Negation Cue Detection

Lahorka Nikolovski 2725815
Panagiota Tselenti 2734736
Shuyi Shen 2734260
YC Roderick Li 2740992

Abstract

Negation cue detection is one of the fundamental tasks in Natural Language Processing (NLP), with particular applications in the biomedical and sentiment analysis domains, and a prerequisite for tackling higher-level negation resolution problems. This paper presents an experimental setup, aimed at finding an effective automatic system for solving the task in the English language. In our study, we approach the task mainly using traditional machine learning methods, as well as a deep learning architecture. Based on previous related work, we propose a set of features encoding lexical and syntactical information, including a novel feature not found in existing literature. We train a simple baseline using a Support Vector Machine (SVM) algorithm to be able to evaluate the impact of proposed features, and further build 5 different systems: two Support Vector Machine (SVM) models, 2 Conditional Random Field (CRF) systems, and 1 Multi-Layered Perceptron (MLP) model. All experimental systems are trained and evaluated on datasets originating from the *SEM 2012 Shared Task (Morante and Blanco, 2012). The performance of all models is discussed, with our highest scoring system achieving a 0.910 F1 macro average score on the development data. The same system obtains a 0.645 F1 on the test data, revealing a deficiency in detecting unseen multi-word negation cues. The results are followed by a detailed error analysis and suggestions for improving the system in future work.

GitHub repository: `https://github.com/LahiLuk/TMgp4-negation-cue-detection`

1 Task Description

The task of Negation Resolution (NR) in NLP aims at detecting negation cues and the scope of these cues in natural language. According to Lapponi et al. (2012) NR is the task of determining which tokens are affected by a negation cue in a sentence level.

Negation is a linguistic phenomenon that can modify the meaning of a lexical item (Chowdhury, 2012). It is a phenomenon present in all languages (Tottie, 1993 and Horn, 1989 as cited in Abu-Jbara and Radev (2012)) and it functions as a transformer of the semantic meaning of an affirmative statement into its opposite (Abu-Jbara and Radev, 2012). Negation can take many forms; it can be single words that trigger the negation in a sentence, affecting the meaning of other words or phrases, it can have the shape of multi-word-expressions or it can be affixal.

According to Morante and Blanco (2012), the interest in automating negation processing originated in the medical domain (Chapman et al., 2001 as cited in Morante and Blanco (2012)), since there was a need of clinical reports and other types of medical documents to be reliably interpreted and indexed. According to Abu-Jbara and Radev (2012) NLP research in the biomedical domain is focusing on extracting factual relations and pieces of information from unstructured data. Thus, correctly detecting negation is of vital importance in this domain, because information that falls in the scope of a negation cue cannot be treated as a fact.

In addition to the biomedical domain, automatic detection of negation and its scope is a task that has occupied multiple other NLP applications including data mining, relation extraction, question answering, and sentiment analysis. For example, in sentiment analysis, failing to detect negation could result in false answers in question answering systems or in the prediction of the opposite sentiment (Abu-Jbara and Radev, 2012). In the same context of sentiment analysis, NR has been explored by studying the contextual features that affect text polarity (Abu-Jbara and Radev, 2012).

The task of negation resolution has different sub-tasks, each one a task in itself, all tightly connected or relying on the other. The most prominent and explored sub-tasks to resolve negation according to literature, are negation cue detection, negation scope resolution, focus of negation detection and negated event identification.

A negation cue is a word, a phrase, a prefix, a postfix or an infix that triggers negation. Scope can be defined as that part of the meaning that is negated by the presence of a negation cue and focus the part of the scope that is most prominently negated (Huddleston and Pullum, 2002 as cited in Morante and Blanco (2012)). Focus can also be defined as "the element of the scope that is intended to be interpreted as false to make the overall negative true"(Morante and Blanco, 2012, 267). The negated event is the event or the entity (broadly referring to a person, an institution etc.) that the negation pinpoints its absence or denies its occurrence.

According to Mehrabi et al. (2015), "rule-based techniques have been shown to be effective in negation detection and widely used in many NLP systems. Rule based negation systems can be token-based (e.g. NegEx, NegExpander, NegFinder, NegHunter) ontology-based, or utilize syntactic parsing results (e.g., DepNeg, ChartIndex, Ballesteros et al. as cited in Mehrabi, 2015). For example, NegEx processes one sentence at a time by finding negation and termination terms. Termination terms are conjunctions, such as *but*, that end the scope of negation terms" (Mehrabi et al., 2015, 214).

Another widely used approach is using Machine Learning algorithms, with the most prominent being Conditional Random Fields (CRF) (since NR is mostly approached as a sequence labeling task) and Support Vector Machines (SVM) for classification sub-tasks such as negation cue detection (Chowdhury (2012); Abu-Jbara and Radev (2012)-Radev; Lapponi et al. (2012)).

Deep learning-based approaches to the task have also been explored. For example, Britto and Khandelwal (2020) apply three popular transformer-based architectures, BERT[1], XLNet[2] and RoBERTa[3], breaking the task in two sub-tasks (that of speculation cue detection and scope resolution) and comparing their performance.

[1] https://huggingface.co/docs/transformers/model_doc/bert
[2] https://huggingface.co/docs/transformers/model_doc/xlnet
[3] https://huggingface.co/docs/transformers/model_doc/roberta

1.1 Negation Cue Detection

Negation cue detection is the task of identifying tokens that express negation in a text. According to Abu-Jbara and Radev (2012), negation cues are textual segments that indicate the existence of negation in a sentence. A negation cue is the lexical item that triggers negation.

Since the occurrence of negation in a sentence is defined by the presence of a negation cue (Abu-Jbara and Radev, 2012), the negation cue detection is essential as a prior step to any other NR task. Detection of negation cues and consequently their scope and corresponding negated events, can increase the accuracy of other natural language processing (NLP) tasks as well, such as extraction of factual information from text, sentiment analysis, etc. (Chowdhury, 2012).

Tokens that are negation cues can be single words (e.g.,*not*), more than one word or so called multi-words (e.g., *no longer*, *by no means*), affixes (prefixes or suffixes, for example *im-*, *-less*), or infixes, for example breath*less*ness). Multi-word negation cues can be discontinuous, as in the case of *neither ... nor* (Morante and Blanco, 2012).

Negation cue detection is considered a classification task and solving it includes, to our best knowledge, two main types of approaches. First, approaching it by classifying a token as being a negation cue or not or secondly, posing the problem as a sequence labeling task; in both cases one makes use of machine learning algorithms.

In cases where identification of negation cues is based on their classification as such, annotations of the training set can be binary (on whether something is a negation cue or not) or multi-class (e.g. type of negation cue).

Reasons behind a sequence labeling task approach is that some negation cues may be false negations, meaning that they do not indicate negation in some contexts. "For example, the negation cue *not* in the phrase *not to mention*, does not indicate negation. Also, some negation cues may consist of multiple words, some of them are continuous and others are discontinuous. Treating the task as a sequence labeling problem helps model the contextual factors that affect the function of negation cues" (Abu-Jbara and Radev, 2012, 331). It can also lead to avoiding errors such as the ones mentioned in Lapponi et al. (2012) where the researchers find five so-called false negation cues (Morante et al., 2011 as cited in Lapponi et al., 2012), including three instances of *none* in the fixed expression *none the less*. The others are affixal cues, of which two are wrong (*un*derworked, *un*iversal) while others might be attributed to annotation errors (*in*superable, *un*happily, end*less*, list*less*ly).

Negation cues should not be confused with negation scope. Negation cue is not considered as part of the scope (Chowdhury, 2012). Each negation cue, though, can be linked to one or more negated concepts or events belonging to the scope of the cue. Alternatively, in-scope tokens and negated events can be paired to the cues they are negated by (Lapponi et al., 2012).

2 Related Work: Classification Approaches and Features Used

In this report, we will approach the negation cue detection task with a machine learning method in order to explore different features. Neural network would require massive data to reap better results, and machine learning has already been proven quite successful for this task. Selected studies that applied a machine learning approach to negation cue detection on the *SEM 2012 Shared Task data are reviewed in the following paragraphs.

The best performing system overall on negation cue detection on *SEM 2012 Shared Task was FBK, which obtained an F_1 score of 92.34 (Morante and Blanco, 2012). System architecture is described in Chowdhury (2012). The task is approached as a problem of sequence identification, and the learning algorithm used is the 1st order Conditional Random Fields (CRF) classifier. Since this system was developed as a part of the closed track, it can only use information provided in the corpus by the organizers. The original datasets include the following features: token, lemma, PoS and parse tree information. From those features, FBK system uses only two: lemma (converted to lower case) and PoS, and additionally uses the lemma of preceding token as a feature. The FBK system also uses a special procedure to detect affixal negations, supplemented by a list of negation cues that is automatically extracted from the training data based on cue frequency. The cue list consist of terms *nor*, *neither*, *without*, *nobody*, *none*, *nothing*, *never*, *not*, *no*, *nowhere* and *non*. The list of negation affixes includes prefixes *un-*, *dis-*, *im-*, *in-*, *non-* and *ir-* and the suffix *-less*. Lastly, the author mentions that certain expressions were marked as negation cues after the classifier's prediction, which is not something we encountered elsewhere. The expressions *neither*, *nobody*, *save for*, *save upon* and *by no means* were marked as negation cues, and the phrase *none the less* was marked as not being a negation cue, both after prediction. In his analysis, Chowdhury poses that the detection of negation cues does not depend so much on the context, but instead, largely on the token itself. Additionally, he stresses the importance of using lists of negation cues, and proposes that one such list could be collected from Morante (2010).

