NLS CW2 - Named Entity Recognition & Sentiment Analysis

- author: Eu-Bin KIM (10327741)

Task 1 - Named Entity Recognition

NER processing. The Inaugural corpus is ner-processed with NLTK NER in `nls_cw2/task_1/ner_with_nltk.py`, and with Stanford NER in `nls_cw2/task_1/ner_with_stan.py`. The results of this process is saved in `data/task_1/ner_with_nltk.ndjson` and `data/task_1/ner_with_stan.djson`.

NER-tagged with NLTK NER	NER_tagged with Stanford NER
In tendering this homage to the **(ORGANIZATION Great Author)** of ...	In tendering this homage to the **Great Author** of ...
How did we accomplish the **(ORGANIZATION Revolution)** ?	How did we accomplish the **Revolution** ?

Table 1: Two examples of NER-tagged results obtained with different methods.

Boundary detection discussion. NLTK NER performs worse than Stanford NER. This is because we observe that NLTK NER tends to produce more false-positives than Stanford NER does. For instance, Washington was referring to the jesus with "the Great Author"(**Table 1**, first row), yet NLTK NER has falsely recognized it as an organization. Likewise, NLTK NER has failed to recognize "Revolution" (second row) as an achievement. In contrast, Stanford NER has correctly recognized both entities as non-organizations.

exact matches	partial matches	total
389 (75%)	124 (25%)	513

Table 2: Exact & partial matches of ORGANIZATION entities between the results of NLTK NER and Stanford NER.

Agreement between tools. In the case they have agreements, there are more exact matches than there are partial matches. 75% of the matches were exactly the same and 25% of the matches were partially the same (**Table 2**). These measures are reproducible with the `nls_cw2/task_1/measure_agreements.py` script.

Task 2 - Sentiment Analysis

Part A - Bootstrapping Sentiment Lexicons

Basic patterns for finding new adjectives. We collect any adjectives if they are conjoined with any of the bootstrap lexicons. That is, if they are and-conjoined with a positive/negative boostrap lexicon, then they are labeled positive/negative (e.g. *brutal and corrupt*). If they are but-conjoined with a positive/negative boostrap lexicon, then they are labeled negative/positive (e.g. *brutal but impressive*). The collection process is implemented in `nls_cw2/task_2/part_a/collect_adjs_basic.py`, where we use NLTK's `RegExpParser` to collect any adjectives that match the `CONJOIN: {<JJ><CC><JJ>}` pattern.

Additional patterns implemented. We also collect any adjectives if they are comma-enumerated together with any of the bootstrap lexicons. For instance, provided that we know *weak* is negative, and given a sentence, *a weak, manipulative, pencil-thin story that is miraculously able to entertain anyway*, we can derive from the sentence that *manipulative* and *pencil-thin* are negative because they are comma-enumerated alongside *weak*. This works under the assumption that we tend to lay out things that are similar to each other when we comma-enumerate words. The collection process is implemented in the `nls_cw2/task_2/part_a/collect_adjs_more.py` script, where we use the `RegExpParser` to collect any adjectives that match the `ENUM: {<JJ><,><JJ><,><JJ><,><JJ>}...` pattern.

Top 5 positive adjectives	Top 5 negative adjectives
funny(0.18), entertaining(0.06), romantic(0.04), fresh(0.03), emotional(0.02)	dull(0.08), predictable(0.07), silly(0.04), unfunny(0.04), repetitive(0.03)

Table 3: The sentiment likelihoods of the newly collected adjectives.

Reconciliation. We assign polarities to each of the newly collected adjectives with their sentiment likelihood. That is, we compute `p(adj|sentiment)` for each adjective with respect to their sentiment class, the result of which is presented in **Table 3**. For instance, *funny*/*dull* are the most frequent in the positively/negatively labeled sentences, hence it is assigned with the highest polarity in each sentiment class. Assigning polarities is implemented in the `nls_cw2/task_2/part_a/assign_polarities.py` script.

Discussion and evaluation of the extended dictionary. Labeling new adjectives with the two patterns achieves over 80% accuracy (reproducible with the `nls_cw2/task_2/part_a/evaluate.py` script), but this is not without errors. As we are collecting adjectives from a set of movie reviews, the patterns are prone to the false-positives & false-negatives unique to the movie reviews.

For instance, a movie being *tough*, *risky* or *scary* is considered a compliment, but they are typically considered negative. Likewise, a *predictable* or *familiar* movies are considered boring, yet both words are typically considered positive. Mining typical opinions from a corpus is susceptible to false cases that are unique to the corpus.

