Information Extraction on Medical Research Articles

Jeffrey H. Hu and Jose I. Velarde

December 14, 2016

Abstract

Researchers at MGH are developing statistical models of patient risk for certain types of cancer by aggregating findings of numerous studies. To do this, determining the reliability of an article is a crucial, but time consuming step. In this paper, we look to automating the process of extracting one highly informative heuristic: the size of the population. We develop a system that uses a convolutional neural network to identify sentences that talk about population and a feed forward neural network to extract exact population size.

1 Introduction

This project was done in collaboration with researchers at Massachusetts General Hospital looking to build models for calculating patient risk of cancer based on genetic mutations, race, previous history of cancer, etc. Their approach was to comb through cancer research and find articles linking specific traits or genetic mutations to occurrences of certain types of cancers. They then synthesized the results, resolved conflicting findings, and produced conclusions. Reviewing the literature was a very time consuming process. We collaborated with them to create a natural language processing system that automatically extracts metrics and speeds up the process of evaluating a paper's reliability.

We focused on extracting population size because it is the feature that correlates most strongly with an article's reliability. In this paper, we present a simple n-gram classification method and attempt to beat it with a two step CNN model. By first identifying sentences that talk about population and then classifying the individual words as being the population size or not, we hoped to achieve greater accuracy.

2 Related Work

While there has been much work on extracting PICO (Population/Problem, Intervention, Comparator, and Outcome) sentences from articles or full text articles, little work has been done on extracting data elements from full text articles. Lin et al. (2010) used a conditional random field to extract number of patients, geographical area, and information from full text articles. The authors achieved a precision of 43% and a recall of 10.7% for identifying the number of patients. They evaluated their system using threefold cross validation on 93 full text articles. De Brujin et al. used support vector machines to identify sentences describing certain information, such as sample size or eligibility criteria. This was then combined with manually crafted extraction rules to extract the relevant information. This architecture achieved a precision of 62% and a recall of 80% for identifying sample size in articles. This was the system with the best performance we found in the literature. Our work differs from previous work in the following ways: 1. we used word embeddings to incorporate semantic information about words into the information extraction. 2. we used neural networks for classification, which have outperformed nearly all other classifiers in most NLP tasks. We were able to surpass the performance of Lin et al., but fell short of the system described in De Brujin et al.

3 Method

3.1 Initial Approach

We approached the problem of extracting an article's population size as a classification problem. We would go through the entire article and classified each word as being the total population size of the article or not. Our dataset was 132 full text articles manually annotated with the total size of the population. They are all freely available on PubMed.

3.1.1 Hand-crafted Features

For a baseline, we implemented a simple model using an SVM Classifier and hand crafted features. The features for the baseline were the the current word, its two neighbors and all of those words' part of speech tags. We refer to this as a context of size 1 (one word before and one after). The features were all one hot encoded. This baseline achieved an F1 score of 31% on classifying words as being the population. We also tested a the same model with a Logistic Regression Classifier and achieved slightly better results (F1 score of 33%). More detailed information about the performance is provided in Table 1.

	Precision	Recall	F1
LR c=1	0.40	0.29	**0.33**
LR c=2	0.65	0.39	**0.49**
SVM c=1	0.33	0.29	0.31
SVM c=2	0.52	0.39	0.45

Table 1: Performance with hand crafted features, best results are bold faced.

3.1.2 Word Embedding

We were able to improve on our baseline performance by removing stop words and increasing the context size to 2. The largest improvement however, came from the combination of using a Word2Vec representation of words that had been trained on a medical corpus, and removing stop words from the texts. Word2Vec is a word embedding that is trained using a skip gram model. Given a word, a skip gram model tries to predict the word's context. In learning how to predict a word's context, the model learns a semantic representation of that word. This gives us a mapping between words and vectors. With this mapping, we let each word's feature vector be concatenation of the vectors representations of its neighbors.