In another study by Abu-Jbara and Radev (2012), their CRF system obtained an F_1 score of 90.98. They also used provided features including tokens, lemmas and PoS tags and simpler features like whether the token has a negation prefix or suffix (prefixes *un-*, *in-*, *im-*,*il-*, *dis-*, and suffix *-less*). In addition, they incorporated PoS n-grams, PoS tag categories, and a feature to indicate whether the token is puncuation (is-punctuation). PoS bigrams and trigrams were used. Besides, PoS tags were reduced into 5 categories: Adjective (ADJ), Verb (VB), Noun (NN), Adverb (ADVB), Pronoun (PRO), and other (OTH). Abu-Jbara and Radev found out that the closed track approach limited their result as cues that did not exist in the training data (e.g., prefix *un*) were destined to be overlooked. They proposed to use a lexicon of negation words, which is consistent with Chowdhury's (2012) conclusion, and of words that can be negated by adding a negation prefix to them (Abu-Jbara and Radev, 2012).

Lapponi et al. (2012), on the other hand, achieved a 91.31 F_1 score with an SVM negation cue classifier. Similar to previous studies mentioned, they adopted token-level features to identify lexical negation cues. These features include token n-grams and lemma n-grams of tokens preceding and following the target. For affixal cues, they used two kinds of features to distinguish de facto negation affixes from confusing counterparts in the same forms. The first kind is character n-grams of the beginning and the end of the substring of a token after getting rid of the affix, up to 5 positions. The n-grams are then combined with the affix itself and the PoS of the token. For instance, the features for the word *impossible* would be character n-grams *possi, poss, pos...* and *sible, ible, ble...*, combined with the prefix *im* and PoS tag *JJ*. The other kind of features would be look-up counts of the base substring. Lexicon counts were first developed with the PoS tagged lemmas and word-initial character n-grams up to 5 positions from the training data. Then, all tokens with matching affix patterns would be given these counts, in the hope of minimizing the misclassification of base substrings that are unlikely to be standalone words. For instance, since the word ***un****derlying* begins with *un*, it will be assigned the lemma and character n-gram counts of the base substring *derly*, which is presumed to be very low if not zero since "derly" is not a word. This should help the machine find out the word *underlying* is actually not a negation cue. Another point to note would be the post-processing filtering, tackling the multi-word cues (MWCs). Since MWCs such as *neither...nor* or *on the contrary* were rather rare in the training set, the study decided to create a

stop-list to filter out these unwanted words from the training samples to avoid overfitting (Lapponi et al., 2012).

2.1 Preprocessing the corpus to obtain features

The dataset provided for our experiments is *SEM-2012-SharedTask-CD-SCO*. It originates from the *SEM 2012 Shared Task (Morante and Blanco, 2012) and was preprocessed in such a way as to only include the following information: name of the book, sentence number, token number, token and label. From those, token is the only potential feature we could directly use. Thus, in order to extend the features for our task, a series of preprocessing steps will need to be performed on the corpus.

With reference to the literature, features we propose to be used in our experiments are listed in Table 1. These features will be reviewed and finalized in Section 3.

#	feature	brief description
1	token	the token
2	lemma	the lemma of the token
3	previous_lemma	the previous lemma
4	next_lemma	the next lemma
5	pos	the PoS tag of the token
6	pos_category	the PoS category of the token
7	affix	whether the token has a negation affix
8	is_negation_cue	whether the token is a probable negation cue

Table 1: Tentative features proposed for this task.

One of the following NLP Toolkits could be selected in order to process the data and obtain lemmas and POS tags: NLTK[4], Spacy[5], Stanza[6] or Stanford Core NLP[7]. Firstly, the tokens might need to be put back together as sentences, which would not be a problem since sentence breaks are encoded as empty lines in the provided corpus. POS tags may be further categorized into Adjective (ADJ), Verb (VB), Noun (NN), Adverb (ADVB), Pronoun (PRO), other (OTH) and saved in another column as an additional feature. After obtaining the lemmas and PoS tags, lemmas can additionally be saved in lower case. Preceding and next tokens, POS tags, and lemmas can then also be extracted and used as features.

Finally, we are considering building a custom vocabulary list that would include highly probable negation expressions, such as *neither*, *nor*, *without* and *nobody*, and a list of affixal and suffixal negations. After composing such lists, additional features that signify whether a token, lemma or character n-gram can be found in the list can easily be extracted.

[4]https://www.nltk.org/
[5]https://spacy.io/
[6]https://stanfordnlp.github.io/stanza/
[7]https://stanfordnlp.github.io/CoreNLP/

3 Data annotation

The annotation experiment that we conducted included annotating negation cues on a corpus of 10 documents. The documents have originally been collected during the creation of the Vaccination corpus, which is described in more detail in Morante et al. (2020). The file names of the 10 documents are provided in Appendix 6.

The corpus consists of online news articles and blog posts discussing vaccination. The main topic of most of the documents our group annotated is the outbreak of measles that happened in the US in 2015, but there are also a couple of documents that have a different topic. For example, one article discusses a vaccination campaign in Kenya.

The task of annotating negation cues was conducted according to the *Annotation of negation cues and their scope Guidelines v1.0*, originally developed for annotating sentence level negation cues, scope and the negated event on two Conan Doyle stories (Morante et al., 2011). Before starting the annotation process, each annotator read the guidelines, focusing on the parts concerning negation cues, since they also provide instructions on annotating the scope and negated event. The annotators did not discuss the guidelines in any way. Instead, each approached the annotation task individually.

The guidelines provide a thorough overview of different spans the negation cues could take and that should be annotated, from including only a part of a word as in the case of a negation prefix, suffix, infix or a contracted negation cue such as *'t* in *can't*, through single words to a combination of words, possibly discontinuous. Special care is taken with regard to false negation cues: elements that have the form of a negation cue, but their function is not negation. Those should not be annotated, and include fixed expressions, like *could not help* in *I could not help asking*, dialogue elements, *like don't you think* or *don't tell me* and modality cues, some of which contain false negation cues, as in the case of *no doubt*. The guidelines also provide a heuristic that should be used if the annotator is not sure whether something is a negation cue or not. It states that one should ask if the negation could be paraphrased using *it is not the case that*, followed by the negated fact. For example, in the sentence *It is not my fault!*, *not* is a negation cue, since we can paraphrase it as *It is not the case that it is my fault*. In the case of potential negative affixes, guidelines stress that one needs to make sure that the meaning of the word without the affix is opposite to that of the original word. Thus, *un*-in *unpleasant* should be marked as a negation cue, since *unpleasant* means *not pleasant*, while *dis-* in *disappear* should not, since *disappear* is not synonymous with *not appear*.

The four authors all participated in the annotation task, each annotating the same 10 documents. Open source tool eHOST[8] was used to annotate the documents, calculate the pairwise inter-annotator agreement (IAA) and generate the agreement reports. Since the eHOST tool does not support comparison of annotation results obtained using the tool on different operating systems, and all the annotators did not have access to the same OS, we made sure that pairs of annotators were using the same system, so that IAA could be calculated. Two annotators used eHOST on Windows, and two used it on macOS.

3.1 Inter-annotator agreement analysis

The IAA is calculated by the number of matches over all annotations and presented as a percentage. Since some span differences are deemed not important in this discussion (e.g., whether *n't* or just *'t* is marked as a cue for *won't*; whether the - is included for *non-*), we mutually decided that the annotations should be considered matching if they have overlapping spans for both groups. In this section, the work of the two annotators working on Windows ("the Windows group") and of the two on Mac ("the Mac group") will be demonstrated and analyzed separately.

True positives are the instances where annotators had matching annotations, meaning that they both thought that a lexical segment was a negation cue. False positives and false negatives are the instances where one annotator thought a lexical segment was a negation cue and the other did not. Depending on whose annotations are set as gold, false positives stand for cases when a lexical item was annotated as a negation cue, but it should not have been, and false negatives stand for when an item was not annotated as a negation cue, but it should have been; always as compared to the other annotator's annotations. Since

[8] https://github.com/chrisleng/ehost

there is not a third reference point outside the annotator's pair, each annotator is each other's reference point, making the false positive and false negative scores identical and just presented in reverse on the table, depending on which annotator is set as the gold standard.

Gold standard	True positives	False positives	False negatives	Recall	Precision	F-measure
Annotator A	129	10	11	92.1%	92.8%	92.5%
Annotator B	129	11	10	92.8%	92.1%	92.5%

Table 2: Pairwise agreement among the Mac group. Precision and recall are given equal weight for the F-score. The Mac group achieved an 92.5% IAA with 258 matches and 21 non-matches.

Gold standard	True positives	False positives	False negatives	Recall	Precision	F-measure
Annotator A	139	5	55	71.6%	96.5%	82.2%
Annotator B	139	55	5	96.5%	71.6%	82.2%

Table 3: Pairwise agreement among the Windows group. Precision and recall are given equal weight for the F-score. The Windows group attained an 82.2% IAA with 278 matches and 60 non-matches.

In total, the Mac group (Table 2) annotated 279 instances as negation cues and the Windows group (Table 3) 338 instances. This demonstrates that the Windows group annotated more segments in general, thus explaining the lower IAA when compared to the Mac group, since more instances means higher probability of disagreement. Furthermore, when observing the recall and precision scores of the Mac group (92.1% and 92.8%, Table 2), they share similar scores demonstrating that this group did well both in identifying the majority of the total of negation cues as they defined them (high recall score) as well as succeeding in having very few false positives, meaning whenever they annotated something as a negation cue, it most frequently indeed it was (high precision score). On the other hand, the same scores are not so similar for the Windows group (71.6% and 96.5%, Table 3). Compared to the 10-11 instances of the Mac group, the Windows group presents a very polarized result in false positives and false negatives, with 5-55 instances. This demonstrates that annotator A annotated more negation cues than annotator B in total.

Category	Mac group	Windows group
Prefixes	independent discount unprecedented	disability irrational discredited dissenting discount
Verbs	deny	failed
Multi-word expressions	too … to kept … from instead of	
False cues	not (only) can't (wait) none (at all)	whether or not couldn't help ... but

Table 4: Overview of the categories and cues on which the groups did not agree.

