Negation and Uncertainty Detection using Classical and Machine Learning Techniques

Abstract—In this project we present several possible solution for a Name Identity Recognition task for medical record. The goal is to detect negation and uncertainty cues with their corresponding scopes. Five main models are presented from a rule-based method to a transformer encoder. Due the limited sized of the dataset, machine learning methods like CRF or HMM, stood out above the others in terms of results.

Keywords—*Negation detection, Uncertainty detection, Clinical text analysis, Rule-based approach, ML*

Contents

1. Introduction

Many details in medical records are written in an unstructured way, often as notes or summaries, making them hard for both automated systems and humans to fully access for clinical or research use. While information retrieval tools can help search these texts, they often fail to distinguish between terms that are affirmed, negated, or uncertain. Negation words are usually treated as stop words and ignored, which leads to misleading results.

The presence of a medical term in a report does not necessarily imply the patient has the condition. It may be explicitly denied or mentioned with uncertainty. These expressions in clinical notes and are crucial for accurate understanding. Identifying not just the negation, but also the uncertainty, and determining which terms are actually affected is essential for meaningful indexing and analysis. Although some medical language processing (MLP) systems can handle negation and uncertainty, they are often not easily reusable.

In this project, we present and test models from the three main realms of NLP: rule-based approaches, machine learning and deep learning. In doing so, we are able to broaden our perspective on the field, which is something that that we could not get from theory alone.

2. Exploratory Data Analysis

For the first part of the project we had to explore the data to understand it to be able to handle it better for our goal. We extracted and transformed the data into a *DataFrame* so that it was more manageable. There are four labels, *NEG* and *UNC* for the cues of negation and uncertainty and *NSCO* and *USCO* for their respective scopes, if any. There are considerably more negation sentences than uncertainty ones, as we can see in Fig. 1.

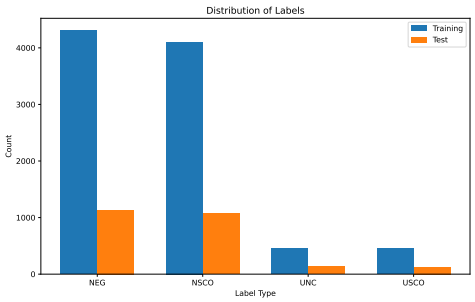

Figure 1. Distribution of label types

3. Preprocessing

3.1. Preprocessing of Data

The preprocessing pipeline employs an `AnnotationProcessor` class designed to structure unstructured text for the annotation tasks. This component performs two core operations: (1) line segmentation via a tokenizer and (2) token span mapping with positional indexing.

Annotation Pipeline (formal representation of the process)

The `process_data` method coordinates the preprocessing workflow by:

1. Generating line-segmented text files with character offset indexes
2. Projecting character annotations to token positions:

$$\tau_i = \arg\min_{\tau \in \mathcal{T}} (\delta_{start}(\tau) \leq \alpha_c \leq \delta_{end}(\tau)) \qquad (1)$$

where α_c denotes annotation character offset and δ represents token span boundaries.

After the mapping, the processed annotations are stored DataFrames for train and test. Each row corresponds to an annotated span and contains the following fields: `doc_id` (document identifier), `line_id` (line number in the document), `annotation_id` (unique ID for the annotation), `start` and `end` (token-level span indices), `label` (e.g., NEG, NSCO), and `text` (the annotated span content).

Making an exhaustive exploration of the data and preprocessing turned out to be very useful for the other parts of the project. As we already a processed dataset in the form of a Pandas Dataframe or CSV file.

4. Rule-Based Approach

4.1. Overview

This approach relies on lexical pattern matching, without using machine learning, focusing on explicit linguistic cues that signal negative assertions or uncertain statements in clinical documentation. This approach creates predictions based only on text pattern matching, without using existing labels.

4.2. Lexicon

The system relies on two primary resources:

1. **Negation Lexicon**: Contains terms indicating absence or denial (e.g., "no", "sin", "negativo")
2. **Uncertainty Lexicon**: Contains terms indicating possibility or doubt (e.g., "puede", "compatible con", "descartar")

Both contain catalan and spanish cues, mostly related to the medical domain, but as we know, in these languages, the morphological variations are common. We aim at catching the different ways in which a cue can present, with variations in number and gender of the word.

4.3. Methodology

4.3.1. Cue Detection

After the medical documents are preprocessed, each tokenized line passes through a function to find its negation cues. It does so by comparing every different word in the lexicon with the token by **exact matching**. When a match is found, the system records in a dictionary useful information related to the indexes, the ID of the document and the line, and both the token and the "canonical" term it had supposedly matched. It first checks for negation and then for uncertainty since it works the same for both.