	Hand crafted features	Word Embedding
LR c=1	0.40/0.29	0.47/0.66
LR c=2	**0.65/0.39**	0.44/0.60
SVM c=1	0.33/0.29	**0.62/0.60**
SVM c=2	0.52/0.39	0.52/0.58

Table 2: Precision/Recall scores with no stop words

3.1.3 Feedforward NN

We then try to classify with a feedforward neural network with one hidden layer. This only made possible by the word embedding representation because one hot-encoded hand-crafted features are too high dimensional to be trained by a neural network. Start and end tokens were represented with the zero vector. Our best performing network architecture for this task was one with one hidden layer of 20 units. We used rectified linear activation, a two node softmax output layer, and cross entropy loss. With this architecture, we achieved an F1 score of 65%. The different models' precision and recall can be found in Table 3.

	Hand crafted	Word Embedding	Word Embedding + no stop
LR c=1	0.40/0.29	0.20/0.29	0.47/0.66
LR c=2	**0.65/0.39**	0.36/0.57	0.44/0.60
SVM c=1	0.33/0.29	0.26/0.36	0.62/0.60
SVM c=2	0.52/0.39	**0.48/0.57**	0.52/0.58
NN c=1	N/A	0.83/0.18	0.82/0.54
NN c=2	N/A	0.62/0.36	**0.88/0.54**

Table 3: Performance with word embedding and no stop words, highest value in each column is boldfaced.

3.2 Two Step Approach

Research Papers are full of tables, annotations, and formatting. The process of converting the PDFs to text left the articles full of noise. To improve our results, we attempted to reduce the search space of our word classifiers by adding a sentence classifier to identify sentences that talk about population. We believed the task of classifying whether a sentence was talking about population or even is properly structured would be easier. The full system would take in a full text article, find the sentences that talk about population, and feed those into another classifier that would classify the individual words as being the population of the article. The classifiers would be trained separately.

3.2.1 CNN for Sentence Classification

Convolutional Neural Networks are well suited for sentence classification because they allows us to represent sentences simply as concatenations of word vectors. Using feed forward neural networks on this representation is extremely impractical because the huge number of input nodes. CNNs use filters so the number of parameters to train is independent of the sentence length. CNN filters of size n also effectively operate as n gram models. Many filters of different sizes can train to identify certain key phrases that can occur anywhere in the sentence. Max pooling gives their activations locational invariance.

3.2.2 Model Architecture and Hyperparameters

The CNN takes as an input an n x k dimensional matrix, where n is the number of words in a sentence and k is the dimension of the word embedding we used. Convolutional filters with different window sizes are applied to the input. The output of the filters are then max pooled and concatenate into a single vector. This vector is then fully connected to two node softmax output layer. See Figure 1.

The network architecture that worked best was to use ReLu activation, 16 filters with window sizes of 3 and 4 (for a total of 32 filters), trained with a dropout rate of 0.5 in the fully connected layer.

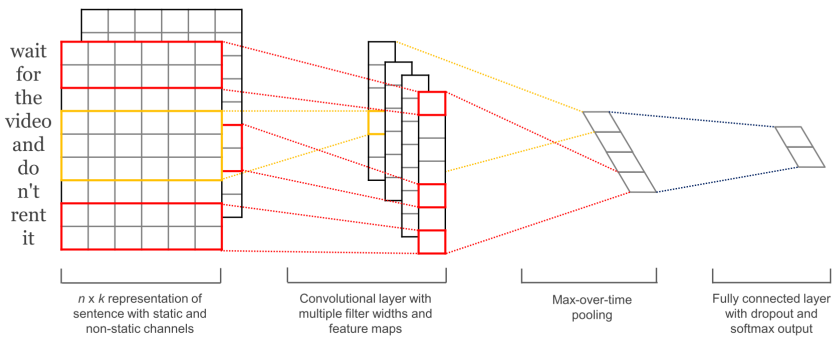

Figure 1: Convolutional Neural Network Architecture

3.2.3 Performance Tradeoffs

In training the CNN, we decided to prioritize high recall over precision, because we wanted to be sure sentences containing the population size would be passed on to the word classifier. The CNN we ended up using in our pipeline had a recall of 92% on the test set, but a precision of only 6%. Nevertheless, it achieved over 85% recall on classifying sentences that were not about population. This means it was effective in narrowing down the search space for the word classifier by almost a factor of 7.