The most common mistakes were simple errors, where the annotators overlooked a cue they otherwise consistently marked. These include 63 cases out of 81 disagreements in total. Not taking into account those misses, the remaining 18 disagreements can be categorized into negation prefixes, verbs, multi-word expressions and false cues. The categories, including examples from our corpus, are summarized in Table 4.

Across the two groups, prefixes and fixed expression false cues were the more frequent types of disagreement. Out of 18 disagreements, 8 are prefix-related and 5 fixed expression-related. The annotation guideline clearly stated that some prefixes should not be counted as negation cues, as semantically they do not negate their bases. Table 4 records the words where we have different opinions about whether the prefixes negated their bases (e.g., whether *dis-* in *discount* negated *count*). The annotation guideline also gave examples about false cues that when a negation word is part of a fixed expression, it should not be counted as a cue. Table 4 also lists some of our disagreements under "false cues", including *not only* (whether *not* negates *only* or it is a fixed expression for compilation), etc.

These disagreements prioritize our attention to work with prefixes and fixed expressions. As some of them constitute "false cues" where consensus is hard to reach by human annotations, they should be even more carefully handled when designing the features for our negation cue detection system. We should have a more consistent understanding in order to select the most relevant features that can help the system's classification.

Given that our disagreements lie mainly in prefixes and fixed expressions, accounting for 13 out of 18 in total, it may indicate that guidelines for these parts might have been insufficient. We found many cases were actually arguable and open to different interpretations. More instructions and training are needed for a higher degree of agreement on what are considered to be negating prefixes and false cues.

We could also see how the level of fluency in the language can also affect annotation. For example, one's English proficiency would potentially affect their interpretation of a negating prefix or fixed expression. Since none of our group members are native English speakers, it is hard for us to be certain whether *independent* is synonymous with *not dependent* or not. It would be interesting to correlate annotator behavior with their English proficiency.

In this annotation task, disregarding the negated events made it harder to decide on the negation cues. When deciding on whether an expression is truly a negation cue, one would have to consider the event it is negating, according to the annotation guideline. Since the scope of this report is to identify negation cues only, it could be actually more confusing to the annotators when we are asked to not consider the events at all. And because the negated events are important indicators of negation cues, this task also reminded us of the importance of sequential context information. Whether an expression is a negation cue depends on the context and the negated event. Therefore, in the upcoming experiment, the task is not just to identify a list of negation words, but also to consider whether that particular expression implies negation in that context. Models and features need to be carefully selected to capture relevant and sufficient sequential information.

Finally, this task had us reflect on the importance of a handy annotation tool. The practicalities of annotating (e.g. highlighting parts of words) were not particularly user-friendly and the members of the Windows group faced numerous encoding problems which they spent a lot of time trying to solve and which affected their IAA scores. Additionally, it is really unfortunate that the eHOST tool does not allow comparing the results across operating systems, since being able to compare the annotations of all 4 annotators would provide us with additional insight, especially regarding the difference in the number of cues annotated by the two groups.

In the end, we all agreed that an annotation task is very complex even when seemingly simple; from the technology required to accomplish it to understanding the guidelines and calculating and interpreting results.

4 Dataset description

For our classification experiments, we are provided with a training and a development dataset. The data are adapted from the SD-SCO corpus, developed for the *SEM 2012 Shared Task (Morante and Blanco, 2012). SD-SCO is a corpus of Conan Doyle stories. The training dataset contains The Hound of the Baskervilles, and the development dataset The Adventure of Wisteria Lodge. Our data differ from the original corpus in two ways: token PoS and parse tree information are not provided, and the negation cues are labeled differently.

Each row in our dataset provides the following information: the name of the book (column 1), the sentence number (column 2), the token number (column 3), the token (column 4) and the label (column 5). Sentence boundaries are indicated by empty lines in the data files. An example sentence from the development dataset is provided in Table 5. We chose the same sentence as shown in Morante and Blanco (2012, p. 268), so that a comparison could easily be made.

wisteria02	108	0	After	O
wisteria02	108	1	his	O
wisteria02	108	2	habit	O
wisteria02	108	3	he	O
wisteria02	108	4	said	O
wisteria02	108	5	nothing	B-NEG
wisteria02	108	6	,	O
wisteria02	108	7	and	O
wisteria02	108	8	after	O
wisteria02	108	9	mine	O
wisteria02	108	10	I	O
wisteria02	108	11	asked	O
wisteria02	108	12	no	B-NEG
wisteria02	108	13	questions	O
wisteria02	108	14	.	O

Table 5: Example sentence from our development dataset.

The last column in our dataset contains the negation cue label. Tokens that are not a part of the negation cue are labeled with O. Single word negation cues are labelled with B-NEG, while for multi-word negation cues, the first token is labeled with B-NEG, and subsequent tokens with I-NEG. In the case of affixal negation, the whole token receives a label, but negation cues contracted with the verb (such as *n't* in *can't*) are tokenized and labeled separately.

The training dataset consists of 3644 sentences, and development of 787 sentences, both containing an average of 17 to 18 tokens per sentence. From those, 23% of training and 18% of development sentences contain at least one negation cue (848 and 144, respectively). In total, there are 987 negation cues in training, of which 130 unique, while for development dataset, those numbers are 176 and 38. Only a fraction of negation cues are multi-word (8 in training and 2 in development dataset), and none of them are discontinuous. The statistics are summarized in Table 6.

When we zoom in on the negation cues themselves, we can see that the five most common single word cues (*not*, *no*, *n't*, *never* and *nothing*) account for the majority of negations in our corpus. There is also only a handful of affixes that account for affixal negation found in our corpus: five prefixes and the infix/suffix *less*. Those statistics are summarized in Table 7, and already give us a good idea of a type of list we could use to aid in recognizing negation cues.

[9] Instead of treating *less* as a negation infix, for the purposes of feature extraction we view variants like *-lessness* and *-lessly* as suffixes. This is also the approach of Lapponi et al. (2012).

	training dataset	development dataset
# sentences	3644	787
# tokens	65451	13567
# tokens, unique	5779	2432
# single word negation cues	979	174
# single word negation cues, unique	125	36
# multi-word negation cues	8	2
# multi-word negation cues, unique	5	2
# sentences with one negation cue	731	115
# sentences with more than one negation cue	116	29

Table 6: Statistics for the training and development datasets.

	training dataset	development dataset
negation prefixes	dis- im- in- ir- un-	dis- im- in- ir- un-
negation suffixes	-less -lessness[9] -lessly[9]	-less
most common single word negation cues and their counts	not, 358 no, 226 n't, 65 never, 59 nothing, 55	not, 42 no, 33 n't, 20 nothing, 16 never, 11
multi-word negation cues and their counts	by no means, 3 on the contrary, 2 rather than, 1 not for the world, 1 nothing at all, 1	by no means, 1 no more, 1

Table 7: Overview of negation cues found in our corpus.

5 Corpus pre-processing and feature extraction

5.1 Preprocessing the corpus

Since the sentences in our corpus are already tokenized (each token is written on a separate line and empty lines signal sentence boundaries), NLTK seemed like the logical first choice for linguistic preprocessing, because it allows parts of the NLP pipeline, like Part-of-Speech tagging or stemming/lemmatization, to be applied independently. The other tools we were considering for this project: Spacy, Stanza and Stanford Core NLP, are optimized to take "raw" text as input and then apply the complete NLP pipeline to it, including tokenization. After some research, we realized that the other tools could also be used to process pre-tokenized text. The process is best documented and most straightforward for Stanza[10]. Still, since those other tools' downstream tasks are optimized to word in conjunction with their tokenizer, we decided to stick with NLTK as our first choice for tokenization, keeping in mind that if the results would turn out suboptimal, we could repeat the process using a different toolkit.

After choosing the toolkit, the first preprocessing step was to extract a list of tokenized sentences from the data file, using empty lines as indicators of sentence boundaries. Next, PoS tags for every token were obtained by using the off-the-shelf tagger available for English through NLTK[11]. The sentences are processed one by one, and the tags obtained are those used in the Penn Treebank Project[12]. Lastly, each token has been lemmatized using NLTK WordNet Lemmatizer[13]. The lemmatized takes as input a token and its PoS tag (obtained in the previous step). Passing the PoS tag to the lemmatizer is essential, since by default, it treats all words as nouns. For example, if the verb *was* is provided to the lemmatizer without a PoS tag, it is lemmatized as *wa*. After indicating the PoS, the correct lemma *be* is obtained.

wisteria02	108	0	After	After	IN	O
wisteria02	108	1	his	his	PRP$	O
wisteria02	108	2	habit	habit	NN	O
wisteria02	108	3	he	he	PRP	O
wisteria02	108	4	said	say	VBD	O
wisteria02	108	5	nothing	nothing	NN	B-NEG
wisteria02	108	6	,	,	,	O
wisteria02	108	7	and	and	CC	O
wisteria02	108	8	after	after	IN	O
wisteria02	108	9	mine	mine	NN	O
wisteria02	108	10	I	I	PRP	O
wisteria02	108	11	asked	ask	VBD	O
wisteria02	108	12	no	no	DT	B-NEG
wisteria02	108	13	questions	question	NNS	O
wisteria02	108	14	.	.	.	O

Table 8: Example sentence from our development dataset after preprocessing.

The PoS tags and lemmas obtained through preprocessing are written to a new file. Table 8 shows how the example sentence from the development dataset introduced in the previous section looks after preprocessing. At least for this sentence, the results obtained for PoS and lemma using NLTK are identical to those provided in the original SD-SCO corpus (compare Morante and Blanco (2012, p. 268)).

5.2 Feature extraction

In this section, we describe which features were selected for our classification experiments and how we extracted them from the preprocessed dataset. Table 9 lists the selected features. Explanation of each feature is detailed below.