4.3.2. Scope Determination

Once cues are identified, determining their scope of influence is a more complex task that involves four different rules based on cue type:

1. **Standard Negation Cues:** It is divided into a Forward and a Backward Scope Rule. The scope typically extends from the cue end to the next punctuation mark. But if the punctuation follows the cue, the scope may affect preceding text.
2. **Negative Adjectives:** It uses similar scope rules but potentially different contextual effects. In medical texts, these often modify preceding noun phrases. This helps capture phenomena like "fiebre: negativo".
3. **Verbal Negation:** More focused scope, usually affecting the noun or noun phrase immediately after it. But if no punctuation mark is found, the end of the scope is set as the end of the text, because terms like "niega" typically precede a list of symptoms being denied. Again, in the case where the cue immediately follows a punctuation mark, it defines the scope backwards.
4. **Uncertainty:** It is a simple rule since the scope typically extends from the uncertainty cue to the next punctuation mark.

Periods, commas, semicolons, and colons act as scope terminators. And for each detected scope, we store the useful information in our prediction dictionary as we previously did with the cues.

4.4. Evaluation

Once both cues and their scopes are determined, the system creates structured annotations with the correct format to evaluate the performance. The total count of labels is very close to the actual one, and the distribution is somewhat close too, as we can see in Table 1.

Label	True	Pred.
NEG	4307	4503
NSCO	4103	3682
UNC	458	563
USCO	451	321

Table 1. Comparison of label counts

We calculated the overall entity matching accuracy, using the **Jaccard Index method**, which, unlike the traditional accuracy, takes into account the false negatives in the denominator so it makes it less optimistic and more realistic, and it got a 42,47%. But then, we computed it again only considering the results of the cues, and it raised to a 75.62%, suggesting that the model struggles with the scopes. This is because an exact match requires identical `line_number`, `start`, `end`, and `label`. This supports the hypothesis that partial matches are penalized too harshly with this metric. For example, in some scopes, the "." is included too and it makes the entire span to be counted as wrong. This is why we made it less restrictive by tokenizing or splitting the annotations into new ones (preserving its correct index and label). Now, with this expanded version of the annotations, the accuracy increased to 58.04%, as we expected.

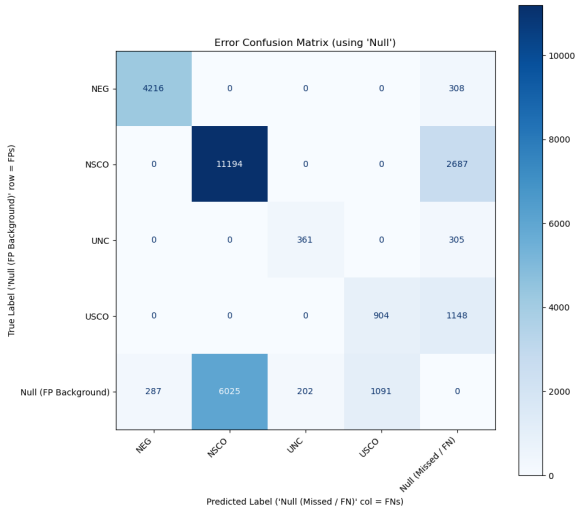

Figure 2. Confusion Matrix made with the predictions on the training data

Finally, we measured the errors with this expanded version of the predictions. In figure **??** we can see a confusion matrix where the rows represent true labels and columns represent predicted labels. The values on the diagonal are the true positives. The special column 'FN' represents missed predictions (omissions) and the 'Null' row represents non-entity states (background text, or mistakes).

By analyzing in which cues errors were made, it can be observed that the majority are for "sin" and "no", which are the cues that are most presented on the train data.

The final results presented were achieved after looking at the errors and performing statistical analysis to detect the most common ones. We removed the keywords that result in more false positives than true negatives. As in the beginning the list of the keywords that triggered

the scope detection was quite large. We believe that with more care the scope detection could be improved.

Developing a Machine Learning model could result on better results, as we believe that it would allow us to distinguish better between the "sin" and "no" examples in the dataset. They result on the great majority of false positives for the NEG and NSCO. The same can be said of "puede" for UNC and USCO. Moreover, due to the simplicity of the rules that we implemented, they were not capable of generalizing over all of the data or of detecting edge cases.

4.4.1. Evaluation on Test Dataset

The same type of evaluation was performed on the test dataset that was provided. The test data was not the basis of any of the rules or lexicon creation that formed part of our model.

