NLP Design Example

Problem Statement

One of the most important elements of clinical healthcare is the correspondence between physician and patient in a normal office visit. During this encounter, the physician understands the patient's problem, its history and course, and translates this interaction into a semi-standardized electronic medical record. At times, this note occupies the physician's attention, leading him or her to spend their time clicking boxes and typing into a computer rather than engaging their patient directly.

How can we free the physician from entering data into an electronic health record and allow them to engage their patients directly while at the same time maintaining proper electronic records?

One solution may be that, if we can record a natural conversation between physician and patient and use natural language processing techniques to yield a clinical note that takes the place of the physician laboriously wrangling awkward medical health records, this would be a great improvement.

To do so, we will discover various machine learning paradigms, understand how they work, and apply them sensibly to generate clinically significant summaries of raw conversations.

General

This task is certainly close to my heart. One of the most important skills all clinicians learn is the art of the interview. The patient interaction, directed by differential diagnosis, should yield a product that is sufficient to convey the absolute essential information to a colleague (or insurance company, or regulatory body) and stands as a lasting commemoration of both observation and inclination.

Here, I present some metadata considerations and discuss both extractive summarization (with weighted word frequency) and abstractive summarization (with LSTM models). I focused here on generating the *Summary* section of the note. To briefly mention the other sections:

- the *Objective* section is expected to be be slighter easier since it is more structured, in practice initialized to negative findings, and matched to significant positives by vocabulary
- similarly, the *Assessment* can match keywords to ICD-10 coding
- the *Plan*, is a (physician-focused) subset of global summarization

A Nice to Have: Speaker Identification

One feature we might want to identify is the speaker. This is a mostly solved problem, made easier by the fact that in our situation, the number of speakers is small. Domain-leading models like SincNet use CNNs to assign identity to speakers in audio segments. Supervised solutions require pre-labeled training data. That could work well in this setting, as each physicians will generate tons of data of their own voice for training. But here, I propose a simpler, more general, unsupervised method (k-means clustering) that has given me good results in the past:

- Use silence detection via de-noised spectral energy to separate the audio signal into silence-segments and speech-segments
- For each speech segment (usually multiple words):
  - extract linear-/log-/mel-frequency coefficients (from different FFTs) to transform the all segments into vectors of equal length
  - extract commonly used characteristics used in audio analysis (recurrence period density entropy, range, spectral centroids, frequency quartiles, peak frequency, average dB) from these transformations as features
- Run the unsupervised k-means clustering with two target classes. The input vector is the combination of the frequency-derived features determined above. These should all be numerics.

The result is a model that will yield centroids (in high-dimensional space) with vectors elements classified based on the closest centroid.

<u>Other Valuable Data</u>

- Patient characteristics (age, gender, previous medical history / charting)
- Physician-based models (Dragon Naturally Speaking, for example, creates individual user profiles to learn a physician's tone, accent, speech patterns, particular vocabulary, etc)

<u>Text Pre-Processing</u>

Almost all methods in NLP require cleaning and tokenizing text. The goal is to split text into standardized words (or sentences) and involves:

- **Case normalization**
- **Punctuation removal:** (strong case is `` `'\w+'` ``, medium case is something like `` `'[\w-]+|\$[\d\.-]+|\S+'` `` for mixed-type words, I prefer the weak case `` `[^,.;!]` ``)
- **Word / Sentence tokenization** (at this point a split on whitespace may suffice)
- **Spelling correction / identity resolution:** for example, in the given transcription *Melius —> malleus*.
- **Stopword removal:** although beware we will loose the operative *no* in "[no] pain"
- **Stemming:** although it causes many awkward scenarios (*ankle—>ankl*) and bad collisions (*does—>doe*)
- **Lemmatization:** unfortunately, *pharyngitis* becomes *laryngitis* (which are not the same)

The result is a sanitized vocabulary.

