Supervised binary classification: detecting counterfactuals

Methodology

Our baseline was established by using a linear classifier (SGDClassifier from scikit-learn) taking as input the raw text from our corpus vectorized with a Bag-of-Words model. Additional features were tested upon this baseline. Each newly introduced feature in the table at the end of this report builds on the previous features that were used to obtain the best results. Features that were kept appear in bold in the table. The feature selection process is discussed in the next sections following a bottom-up approach to the linguistic representation of a sentence — from lexicon to semantics.

Text vectorization

To perform machine learning on our textual data, we need to turn each sentence into a numerical feature vector. Here are two methods we can use:

- **Bag-of-Words representation (BoW)**: This consists in counting the occurence of each token with no regard to syntactic structure. We collect a vocabulary of all occurring tokens, and then we attribute a score to each token, using its count or its frequency.
- **TF-IDF**: The problem with the BoW model is that it may attribute more importance to frequent words. Rarer words specific to a domain are given less importance while they could potentially contain useful information. To deal with this issue, a common method consists in scaling word frequency by how often they appear across all sentences, so that more frequent words are penalized. This method is called Term Frequency – Inverse Document Frequency.

Text preprocessing

Experimenting with different preprocessing steps showed that cleaning the sentences was best kept to a minimum for better results. Removing stop words and removing punctuation yielded worse results, while lowercasing the text improved the results. This fact corroborates our previous linguistic observations. Removing stopwords failed to provide better results since counterfactual sentences are frequently built using modal verbs which would have been featured in the preset stop words lists. Using lemmatization scored slightly lower than the simple preprocessing mentioned above. This can be explained by the fact that counterfactual sentences tend to be constructed with specific verb forms, a common structure being: if + past perfect, would + have + past participle. The following two example sentences, one being counterfactual and the other non-counterfactual, show how lemmatization hinders classification.

- Counterfactual sentence: “*If it had rained, you would have been wet.*”
    - Lemmatized: “*If it have rain, you would have be wet.*”

- Non-counterfactual sentence: “*If it has rained, you will be wet.*”
    - Lemmatized: “*If it have rain, you will be wet.*”

By using lemmatization, we ignore these various inflections that provide relevant information for this task.

Adding lexical information

Our intuition tells us that counterfactual sentences would tend to be formed with a specific lexicon. To confirm this hypothesis, we can use statistical methods. The chi-squared test allows us to test if two categorical variables are related. In this case, the two variables that were tested were counterfactuality and the presence of a keyword. All unigrams and bigrams in the corpus were systematically tested for statistical relevance in this classification task. These unigrams and bigrams were then ordered by their score, i.e. their p-value, and the top 50 were kept as features. Below are some of the most common unigrams and bigrams along with the weight they were attributed by the SGD classifier.
It was noted that specifically adding them as one-hot encoded features did not improve the results. While this lexical analysis brought insights into the construction of counterfactual sentences, their presence in a sentence was already encapsulated by the vectorizers. The classifier automatically learns the parameters and attributes higher weights to the statistically relevant keywords.

10 most common tokens and their weights:
('would', 2.19)
('say', 1.54)
('I', 1.77)
("'s", 0.02)
('may', 0.22)
('Mr.', -0.84)
('even', -0.38)
('year', 0.42)
('get', -0.10)
('take', -0.16)

4 most common bigrams and their weights:
('I would', -0.35)
('Mr. Trump', -0.45)
('I wish', -0.64)
('I could', -0.34)

Adding syntactic information

Syntax plays a central role in conveying meaning in a language. The use of TF-IDF is an efficient way to turn sentences into feature vectors, but it fails to capture syntactic information, i.e. how words are connected to each other. This information is particularly relevant to this task since the construction of counterfactual conditionals relies heavily on the use of syntax. As we have seen earlier, common counterfactual constructions include the use of specific verb tenses but also subject-verb inversion to express an hypothetical meaning

E.g. *"The vote would have been 65 to 35 had all senators been in attendance"*

In order to provide this information to the classifier, individual tokens were merged with their corresponding part-of-speech (POS) tag in a single string, e.g. 'the_DET'. SpaCy provides a convenient way to simultaneously retrieve tokens and POS tags. This merging yielded slightly better results than just having the tokens.
Along the same line, setting the ngram range of the TF-IDF vectorizer to include bigrams and trigrams improved the results compared to just using unigrams or unigrams and bigrams. This allows frequently occurring sequences of words to appear in the feature set. Additionally, since the construction of counterfactuals is intricately tied with the use of specific verb forms, frequent sequences of POS tags containing at least one verb were considered as an additional feature. Below are the distribution ratios for the most common POS tags trigrams, respectively for non-counterfactual and counterfactual sentences. It can be noted that the typical sequence ('MD', 'VB', 'VBN') — e.g. 'would have done' — is more than twice as likely to occur in a counterfactual sentence.

Most common POS tags trigrams (0,1):
('PRP', 'MD', 'VB') : 0.38 | ('MD', 'VB', 'VBN') : 0.65
('MD', 'VB', 'VBN') : 0.28 | ('PRP', 'MD', 'VB') : 0.44
('NN', 'MD', 'VB') : 0.25 | ('IN', 'PRP', 'VBD') : 0.3
('MD', 'RB', 'VB') : 0.23 | ('MD', 'RB', 'VB') : 0.2
('VB', 'DT', 'NN') : 0.22 | ('NN', 'MD', 'VB') : 0.17
('MD', 'VB', 'DT') : 0.18 | ('VB', 'VBN', 'DT') : 0.16

Along POS tagging, spaCy features a dependency parser to extract dependency relations from a sentence. These relations were not deemed useful for this classification task because this type of analysis focuses on the surface level of the sentence. Listing the proportion of all dependency relations for positive and negative examples revealed that there were no significant differences in their distribution and thus confirmed this intuition.

