IDENTIFYING ENTITIES AND THEIR RELATIONSHIPS FROM MEDICAL ARTICLES

SUBMITTED BY: VIKAS GARG (EECS)
TEAM: FRANCK DERNONCOURT AND VIKAS GARG
6.864 FALL 2015

ABSTRACT. Our objective is to automatically extract the influence of various food items on different diseases from a large corpus of articles. Specifically, the problem setting is as follows. We assume a collection of medical documents corresponding to the effect of particular food items (e.g. alcohol, dairy, meat, etc.) on different diseases (e.g. breast cancer, diabetes, etc.) in a specific population group (based for instance, on age or ethnicity or having some previous medical condition). The effect could, for instance, be positive (accelerating the onset), negative, or unknown. Our goal is to extract sentences from these articles that can identify the different entities, i.e. the specific population group, the food item, the medical disease or ailment, and the effect.

We set up our problem in a neural setting and propose two approaches. The main idea is the following. Each sentence can be abstracted in terms of the words describing food, disease, population, effect, and the rest of the sentence. We learn a 5-block hidden representation of each sentence, where each block consists of several neural units. In the first approach, we associate a separate entity (food, disease, population, effect, other) with each block - we sample some training sentences, perturb them slightly by modifying some entity words, and do a block-constrained training by restricting the weight updates to only the units in the blocks that correspond to the modified entities. This hidden block layer is sandwiched between the input and output layers of an auto-encoder, and a subsequent training procedure tries to minimize the discrepancy between the input and the output. In the second approach, we do an end-to-end training using an architecture that contains a GRU encoder, the hidden block layer, and a GRU decoder, and measure the discrepancy between the input and output sentences directly. We form our corpus using the abstracts from the PubMed database. The work is ongoing, and we present some initial results. All the data and code used in the project can be accessed at:
`https://github.com/Franck-Dernoncourt/ieunsup`

1. INTRODUCTION

This project is motivated by the need to automate the generation of meta-reviews about the influence of different food items on diseases from a large corpus of articles. Currently, such meta-reviews have to be generated manually by human experts.[1] It is hard for humans to sift through hundreds of articles before they can summarize their findings in terms of meta-reviews. Moreover, often the veracity of the sources on which these reviews are based can be questionable. Therefore, the need to have a framework to automate the entire process cannot be overemphasized.

The setting is as follows. We have a corpus of medical documents corresponding to the effect of some food items (alcohol, dairy, meat etc.) on breast cancer in a certain population group (based for instance, on age or ethnicity or having some previous medical condition). The effect could for instance, be positive (accelerating the onset), negative, or unknown. We would like to extract sentences from these articles that can identify the different entities, i.e. the specific population group, the medical disease or ailment, and the effect. Having a meta-review that summarizes the effect of different food items on a subset of diseases would be ideal, however, a more realistic immediate goal is to have the different entities extracted along with their effects in a table (without caring about generating proper grammatical sentences for example). Moreover, we would like to use only a limited amount of supervision. For a concrete example of the problem setting, consider the following three sentences.

[1] see, for example, http://foodforbreastcancer.com

Food	Disease/Mental Condition	Population	Effect
Alcohol	Breast Cancer	Adult Women	Positive
Salt	Blood Pressure	-	Positive
Dairy	Rickets	Children	Inconclusive

(1) Alcohol consumption by adult women is consistently associated with the risk of breast cancer.
(2) A high intake of salt has been linked to high blood pressure.
(3) The effect of dairy on the onset of rickets in children remains inconclusive.

We would like a learning framework to automatically generate the adjacent table. Note that not all the entities may be present in a sentence, and these missing entities are omitted from the corresponding row in the table. Nonetheless, we would like to extract as much information from the data as possible.

In the following sections, we describe our methodology in detail. An initial milestone was to determine the feasibility of the project. We describe in detail our experiments towards that goal in Section 2. We then introduce a neural architecture, and outline our approach to automate the process of extracting information from the medical documents with partial supervision in Section 3. We then present some results obtained using this architecture in Section 4. At the time of writing this report, we are in the process of implementing a second architecture. We describe this architecture in Section 5.