We found the same results on the test data, with an overall accuracy of 58.73%, using the individual token evaluation. Just taking into account the NEG and UNC labels, 75.20%. The same types of errors were found, with a prevalence of false positives for "no" and "sin".

These metrics seem to indicate that the test and training data are extremely similar.

5. ML Approach - CRF's

5.1. Overview

Conditional Random Fields (CRFs) are a type of probabilistic model that defines a conditional distribution over possible label sequences given an input sequence. This allows the model to focus on the structure of the output without making important assumptions about the input.

Unlike the rule-based method, CRFs use contextual features and dependencies between neighboring labels, which allows for better generalization to unseen or ambiguous cases of uncertainty and negation.

5.2. Preprocessing

To prepare the training data for the CRF model, we implemented a function that aligns each token with its corresponding annotation label. This function, `data_to_tagging`, processes each clinical document previously split line by line. Using the same tokenizer as in preprocessing, it splits the lines into tokens. It also takes the dataframe containing annotated spans (with start and end indices) and labels. See Preprocessing for more details about the preprocessing steps. Tokens are initialized with the label O ("Other"). For each annotated line, the corresponding token spans are updated with the proper tag. The function returns two dictionaries:

1. One mapping line IDs to token sequences.
2. Another mapping line IDs to their respective tag sequences.

This output serves as the input for the CRF, where each token is paired with its tag for supervised learning.

5.3. Methodology

5.3.1. Feature Extraction

For each token, we generate the following features to train the CRF model, these aim to capture the characteristics of the token and its local context:

Token-level features:

- `word`: The token itself.
- `lower`: Lowercase version, useful for normalization.
- `is_upper`: Boolean flag for all-uppercase tokens (e.g., acronyms).
- `is_title`: Indicates if the token is capitalized.
- `is_digit`: Flags numeric tokens (e.g., dates).

Contextual features:

- `-1:word`: Lowercase form of the previous token (if it exists).
- `+1:word`: Lowercase form of the next token (if it exists).
- `BOS`: True if the token is at the beginning of a sentence.
- `EOS`: True if the token is at the end.

5.3.2. Model Training

We employed two different optimization algorithms for training: L-BFGS and L2SGD.

1. **L-BFGS** is a second-order optimization method that uses both the gradient and an approximation of the second derivatives (Hessian matrix) to optimize the model. It provides faster convergence and more stable optimization and uses both L1 and L2 regularization. L-BFGS tends to outperform other methods in terms of accuracy and convergence for moderate-sized datasets.
2. **L2SGD** is a first-order optimization algorithm. It only considers the gradient and uses L2 regularization, making it more memory-efficient and faster, but it may take longer to converge and can be less stable than L-BFGS.

Both algorithms were tuned using GridSearchCV to optimize hyperparameters such as regularization strength and maximum iterations.

Algorithm	Best Parameters
`lbfgs`	{c1: 0.1, c2: 0.1, max_iterations: 50}
`l2sgd`	{c2: 0.1, max_iterations: 50}

Table 2. Best hyperparameters found for each CRF optimization algorithm using Grid Search

5.4. Evaluation

The goal was to find the best performing algorithm for the dataset by evaluating their performance based on cross-validated accuracy.

Table 3. Classification report for CRF model with LBFGS algorithm

Label	Precision	Recall	F1-Score	Support
NEG	0.973	0.929	0.951	1181
UNC	0.937	0.604	0.735	197
NSCO	0.919	0.817	0.865	3550
USCO	0.892	0.615	0.728	564
Micro Avg	0.930	0.813	0.867	5492
Macro Avg	0.930	0.741	0.820	5492
Weighted Avg	0.928	0.813	0.865	5492

Table 4. Classification report for the CRF model using L2SGD algorithm

Label	Precision	Recall	F1-Score	Support
NEG	0.975	0.924	0.949	1181
UNC	0.943	0.503	0.656	197
NSCO	0.944	0.789	0.860	3550
USCO	0.886	0.441	0.589	564
Micro Avg	0.948	0.772	0.851	5492
Macro Avg	0.937	0.664	0.763	5492
Weighted Avg	0.945	0.772	0.844	5492

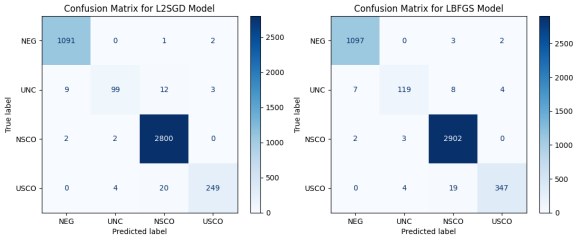

Figure 3. Confusion Matrix Results for the CRF model

5.5. Results

UNC (Uncertain scope) is the most challenging class for both models. However, L-BFGS outperforms L2SGD, particularly in USCO and UNC, which are more context-sensitive.

