

Automated Annotation for Medical Discharge Summaries: A Preliminary Study

Gayle McElvain (mcelvain@stanford.edu) and Anand Madhavan (manand@stanford.edu)

1 Introduction

As the problem of limited training data is ubiquitous in the NLP domain, strategies that can leverage small amounts of manually annotated data are potentially quite useful. This is precisely the central challenge for our project, which involves two separate but related labeling tasks, and not enough training data for either.

Task 1:

The first task is to automatically label medical narratives with annotations from a set of twenty fairly common, high morbidity rate, neonatal complications. The list of annotations provided to us contains complications such as: intraventricular haemorrhage (IVH), pneumonia (PNA), necrotizing enterocolitis (NEC), etc. A full list is shown in Table 1.

Task 2:

The second task requires that the system be able to identify additional complications explicitly mentioned in the discharge summary but not known in advance. These are complications, which are likely to be less common across the patient population but are extremely important to extract from the medical records of certain patients.

Presumably, with sufficient training data, it would be possible to build a classifier for each of the twenty predetermined complication labels. However since our training set consists of only a hundred and forty patients in total, and 14 of the 20 labels were associated with 10 or fewer patients, this approach was clearly infeasible. Similarly for the second task, there were only twenty patients labeled with additional complications. Instead of relying solely on manually annotated training data, we leverage it in order to develop a pattern matcher that automatically generates more “labeled” data. This output is then used to train a classifier that recognizes whether or not a particular sentence contains some mention of a complication. The classifier is feature engineered to capture the general characteristics (linguistic and otherwise) of any complication mention, that is, it captures instances from a much broader set than the twenty complications the pattern matcher was developed for.

2 Background

There is a vast amount of rich medical information captured in non-standardized narrative records, of which discharge summaries are a single type. In order for this information to be used by electronic systems, whether for research purposes or for databases developed to improve the access and quality of patient records, some amount of natural language processing is required. Manual annotation and data entry is simply too costly on the scale required. Most general purpose NLP systems, however, are not designed for a domain like medical narratives, where the text is typically transcribed speech fragments, with a specialized vocabulary and grammar. For this reason, there is strong motivation for the medical informatics community to develop domain-specific medical language processing tools.

One barrier to the widespread development of such tools is the restricted access to medical records. Additionally, automatic de-identification of clinical data is not a trivial problem. Though recent attempts [8] have achieved relatively high accuracy, there still does not currently exist a de-identification system considered robust or accurate enough to make medical data widely available to the public. For our data, we first built an in-house de-identification system based primarily on regular-expression matching and heuristics.

The more interesting problem for medical language processing, of course, is the automatic evaluation of patient records and domain-specific information extraction. There is a significant body of literature on classifying patient outcomes from discharge summaries, which might be more generally be termed text classification in the NLP community. Examples of recent work in this area include: classifying patients’ smoking status [10, 13, 3, 2], adverse event detection[7], and foot examination findings[9]. This goal of labeling discharge summaries with patient outcomes is similar to our task, except that we have a larger number of possible labels (20 predetermined, and an open set of unknown labels) than is typical in this domain (which ranges from binary to five-way outcomes). Partly for this reason, we expect that the nature of our data set may be more amenable to string matching methods than those described in the above papers, at least for labeling the twenty predetermined 20 complications. A second area

Label	Complication	YES	NO
SEPSIS	Sepsis Infection	61	73
NEC	Necrotizing Enterocolitis	7	123
IVH	Intraventricular Hemorrhage	28	106
SEI	Seizure	3	131
HYD	Hydrocephalus	7	127
PDA	Patent Ductus Arteriosus	46	88
BCS	Bacterima, Coagulase Negative Staph	11	123
PPHN	Persistent Pulmonary Hypertension	1	133
RDS	Respiratory Distress Syndrome	95	39
BPD	Chronic Lung Disease (Bronchopulmonary Dysplasia)	18	116

Label	Complication	YES	NO
PNE	Pneumothorax	8	126
PUL	Pulmonary Hemorrhage	3	131
ADR	Adrenal Insufficiency	4	130
ARF	Acute Renal Failure	7	126
ATN	Acute Tubular	4	130
UTI	Urinary Tract Infection	5	129
BAC	Bacterima, NOT Coagulase Negative Staph	7	127
ARR	Arrhythmia	5	129
PNA	Pneumonia	4	130
DEAD	Patient Fatality	8	126

Table 1: Labeled Complications (By Patient)

of related work deals precisely with the task of matching free text to a large dictionary/lexical web of medical annotations[4, 12, 11, 6]. These annotations can be drawn from disease outcomes, medical procedures, diagnoses, and common medical terms as well as a wide range of annotation modifiers. Our task is somewhat more limited than the references above, though, which describe systems for extracting a very broad range of annotations, typically mappings to UMLS phrases.