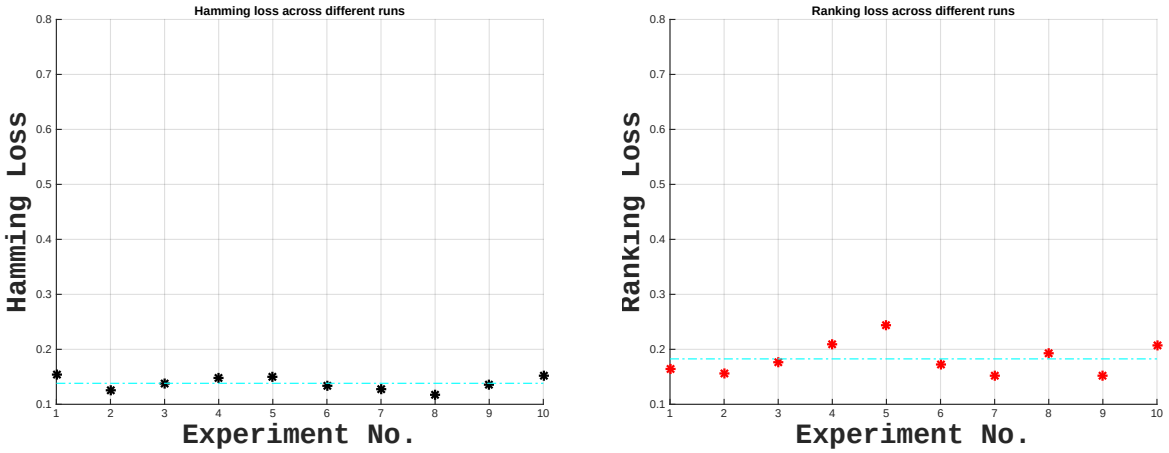
2. Feasibility Check

It is not clear whether it is possible to automate the process of generating the table. In order to check the feasibility of the project objectives, we investigated whether the population specific words could be identified with some supervision, so that we could proceed to a less supervised setting if the results were satisfactory. The basic premise for our experiments was that if we could not achieve good performance with a supervised model involving a single entity, it would be very hard to obtain meaningful results with less supervision in the original setting where the goal is to identify all the entities simultaneously.

We selected 179 sentences from the different food items for breast cancer using the PubMed database, and manually annotated the words that identified the population. Many sentences had more than one word for the population category - the average population words per sentence turned out to be 2.5. A few sentences (< 5) were taken to have no food item in them to model noise in the downloaded abstracts. After removing stop-words, each word in the vocabulary was represented by a 3-dimensional skip-gram vector using word2vec with a context window size of 4 for 200 iterations with 100 negative samples. The maximum length of a sentence after removing stop-words turned out to be 22, and the average length 12. Therefore, we represented each sentence as a 66-dimensional vector formed by concatenating the vector representations of the words in the sentence, in the order that they appeared. We padded sentences that had fewer than 22 words with zeros at the end, so that we could use a fixed length vector to represent any sentence.

We formulated the problem as a multi-label classification problem - given an input vector $x \in \mathbb{R}^{66}$, the goal was to identify all the positions, ranging from 1 to 22, where a population identifier occurred. This allowed us to study the problem in a neural network setting introduced by [1]. We then conducted 10 independent runs of the following experiment. In each run, we divided these 179 sentences randomly into a training set of size 150 and a test set of remaining sentences. We trained a one-layer hidden network with 12 hidden units, a learning rate 0.05 and 40 epochs. These settings closely follow a typical configuration that performed well in the experiments conducted by [1] - hidden units 20% of the input dimensionality and a learning rate of 0.05. We then trained a separate net in each iteration and observed the performance on the test set for that iteration. We report the average performance and the standard deviation across the runs.

We measured the performance in terms of the average Hamming loss, ranking loss, average precision (AP), one-error, and coverage [1]. Essentially, for each position in a sentence we need to make a binary decision whether it contains a population word or not - that is whether it is included in a set or not. The Hamming loss measures, on average, the error in predicting an incorrect position or missing a correct position - it degenerates to the zero-one classification loss for the standard single label classification problems.