[10] https://stanfordnlp.github.io/stanza/tokenize.html#start-with-pretokenized-text
[11] https://www.nltk.org/api/nltk.tag.html
[12] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html
[13] https://www.nltk.org/api/nltk.stem.wordnet.html

As features, tokens and lemmas are taken from the preprocessed dataset and lowercased. Lowercasing is done because our system would otherwise treat a token, such as *No*, differently from the same word without the capital letter (*no*). Lowercasing should thus lead to an increase in accuracy, since knowing whether a token start with a capital letter is not a useful feature for detecting negation cues (as it would be for named entity recognition, for example). In fact, only lemmas of these tokens will be used in our experiment, as they retrieve the basic form of words (e.g., turning *recorded* into *record*) and are presumed to minimize vector dimensions of one-hot encoding. Tokens are kept in the feature file to build a baseline system for evaluation.

#	feature	brief description
1	token	the token
2	lemma	the lemma of the token
3	previous_lemma	the previous lemma
4	next_lemma	the next lemma
5	pos_category	the PoS category of the token
6	is_single_cue	whether the word is a single negation cue
7	has_affix	whether the lemma has a negation affix
8	affix	the affix itself
9	base_is_word	whether the lemma without the affix (the base) is a valid word
10	base	the base itself if it is a word

Table 9: Features selected for this task.

Next, we extracted the preceding and the next lemmas to improve the performance of our model. By processing only the lemmas, our model would overlook false cues and multi-word negations. For example, *no* is almost always a negation cue, but in fixed expressions like *no wonder* or *no doubt*, this *no* is in fact a false cue. Words like *by* and *means* also need to be recognized as a part of a negation cue when they appear in the expression *by no means*. Looking at the previous and the next lemmas should help identify these multi-word expressions. Sentence boundaries are indicated by using 'bos' (beginning of sentence) and 'eos' (end of sentence) as the previous or the next lemma, where applicable.

PoS tags are also useful to identify potential negation cues. For example, words that contain either negation prefix or suffix, such as unambiguous, unknown, or incredulously, are usually adjectives or adverbs. However, instead of using the full variety of PoS tags, we reduced them to seven categories as follows: adjective (ADJ), noun (NN), adverb (ADV), verb (VERB), pronoun (PRO), punctuation (PUNCT) and other (OTH) (*pos_category*) to reduce vector dimensions. Besides, considering our system might treat a negation cue, such as fail, save, or refuse in the present tense (VB) different from the same word in the past tense (VBD), categorizing PoS might help increase the accuracy of the classifier.

Aiming at increasing the probability of recognizing a negation cue, *is_single_cue* feature is introduced. According to literature, some specific words are always negation cues. Thus, it was deemed useful to compile a list, explicitly including these cues and use the fact whether the lemmas in our corpus belong to this list or not, as a feature. To encode it, a set of unique single word negation cues was first extracted from the training dataset. Cues containing affixes were removed from this set, leaving 18 unique single word negation cues. This final set was then written to a file. Wanting to extend this small lexicon of single word negation cues obtained only based on the training data, Morante (2010) was used as an external resource, and we manually added more cues listed in their work. The complete list of negation cues we compiled can be found in Appendix 7. To extract the feature *is_single_cue*, we check for each lemma, whether it can be found in this list or not. Single word cues that can frequently trigger negation but are not always negation cues, e.g. the word *save*, were included in the list. Our choice was motivated by the fact that we are aiming at detecting anything that triggers negation and in the cases of words like *save* that can function as negation cues, we are hoping context and PoS information will be captured by our other features, making them and *is_single_cue* complimentary.

Another feature tells whether a lemma has a negation affix (*has_affix*). A lemma containing a negation

affix is more likely to be a negation cue. A set of negation prefixes and a set of negation suffixes is acquired from our training dataset, as listed in Table 7. To broaden the scope of the set to cover potential unseen prefixes, we considered including additional ones with reference to van Son et al. (2016), i.e. *a-, an-, de-, il-, non-, off-, mis-, anti-* and *counter-*. However, since some of them are common beginnings of non-negation words (like *a-* or *de-*), we thought that adding them might not be helpful. After careful consideration, we decided to add only *non-* to the set because it is a more common, bona fide negation cue that is less likely to happen to be just the beginning of a non-negation word. Additionally, we observed that, for lemmas that have a negation prefix but are not negation cues, it is very often the case that the base (part of the lemma that is left after excluding the affix) is very short, e.g. ***in****ch* or ***un****til*. Therefore, only when the lemma includes the affix and the base is longer than the threshold (3 characters for prefixes and 2 for suffixes), then it is considered to be having a negation affix.

In addition to whether a word has a negation affix, the surface form of the affix is also added as a feature (*affix*). This is to capture the differences among the affixes. For example, words with prefix *un-* are more likely to be negation cues than those with prefix *dis-*. If a token has an affix from the affix sets (has_affix=1), the surface form of that affix is written separately in another column.

base_is_word records whether the base is a word by itself and exists in the corpus. This is to further help the system identify words that have the same affix but are not negation cues. Similar to Chowdhury (2012), the rationale is that words of which the bases are not standalone words (like *creet* in ***dis****creet*) are less likely to be a real negation cue. In this study, a vocabulary set is built from the Gutenberg corpus, the largest NLTK corpus[14] and the same genre where our datasets are. The base of a lemma is matched with items in this vocabulary set to see if it exists.

Our novel feature is (*base*) which captures the complete form of the base of the lemmas containing negation affixes. Despite indicating whether the bases exist in a corpus or not, there are still many non-cue words bases which also happen to be standalone words, for example, ***dis****cover* or ***in****deed*. This feature is to strengthen the correlation between the bases themselves and the labels. If a token has a value 1 for stem_is_word, the stem itself is also used as a feature. This feature is inspired by literature but has not yet been approached in the same way to the best of our knowledge.

[14]https://www.nltk.org/nltk_data/

6 Experiments

In this section, we describe the experiments undertaken in order to design a system that will perform well on our negation cue detection task. First, a baseline system is built, the performance of which will be used to evaluate our experimental systems against. Of the 5 experimental systems, two were designed using the SVM algorithm, two using the CRF algorithm and one uses the Multi-Layered Perceptron (MLP). The systems are trained on the training data, and evaluated on the development data, described in more detail in Section 4.

This study mainly investigates machine learning methods. This is because machine learning algorithms have proven to be quite successful for the task of negation cue detection (Chowdhury (2012), Abu-Jbara and Radev (2012), Lapponi et al. (2012). Compared to deep learning methods, machine learning ones are also more transparent about how the features work and cost less computational resources. That said, we evaluated an additional MLP classifier just to see if it gives better results.

Out of different ML algorithms, results with SVM and CRF are compared. The SVM classifier is chosen because it has given robust performances in negation cue detection in past related work (Lapponi et al., 2012). In addition, with a multi-class classification task at hand, where we expect some of our features will interact (e.g. has_affix and affix or lemma and is_neg_cue), an SVM classifier can learn the weights in such a way that the interdependence of features should not have a negative impact on the results. CRF is also deemed suitable for this task since this method considers the neighboring context and performs well on sequence labeling. Considering sequences is important for this task for labeling multi-word negation cues (e.g. labeling the sequence *"by no means"* as B-NEG I-NEG I-NEG) and fixed expression false cues (e.g. labelling *"nothing but"* as O). In this sense, CRF is expected to perform better than SVM. It is also the classifier that performed the best on negation cue detection in the original *SEM 2012 task (Chowdhury, 2012). Abu-Jbara and Radev (2012) also chose CRF for this task.

Deep learning is also explored in our experiments because it may produce better results for this task. Approaches such as recurrent neural network (RNN), convolutional neural network (CNN), or multi-layer perceptron (MLP) are capable of learning non-linear models for complex mappings between the network's inputs and outputs. We chose MLP because it is a classic type of neural network and can be used as a baseline point of comparison to confirm the performance between machine learning and deep learning approaches. Additionally, MLP is suitable for classification problems (Brownlee, 2018). Its predictive capability comes from the hierarchical or multi-layered structure of neural networks, and their data structure can learn to represent features at different scales and combine them into higher-order features. Taking into account these capacities, we expect an MLP might better capture the intricacies of the imbalanced class I-NEG and therefore get a higher macro F1.

6.1 Baseline system

As a baseline, we designed a simple system using the linear kernel SVM classifier[15] with default parameters. The baseline uses only one feature, already present in the dataset before preprocessing - the token itself. Tokens are lowercased and represented by one-hot vector encoding. The resulting sparse vectors of length 5444 (number of unique lowercased tokens in the training dataset) are passed to the classifier, and one label per vector (token) is obtained after classification.

We chose the SVM algorithm for the baseline because it was deemed the most basic of the three algorithms selected for the experiments. It can facilitate a comparison of the results that would be meaningful. It was also deemed valid for systems like CRF and MLP with different architectures to be compared to the same baseline, in order to observe how CRF's sequence labeling capacities and MLP's neural network learning influence the results.

6.2 Classification experiments

The classification task performed in these experiments aims at detecting and classifying negation cues. These cues can be of three main types: single words, multi-word expressions and affixal negations, which also constitute single word negations but were addressed explicitly by our features as a separate

[15] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

type of cue. In each case, our classification units are tokens, meaning the assignation of classes is done on token-level.

An overview of the systems run in the experiments is shown below:

#	System	Brief description
1	Baseline	SVM classifier with default parameters and token as feature
2	System1	SVM classifier with default parameters and all features
3	System2	SVM classifier with fine-tuned parameters and all features
4	System3	CRF classifier with default parameters and all features
5	System4	CRF classifier with fine-tuned parameters and all features
6	System5	MLP classifier with all features

Table 10: Overview of systems built.

System 1 is a linear kernel SVM using all the features extracted (as described in Table 9). Features are represented by one-hot encoding and concatenated, which leads to vectors of length 13487 after accounting for all the possible values found in the training data. The high dimensionality of data vectors also explains our decision to use the linear kernel: since we are using very long and very sparse vectors, a different kernel would not fit our data as effectively.