Precision	Recall	F1
0.70	0.17	0.29

Table 4: Performance of two stage classifier

4 Results and Discussion

The scores of our two stage classifier are shown in Table 4. Unfortunately, our two stage architecture was unable to surpass our best performing word classifier. The recall was simply not high enough. Neural networks traditionally need a lot of data to train on and we believe there many not have been enough positive samples in the data. There were too few sentences about population for the neural network to learn the features that make a sentence talk about population. Our best performing model for extraction was the feed forward neural network with a context size of 2 (four neighboring words). Details of its performance and a comparison with the state of the art can be found in Table 5.

	Precision	Recall	F1
Neural Network	**0.88**	0.54	0.67
De Brujin et al.	0.62	**0.80**	**0.70**
Lin et al.	0.43	.11	.18

Table 5: Performance of feedforward neural network

4.1 Data and Classifiers

Our data set was very sparse in both cases. Sentences that talked about the population made up less than 1% of all sentences in the articles in our data set. Similarly, the population size is usually only mentioned once or twice in each article. As a result, less than 1% of all words in our data set constituted positive samples. At first, the SVM and the logistic regression would always classify words as not population. We decided to modify the hyperparameters of both classifiers to make them more aggressive. For the SVM, we increased the value of C to 10,000,000, effectively making it into a hard SVM. This means that it would not allow the resulting hyperplane to misclassify training samples. For the logistic regression, we removed the regularization term, allowing it to fit as closely to the data as possible. Both of these modifications allowed the models to make more accurate predictions.

4.2 Hand Crafted Features

When using hand crafted features, the logistic regression classifier consistently outperformed the SVM by two percentage points on the F1 metric. With the hand crafted features, the SVM and the logistic regression achieved identical recall scores, but the logistic regression had a higher precision. We believe this is because the SVM only considers the training examples that are close to the decision plane. This works well when the data is linearly separable, but we believe that it isn't in our case. The logistic regression, on the other hand, learns from all of the training samples. This allows it to better discriminate the population based on the features.

4.3 Word Embedding

As previously mentioned, using the word embedding to create feature vectors greatly increased the performance of some of our classifiers, specifically the SVM and the feedforward neural network. The word embedding had the same general effect on both the SVM and the logistic regression: it increased the recall but hurt the precision. However, the drop in precision was worse for the logistic regression than for the SVM. As a result, it was a net gain for the SVM, but a net loss for the logistic regression. Both models achieved better results with smaller context sizes.

4.3.1 Word Embedding and Stop Words

The models with word embedding could be very sensitive to the presence of stop words. Their sensitivity to the presence of stop words depended on the context size. With a small context size, the presence of stop words caused a considerable decrease in performance. However, with a large context size, the difference in performance was much smaller. We believe that this is because the word embedding for stop words probably does not contain much information. As a result, words that have stop words in their context become much harder to classify.

4.4 Stop Words

Removing stop words provided a large boost in performance for all the classifiers we tested. In all cases it increased both the precision and the recall. The neural network was only able to outperform the other classifiers when the stop words had been removed, otherwise, it had an average performance. We believe that this is because the neural network has to learn many parameters, which makes it prone to overfitting to noise. Removing stop words removes noise that the network could have overfit to.

5 Conclusions and Future Work

We have presented the different approaches we evaluated to extracting the population size of an article. We worked with a small data set, and we believe that the first step in improving the performance of the system would be to obtain or create a larger data set. The common trend amongst all the classifiers we trained was a low recall. We believe more training data would allow the models to better learn to identify different phrases that contain or describe the population size.

Additionally, insights could be obtained by performing a more in depth analysis of the data set to find trends that could be used as heuristics to improve system performance, such as in what section of articles is the population usually in. Another approach we would like to try in the future is using a recurrent neural network to classify the words.

6 Team Member Contributions

Jeffrey wrote the code that parsed the articles into sentences and preprocessed the text from the articles. Jeffrey also wrote the code that created training matrices and implemented the convolutional neural network for classifying sentences in TensorFlow.

Jose wrote the code that converted PDF articles into text files. He also implemented the all different word classifiers mentioned in this paper. He also wrote the code for hand crafting the feature vectors for words and for uploading and using the word embedding. The writing of this paper was split evenly.

References

[1] Sampo Pyysalo, Filip Ginter, Hans Moen, Tapio Salakoski and Sophia Ananiadou. *Distributional Semantics Resources for BioMedical Text Processing* LBM 2013.