation and ground truth extraction

To obtain the groundtruth we used the same method as in the machine learning algorithm, a tagging method that tracks through the json object and the tokenized text the different NEG, NSCO, UNC & USCO tags, all of the tokens that are not marked as any of the 4 before mentioned tags were marked as 'O'.
A big problem with the data is its low presence of UNC cue words and USCO tagged words, as we can see in this tag data exploration of the groundtruth, this will cause some difficulties that we tried palliating later on.



Data Preprocessing

We cropped the unnecessary bits of the JSON text: that being the beginning patient information in all of its appearances throughout each text and from 'nhc' to 'lopd', so removing the potential false positives. We then treated the text as sentences until punctuation or 200 tokens.

Feature extraction

We extracted the following features from the data, per token:

Raw word: This being the original word in a string format ‘induccion’
Initial Scopes & Final Scopes: Obtained from the tokenization step and given as a tuple. Example: (319, 328).
Tag: This will be used as out y_train/label and it is the tag of the token (O, NEG, NSCO, UNC, USCO)
POS: Part of speech tagging meaning the actual lexical meaning of the word, in context: ‘PROPN, ADP, NOUN’
LEMMA: This being the lemmatized word in a string format ‘semanas’ -> ‘semana’

This features are given a lesser importance in the deciding factor of the model:
Number: If the whole token is numbers ‘1001’ -> 1 ‘10mg’ -> 0
Contains Number: If the token contains a number ‘hola’ -> 0 ‘h0l4’ -> 1
Maj Number: If the majority of items in the token are numbers ‘paraceta1’ -> 0 ‘12.4g’ -> 1

These features are then loaded into a pandas dataframe and later converted into a .csv for convenience, we also included text id (which text from the json dataset are in), the loading process is done both for the train and test set.

feature_extracted_data

Word	Initial Scopes	Final Scopes	Tag	POS	LEMMA	NUMBER	Contains NUMBER	Maj NUMBER	text_id
paciente	315	323	O	NOUN	paciente	0	0	0	0
que	324	327	O	SCONJ	que	0	0	0	0
ingresa	328	335	O	VERB	ingresar	0	0	0	0
de	336	338	O	ADP	de	0	0	0	0
forma	339	344	O	NOUN	forma	0	0	0	0

feature_extracted_dataTEST

Word	Initial Scopes	Final Scopes	Tag	POS	LEMMA	NUMBER	Contains NUMBER	Maj NUMBER	text_id
induccion	319	328	O	PROPN	induccion	0	0	0	0
al	329	331	O	ADP	al	0	0	0	0
parto	332	337	O	NOUN	parto	0	0	0	0
por	338	341	O	ADP	por	0	0	0	0

Model

First Approach

Our model was built on pytorch upon the basis of the Deep Learning approach for Negation cues Detection in Spanish paper[8] where it is explained that our features should pass first through an embedding layer then concatenated and passed through a dense neural network. After we complete this step we then pass our data through a BiLSTM, then a dense neural network and another dense neural network inside a time-distributed wrapper. Given later our output layer.

To train this model we used the GPU functionalities of pytorch and first encoded our data using sklearn's Label encoder, obtained the indexes and handled unknown data adding the 'unknown' tag to those words so the training could be successfully completed. After this we created a Custom Dataset to load our data into the model through the DataLoader. We initialized our model using Cross Entropy Loss and the Adam optimizer, we trained through 10 epochs using our training loop. We trained without the number features as the training was not optimized for them and resulted in nearly untrainable models because of our time limitation, this might have been because of an erroneous pre-processing of said features.

Our training presented quite a bit of overfitting as we can see in our loss graph, so we decided in our revised approach we would try regularization.

[8] Fabregat, Martinez-Romo, and Araujo, "Deep Learning Approach for Negation Cues Detection in Spanish."

Model Evaluation

For this evaluation we are considering first a Validation score, based on an 80:20 split of the training data and the Test score. We are using accuracy, precision, recall and F1 score as metrics. We provide tables analyzing the results for both models.