For System 2, we also train an SVM classifier, but this time we use cross-validation on the training dataset to choose the best parameter settings. The parameters we try optimizing are loss, regularization parameter (C), tol and max iterations. We consider these the most important parameters for optimizing our model since they are vital for convergence (max_iter), flexibility on misclassification allowing for an increase in the margin to avoid overfitting (C), affecting how our model learns patterns in the data by instructing it where to stop (tol) and how it calculates loss. We make use of GridSearchCV[16] and decide to run a 5-fold cross-validation. From the experiment, we observe that the only parameters that seem to make a difference for our training data are loss and C. The system performs better when using the default setting of 'squared_hinge' as the loss argument, and a value of C smaller than the default value of 1 (0.8). The result is elaborated in the next section.

For System 3, we train a CRF classifier using the default parameters[17] and again making use of all of our features extracted before. The CRF system takes a list of lists as input. Each list represents a sentence and contains features associated with each of the tokens, retaining their order in the sentence. As the output of the system, we similarly receive a list of lists, one per sentence, each containing the predicted label for each token in order.

For System 4, we again use the same set of features, and deploy cross-validation on the training dataset in the hope of finding the best parameters for our system. We employ a slightly different strategy to search for the best parameters, based on our experience with System 2. Since we expect that, similarly to what was observed in the SVM experiment, regularization parameters will have the biggest impact on the result, we deploy Randomized Search 5-fold cross-validation[18] that allows us to experiment with a range of values for both the L1 and L2 regularization parameters[19]. The results are described in the next section.

For System 5, a combination of dense representation and sparse representation of features was adopted. We first converted the data into a numerical format and combined traditional features with word embedding using gensim [20] to load the word2vec pre-trained Google News word vector (300d) model[21]. Traditional features, such as *affix* and *base* are converted into one-hot encodings. For word embeddings, *lemmas*, *preceding lemmas* and *next lemmas* were converted into 300 dimensional dense vectors and

[16] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[17] https://sklearn-crfsuite.readthedocs.io/en/latest/api.html#module-sklearn_crfsuite
[18] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
[19] Inspiration for this is found in the sklearn-crfsuite tutorial: https://sklearn-crfsuite.readthedocs.io/en/latest/tutorial.html
[20] https://radimrehurek.com/gensim/models/keyedvectors.html
[21] https://code.google.com/archive/p/word2vec/

further concatenated with one-hot encodings.

In respect to the hyperparameter settings, we randomly test the hidden layer between 100 and 500 and set random state to 2. According to the documentation[22], default solver *adam* works pretty well on relatively large datasets (with thousands of training samples or more) in terms of duration of the training time and validation scores. For the activation function, considering our model has many layers, default ReLU was used to add non-linearity to our model and to prevent vanishing gradients. Given that the neuron might get stuck in negative values and not contribute in discriminating the input when the learning rate is too high, we also set the default value to *0.001*. In terms of alpha, we decided to keep the default L1 regularization, since the output from ReLU is not restricted by any value limit by design, which means that in some cases, the output can continue to grow in size. Considering this, keeping the default regularization parameter helps us reduce the complexity of the model and prevent overfitting. For the random state setting, it is important to note that the bias weight in each neuron is set to zero by default and not for example, a small random value. However, nodes that are side-by-side in a hidden layer connected to the same inputs must have different weights for the algorithm to update them while learning. In other words, setting weight to zero will fail in making any changes to the network weights and the model will remain stale. That is why we decided not to keep the default settings of random state (Brownlee, 2018).

The output of all systems is in the format of the BIO tags. One label (B-NEG, I-NEG or O) is assigned to every token. B-NEG stands for ”beginning of negation”, I-NEG for ”inside negation” and O for ”outside negation”.

6.3 Evaluation

An overview of the experiments' results can be seen below:

System	F1 B-NEG	F1 I-NEG	F1 macro avg.
Baseline	0.895	0.000	0.631
System1	0.934	0.500	0.811
System2	**0.937**	0.500	0.812
System3	0.922	0.000	0.640
System4	0.931	**0.800**	**0.910**
System5	0.926	**0.800**	0.908

Table 11: Overview of results for all systems, best results in bold print.

The systems are evaluated by their precision, recall and F1-scores. The macro average score is chosen for evaluating the systems' overall performance instead of accuracy or the weighted average score, since we are dealing with a dataset that is extremely imbalanced. In our training and development data, I-NEG is a very rare class; we only find a few instances of it. The size of the B-NEG class is also very small in comparison to the O class. More specifically, evaluating on accuracy or the weighted average would not give an appropriate representation of the classifier's performance, since it would mostly be influenced by the O label that dominates the dataset.

In addition to the macro average scores, we evaluate our systems on each of the classes separately. Aside from the irrelevance of the scores for class O, the evaluation on the I-NEG class in particular will allow us to explore the extent that we can capture multi-word-expressions, even though we expect it to have a great impact affecting our evaluation scores in the case of misclassification.

In table 11 we select to present a summary of F1 scores for our two classes B-NEG and I-NEG and macro average F1 score for each of the systems. The full tables with our experiment results can be found in Appendix 8. We prefer to demonstrate the F1 scores of classes B-NEG and I-NEG instead of precision and recall scores to highlight how each system responded to the I-NEG rarity issue and how our methods of tackling this affects our results.

[22]https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

System 1 and System 2, SVM classifiers that use the full set of features, demonstrate an improvement over the baseline with a 0.811 (System1) and 0.812 (System2) macro average F1 score, compared to the baseline's 0.631. This demonstrates the contribution of adding additional, carefully-crafted features to the performance of classifiers. Still, we can observe that the improvement is mainly based on the system recognizing more B-NEG instances. It seems that our features themselves are not enough for a significantly better performance on the I-NEG class. We hypothesize those results might be improved if we used a bigger context window around every token as features, for example also representing one more lemma before and after the previous / next lemma.

System 3 using a CRF classifier with default parameters doesn't show much improvement in comparison to the baseline with a macro average F1 score of 0.640 (0.01 points better), failing to capture the I-NEG class completely. System 4 though, using a tuned CRF classifier with custom parameters, performs much better than the baseline, scoring 0.910; again this score being the result of managing to capture some I-NEG sequence-based classes.

Finally, System 5 using an MLP classifier, also has quite impressive results with an 0.909 F1 macro average score and manages to capture 2 I-NEG instances, but ultimately falls slightly short of the performance of System 4.

6.3.1 Ablation study

Since a CRF with fine-tuned parameters reaped the best macro F1 among all the systems, a feature ablation study has been done on this system to find out the best feature combination. It is done in two parts. In the first part, the features are removed one by one and the rest of the features are fed into CRF classifiers with fine-tuned to perform the same task. The more the F1 macro average drops, the more important that missing feature is. Table 12 lists the macro avg F1 of each missing feature and those of the CRF with no features removed ("NONE") from low to high.

Feature removed	F1: macro avg (low to high)
lemma	0.898
affix	0.903
base_is_word	0.906
next_lemma	0.907
prev_lemma	0.908
pos_category	0.908
has_affix	0.909
is_single_cue	0.910
NONE	0.910
base	**0.911**

Table 12: The macro avg F1 of each removed feature compared to "NONE" from low to high.

Out of the features, *lemma* and *affix* top the list, proving their significance in this negation cue detection task in general. *next_lemma* and *base_is_word* also considerably contributed to the system. On the other hand, feature *base* is detrimental to the system's performance, as removing it actually increases macro F1. Removing it from our feature set would improve the performance of the system on the development set and might lead to better performance on a different test set as well.

The second part is to remove more features to see if this will further improve the system. In addition to *base*, the least contributing features, i.e. *is_single_cue*, *pos_category* and *prev_lemma* are removed in a series of experiments with all possible combinations. Table 13 lists the macro F1 for each experiment.

As the results show, removing only the feature *base* or both *base* and textitis_single_cue features leads to a slight improvement of the classifier result on the development set, with the systems predicting one more B-NEG instance correctly or one less FP B-NEG respectively. Removing any other features from

Feature(s) removed	F1: macro avg (low to high)
base, is_single_cue, pos_category, prev_lemma	0.905
base, pos_category, prev_lemma	0.907
base, is_single_cue, prev_lemma	0.907
base, is_single_cue, pos_category	0.908
base, prev_lemma	0.908
base, pos_category	0.908
NONE	0.91
base, is_single_cue	**0.911**
base	**0.911**

Table 13: The macro avg F1 of the removed feature(s) compared to "NONE" from low to high.

the feature set would lower the performance of the classifier. This implies that, based on this ablation study, the best combination of features would be to use the features, i.e., *lemma*, *prev_lemma*, *next_lemma*, *pos_category*, *has_affix*, *affix*, *base_is_word*.

Similar ablation studies are also done with SVM and MLP, respectively. However, no system obtained a higher macro F1 than the fine-tuned CRF.

6.3.2 Impact of parameters/hyperparameters optimization

We experimented with optimizing the parameters of our SVM and CRF-based classifiers. Here, the performance of System 2 (SVM with fine-tune parameters) is compared to the performance of System 1 (the same system build using default parameters). The same is done in the case of CRF, comparing the fine-tuned System 4 to System 3 that uses the default parameters. The process of fine-tuning is described in more detail in the Classification experiments section.

In the case of the SVM systems, the difference in performance between using the linear kernel algorithm with default parameters (System 2) and fine-tuned parameters (System 3) is only slight. Predicting the development data labels using the combination of parameters that led to the best performance on the training dataset during cross-validation (the highest f1 macro average score obtained during cross-validation was 0.735), results in the F1 macro average score of 0.812. This is only 0.001 better than the System 2 performance: System 3 classified one more B-NEG instance correctly.