Both models outperform the rule-base method and L-BFGS is preferred due to higher accuracy in the harder-to-predict classes and a better balance between precision and recall. It's computationally more expensive but more stable for this task.

6. ML Approach - HMM

6.1. Overview

Hidden Markov Models (HMMs) are statistical sequence models where each token's label only depends on the previous token's label (Markov property). We thought this limitation might cause HMMs to struggle with negation and uncertainty detection, since these usually need understanding of longer dependencies. Also, because of the chain rule, we expected errors to spread when one token is wrong, the next ones are likely to be wrong too. Even with these challenges, we still wanted to try HMMs to get a statistical baseline and see if we could make improvements to overcome these limitations.

6.2. Methodology

For our models, hidden states are the labels (NEG, NSCO, UNC, USCO, O) and observations are the tokens. We tried three different versions, each one more complex than the last.

1. **Baseline HMM**. Just uses word tokens as observations, learning emission probabilities $P(word|label)$ and transition probabilities $P(label_t|label_{t-1})$.
2. **BIO+POS HMM**. Adds two improvements. First, BIO tagging to better find entity boundaries. Second, Part-of-Speech features combined with tokens to get richer observations.
3. **Second-Order HMM**. Goes one step further by looking at two previous states $P(label_t|label_{t-1}, label_{t-2})$, trying to fix the limited dependency window.

We trained all models with Laplace smoothing (parameter=0.01) and used Viterbi for inference.

6.3. Results

Figure 4 shows how the different models performed. The baseline got an F1 score of 0.56, then the BIO+POS model jumped to 0.73 (a 30.4

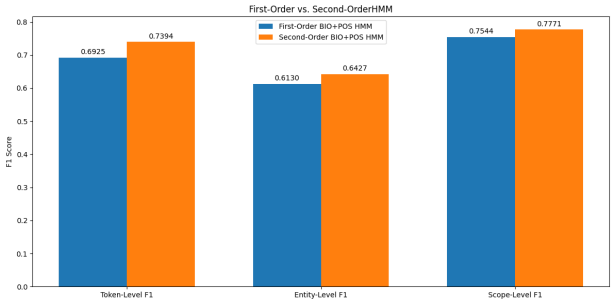

Figure 4. Performance comparison of our HMM models showing F1 scores at token, entity, and scope levels.

The second-order model shows significant improvements because it captures longer dependencies between labels. This is especially important for scopes (NSCO, USCO), where context beyond the immediate previous word matters. By using trigram transitions, our model better handles the continuity of scope labels and reduces the error propagation problem that affected the first-order models. As shown in the table, we achieved good results for negation detection, while uncertainty detection remains more challenging.

Table 5. Second-order HMM model classification report with BIO tagging

Label	Precision	Recall	F1-Score	Support
NEG (B+I)	0.91	0.94	0.93	1152
UNC (B+I)	0.65	0.71	0.68	194
NSCO (B+I)	0.79	0.79	0.79	3273
USCO (B+I)	0.44	0.51	0.47	495
Macro Avg	0.73	0.76	0.74	5114
Weighted Avg	0.76	0.78	0.77	5114

6.4. Error Analysis

When we looked at the results more closely, we found that HMMs work well in some cases but not so well in others.

Success Case. Simple Negation with Clear Scope
Text. "actualmente presentan buen aspecto sin signos de sangrado ni infeccion"
True labels. NEG (sin), NSCO (signos, de, sangrado, ni, infeccion)
Baseline predictions. Got both the negation marker and its entire scope right.
Analysis. The baseline HMM handles common patterns like "sin" with a straightforward scope pretty well.

Error Case. Uncertainty vs Negation Confusion
Text. "la motilidad compatible con contractilidad ausente nota no se observa"
True labels. UNC (compatible, con), USCO (contractilidad), NEG (no), NSCO (se, observa)
Baseline errors. Messed up by labeling the uncertainty cue and scope as negation scope (NSCO).
Analysis. The model cannot really tell the difference between uncertainty and negation when they appear close to each other, which shows how limiting it is to only look at the previous state.