(A) Hamming Loss (B) Ranking Loss

FIGURE 1. Hamming Loss and Ranking Loss. Both take values between 0 and 1 - the lower the better.

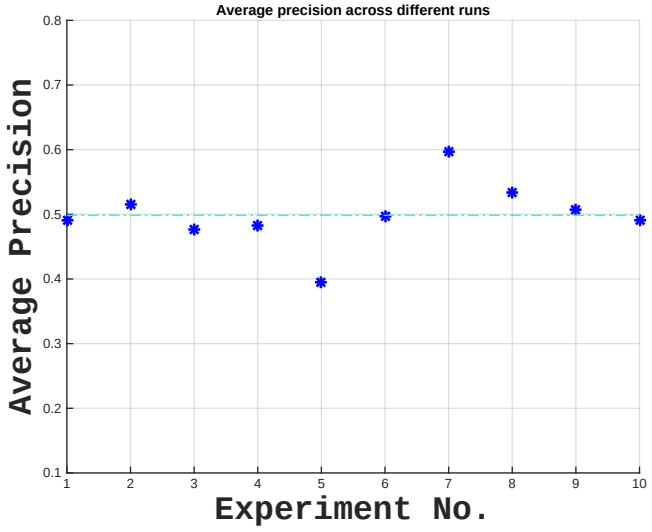


FIGURE 2. Average Precision. Takes a value between 0 and 1 - the higher the better.

The pairwise ranking loss, on the other hand, measures the fraction of label pairs, each containing one correct position and one incorrect position, that are flipped for a sentence in terms of the score assigned by a

real valued function (that is, we incur a penalty for each pair where an incorrect position gets a higher value than the correct position). AP measures the average number of labels that are scored higher than a particular correct position being considered, which actually belong to the set of correct positions - unlike Hamming loss and ranking loss, a higher value of AP suggests a better performance. Coverage is similar to the notion of precision at the level of perfect recall - it conveys the average number of steps to go down the list of labels in order to cover all the proper labels of a sentence. Finally, one-error counts the occurrences the top-ranked position is not in the set of correct positions of the instance. In summary, we want AP to be high and all the other quantities to be low.

The attached plots show the Hamming loss (Fig. 1(A)), ranking loss (Fig. 1(B)), and AP (Fig. 2) across independent experiments. The line in cyan indicates the average performance. The overall stats were:

(1) **Hamming Loss** = 0.14 +/- 0.01
(2) **Ranking Loss** = 0.18 +/- 0.03
(3) **OneError** = 0.66 +/- 0.08
(4) **Coverage** = 6.11 +/- 0.84
(5) **Average Precision** = 0.50 +/- 0.05

We achieved a low value of ranking loss and hamming loss, and a good AP score. Since the average length of a sentence comprised 12 words, we effectively need to traverse half the sentence to identify all the population words. Since the correct positions per sentence averaged 2.5 in our experimental setup, a Hamming loss of 0.14 is quite low.

In order to look for further evidence, we trained a Maximum Entropy model on 80% train (= 143 sentences) and tested on the remaining 36 sentences. Based on frequency of the words in the training set, we used 8 indicator word features corresponding to whether the word contained

(1) gender root (wome, mice, male etc.)
(2) type (pre, post, young, adult, etc.)
(3) orientation (hetero, lesbian etc.)
(4) nationality (american, african etc.),
(5) ethnicity (black, white, hispanic etc.)
(6) food habits (veg, omni etc.)
(7) general indicators (-, non)
(8) more than one capital letter

Additionally, we used a 3 feature bigram hot-vector to model whether the previous word is relevant, irrelevant, or the start of the sentence. Using these simple 11 features, we obtained the following results averaged over 20 independent runs:

- **Precision**: 0.79 +/- 0.04
- **Recall**: 0.71 +/- 0.06
- **F1-Score**: 0.75 +/- 0.05

Since these results were promising, we decided to consider the problem of simultaneous extraction of the different entities in a sentence.