As for the CRF systems, the impact of choosing the parameters that are different from the default ones is more significant. The fine-tuned System 4 obtains the f1 macro average score of 0.816 on the training data during the cross-validation process. On the development dataset, it obtains a macro average F1 score that is 0.270 higher when compared to using System 3 with default settings (0.910, compared to 0.640 in case of System 3). The improvement in performance stems from the fine-tuned system correctly classifying 2 out of 3 I-NEG instances, while when using the default settings, none were predicted. Also, System 4 correctly predicts two more B-NEG instances than System 3.

6.3.3 Problems and Conclusion

This section discusses a set of experiments undertaken in order to explore the impact of using different machine learning algorithms, features and parameters for the token-level negation cue classification task. Each token is classified to the B-NEG, I-NEG or O class. Two machine learning methods — SVM and CRF are compared. After parameter fine-tuning, a CRF classifier's macro F1 outperforms an SVM one. Although not having significant impact on SVM, parameter setting is proven to be important to improve our CRF systems. A Multi-Layer Perceptron algorithm is also attempted. This deep learning

method concatenating word embedding for target and neighboring lemmas and one-hot encoding of other features. The result is quite impressive yet slightly worse than a fine-tuned CRF. After feature ablation experiments, our best system is a fine-tuned CRF with all the features.

There are a few problems that we encountered during the experiments. Firstly, there are too few I-NEG in the dev set to evaluate. Correctly labeling as few as 1 I-NEG already makes a huge difference on the macro F1. The results on this dev set may not accurately reflect the real performance on a new data set with many more I-NEGs.

Secondly, the differences between the systems' performance are sometimes too minuscule to indicate which is better. Macro F1 of different systems can differ by only 0.001. This is especially prevalent in parameter tuning, where tiny adjustments to the C value can lead to slightly different results. Us choosing the best setting by the best score with a margin down to 3 digits after decimal risks overfitting the dev set.

Besides, hyperparameter optimization for MLP under cross-validation is practically infeasible. In our attempt, GridSearch is used for searching the best hyperparameter values. However, it took more than 90 minutes to run and was thereby deemed too costly for this task. If the system can be cross-validated and more hyperparameters are adjusted, it may generate even better results than the current ones.

In spite of all the complications, our fine-tuned CRF system still obtains a decent score. For the final evaluation on the test dataset, we intend to use this system and additionally experiment with feeding it the development data as additional training data and fine-tuning it using cross-validation on a combined training and development dataset.

6.4 Evaluation on the test set

For the final evaluation on the test set, we decided to use our best performing System 4 as described in the previous sections. Even though our ablation study showed that feature *base* has a slightly negative impact on the system, and feature *is_single_cue* doesn't have an impact, when testing on test data, we decided to keep all of our features because the difference in performance on the dev set was minimal, and we still felt that our feature set was well motivated. Besides, our system's parameters have been tuned using all the features, thus it made sense to keep the model intact. The performance of the system on development data is provided in Table 14.

	precision	recall	f1-score
B-NEG	0.947	0.915	0.931
I-NEG	1.000	0.667	0.800
O	0.999	0.999	0.999
macro avg	0.982	0.860	0.910

Table 14: Evaluation of System 4 predictions on the development dataset.

To test the classifier, we use the test dataset as described in Morante and Blanco (2012, p. 268), combining both The Adventure of the Red Circle and The Adventure of the Cardboard Box. The results are presented in Table 15.

	precision	recall	f1-score
B-NEG	0.919	0.922	0.920
I-NEG	0	0	0
O	0.999	0.999	0.999
macro avg	0.639	0.640	0.640

Table 15: Evaluation of System 4 predictions on the testing dataset.

We can observe that performance is significantly worse than on the development dataset. The drop in performance comes mainly from the fact that the model does not predict a single I-NEG instance. The

performance on the B-NEG class is impacted to a lesser degree, with the classifier now having worse precision, but slightly better recall than the classifier evaluated on the dev dataset.

Our final experiment consists of training the CRF classifier on the combined training and development datasets, hoping that more training data will lead to an increase in classifier performance. Fine-tuning is again used in the process of training the classifier, and we run a 5-fold cross-validation on the combined training and development set to obtain the best c1 and c2 parameters for the model before applying it to the test dataset. The best result obtained during this cross-validation is 0.798 macro average f1-score, which is lower than the score obtained previously while fine-tuning the system on only the training dataset (0.816).

The evaluation results are presented in Table 16. The performance after training on the combined dataset does improve, and the classifier now performs slightly better than it did on the development dataset. The performance of the model on the I-NEG class, however, is still regrettable with no I-NEG instances predicted, despite being trained on data containing 2 more multi-word negation cues. In this respect, the performance of the system is comparable to that of the two lowest performing systems we tested during our experiments, SVM with only token as feature (Baseline) and CRF with default parameters (System 3).

	precision	recall	f1-score
B-NEG	0.940	0.929	0.935
I-NEG	0	0	0
O	0.999	0.999	0.999
macro avg	0.646	0.643	0.645

Table 16: Evaluation of predictions on the testing dataset, after training and fine-tuning the CRF system on combined training and development datasets.

A possible explanation for this is the fact that, of the multi-word negation cues present in the testing data, only one is also present in the combined training and development data. That one cue, moreover, is *no more*, which was also misclassified by the model when evaluating it on the development data (see also the next section on error analysis for a more detailed account). Still, we would have hoped that adding this example to the training corpus will improve the performance of the classifier. We have to conclude that treating the I-NEG class is the most problematic part of the task, as already touched upon in the previous subsection. The fact that there are so few instances of this class in the data, and that we haven't designed any features to explicitly deal with them, probably contributed to the low performance.

7 Error Analysis

We carried out our error analysis on the predictions of our best performing System 4 on the development dataset. This provides the opportunity to examine in depth the sources of errors, since they are relatively few, as well as study our carefully designed final model more closely.

	B-NEG predicted	I-NEG predicted	O predicted
B-NEG gold	161	0	15
I-NEG gold	0	2	1
O gold	9	0	13379

Table 17: Confusion matrix for the model predictions on the development dataset.

Table 17 presents a confusion matrix demonstrating the misclassifications produced by the system. We observe that the most errors (15) happen because the system predicts the B-NEG instance as belonging to the O class ("false negative single cue prediction"). This is followed by the opposite case of predicting O class as B-NEG ("false positive single cue prediction"), which happens 9 times. Finally, there is one case of predicting a I-NEG class instance as O. Since our system only produced 25 errors, we decided to look at each error in detail. Below, we first discuss the errors belonging to each of the 3 classes just mentioned (false negative and false positive single cue prediction and false negative prediction of a multi-word cue). Then, we attempt a systematization of the observed errors that also provides inspiration for improving the system.

Of the 15 B-NEG instances predicted as belonging to the O class, two-thirds are prefixal negations, and the rest are words with the feature *is_single_cue* (words that are always or frequently negation cues). Looking at each in turn, we first observe that the system does not seem to have the same difficulty with all prefixes. The most commonly misclassified prefix is *in-* (4 times), followed by *dis-* and *im-*, which both occur twice, while *ir-* and *-un* are misclassified once each. This probably has to do both with the frequency with which each of the prefixes occurs in the dataset (for example, the most frequently misclassified prefix is also the most frequent prefix in general), and how frequently they belong to the B-NEG or O class in the training data. For example, with *dis-*, we observe that words starting with this prefix are rarely negation cues (5 out of 94 instances in the training dataset), while the opposite is true for *un-* (more than half of 141 instances in the training set are negation cues). This might explain why *un-* is so rarely misclassified, despite being more frequent than *dis-*. This polarizing classification of *dis-* and *un-* is also supported by the fact that all *dis-* in the dev set are predicted to be Os, and that this system correctly predicts 9 more TP B-NEG with prefix *un-* than the baseline. These prove the effect of having prefix as a feature.

Additionally, we observe that only 2 of those 10 words ever occur in the training dataset. Moreover, one of them - *impatience* - is misclassified in training, where it is assigned the label O. Only *inexplicable* is both present in the training dataset and labelled correctly. Furthermore, we observe that some features associated with those instances are incorrectly extracted. In the cases of *unkempt*, *irreproachable* and *inexplicable*, the feature *base_is_word* is assigned the value 0, and the feature *base* is not extracted. The same is true for *inadmissable* [sic], in which case incorrect feature extraction is probably the consequence of the fact that it is misspelled in the development dataset.

Finally, observing more carefully our training data, we notice a pattern: lemmas with the prefix *dis*, are frequently preceded and followed by lemmas, whose POS category is "OTH" (OTH stands for *other*, and it is a possible value of our feature *pos_category*). Despite not explicitly providing our classifier with information on the POS tags of previous and next lemma, having OTH as part of our features, could be a reason for the misclassification we are examining (tokens with prefix 'dis'). To further inquire on our hypothesis, we looked at the ratio of this pattern occurring as such also in the development data. Indeed, the result shows that the ratio is similar in both datasets. When we, on the other hand, examine instances of tokens starting with prefix 'dis' which are gold labeled as "B-NEG" in the training data, only 1 out of 5 instances complies with the pattern "preceding POS tag OTH + dis- + following POS tag OTH"

we described. In other words, our model, to some degree, recognizes the pattern and therefore perhaps makes a wrong prediction, that tokens with a 'dis' prefix, preceded and followed by "OTH" POS tags are more likely to be "O" instead of "B-NEG."

Of the 5 instances without negation prefixes, 4 are bona fide negation cues, where we are unsure of the possible reasons for misclassification. Those instances are *neither*, *nobody*, *not* and *nothing*, all of which appear in the development dataset multiple times, and are, with one exception each, otherwise correctly identified. Moreover, the only difference in terms of features between them is in the *next_lemma* and *previous_lemma* values. For example, the case of *not* could be the result of its *next_lemma* being *doubt*, as ***no*** *doubt* is sometimes a false negation. The more informative error in terms of possible improvements of the classifier is misclassification of *save*, since that very often not a negation cue. In this case, including the PoS category of the next lemma could help since when *save* serves as a negation cue it usually has a preposition as next lemma (*save upon*, *save for*).