Validation Score

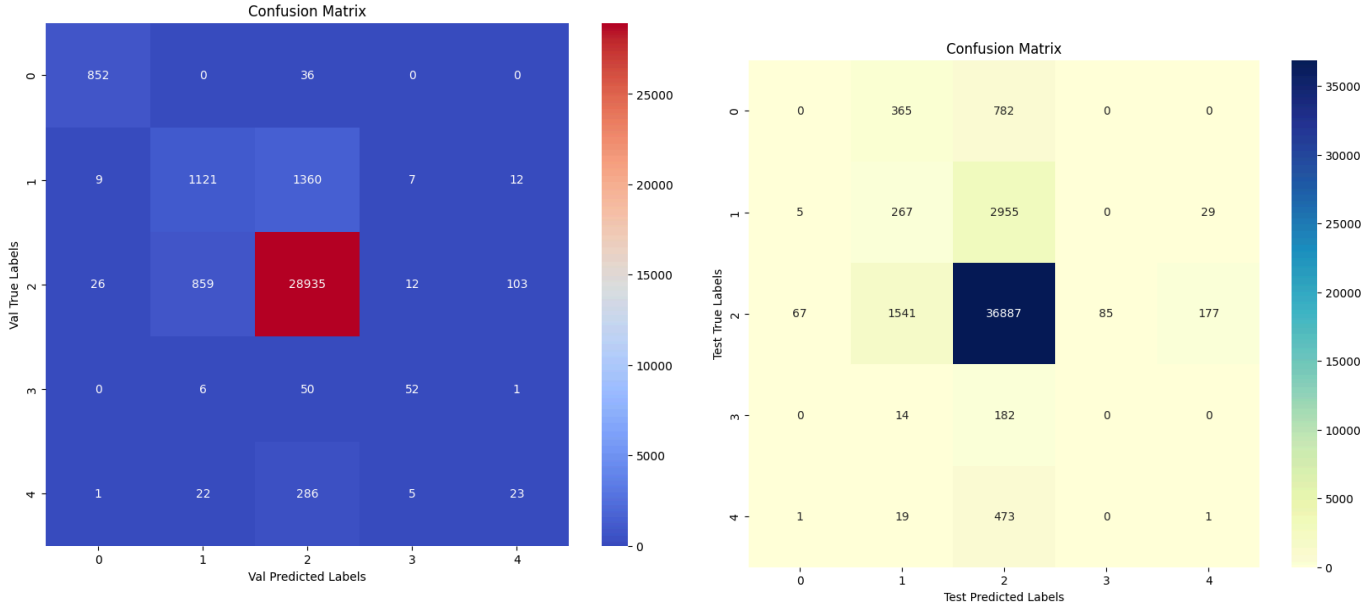
Metric/Model	**Bidirectional LSTM**	**Revised Bidirectional LSTM**
Accuracy	91.72%	76.59%
Precision	90.67%	90.46%
Recall	91.72%	76.59%
F1-Score	91.11%	82.13%

Test score

Metric/Model	**Bidirectional LSTM**	**Revised Bidirectional LSTM**
Accuracy	84.73%	74.55%
Precision	79.88%	90.80%
Recall	84.73%	74.55%
F1-Score	82.20%	80.96%

Confusion matrix

On the index of the confusion matrix 0-'O', 1-'NEG', 2-'NSCO', 3-'UNC', 4-'USCO'. The blue and red confusion matrix refers to the Validation Set and the yellow and dark blue to the Test Set.



Print-outs

Here are some selection of the print outs from the models validation set:

```
True Label: O, Predicted Label: O
True Label: NEG, Predicted Label: NEG
True Label: O, Predicted Label: O
True Label: O, Predicted Label: NSCO
True Label: NSCO, Predicted Label: O
True Label: NSCO, Predicted Label: NSCO
True Label: NSCO, Predicted Label: O
```

Here are some selection of the print outs from the models test set:

