Contradiction Detection with Spacy, Scikit-Learn, and Flask

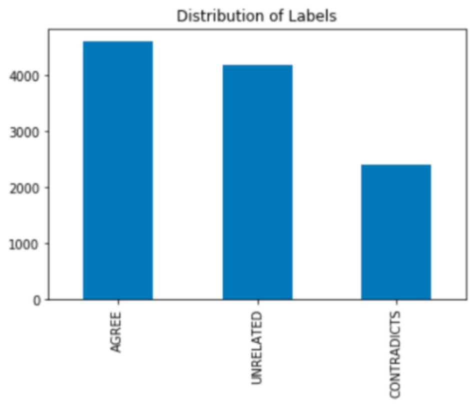
Gautier Dagan

Problem: The aim of this exercise is to design an NLP system that can detect contradictions in a given pair of sentences.

Proposed Solution: Due to the nature of the small dataset, the approach chosen was *feature engineering followed by a logistic regression* model. This has several advantages: out-of-vocabulary words will not have much of an impact on the accuracy of the model, it requires a minimal setup and little data, it is explainable, and it is extremely quick at inference. Its limitations are that it remains a hand crafted solution, where the features are only as good as they have selected to. All features also treat the sentences as a Bag of Words, thus we also lose the sequential information contained which might be crucial in some cases for additional context.

Dataset: I use the Stanford Contradiction Corpora namely all the three way RTE 1,2, and 3 sets and the two negated contradictions and real-life contradictions datasets. The three way dataset labels are mapped from "YES" to "AGREE", "NO" to "CONTRADICT" and "UNKNOWN" to "UNRELATED". The negated contradictions are mapped from "YES" to "CONTRADICT" and "NO" to "UNKNOWN".

All datasets are concatenated to form the dataset which I will use to build the model. Additionally, I reverse the *text* and *hypothesis* sentences and replicate the dataset with the switched sentences but same labels. The terms *text* and *hypothesis* are used to refer to the sentence pair, where the hypothesis sentence is typically tested for contradiction against the text sentence. However, since we expect to deal with real world data, neither sentence will be determined to be "text" or "hypothesis", we desire an algorithm which would predict the same regardless, we can augment the dataset by switching the two pairs.



The final dataset comprises of 11,194 examples with the label distribution shown above.

Feature Extraction: I use **spacy** in order to preprocess sentences (remove stop words and punctuations) and lemmatize. Using the spacy en_core_web_sm which is a small English language model trained on blogs, news, and comments. It is able to also do some basic Named Entity Recognition and therefore during the preprocessing I also extract the set of named entities for each sentence.

I extract features from the sets of preprocessed words and named entities for both the *text* and *hypothesis* sentence.

Features used:

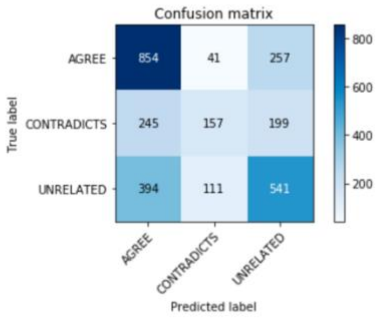
- **Overlap**: Overlap is calculated as the size of the intersection between two sets. This is calculated for both the preprocessed word sets and the extracted named entity sets.
- **Text or Hypothesis Extra**: Text set minus the hypothesis set and vice versa. This indicates how much longer is either sentence compared to the other but also how many words exist outside the intersection of both. This is also calculated for the named entity sets.
- **Jaccard Similarity**: Jaccard Similarity or Intersection over Union describes the intersection of the two sentences over their union. This is a number between 0 and 1 where 1 happens when the two sentences are the same or contain exactly the same set of words. This gives the model an intuition on the similarity between sentences.
- **Small Set Jaccard Similarity**: JS normalized over the smaller of both sets instead of the union. The idea was to capture the similarity when the difference between the sizes of the two sentences is large

- **Negation feature** number of negations present in each sentence where a negation is a word in the set: (deny, fail, never, no, nobody, noone, not, nothing, reject, without, none, cannot, nor, n't). This is calculated for the non-lemmatized word sets.

This gives us a total of 10 features.

I also tried using the **spacy document similarity** based on the average word embeddings – in order to capture semantic information. This improved F1 and mean accuracy by 0.02 and 0.04 respectively, but I decided against because of the size of the larger embedding file required. Though this indicates that using a transfer learning approach using pre-trained embeddings could help the accuracy of the model further.

Results and Evaluation: In order to select the model architecture, I first split the dataset using a stratified split to obtain a training set and test set. The test set was 25% of the total data. I then ran k-fold cross validation with a fixed seed on the training set, and tested multiple architectures such as a support vector machine , a 2 layer MLP, a random forest classifier and logistic regression. I noted very minimal differences in mean accuracy between each approach, I settled on using logistic regression due to its simplicity as well as comparably high performance.



The multi-classification logistic regression model by using 'ovr' or One vs Rest classification. I tested the model on the remaining test set and obtained a mean accuracy of 55%, a precision of 0.54, recall of 0.50, and F1-Score of 0.5. The confusion matrix for the results are shown above. The results show that the model can guess a majority of the "AGREE" and "UNRELATED". This might be partly due to the uneven distribution of data in which both "AGREE" and "UNRELATED" appear more often. The model is less successful at predicting contradictions between sentences, and this could be because it is indeed a harder task than finding if two sentences are unrelated and often requires context or knowledge not contained within the features. For instance, if two sentences are the same but one contains the antonym of the verb from the first, essentially contradicting it, this model does not have access to the meaning of words and so cannot figure out the contradiction.

Flask API: The Logistic Regression is then ran on the entire dataset and a model is saved to a local pickle file (see build_model.py). The Flask API is contained with the app.py file and can be started from the command line by calling: *python app.py*

You can then query the app by sending a GET request to the instantiated flask server. For instance if the flask is started at http://127.0.0.1:5000/ (default), you can send a GET request passing each sentence with the attributes "txt" and "hyp" respectively (note it does not matter which is which, the notation was maintained simply for consistency).

For instance, you can also curl request from the terminal the running flask server:

```
curl -X GET http://127.0.0.1:5000/ -d txt="that movie was boring" -d hyp="hi that was great seeing you"
```

To obtain the prediction:

```
{
  "prediction": "UNRELATED",
  "confidence": 0.753
}
```

Reproducing: Running the code requires python 3.7+ and installing the dependencies listed in requirements.txt. This can be done through a virtual environment such as conda and calling *pip install -r requirements.txt*

Additionally, the code and results which generated the plots above can be found in the **main.pynb** notebook, which also contains examples on how to invoke the API straight from the notebook using the requests library.