3. Multiple Entity Extraction

Our approach for extracting all the entities is based on the intuition that a sentence can be abstracted in terms of a hidden representation of 5 blocks or constituents - a food block, a population block, a disease block, an effect block and the rest of the sentence - that can essentially recover the entire sentence. A neural auto-encoder based on this intuition is shown in Fig. 3. The architecture consists of the following two main components:

- **Sentence2Vec**: The Sentence2Vec module takes as input a sentence of variable length and produces a fixed vector representation for the sentence. The sentence vectors are initialized using the gensim model.

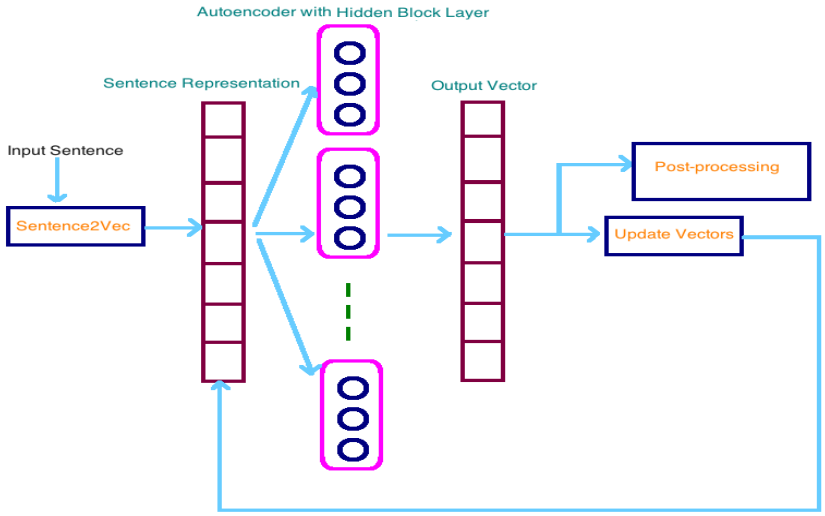


FIGURE 3. Architecture 1

- **Hidden Block Layer**: The hidden layer consists of 5 blocks, each of which has several neural units. Each block identifies with either an entity, or relationship or rest of the sentence. Every block produces a feature vector, and we concatenate the feature vectors of the different blocks to produce an aggregate feature vector.

We train the model by minimizing the discrepancy between each sentence vector and the corresponding aggregate vector. However, unlike the standard training procedures, we follow a block-constrained training for updating the weights of the parameters. Specifically, we assume we have a subset of training sentences which we call the seed sentences, for each of which we know the words that identify the entities and the effect in the sentence. For each seed sentence, we perturb the sentence slightly by removing one of the entities and allow only the weight parameters associated with the block corresponding to the omitted entity and the block corresponding to the rest of the sentence to change, while keeping the other parameters unchanged. For example, if we remove a food item from a seed sentence, then we only update the parameters associated with the food block. This way we are able to bias the representation from each hidden block towards the different entities locally. Also, this procedure results in data augmentation since we obtain new sentences by perturbing the entities in the sentences.

To infer the entities from a test sentence, we first produce the hidden layer representation S_i of the sentence i. Then, we follow the following procedure to infer the category of each word j in sentence i. We delete j and obtain the hidden representation $S_{i/j}$ of the resulting sentence. We then measure the discrepancy between the vectors S_i and $S_{i/j}$, and choose the category associated with the block that changes the most. Intuitively, as a result of the way we biased the entities towards their corresponding blocks, if we remove a food word then most of the effect on the representation should be restricted to the block that corresponds to food. We follow this procedure to infer the category for each word in a sentence.

4. EXPERIMENTAL RESULTS

We now present results of our experiments with Architecture 1. We used a 200-dimensional Sentence2Vec representation for each sentence, and let each of the 5-blocks have 10 units each, resulting in a 50-dimensional hidden block representation. We assumed each sentence in the training set (comprising 8000 original sentences) to be a seed sentence, and therefore obtained an augmented dataset by perturbing the different entities in each sentence. Our test set comprised of about 5000 sentences. Fig. 4 shows the mean squared error on the training and test sets obtained as a function of the number of epochs, when standard training was used