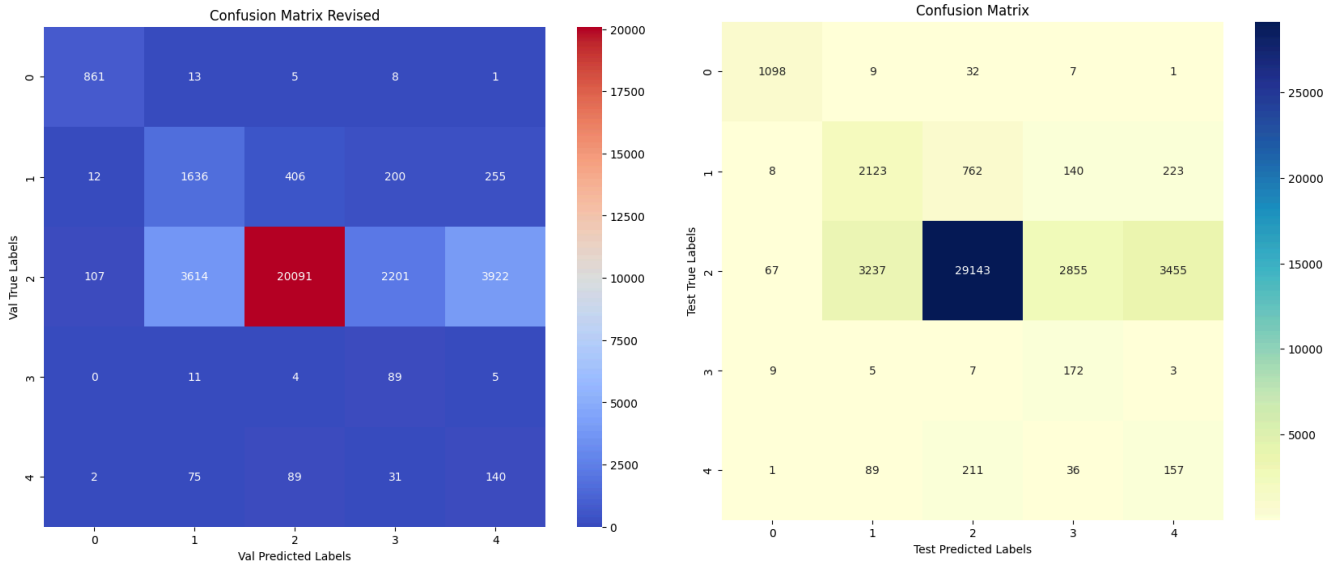
```
True Label: O, Predicted Label: O
True Label: O, Predicted Label: NSCO
True Label: O, Predicted Label: O
True Label: O, Predicted Label: O
True Label: NEG, Predicted Label: O
True Label: NSCO, Predicted Label: O
True Label: NSCO, Predicted Label: O
```

As we can see the model tends to tag many of the tokens as 'O' when possible, we nicknamed this problem as the All-Os Problem and we set upon making a model that solved this problem (although it might be at a lower accuracy cost); our Revised Approach.

Revised Approach

In this model we use regularization in the models definition to palliate the overfitting problem and we also used class weights to solve our imbalance dataset problem, we also implemented an early stoppage, here are the results.

Confusion Matrix



Print-outs

Here are some selection of the print outs from the models validation set:

```
True Label: NSCO, Predicted Label: NSCO
True Label: O, Predicted Label: O
True Label: NSCO, Predicted Label: NSCO
True Label: O, Predicted Label: O
True Label: USCO, Predicted Label: USCO
True Label: NSCO, Predicted Label: O
True Label: O, Predicted Label: NEG
```

Here are some selection of the print outs from the models test set:

```
True Label: NEG, Predicted Label: NEG
True Label: NSCO, Predicted Label: NSCO
True Label: NSCO, Predicted Label: NSCO
True Label: O, Predicted Label: USCO
True Label: O, Predicted Label: O
True Label: O, Predicted Label: NSCO
```

Conclusions

Our model works but it still struggles with the All-Os problem, this time specially with UNC and USCO labels this is strongly caused by the imbalanced dataset with a great lack of these types of tags, also we have thought about how our model could be improved by using more features relevant to these task such as NER and punctuation derived features. Also, another thought could be to fine tune our model with optimal parameters and stronger regularization.

Who did what Deep Learning Approach?

Code:

- Data Analysis, annotation and groundtruth: Marino
- Data Preprocessing: Marino
- Feature Extraction: Andreu, Pere & Marino
- Model: Marino
    - Model: First Approach
    - Model: Revised Approach

Report: Marino