In the category of instances classified as B-NEG and with the gold label O, we again have 5 instances from our list of negation cues (*no*, *none* – which occurs twice, *absence*, *never*), and 4 times we have affixal negation (two times the prefix *un-*, and suffixes *-less* and *-lessly*). For the group of errors belonging to affixal negation, it actually seems that there are three instances where the gold label annotation is incorrect (*unhappily*, *endless*, *listlessly*). This also seems to be the case for two single word negation cues, *no* and *never*, which we would classify as B-NEG according to their context (*and no pain*, *man never lives*). Thus, it seems that in this group there are actually only 4 errors and not 9: *understand*, *absence* and *none* (twice). The case of *understand* is interesting, since it occurs in the dev set multiple times. Every time it is correctly classified, it also has a correct PoS tag indicating it is a verb, while the one erroneous prediction is done in the only instance where an incorrect PoS tag is provided as a feature (adjective). Improving preprocessing might help in this case. *Absence* is a similar case as *save*, since it is again a word that is frequently not a negation cue. Again, a feature like next lemma PoS tag might help, because it also usually has a preposition following it when it functions as a cue (*absence of*). Finally, the word *none* is misclassified twice, both times in the phrase *none the less*. Here, a bigger window could potentially help, capturing both *the* and *less* as features. Or, we could possibly incorporate a ruled-based component in our system, as was done by Chowdhury (2012) for the same case.

The last group of errors concerns multi-word negations, of which there are only two occurrences in our corpus. The first, *by no means*, is predicted correctly by our model, but in the case of *no more*, only *no* is correctly labelled as B-NEG and *more* with gold label I-NEG is falsely labelled as O. The reason for this could be that *by no means* is also present in the training data, while *no more* is not. While it is already difficult for the machine to label an unseen I-NEG without enough features, the ill-defined PoS categorization might have worsened our system's ability. In the training data, 10 out of 16 I-NEG instances belong to the "OTH" category, followed by 4 NNs and 2 ADJs. It is possible that our system built an overpowering correlation between OTH and the label I-NEG, thus lacked clues to identify *more*, an ADV, as I-NEG. Moreover, we suspect that capturing multi-word negation cues is a challenge which might not be sufficiently tackled by the only features we have at the moment to address it: previous and next lemma. It should be beneficial for future endeavors, to have a feature dealing more directly with multi-word cues and their particularities.

It seems that the 25 classification errors we found in the development dataset can be classified in a few broad categories. The first big group are errors that stem from the dataset itself, predominantly from incorrectly labelled instances. Those can be further subdivided into erroneous labels in the development dataset, where the prediction is actually right but reported as wrong in the confusion matrix (*unhappily*, *no*, *never*), and false labels in the training dataset, based on which the system learned wrong predictions (*impatience*). Moreover, we frequently observe misclassifications for instances that were never encountered during training: training on more data would probably improve the classification. Then, there are instances where features have been incorrectly extracted, either during the preprocessing phase (incorrect PoS tag), or when deciding whether the base of the instance containing a prefix was a valid word. Improving the preprocessing and feature extraction methodology might lead to a more correct classification. Additionally, there are cases where we feel we might be lacking a certain feature that could aid

classification. As mentioned, in the cases of words that are often not negation cues, providing the PoS tag of the next lemma might be beneficial. For cases such as the false negation cue *none the less*, having a larger context window might aid classification. Lastly, we found a number of errors where we can't hypothesize, at this point, what might have gone wrong. Fortunately, these types of errors account for only around 20% of errors we investigated.

8 Conclusions and Discussion

This paper attempted an approach towards the task of negation cue detection. To experiment with machine learning methods, SVM and CRF classifiers were tested and evaluated to find out the best performing set up of a system.

Regarding parameter optimization, despite not finding significant difference when it came to the SVM models, parameter fine tuning greatly elevated the CRF systems' macro F1, thanks to much higher F1 for I-NEG. In addition, experiments were run on a multi-layer perceptron (MLP) classifier. Word embeddings for target and neighboring lemmas were concatenated with one-hot encodings of traditional features. Still, a fine-tuned CRF performed the best with better results on the B-NEG class. A feature ablation study was conducted to find out the best feature combination, leading to no significant improvement by removing any of the features.

The fine-tuned CRF with all features was finally tested on an unseen test dataset. Regrettably, the system performed slightly worse for B-NEG compared to the development set and fell short of predicting any I-NEG, making the macro F1 plunge drastically. This might have been because all the I-NEGs in the test set did not exist in the training data at all, proving the features' failure in helping the system with unseen I-NEGs. Another experiment was run when the same setting was trained on not only the training dataset but with the development set combined. Although no improvement was found for I-NEG, the F1 for B-NEG increased from 0.920 to 0.935, proving that a larger corpus of training data could improve the system's performance.

Finally, an error analysis was performed on our best system's predictions on the development set. The errors indicated deficiencies of our features. To begin with, the features did not help the system to classify unseen words. The system also could not differentiate false and bona fide affixal negations. Then, necessary sequential information was lacking with the current set of features. Not only did the system lack access to a larger context for phrases, it was affected by improper PoS categorization. Its performance in distinguishing B-NEG in fixed expressions and I-NEG was unsatisfactory. Lastly, some suspicious cases of PoS tagging and *base_is_word* extraction in earlier stages that could have caused some errors were found.

When reflecting on negation cue detection as a problem solving task, one realises there will always be the issue of an imbalance of classes in the training data. Negation cues are always quite scarce in natural language compared to non-negation cues, as they may only occur, for example, once in a few sentences. It seems then, that a huge O class will always be a challenge to tackle. I-NEG is even rarer, thus posing a major difficulty for the classifier to correctly predict I-NEGs in particular. It was especially disappointing when the system detected none of the I-NEGs on the test set.

Despite the effort of crafting features to tackle these issues, the attempt was proven inadequate and our strategy of approaching the task would need improvement. The annotation analysis helped us identify the challenges with false cues, including prefixal and single ones in fixed expression. For example, features including *affix, base_is_word* and *base* were designed to distinguish between e.g. *disturb* – O and *dislike* – B-NEG. Features *prev_lemma* and *next_lemma* were designed for contextual information, e.g. *save* is B-NEG when *next_lemma* is *for* or *upon*. As the system failed to make such distinctions, these features were proven not useful enough. Our system would need more features to solve the unseen word problem. The predictions of "unseen" words that do not exist in the training set, like *irreproachable* or *insensibly*, and unseen I-NEGs like *no* ***more***, were unsuccessful. Features beyond the affix and the complete base substring, and beyond the previous and the next lemma, would be needed to help the machine decide.

Apart from the features, whether CRF is the best algorithm to approach this task is still open to discussion. In this study, the reason why our MLP classifier had a slightly inferior macro avg F1 could be that it did not adopt hyperparameter tuning. Initially, we used Grid Search to find the best hyperparameters. The system was trained on 65451 samples of 6 traditional features with 5 parameters: solver, alpha, hidden layer, activation, and learning rate, each specified with 2 options. In Grid Search, this setting tried 2 x 2 x 2 x 2 x 2 = 64 different parameter combinations for each fold. Over the parameter space of 64 with a 5-fold cross-validation with roughly 2 minutes to run a set a parameters, it would take almost 64 X 5 X 2 = 640 minutes in total to obtain the result. It is deemed impractical to spend such computing cost

for this task. It can be argued that with a cost-effective way to fine-tune, a deep learning algorithm may actually serve as a better choice than a CRF.

With reference to the constraints of the current study, the following suggestions are made for future work.

Firstly, there are negation cues that need more sequential information, including single cues like *save* and *absence* where the context matters, as is also the case with multi-word cues (MWCs). In future work we can incorporate more features, like neighboring PoS, as some patterns related to the previous or the next PoS were observed. This may be the reason why PoS n-grams were also used by Abu-Jbara and Radev (2012). Another feature possible feature is to have bigger window sizes for neighboring lemmas for expressions like *none the less*. Besides, an MWC list similar to the single cue list used in this paper can be built to help detect MWCs, especially unseen MWCs that do not exist in the training data. Features for distinguishing false and bona fide affixal negation cues can also be experimented in future work. In addition to capturing the complete form of the base, other features like character n-grams and lexicon counts of the base can be incorporated, as they are also attempted by Lapponi et al. (2012). In general, future studies can look into a more conscious matching of features and system design. Various models can be experimented with features crafted to complement the respective algorithm each time, instead of using the same features for all models as done in the present study.

Moreover, some details of our experiments are worth adjusting and comparing. The first adjustment is using a different PoS tagger during preprocessing. In the current study, NLTK is used to get the PoS tags of the tokens. However, some suspicious PoS tags were found (e.g. ADJ for *understand*) and it presumably caused an error in classification. Using a different PoS tagger is likely to yield different behavior of the classifier.

The second adjustment is the process of feature extraction. This include the matching of *base_is_word* and the categorization of *pos_category*. For *base_is_word*, it has been found that *unkempt*, *irreproachable* and *inexplicable* were incorrectly given the value 0. Their bases, i.e. *kempt*, *reproachable* and *explicable*, exist in the Oxford dictionary [23] but not the Gutenburg corpus used in this study, nor other large corpora like Brown or Reuters accessible in NLTK. In order to have a more accurate assessment, some other lexical resources may be needed to determine whether the base should be treated as a valid, stand-alone word. The "OTH" category may also be an overgeneralization of several PoS types, which may have caused the system to develop some erroneous correlation between "OTH" and the class. Future work that adopt finer subdivision of PoS categories, such as preserving determiners (DT) and prepositions (IN), is encouraged.

The final adjustment is to use Randomized Search instead of Grid Search for our MLP classifier. With Grid Search being too time-consuming, Randomized Search could be adopted to yield similar results with less experiments. If we have 3×3 set of parameters, for example, Grid Search may not find the optimal parameters since we do not have them in our grid. With Random Search, on the other hand, the parameters are selected randomly with a specified iteration number while the process takes much less time. It would be interesting to fine-tune an MLP classifier and compare it to a CRF, to find out which algorithm would be optimal for approaching the task of negation cue detection.