<u>Tokenization</u>

We will need an embedding to convert our vocabulary and sentences to numerical encodings. At small scale, it's common to use a one-hot-encoding in which each word in a sample is indexed and a sentence consists as a sparse NxM matrix where N is the number of words in the sentence and M is the size of the entire vocabulary. Unfortunately, even with condensed representations, this quickly stresses memory resources. Another approach uses skip-grams. These are based on the hypothesis that a word's context influences its meaning: each word exists in a vector that also encodes its neighbors, i.e. co-occurrence with other words in the vocabulary, such that closely related words also lie close to one another in the learned vector space (similarity often calculated as the cosine of the angle between vectors). Finally, we can used indexed sentence representations wherein a sentence is simply a vector of the indices of each word. We'll use this last encoding for the LSTM model (since it is the model's job to maintain context, not the encoding).

<u>Weighted Word Frequency / Approach</u>

As far back as 1957, H. P. Luhn suggested that word frequency analyses can yield extractive models; given a corpus to summarize, we identify sentences that are highly enriched in target words or phrases, extract those sentences directly (or clean them slightly) and place them into a summary. Naive approaches can take a set of documents, such as our transcribed patient interactions, and calculate relative word frequencies, the hope being that 'rare' words indicate a sentence to be extracted. This is useful if there is absolutely no knowledge known about the content of the input documents.

Domain specific wordlists can be generated using TF-IDF, the product of Term Frequency (Number of time the word *t* occurs in the text / Total number of words in text) and Inverse Document Frequency (Total number of documents / Number of documents with word *t* in it). We can measure the significance of a particular term in a set of specific (medical) corpuses compared to a wider range of text (often taken from books or news articles). In fact, if we have access to a set of complete SOAP notes, we can use this as our corpus of significantly useful words.

More recent extraction methods may also use SumBasic, TextRank but the general theory remains the same. For an input document:

- Tokenize
- Generate of word-level weights
- For each sentence in the input document, only consider the sentence if it has less than N words (a **hyper-parameter** used for conciseness), or consider partitioning sentences on non-significant words
- Tokenize every word in that sentence as above
- Sum the weighted frequencies of every tokenized word in the sentence and place them in a sorted structure (like a heap)
- Pull the sentences with the largest scores
- Likely modify the sentences, potentially by pulling out *only* the significant part with some modification ("and my {hand hurts}" —> "Patient reports their {hand hurts}")

The result is summary comprised of sentences enriched in 'significant' words. Unfortunately, we are limited to what directly appears in the text and we also cannot model sentiment that spans more than one sentence.

Encoder-Decoder Approach

Abstractive summarization is in some ways a more powerful set of techniques as we are able to generate new sentences that may not have been in the original text. One common standard is the encoder-decoder paradigm, designed to address sequence-to-sequence problems. The architecture is comprised of two components: the encoder reads input sequences, encoding them into fixed-length vectors while the decoder maps the vector representation back to a variable-length target sequence.

Neco & Forcada, 1997, posited the idea of using such at structure with RNN encoders to perform machine translations. The goal is to decompose the state of the translation into a series of time steps with "cells" at each time-step. RNNs are called *recurrent* because they perform the same task for every element of a sequence, with the output being depended on the previous computations.

RNNs often suffer from what's called the vanishing gradient problem, which is related to the dependence on the distance between sequenced information. In or case, for example, a patient may say "I just, my **ankle** has always been fine, and a lot at time at the warehouse, because I often have to lift boxes up and down in the loading bay, but for some reason the past two weeks, it's been really **painful**." Here, the sentence is somewhat disjointed and the distance between the relevant information **[ankle]** and the point where it is needed to make a prediction **[painful]** is very wide, introducing computational complexity.

To solve these kinds of problems, Sutskever et al. (2014) and Cho et al. (2014) developed methods to use RNNs (or GRUs) for both encoder and decoder, and popularized Long Short-Term Memory.

LSTMs are an evolution from a traditional RNN in part designed to mitigate the gradient problem. LSTM has the ability to add or remove information through structures called gates: the forget gate, input gate, candidate gate and output gate layer. Thanks to the gate mechanism that allows for explicit memory deletes and updates in LSTM, the problem of exploding/vanishing gradients is controlled so that the model can capture long-distance dependencies in a sentence much better.