Part B - Classifier Implementations

All the classifiers below are implemented in the `nls_cw2/task_2/part_b/evaluate_models.py` script.

Baseline classifier. `BaselineClassifier` (at lines 90-117) classifies the sentiment of a sentence by simply polling the occurrences of the positive and negative words. That is, if there are more positive occurrences than there are negative ones, the sentence is classified as `pos`. Otherwise, it is classified as `neg` unless the occurrences are equal. If the occurrences are equal, then the sentence is classified as `neu`.

BoW classifier. As for the machine learning approach, we train a `RandomForestClassifier` model by feeding Bag-of-Words vectors of each sentence as the features, and the sentiment class (i.e. `pos`, `neg`) as the label. We build BoW representations of each sentence in `preproc_dataset_bow()` function (at lines 43-60). The BoW features are then used to train a `RandomForestClassifier` in the function `evaluate_bow` (at lines 157-168), using a scikit-learn library.

Word2Vec classifier. we also train another `RandomForestClassifier` model by feeding the average of Word2Vec vectors as the features. We use a pre-trained Word2Vec model to get 200-dimensional word vectors of each word in a sentence, and average all of them to get the vector representation of each sentence. This is implemented in the `preproc_dataset_w2v()` function at lines 63-87. The W2V features are then used to train another `RandomForestClassifier` in the `evaluate_w2v()` function at lines 171-183.

Additional features implemented. In addition to the vector representation of each sentence, we use the existence of negations in each sentence as another feature. That is, if the sentence contains either "n't", "not" or "no", we flag the sentence as 1 and 0 otherwise. For instance, the sentence *effective but too-tepid biopic* is flagged 0 as it contains none of the three negations. In contrast, *emerges as something rare , ... that it doesn't feel like on* is flagged 1 as it contains "n't" in "does**n't**". The feature extraction process is implemented in the `build_dataset.py` script, the result of which can be viewed in `data/task_2/part_b/dataset.tsv`.

baseline	vanilla bow	bow (negation added)	vanilla w2v	w2v (negation added)
0.5139	0.6459	0.6475	0.6937	0.7009

Table 5: The accuracy of the baseline and the mean-accuracies of the machine learning models on Corpus 2.

Accuracy and cross-validation We evaluate the baseline model by testing its accuracy on Corpus 2, and evaluate the two machine learning models by cross-validating them on Corpus 2. We evaluate the baseline model with the `evaluate_base()` function at lines 120-146. As for the BoW and W2V classifiers, we test them with K-fold cross validation in order to get a more reliable accuracy. The cross validation is done with 4 folds, which is implemented in both `evaluate_bow()` and `evaluate_w2v()` functions. We also cross-validate the models with the additional negation features. The results of all the evaluations are presented in **Table 5** above.

Comparisons to baseline. Both BoW and W2V models perform far better than the baseline model. As can be seen in **Table 5**, The vanilla BoW model outperforms the baseline by 10% in accuracy, and the vanilla W2V model outperforms the baseline by nearly 20%.

Overall discussions of results. Although the baseline model performs the worst, it should be noted that a rule-based approach as such would have been more desirable than the others had we had a shortage of data. While the BoW model outperforms the baseline, it is outperformed by the W2V model. This is because a BoW vector (16674) is far sparser and a W2V vector (200), and thus the BoW model is more prone to the Curse of Dimensionality than the W2V model. Training the models with the negation feature included only so much improves upon the vanilla cases. This might be because, just because a sentence includes "n't", "not" or "no" does not always mean that the sentence is negative. For instance, the sentence *perhaps **no** picture ever made has more literally showed that the road to hell is paved with good intentions.*, does contain "no" in the beginning, but it is used to emphasize how novel the product is; the sentence bears a positive sentiment although it contains a negation.

Other ideas. Breaking free of the Bag-of-Words assumption by leveraging sentence-level features could improve upon the W2V model. Sentence-level features largely affect the sentiment of a sentence, and yet these are simply unobservable under a Bag-of-Words assumption. For instance, sarcasm (e.g. *What a great movie, huh.*) or figures of speech (e.g. *The director has an emotional maturity of a potato.*) effectively determines the sentiment of a sentence, but they are completely lost when sentences are assumed to be simply a bag of words. Provided that there are sufficient amount of data, we could try training a sequential model (e.g. LSTM, Transformer) to model the sentence-level features as such, which may classify sentiments better than the W2V model.