Identifying Novel Treatments for Rare Diseases using Radio Transcripts

Rahul Yedida

Abstract

Identifying novel treatments for rare diseases is a difficult medical problem, exacerbated by the few number of cases reported and clinical trials conducted. This document describes an approach using radio transcript data to mine novel treatments for diseases, along with the existing files and code.

Data

The data is transcripts of a medical radio show, where episodes consist of people describing their experiences living with various diseases. Some episodes discuss specific courses of treatments they took, while others simply discuss different stages of such diseases. It is known that there are no nonsense statements in the audio. This audio was passed to Google Cloud to generate text transcripts.

Initial proposal

I initially proposed a 7-step approach to the problem, after going through the data. For a detailed intuition behind the steps, please refer to, "initial writeup.pdf".

1. **Low confidence text pruning:** The transcripts are not official, and are simply extracted from MP3 files using Google Cloud. The Google Cloud engine chunks the text, and assigns confidence scores (from 0 to 1). I think a threshold of 0.96 to 0.965 should be reasonable, after going through a few excerpts.
2. **Irrelevant term removal:** In this step, we remove healthcare terms that provide no useful information, such as "medicine", "healthcare", "pharmacy", etc. For example, it is not useful to know that a patient cured symptom X simply by taking "medicine"; it is more helpful to understand the specific medication taken.
3. **Classification:** At this stage, we run a recurrent neural network, such as an LSTM with a one-one structure (i.e., one output for every word, that describes whether it is a medical term or not). The outputs could be binary, or multi-class (drug, disease, or irrelevant).
4. **Speculation removal:** In this stage, we remove speculative phrases, such as "X leads people to do Y". However, because the transcripts do not have periods, it is difficult to find out where the sentences begin and end. It is possible to instead simply remove the words on either side of such phrases.
5. **Adjacent duplicates removal:** Once we've highlighted medical terms that are of interest to us, we wish to form pairs of adjacent important terms. At this stage, we remove adjacent duplicate words.

6. **Pair formation:** Form pairs of adjacent medical terms.
7. **Validation:** Validate the pairs with the help of an expert or a database.

Modifications

The approach above has since been largely modified. After a discussion with Daniel, I learned that:

1. The focus is not on the validity of the treatments proposed, since they can be checked. The focus of the project is to simply generate reasonably good pairs; that is, false positives are acceptable, but not false negatives. Therefore, we do not do step 4 above.
2. PubMed is a valuable source of data for training a language model. PubMed is a comprehensive collection of medical papers. PubMed itself only gives access to abstracts, but a subset called PubMed Central (PMC) gives access to the full papers (https://www.ncbi.nlm.nih.gov/pmc/tools/ftp/). PubMed has about 28M papers, while PMC has about 2M papers.
3. An alternative to learning the medical terms is to simply use a dictionary file.
4. Yet another alternative is to build a language model to obtain word embeddings and see what terms cluster together—do drugs cluster together? Do diseases cluster together? Are these two distinct clusters?
5. A lot of medical terms are multi-word, which would make the above process a bad idea. A paper I found later (https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7747810) uses this concept to tag the beginning, inside, and outside of medical terms. Their results show that neural networks are not necessarily better than other models, such as Conditional Random Fields (CRFs). Their paper does not link to code, but they seem to imply that passing word embeddings learned by a language model to a CRF may be a good idea.

Later, Dr. Chirkova linked me to BioBERT (https://github.com/dmis-lab/biobert), a language model for medical text. BioBERT is the BERT language model by Google, fine-tuned on PubMed data. BioBERT can perform Named Entity Recognition (NER), which is essentially step 3 in the approach above. After NER, it is trivial to form pairs.

Code

All code developed is available at https://github.com/yrahul3910/csc830-drug-disease-discovery (private repository link). Some files have not been committed to the repository. The files are described below:

1. `scripts/01_low_confidence_prune.js`: Removes all excerpts with confidence below 0.96, and generates one JSON file. This creates `extracted.json`.