All HMM variants struggled with discontinuous scopes and sentences with multiple overlapping expressions. The second-order model performed better on these complex cases, but uncertainty detection remained challenging across all models, with UNC and USCO having lower F1 scores than their negation counterparts.

7. Deep Learning

7.1. Overview

In recent years, deep learning and especially the Attention mechanism have proven to be very useful for general NLP tasks. However, this project feels very narrow in comparison. Hopefully, this section serves as an introduction on how a more advanced deep learning model can be developed.

7.2. Architecture

Three transformer models are presented, one entirely pre-trained and other with just a pre-trained token encoder. Other, more resource-efficient methods were not included for lack of time. In addition, we wanted to explore more advanced models like transformers out of curiosity, since we felt that we could not surpass the great results of the CRF with other more simple architectures.

1. Fully Pretrained (Multilingual): 'longformer-roberta-xl'
2. Fully Pretrained (Spanish): 'roberta-base-biomedical-clinical-es'
3. Custom Transfomer (Spanish): 'roberta-base-biomedical-clinical-es'

Label	Precision	Recall	F1-Score	Support
NEG	0.92	0.93	0.92	455
NSCO	0.74	0.79	0.77	439
UNC	0.78	0.82	0.80	44
USCO	0.22	0.28	0.24	43
Micro Avg	0.79	0.84	0.82	981
Macro Avg	0.66	0.71	0.68	981
Weighted Avg	0.80	0.84	0.82	981

Figure 5. Classification report for the specialized transformer tokenizer.

The choice in pretrained models was deliberate for the sake of comparison, as it clear that the method that we expected to work the best would be the second one, as the tokenizer is specifically built for the context of the project. Moreover, the first tokenizer is a 'longformer', a class of model that allow us to feed an entire document from our dataset into the model. This was not possible with the more specialized tokenizer, as it has a much lower context windows.

7.3. Methodology

For the first two model with the fully prebuilt tokenizers, the whole model was being finetuned with a classification head on top. Meaning that the batch sizes and epochs had to be reduced in comparison with the third one. A BIOES scheme is used for the label creation; however, due to the nature of the 'transformers' library finetuning, this is assumed on the final results. The labels are simplified, presenting the metrics with the four labels used in our dataset. The 'transformers' 'Trainer' class was used to carry out the training, therefore we cannot adjust all of the hyperparameters that we are accustomed in pure 'pytorch' code.

The custom consists of a transformer encoder that takes as input the tokens from the pretrained tokenizer. A classification head sits on top of the encoder, taking each output token and assigning it to a class. A BIOES scheme was used to do so. At maximum 5 layers were used in the transformer encoder. Since we have an unbalanced dataset, we decided to use Focal loss instead of a Cross-Entropy as the model's criterion. This was an advantage of making a custom model instead on relying in pre-made classes as in the other two models.

Due to the variance in size of both approaches -the pretrained models are much larger in terms of parameters-, we opted to balance the training time instead of other aspects like epoch count.

7.4. Results

Because of time and space restrictions, we present only results from the second model. The other results are not worth to look into them, since they do not generalize the UNC and USCO classes, both achieving zero or a very small number -less than 2%- in all the metrics. Only the NEG and NSCO classes were decently classified, specially the NEG due to its prevalence on the dataset. We also believe that the fact that this labels is usually presented in 'SINGLE' form played also a role in its great results.

As we can see in the table below, the results for the UNC class are fairly decent; however, we cannot say the same for the USCO class. In general, this deep learning model performs significantly worse than the CRF model.

To us it is pretty clear that the size of the dataset is not large enough to train a transformer -or similar- model. When training the custom model, the precision for the UNC and USCO could never the elevated above 5%, even when purposely trying to overfit the model over the validation/test dataset. The fact that we did not have pretrained weights for the transformer encoder seems to be the reason for this.

In regard to our best model, it seems that the finetuning of the text tokens was a much simpler task for our dataset, despite being a larger model in terms of parameter count.

As a final conclusion: this is an example of the fact that choice of the correct finetuned model for your task is the key when working with pre-made model or libraries like 'transformers'. We also need to highlight the fact using a simpler model like RNNs or a bi-LSTM could have worked better, since they are less data-hungry models.

8. Conclusions

Despite our unfamiliarity with NLP tasks and models, we feel that the goals of the project were achieved, as we have reason about our results. Our main conclusion would be that ML methods are the most adequate for this task, both in terms of implementation complexity and results. Perhaps if we had invested more time on the rule-base method, we could have achieved better results, specially for the USCO label. We would like to highlight the CRF model, as it provided the best results, being one of the simpler models to implemented.