The following rubric follows the encoder-decoder architecture as per Nallapati et al (2016) and also refined by Liu et al (2018) at Google Brain. To train, we need a set of transcriptions combined with properly written Subjective summaries. Then we:

1. Index-tokenize sentences as above
2. Zero-pad our indexed-sentence matrix to some max length (a **hyper-perameter**), then transpose for proper operation
3. Separate a training and testing set
4. The model is:
    1. **Input: B**atch of encoded, padded sentences
    2. **Droupout:** During training, a dropout layer randomly sets input units to 0 with a particular rate [**hyper-parameter**, commonly 0.2] to help prevent overfitting
    3. **Bidirectional LSTM (built on GRU):** Some suggest two layers, problem dependent. Important **hyper-parameters**:
        - Number of hidden neurons in (each) layer. Hagan et al. 2002 suggest $N_h = N_s / (\alpha * (N_i + N_o))$ where N_i is the number of input neurons, N_o is the number of output neurons, N_s is the number of samples in the training data set and α is an arbitrary scaling factor in the range between 5 and 10. Others suggest 2/3 of total number of inputs.
        - **Activation:** typically *tanh* or *relu*, though I am oddly fond of *LeakyReLU*, purely because of its neuronal biological analogy
    4. **Activation:** *softmax* works best because it allows the model to interpret the outputs as probabilities (i.e., what word comes next)
5. *Loss function: C*ategorical cross-entropy
6. *Optimizer:* Adam (almost always the first choice)
7. *Evaluation metric:* Full length F1/ROGUE-6 (ROGUE-N are accuracy metrics of overlapping N grams, in this case 6; F1 refers to F1-score calculated from precision and recall)

Once trained, the decoder generates the response sentence in a token-by-token fashion. It uses the encoder's context vectors, and internal hidden states to generate the next word in the sequence. Attention mechanisms, a la Bahdanau et al. 2014, can be used to limit the decoder's attention to certain parts of the input sequence, rather than the entire fixed context.

Some Edge Cases

Clinical conversations can be long include long stretches of time entirely unrelated to the medical case at hand. For example, "It's a beautiful day, isn't it, doc?" is unlikely to be clinically relevant. The output should be null, i.e. it should not contribute to the end product. But there are many edge cases that can be trickier:

Usually But Not Always Inane information

Often in the course of patients' self-described illness history, you'll hear "I was with my wife", or "my boyfriend was driving me home" or "my buddy told me. . ."; this is also usually non-contributory
The special case here is that of infections disease. The "buddy" may have experienced some symptoms as well ("my buddy also got sick"), which can give context and might be clinically relevant

Potentially But Not Always Diagnostic Information

In the given transcript, the patient had taken a skiing trip. Usually, ”I took a trip. . .” is benign but there are certain cases where it is relevant. In the context of gastro-intestinal or infectious disease, for example, travel history can be pertinent.

Contextually Unnecessary Information

Prior histories can be really long. For example, it’s common to ask about prior surgeries, regardless of the current scenario, especially for an initial visit. That will elicit a lot of very medically relevant phrases that could potentially be distracting if these surgeries were many years ago and have nothing to do with the current chief complaint.

Bibliography

Bahdanau D., Cho K., and Bengio Y. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473, 2014.

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Proceedings of the Empirical Methods in Natural Language Processing (EMNLP 2014). to appear.

Hagan, T., Demuth, H., Beale, M., and De Jesús, O. Neural Network Design, 2nd Edition (2002).

Luhn HP (1958) The automatic creation of literature abstracts. I.B.M. Journal of Research and Development.

Neco, R. P., & Forcada, M. L. (1997, June). Asynchronous translations with recurrent neural nets. In Neural Networks, 1997., International Conference on (Vol. 4, pp. 2535-2540). IEEE.

See A., Liu P., and Manning C.. Get to the point: Summarization with pointer-generator networks. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 1073–1083, July 2017.

Sutskever, I., Vinyals, O., and Le, Q. (2014). Sequence to sequence learning with neural networks. In Advances in Neural Information Processing Systems (NIPS 2014).