2. `scripts/02 - Fetching Data.ipynb`: Jupyter notebook to test fetching data from the server. I like having a notebook to test some part of the code, and then developing a larger notebook/script that does the entire job.
3. `scripts/03 - Download xml files.py`: Script based on the above to fetch all data from the server and parse the resulting XML.
4. `scripts/check-drug-name-percentage.py`: I was asked to check what percentage of the drugs in DrugBank were present in our transcripts data. This file computes all three interpretations of that question. This depends on a file called `drug-names.txt`.
5. `scripts/generate-tsv.py`: Generates a `test.tsv` file for the BioBERT model to generate predictions.

The rest of the code files are BioBERT code. Download the BioBERT code from the repo link above, then download their pre-trained weights from https://github.com/naver/biobert-pretrained. To run their NER code, you need to fine-tune the pre-trained models on your own dataset first, then run predictions. This requires manual labeling, which I have not done yet. Instead, I downloaded their NER datasets (https://drive.google.com/open?id=1OletxmPYNkz2ltOr9pyT0b0iBtUWxslh), and used the `s800` data (I randomly picked this one) to train (you need the `train.tsv` and `train_dev.tsv` files from here). For the `test.tsv` file required, use generate-tsv.py.

With these files set up, you can start training the BioBERT model. `cd` into the directory with the pre-trained weights, and run

```
$ export BIOBERT_DIR=$PWD
```

Next, go to the directory where the different .tsv files are present, and run

```
$ export NER_DIR=$PWD
```

Now, move to the BioBERT code directory. Train the model using:

```
$ mkdir ./tmp/
$ python3 run_ner.py \
            --do_train=true \
            --do_eval=true \
            --vocab_file=$BIOBERT_DIR/vocab.txt \
            --bert_config_file=$BIOBERT_DIR/bert_config.json \
            --init_checkpoint=$BIOBERT_DIR/model.ckpt-1000000
            --num_train_epochs=10 \
            --data_dir=$NER_DIR \
            --output_dir=./tmp/
```

Presumably, you can then run predictions by setting `--do_train=false` –`do_predict=true`.

However, BioBERT code seems to keep crashing, complaining about a missing `token_test.txt` file, for which I raised a GitHub issue (see https://github.com/dmis-lab/biobert/issues/50). Other people also seem to be facing this same issue (for example, see https://github.com/dmis-lab/biobert/issues/54). In the meantime, I found another language model called FLAIR (https://github.com/zalandoresearch/flair/), which has an easy API. There has been an attempt to fine-tune this on PubMed data, called BioFLAIR (https://github.com/shreyashub/BioFLAIR/). Using FLAIR's documentation with the data and code from BioFLAIR should be a good alternative to BioBERT.

Current Progress

The pipeline below shows the completed steps in green, and the others in black:

1. **Low confidence text pruning:** Achieved by `scripts/01_low_confidence_prune.js`. For a pipeline, it may be a good idea to either
   (a) port this to Python or
   (b) instead of the pipeline being a Python API, create a *nix-style set of commands that can be piped from one to the next.

   Both are easy to do, and take 30 minutes or less.
2. **Irrelevant term removal:** A word list has been built, and while there's no code in the source to do this, it's one line in Python using a list comprehension:

   ```
   [word for word in excerpt.split() if word not in excluded_words]
   ```
3. **Classification:** We need to place either BioBERT or BioFLAIR here.
4. **Adjacent duplicates removal:** Again, while there's no code file, this is a one-liner (credit: https://stackoverflow.com/a/3463582):

   ```
   [key for key, group in itertools.groupby(medical_terms)]
   ```
5. **Pair formation:** Trivial in Python (credit: https://stackoverflow.com/a/21303286):

   ```
   pairs = zip(terms, terms[1:])
   ```
6. **Validation:** We currently have no method of doing this.