CS839 Project Stage 1 Report

Mohammed Danish Shaikh, Somya Arora, Swati Mishra
{mshaikh4, somya.arora, smishra33}@wisc.edu
University of Wisconsin, Madison

1 Introduction

The goal of this project stage is to perform information extraction (IE) from natural text documents, using a supervised learning approach.

2 Methodology

The extraction process can be broken down into the following steps:

1. Collect 300 text documents containing entities that can be extracted (eg. Person, Place, etc). These documents must contain well-formed sentences.
2. Decide an entity type to be extracted from these documents.
3. Go through all the documents and markup the entity names to be extracted.
4. Split the set of documents into two sets - I containing 200 documents and J containing 100 documents.
5. Set I will be used for development and debugging of the extractor (dev set). Set J will be used for reporting the accuracy of the extractor (test set).
6. Perform cross validation on set I to select the best classifier M. Consider at least linear regression, logistic regression, support vector machines, decision trees and random forests.
7. Train the classifier M using set I. Test the accuracy of the resulting model using set J. You should aim at least for a precision of 90% and recall of 60%.
8. Debug and improve the classifier M if required accuracy isn't being met. Cross validation has to be performed after each debugging step.
9. Obtain a final classifier M giving the best possible precision and recall.
10. Apply post processing rules in order to further improve the accuracy of the classifier.
11. Apply the final classifier M and post processing rules on the set J to obtain the precision and recall. Note that the set J shouldn't be touched until this step.

3 Environment Setup & Implementation

3.1 Corpus

We used the BBC full text data set. 100 documents each from entertainment, sports and politics categories were obtained and labeled.

3.2 Package

We used the scikit-learn [3] package in python to implement linear regression, logistic regression, decision tree, random forest and support vector machine classifiers.

3.3 Candidates Generation

Algorithm 1: Candidates Generation

```
D = set of documents
C = []                                                    // List of candidates
L = []                                                    // List of labels
T = 4                        // The threshold maximum number of words in a candidate
for each d in D do
    words = list of words in D
    for each word in words do
        for i in 1 to T do
            candidate = candidate of length i beginning at word   // word along with i -1 words ahead
            if candidate is invalid then
                break                              // Prune early, this is the blocking step
            else if candidate is labeled with entity then
                C.append(candidate)
                L.append(1)
            else
                C.append(candidate)
                L.append(0)
```

3.4 Features

We generated a *321-dimensional* feature vector for each candidate using the following features:

3.4.1 Synset Types

We used the nltk wordnet corpus [2] to create a feature vector encoding the synset types of a given phrase. Noun, Verb, Adjective, Adjective Satellite, Adverb synset types were encoded for each of the following phrases for each candidate:

1. Two word prefix appearing just before the candidate.
2. Two word suffix appearing just after the candidate.
3. The candidate itself.

For a given phrase above, the feature vector was generated by using algorithm 2. This way, we generated *5 (synset types) * 3 (number of phrases) = 15 features.*

Algorithm 2: synset_type(phrase)

```
feature = [0, 0, 0, 0]              // Initialize synset types feature vector to zero
for each word in phrase do
    synset_types = get synset types from wordnet for word
    feature = feature OR synset_types
return feature
```

3.4.2 Valid Phrase

We used the nltk wordnet corpus [2] to create a feature vector encoding whether a given phrase is valid or not, wherein a phrase being valid implies that at least one word constituting the phrase is present in the wordnet corpus (algorithm 3). The following phrases were considered for each candidate:

1. Two word prefix appearing just before the candidate.
2. Two word suffix appearing just after the candidate.
3. The candidate itself.

This way, we generated *1 (valid phrase or not) * 3 (number of phrases) = 3 features.*

Algorithm 3: valid_phrase(phrase)

```
1 is_valid = False                                    // Initialize valid to False
2 for each word in phrase do
3     word_validity = check if word is present in wordnet corpus synset
4     is_valid = is_valid OR word_validity
5 return is_valid
```

3.4.3 Missing Prefix or Suffix

We created features encoding the existence of prefix or suffix for a given candidate (algorithm 4). For example, a candidate at the start of a sentence won't have a prefix. This way, we generated *2 features.*

Algorithm 4: has_prefix_suffix(candidate)

```
1 has_suffix = 0                          // Initialize feature for empty suffix
2 has_prefix = 0                          // Initialize feature for empty prefix
3 if candidate has suffix then
4     has_suffix = 1
5 if candidate has prefix then
6     has_prefix = 1
7 return has_suffix, has_prefix
```

3.4.4 Has Apostrophe

We created a feature encoding the existence of apostrophe in a given candidate (algorithm 5). The candidates ending with *'s or s'* were ignored in this feature.