Adding semantic information

While TF-IDF does provide semantic information, it is rather limited as words are considered in isolation, omitting the fact that words are relevant in the context they occur in. Linguist J. R. Firth famously said: "You shall know a word by the company it keeps". The distributional hypothesis suggests that words that are semantically similar tend to occur in similar contexts. The idea of learning representations for words has been around for many years in

the field of NLP: TF-IDF is used to do so by creating a document-term matrix where each row represents one example from the corpus and each column represents a word from the vocabulary. This results in a large sparse matrix. In 2013, a novel method to learn dense embeddings called Word2Vec was introduced by Mikolov *et al* in their seminal paper[1]. In a nutshell, this method trains a language model to predict a word given its context and then assigns the weights that have been trained for each word as embeddings. In this way, words with similar meanings are attributed similar numerical representations and it turns out that dense vectors perform better than sparse ones in every NLP task.
Using the library Gensim, we are able to train our custom word embeddings on the dataset using Word2Vec, since the corpus at hand is relatively small. Words with similar meanings appear close to each other in the vector space (see the words related to 'if' below).
Word2Vec's shortcoming is that only tokens are considered for the embedding. In order to produce a single vector for one training example (i.e. a sentence), we can sum all the words and average them, thereby obtaining a vector representation of the sentence.
Another useful application of word embeddings is that one can reuse embeddings that have been pre-trained on a very large corpus. Using Gensim, we can access models that have been pre-trained on various datasets. The model 'word2vec-google-news-300' was used to create embeddings for sentences in the same manner as the previous step.
While this served as an interesting investigation of word embeddings, using the embeddings from these two methods as features yielded worsened results.

```
model.wv.most_similar('if'):
 ('unless', 0.75),
 ('though', 0.73),
 ('assuming', 0.72),
 ('once', 0.71),
 ('somehow', 0.68),
 ('convinced', 0.68),
 ('supposing', 0.68),
 ('necessarily', 0.67),
 ('actually', 0.66),
 ('glad', 0.66)
```

The last features that were considered are mood and modality. The Pattern library provides a simple way to extract these two features from sentences. Grammatical mood refers to the use of auxiliary verbs (e.g., could, would) and adverbs (e.g., definitely, maybe) to express uncertainty. The mood() function returns either INDICATIVE, IMPERATIVE, CONDITIONAL or SUBJUNCTIVE for a given parsed sentence. For this specific task, establishing whether a sentence's mood is conditional could be useful to classify it as counterfactual. The modality() function returns the degree of certainty as a value between -1.0 and +1.0, where values > +0.5 represent facts. For example, "I wish it would stop raining" scores -0.35, whereas "It will stop raining" scores +0.75. Accuracy is about 68% for Wikipedia texts. While accuracy is rather low, a brief analysis on the data showed that the mean modality value for

[1] https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

non-counterfactual sentences was higher than for counterfactual sentences (0.14 against -0.02). Adding modality as a feature slightly improved the model's accuracy. While mood did seem like a promising feature to include as well (see distributions below), it did not improve the results further.
Mood distributions (0,1):
('indicative', 10.86), ('indicative', 2.0)
('imperative', 0.85), ('imperative', 1.0)
('conditional', 84.25), ('conditional', 88.33)
('subjunctive', 4.04), ('subjunctive', 8.67)

An examination of the library's source code[2] showed that the contributors wrote thorough linguistic rules to analyse sentences. These rules could serve as inspiration to combine a machine learning classifier with a rule-based system. This method could potentially improve accuracy by filtering false positives and dealing with false negatives[3].

Hyperparameter optimization

The following classifiers available in scikit-learn were tested: RandomForestClassifier, svm.SVC, MultinomialNB, SGDClassifier. Among these estimators, SGDClassifier performed the best with any features. The final step to make the predictions a tad more accurate is to tune the model's hyperparameters. To do so, an exhaustive grid search was performed using GridSearchCV with a grid of parameter values and the best combination was kept.

Macro-averaged metrics (SGD)

	Precision	Recall	F1-score
Raw text - Count vectorizer	0.83	0.68	0.70
Raw text - TF-IDF	0.82	0.68	0.73
Adding subtask2 data	0.85	0.82	0.84
Preprocessing			
Lemmatization	0.83	0.80	0.82
Tokenization	0.85	0.82	0.84
Syntactic information			
Merging Token_POS tag	0.86	0.83	0.84
Ngram range(1,3)	0.88	0.85	0.86

[2] https://github.com/clips/pattern/blob/master/pattern/text/en/modality.py
[3] https://www.semanticscholar.org/paper/Hybrid-Approach-Combining-Machine-Learning-and-a-Villena-Rom%C3%A1n-Collada-P%C3%A9rez/6e2f9f203db02be6d7a90d7b6c6819c1c65ab6ae

POS tags sequences	0.88	0.87	0.87
Semantic information			
Custom embeddings	0.85	0.79	0.80
Pretrained embeddings	0.86	0.80	0.81
Modality	0.89	0.87	0.88
Mood	0.87	0.87	0.87