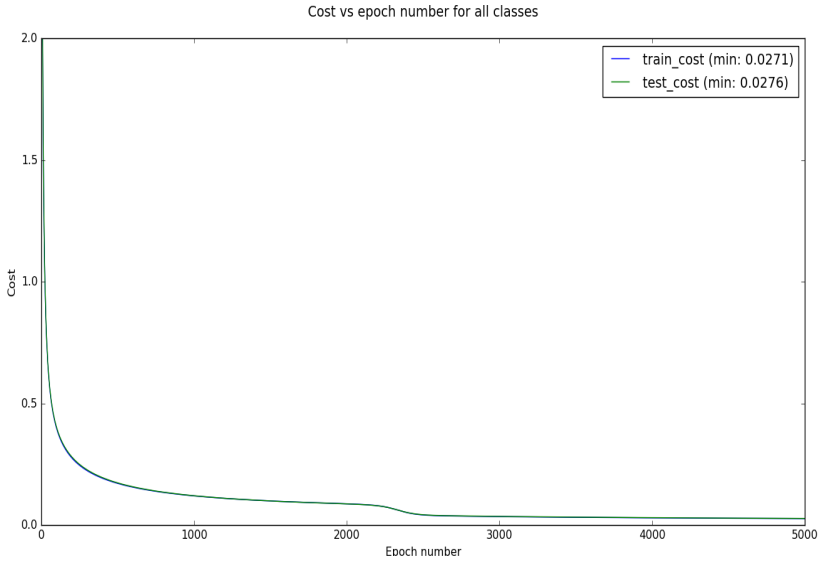


FIGURE 4. Error vs. epoch curve obtained with the standard training

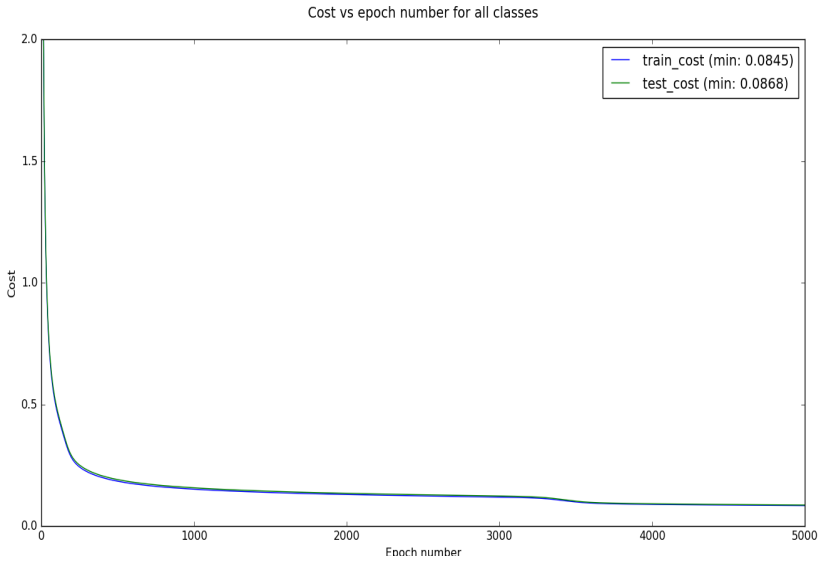


FIGURE 5. Error vs. epoch curve obtained with block constrained training

where we update all the weights during the training. On the other hand, Fig. 5 shows the training error when block constrained training was used. Recall that in the block constrained training, only those weights were updated that were associated with the block whose corresponding entity was removed from the original sentence, and the block corresponding to the rest of the sentence. Therefore, we expect block constrained training to perform worse in terms of the squared error, since the block-constrained updates form a strict subset of the updates allowed with the standard training. Nonetheless, we observe that the performance of the block-constrained training is not much worse to that of the standard training in terms of the minimum error achieved by each method.

We now report the performance of Architecture 1 with block-constrained training in terms of the F1-score on the entire set. Table 1 shows the individual performances on the different entities on Architecture 1, as

		Architecture 1				**Baseline 1**		
Entity	Precision	Recall	F1-score	Support	Precision	Recall	F1-score	Support
Food	0.03	0.01	0.01	10966	0.00	0.00	0.00	10966
Disease	0.05	0.03	0.04	8386	0.00	0.00	0.00	8386
Population	0.01	0.02	0.01	1196	0.00	0.00	0.00	1196
Effect	0.06	0.24	0.09	4939	0.00	0.00	0.00	4939
Other	0.87	0.84	0.86	168489	0.87	1.00	0.93	168489

TABLE 1. Comparison of the performance of Architecture 1 (block-constrained training) on the entire corpus with the mode of the training set as the baseline.