Algorithm 5: has_apostrophe(phrase)

```
1 contains_apostrophe = False
2 if phrase contains ' and doesn't end with 's or s' then
3     contains_apostrophe = True
4 return contains_apostrophe
```

3.4.5 Word2Vec

We used Google's word2vec [1] to encode a 300-dimensional feature vector. For each candidate, we generated the feature vector as shown in algorithm 6

Entity	Person
Total number of mentions marked up	6888
Number of documents in set I (Dev set)	200
Number of mentions in I	4552
Number of documents in set J (Test set)	100
Number of mentioned in J	2336
Classifier (M) settled on after cross-validation	SVM with rbf kernel
Precision, Recall and F1 of M on I	P=90%; R=96%; F1=0.93
Classifier (X) settled on before post-processing	SVM with rbf kernel
Precision, Recall and F1 of X on J	P=90%; R=91%; F1=0.91
Post-processing rules applied or not	No
Precision, Recall and F1 of Y on J	P=90%; R=91%; F1=0.91

Table 1: Reporting Metrics

Algorithm 6: word2vec(candidate)

```
1 w2v = [0] * 300    // 300 dimensional word2vec feature vector for candidate initialized
  with zero
2 num_words = 0                                        // Number of words in the candidate
3 for each word in candidate do
4     current_w2v = word2vec(word)
5     w2v = w2v + current_w2v
6     num_words = num_words + 1
7 w2v = w2v / num_words
8 return w2v
```

4 Experimental Data & Discussion

4.1 Metrics

The experimental metrics have been summarized in Table 1. Notations used correspond to those in the assignment page.

4.2 Post-Processing rules

We debugged the wrong predictions (False Negatives and False Positives) of our model to see if we can add any post-processing rules. However, we couldn't find any trend that we could formalize as a rule. We could have just post-processed some of those predictions specifically to increase our accuracy, but we didn't do this since we were already reaching the required precision and recall without post-processing rules.

4.3 Debugging

Even though debugging the wrong predictions wasn't of any help to us in adding post-processing rules, we did find some grammatical errors in the text or that in labelling that were leading to wrong predictions (For example, *The consequences were suffered by William.* was labeled as *The consequences were suffered by ¡person¿William.¡/person¿* and later corrected to *The consequences were suffered by ¡person¿William¡/person¿.* after analyzing wrong predictions). This, in turn, improved our recall slightly.

4.4 More Features

We believe that adding the following features could have further improved the accuracy of our classifier:

1. Length of the candidate.

2. Whether the prefix of a candidate is a title like *Sir, Mr, Lord, Ms, etc* or not.

3. Whether the candidate ends with a *'s or s'* or not.

5 Conclusion

This project stage was a great learning experience for us. We learned how to build an end-to-end pipeline for extracting information from natural text documents. This required us to learn to use and evaluate different types of classifiers against each other, engineer features relevant to the problem at hand, debug the outcome and improvise the model based on the findings to finally come up with a model satisfying the required accuracy constraints. The labeled documents, code, evaluation and models can be found in our GitHub repository. Additionally, it can also be found at this website.

6 Acknowledgements

We would like to thank Professor AnHai for giving us the opportunity to work on this project stage, which served as a way for us to learn how an end-to-end pipeline is implemented practically. We would also like to thank our classmates for asking interesting questions and helping us broaden our thinking.

References

[1] Google word2vec Contributors. *word2vec*. `https://code.google.com/archive/p/word2vec/`. Accessed 2019-03-08.

[2] NLTK Wordnet Contributors. *wordnet*. `http://www.nltk.org/howto/wordnet.html`. Accessed 2019-03-08.

[3] Scikit Learn Contributors. *scikit-learn*. `https://scikit-learn.org/stable/`. Accessed 2019-03-08.