[23] https://www.lexico.com/

Appendix 1: distribution of work for assignment 1

Lahorka Nikolovski Wrote the Chowdhury paper review. Worked on writing section 2.1. Assisted in reviewing the report. Created the GitHub repository.

Shuyi Shen Wrote the AbuJbara-Radev paper review. Worked on writing section 2.1. Assisted in reviewing the report.

YC Roderick Li Wrote the Lapponi-et-al paper review. Worked on writing section 2.1. Assisted in reviewing the report.

Panagiota Tselenti Wrote the task description section. Assisted in reviewing the report.

Appendix 2: distribution of work for assignment 2

Lahorka Nikolovski Annotations, calculating IAA, wrote the description of the annotated corpus, annotation task and annotators, edited the final draft and made LaTeX tables.

Shuyi Shen Annotations, calculating IAA, disagreement analysis and reflection for the Windows group.

YC Roderick Li Annotations, calculating IAA, wrote the disagreement analysis and reflection for the Mac group and edited that section.

Panagiota Tselenti Annotations, calculating IAA, contributed in writing sections: scores overview, disagreement analysis, reflection, edited & reviewed the assignment.

Appendix 3: distribution of work for assignment 3

Lahorka Nikolovski Wrote the data description and corpus pre-processing parts of the report. Analyzed the data. Wrote the preprocessing script. Reviewed the feature extraction part of the report and helped with the feature extraction script. As the group leader, coordinated the work done this week.

Shuyi Shen Wrote the report on the description and motivation of feature extraction for features #1-4. Reviewed others' work. Wrote the feature extraction script and revised it.

YC Roderick Li Wrote the report and script for affix-related features (#6-9). Reviewed others' work.

Panagiota Tselenti Wrote the report and script for list-related feature (#5). Reviewed others' work.

Appendix 4: distribution of work for assignment 4

Lahorka Nikolovski Worked on the design and implementation of the SVM system. Designed and implemented the CRF system. Wrote sections of the report about the CRF system and worked on the sections concerning SVM. Reviewed others' work.

Shuyi Shen Build baseline using SVM classifier. Finalize the code and wrote description and motivation for building MLP classifier with ablation analysis.

YC Roderick Li Feature ablation. Problems and conclusion. Reviewed others' work.

Panagiota Tselenti Worked on composing multiple sections of the report. Assisted with decision-making on system design and ablation analysis. Reviewed others' work.

Appendix 5: distribution of work for assignment 5

Lahorka Nikolovski Wrote code to evaluate the classifier on test data. Wrote Evaluation on the test set subsection. Performed error analysis and wrote a part of the Error analysis section. Updated some previous parts of the report based on received feedback. Reviewed the abstract.

Shuyi Shen Contributed to error analysis. Drafted conclusion. Updated suggestion and reflection parts of fine-tuning for the report. Reviewed others' work.

YC Roderick Li Contributed to error analysis. Drafted conclusion. Updated previous parts of the report. Reviewed others' work.

Panagiota Tselenti Wrote abstract, co-wrote error analysis, co-wrote conclusions, reviewed others' work.

Appendix 6: list of documents our group annotated

The file names of the 10 documents from Batch 4 are provided below.

1. huffingtonpost-com_20161221T170356
2. info-cmsri-org_20170226T070402
3. Infowars_20160227T205616
4. justthevax-blogspot-nl_20161107T201500
5. Kid-Nurse_20170619T190950
6. Kid-Nurse_20170627T195033
7. latimes-com_20170206T032149
8. latimes-com_20170628T014330
9. LDI_20150623T130008
10. LifeSiteNews_20170611T045238

Appendix 7: list of single word negation cues used to extract the feature *is_single_cue*

Lemmas marked with * are found in the training dataset; others were included based on Morante (2010).

1. absence*
2. absent
3. cannot
4. decline
5. deny
6. either
7. except*
8. exclude
9. fail*
10. failure
11. favor
12. lack
13. loss
14. miss
15. n't*
16. negative
17. neglect*
18. neither*
19. never*
20. no*
21. nobody*
22. none*
23. nor*
24. not*
25. nothing*
26. nowhere*
27. prevent*
28. refuse*
29. resolve
30. save*
31. without*

Appendix 8: full results for the 5 experimental systems, best results in bold print

Baseline system	precision	recall	f1-score
B-NEG	0.922	0.869	0.895
I-NEG	0.000	0.000	0.000
O	0.998	0.999	0.999
macro avg	0.640	0.623	0.631

Table 1: Evaluation report for the baseline SVM system

Baseline system	B-NEG predicted	I-NEG predicted	O predicted
B-NEG gold	153	0	23
I-NEG gold	1	0	2
O gold	12	0	13376

Table 2: Confusion matrix for the baseline SVM system

SVM system	precision	recall	f1-score
B-NEG	0.942	0.926	0.934
I-NEG	**1.000**	0.333	0.500
O	0.999	0.999	0.999
macro avg	0.980	0.753	0.811

Table 3: Evaluation report for the SVM system with default parameters, using all features

SVM system	B-NEG predicted	I-NEG predicted	O predicted
B-NEG gold	163	0	13
I-NEG gold	0	1	2
O gold	10	0	13378

Table 4: Confusion matrix for the SVM system with default parameters, using all features

SVM system, ft	precision	recall	f1-score
B-NEG	0.943	**0.932**	**0.937**
I-NEG	**1.000**	0.333	0.500
O	0.999	0.999	0.999
macro avg	0.980	0.755	0.812

Table 5: Evaluation report for the SVM system with fine-tuned parameters, using all features

SVM system, ft	B-NEG predicted	I-NEG predicted	O predicted
B-NEG gold	164	0	12
I-NEG gold	0	1	2
O gold	10	0	13378

Table 6: Confusion matrix for the SVM system with fine-tuned parameters, using all features

CRF system	precision	recall	f1-score
B-NEG	0.941	0.903	0.922
I-NEG	0.000	0.000	0.000
O	0.999	0.999	0.999
macro avg	0.646	0.634	0.640

Table 7: Evaluation report for the CRF system with default parameters, using all features

CRF system	B-NEG predicted	I-NEG predicted	O predicted
B-NEG	159	0	17
I-NEG	1	0	2
O	9	0	13379

Table 8: Confusion matrix for the CRF system with default parameters, using all features

CRF system, ft	precision	recall	f1-score
B-NEG	**0.947**	0.915	0.931
I-NEG	**1.000**	**0.667**	**0.800**
O	0.999	0.999	0.999
macro avg	**0.982**	**0.860**	**0.910**

Table 9: Evaluation report for the CRF system with fine-tuned parameters, using all features

CRF system, ft	B-NEG predicted	I-NEG predicted	O predicted
B-NEG gold	161	0	15
I-NEG gold	0	2	1
O gold	9	0	13379

Table 10: Confusion matrix for the CRF system with fine-tuned parameters, using all features

MLP system	precision	recall	f1-score
B-NEG	0.926	0.926	0.926
I-NEG	**1.000**	**0.667**	**0.800**
O	0.999	0.999	0.999
macro avg	0.975	0.864	0.908

Table 11: Evaluation report for the MLP system, using all features

MLP system	B-NEG predicted	I-NEG predicted	O predicted
B-NEG gold	163	0	13
I-NEG gold	0	2	1
O gold	13	0	13375

Table 12: Confusion matrix for the MLP system, using all features

References

Amjad Abu-Jbara and Dragomir Radev. 2012. UMichigan: A conditional random field model for resolving the scope of negation. In **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 328–334. ACL.

Benita Kathleen Britto and Aditya Khandelwal. 2020. Resolving the scope of speculation and negation using transformer-based architectures. *ArXiv*, abs/2001.02885.

Jason Brownlee. 2018. When to use mlp, cnn, and rnn neural networks. `https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/`. Accessed: 2022-01-27.

Md Faisal Mahbub Chowdhury. 2012. FBK: Exploiting phrasal and contextual clues for negation scope detection. In ** SEM 2012: The First Joint Conference on Lexical and Computational Semantics–Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 340–346. ACL.

Emanuele Lapponi, Erik Velldal, Lilja Øvrelid, and Jonathon Read. 2012. Uio 2: sequence-labeling negation using dependency features. In ** SEM 2012: The First Joint Conference on Lexical and Computational Semantics–Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 319–327. ACL.

Saeed Mehrabi, Anand Krishnan, Sunghwan Sohn, Alexandra M Roch, Heidi Schmidt, Joe Kesterson, Chris Beesley, Paul Dexter, C Max Schmidt, Hongfang Liu, et al. 2015. Deepen: A negation detection system for clinical text incorporating dependency relation into negex. *Journal of biomedical informatics*, 54:213–219.

Roser Morante and Eduardo Blanco. 2012. * SEM 2012 shared task: Resolving the scope and focus of negation. In ** SEM 2012: The First Joint Conference on Lexical and Computational Semantics–Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 265–274. ACL.

Roser Morante, Sarah Schrauwen, and Walter Daelemans. 2011. Annotation of negation cues and their scope: Guidelines v1. *Computational linguistics and psycholinguistics technical report series, CTRS-003*, pages 1–42.

Roser Morante, Chantal Van Son, Isa Maks, and Piek Vossen. 2020. Annotating perspectives on vaccination. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 4964–4973. ELRA.

Roser Morante. 2010. Descriptive analysis of negation cues in biomedical texts. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, pages 1429–1436. ELRA.

Chantal van Son, Emiel van Miltenburg, and Roser Morante. 2016. Building a dictionary of affixal negations. In *Proceedings of the Workshop on Extra-Propositional Aspects of Meaning in Computational Linguistics (ExProM)*, pages 49–56. The COLING 2016 Organizing Committee.