		Architecture 1				**Baseline 2**		
Entity	Precision	Recall	F1-score	Support	Precision	Recall	F1-score	Support
Food	0.03	0.01	0.01	10966	0.06	0.06	0.06	10966
Disease	0.05	0.03	0.04	8386	0.04	0.06	0.05	8386
Population	0.01	0.02	0.01	1196	0.00	0.01	0.00	1196
Effect	0.06	0.24	0.09	4939	0.03	0.02	0.02	4939
Other	0.87	0.84	0.86	168489	0.87	0.85	0.86	168489

TABLE 2. Comparison of the performance of Architecture 1 (block-constrained training) on the entire corpus with the baseline 2. In this baseline, we count the fractions of each category in the training set, and then assign each word in the corpus a category with probability equal to its fraction.

well as a baseline in which the mode of the sentences over the training set was selected as the class. Since the relevant entities (food, disease, population, and effect) are outnumbered by the remaining part, the mode heuristic assigns the class to be the 'other' class for every word in the corpus. On the other hand, while its overall performance is low, the neural model has a non-zero F1-score on the relevant entities. Table 2 shows the comparison with another baseline, where each word in the corpus is independently assigned each category with a probability equal to the fraction of occurrence of the category in the training set. This weighted random baseline performs better than Architecture 1 on identifying food and disease, but performs worse in terms of correctly extracting population and effect. This suggests that Architecture 1 does not seem to provide any significant benefit over this baseline. Therefore, we are currently looking into another architecture that we describe in the next section.

5. ONGOING WORK: ARCHITECTURE 2

We are currently in the process of implementing another architecture that directly takes as input a sentence, and produces a sentence as output. The architecture is shown in Fig. 6. The input to the model is an input sentence that feeds into a GRU Encoder. The representation obtained from the GRU encoder is then fed into a hidden block layer similar to Architecture 1. Finally, the GRU decoder takes the hidden block representation and produces an output sentence. We expect this architecture will improve over the performance of Architecture 1, since we can (a) bypass some of the issues associated with imposing a fixed vector representation on the variable length sentences, (b) model the long-term dependencies, and (c) learn to focus on parts of the sentence that are more likely to contain the entities and the relationships.

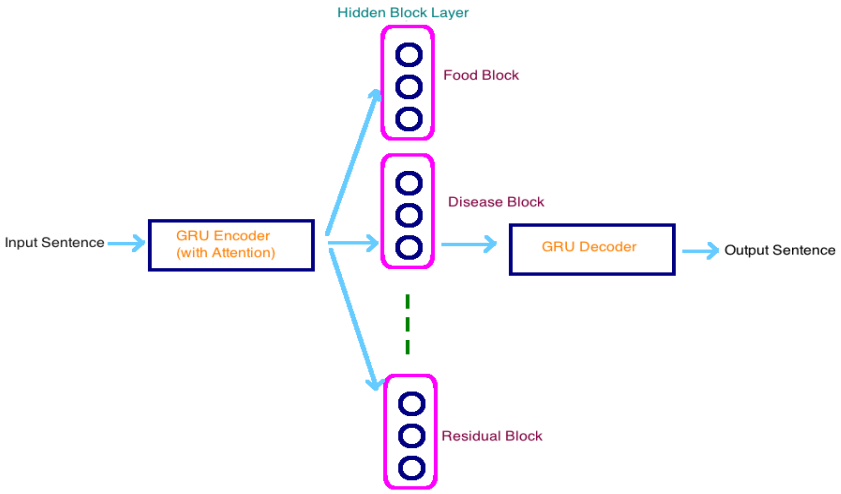


FIGURE 6. Architecture 2

REFERENCES

[1] M.-L. Zhang and Z.-H. Zhou. Multi-Label Neural Networks with Applications to Functional Genomics and Text Categorization, *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2005.