We are not planning to use the UMLS knowledge base for this project, though it may be something to be explored in future work. Our current task is limited to such a small subset of what’s available in the UMLS knowledge base that we are focusing on methods engineered for this specific data set. Later we may compare the performance of our system with a more general purpose approach provided by systems like MEDLEE[5].

3 Data

Source of Data

This data was collected as part of an ongoing collaboration between Stanford Medical Center and the Probabilistic AI Lab. Suchi Saria, a student in Daphne Koller’s group working on this project, provided us with discharge summary documents for 1000 patients. As training data, one hundred and forty patients had been annotated Y/N for each of the twenty predetermined labels. The annotators were nurses at the Stanford medical center. They were instructed that all annotations should derive from explicit mentions in the text, as the automated task is not supposed to require reasoning about additional world/domain knowledge or implicit information. The annotators were also instructed to note any additional complications relevant for a specific patient that were not covered by the twenty labels provided. Table 1 shows the distribution of labels provided for the first task.

Data Preparation

DEIDENTIFICATION

Discharge summaries were first extracted from long text files containing different types of narrative reports. The discharge summaries were then stripped of all identifying information including patient name, account information, dates within the header and the document, and names of medical staff (doctors, nurses, etc). This was primarily done with regular expressions and the code runs from Deidentifier.java.

GENERATING LABELED TRAINING FILES

Initially we understood that the training data would consist of documents annotated independently, solely on the basis of explicit information in the text. As it turns out, the data we currently have access to, actually consists of one set of labels per patient. And many patients are associated with multiple discharge documents, which may contain different and/or conflicting information about a single patient. For the work presented here, we did not attempt to add a component to the system that determines patient outcomes from multiple documents. There were several reasons for this decision. First, during the de-identification process all dates and times were stripped from the documents. Also, often but not always, a discharge summary can include information about previous conditions and hospital stays, such that a significant amount of temporal reasoning within the document would also be required to disambiguate instances of a particular complication. Finally, there remains some uncertainty at the present point in time as to whether or not we actually have access to all the relevant discharge documents for each labeled patient. We do intend to add this component for reasoning over multiple documents in the future, once some of these issues have been resolved. The consequence, of course, is

that not all our training labels accurately reflect content in a given training document, and this has some effect on the number of false positives / false negatives reported by our RegexpMatcher (RegexpTest.java). Since all the training data is currently be used for development anyways, we plan to test on new data with per document labels, as it becomes available. Both these facts need to be taken into account when reviewing the results reported for the RegexpMatcher below.

TOKENIZER

The next step after deidentification was to separate the document into spans (Span.java), each approximately one sentence long. All our regular expressions are designed to match on only one sentence at a time, but the main reason for tokenizing the document is to be able to use output from the RegexpMatcher as input to the classifier. More is said about input to the classifier in Sec. 4. A span includes not only the sentence content, but also the section header, sentence location, and header location within document. Since these are text files, the Tokenizer does some document cleaning (removing large gaps & erroneous characters) and uses regular expressions to identify the section headers. The code for this is the method tokenizeDocument() inside Item.java. Once the spans created by the tokenizer have been processed by the RegexpMatcher, they also contain information about any patterns that they matched.

4 Methods

RegexpMatcher

For the first task, labeling discharge documents with the patient outcomes listed in Table 1, we engineered regular expression lists for each patient outcome. Initially, we guessed that names of rare complications were unlikely to occur in a document unless a patient was ultimately labeled with that complication. This intuition turned out not to be entirely correct. For many complications with only a few positive instances (i.e. phrases in the text indicating the patient actually has some complication), there were just as many or more negative instances indicating the patient did not have the complication mentioned (detailed comments on each pattern list indicating the different combinations of regular expressions tried and their outcomes are provided in the file LabelPatterns.java). In order to catch negated instances, a set of negative regular expressions was also generated for each label. The mode runTest() inside RegexpTest.java, shows the logic used to classify a single discharge document. The program iterates over each span in the document once per complication. It looks for a match to any one of one of the positive patterns for that complication. If a match is found, that span is checked against all of the negative

Positive Patterns:

1. (?i)ventricular[^\s]*{0,20}?(haemorrhage|hemorrhage)
2. (?i)intracranial\s*(haemorrhage|hemorrhage)
3. (?i)head\s*\s*ultrasound[^\s]*?(haemorrhage|hemorrhage)

Negative Patterns:

1. \b(no|never|none)\b[^\s]*?(haemorrhage|hemorrhage)
2. negative[^\s]*?(haemorrhage|hemorrhage)
3. haemorrhage|hemorrhage[^\s]*negative

Figure 1: Positive and Negative Patterns for Intraventricular Hemorrhage

patterns specific to that complication. The strategy used to label a document (Y/N) for a given complication, ensures that a document is labeled “N” if no spans matched a “positive pattern” for that complication or if *any* span matched a negative pattern. If at least one span matched a positive pattern and *none* matched a negative pattern, the document is classified “Y”. To illustrate the type of patterns used, we show the relevant positive and negative patterns for IVH in Fig. 1 below.

In addition to engineering label-specific negative patterns like those listed in Fig. 1 above, we also experimented with “NegEx”, a freely available regular expression algorithm for identifying pertinent negations in medical narratives[1]. A listing of the all the phrases used by the algorithm to contextualize negative mentions of medical findings is available at <http://www.dbmi.pitt.edu/chapman/NegEx.html>. To integrate NegEx with our RegexpMatcher, we simply had to pass the list of positive patterns for the label in question to a wrapper function which would then return whether or not any of the patterns had been found within a negated context. The method runTest() in RegexpTest.java allows for using both negative patterns and the NegEx method for a given complication.

Classifier

The second task described above aims to automatically extract pertinent patient outcomes from discharge summaries. These outcomes map to an open class of labels, which is not known in advance. We chose to reformulate this problem as deciding, for each sentence in a document, whether or not any medical complication is mentioned. Once the relevant sentences are identified within a document, it is left as a post-processing step to filter out any excluded complications and/or negated findings. Additionally, one could extract from each remaining sentence the least frequent noun phrase (e.g) to generate a list of “labels” for each document. Given the limited amount of manually annotated training data, we instead used the RegexpMatcher described above to generate positive training examples for the classifier. The key to using these sentences as positive training examples is not

to engineer features which benefit from knowledge of the *specific* complication matched to a sentence. Features must instead try to capture general characteristics of the context, content and document location, which are likely to be true of any complication mentioned within a discharge summary. In order to generate negative examples for input to the classifier, 15 randomly selected summaries from the unlabeled pool were manually edited. This process consisted only of removing sentences that explicitly mentioned complications from the original document and took only slightly more time than reading the files straight through.

Feature Engineering

We used an off the shelf decision-tree based classifier (J48) from WEKA, a Java based machine learning toolkit, for our purposes.

The first kind of features we tried were features specific to the section that the span came from, such as header information (whether a span belonged to certain header sections), the location of the section in relation to the document, and the length of a section. Since section headers are free formed text, spelling errors, and different ways of saying the same thing (such as ‘Diagnoses at time of discharge’, ‘Diagnoses at discharge’, ‘Diagnoses at date of discharge’ etc refer to similar sections) and our features needed to have some amount of leeway to capture these. We also looked at sentence position within the section.

The second kind of features we tried were specific to the sentence and its contents. We engineered features for specific keywords and string patterns. Whether a span contained abbreviations (eg: MRI), whether a span belongs to certain sections (eg: ‘Diagnoses section’), or contains: percentage symbols, time formats (eg: 2:30), date formats, title tokens (eg: such as ‘Dr.’, ‘R.D.’, ‘M.D.’, etc), integers, or decimal numbers were all incrementally useful. Certain keyword groups, such as ‘treatment’, ‘laboratory’ were seen to presage outcomes, or other groups such as ‘acute’, ‘severe’ indicated conditions, or yet other groups such as ‘mother’, ‘delivery’, ‘pregnancy’ were features going the other direction, since they tend to describe conditions not attributable to the patient. Other useful contextualizing phrase groups included: ‘diagnosed with’, ‘history of’, ‘symptom of’, and ‘evidence of’. We also used morphological features (e.g., does the sentence contain a word that ends in ‘mia’, ‘nal’ or ‘ral’).

The final kind of feature we tried were based on term frequencies over the entire discharge corpus. Mean word frequency and minimum word frequency were useful numerical features, which was expected since names of medical complications are typically the least frequent words in sentences where they occur.

In order to determine the usefulness of each feature we used Weka’s InfoGainAttributeEval module with a

Rank	Info Gain	Feature
1	0.177826508	Header length
2	0.166078434	Header location
3	0.134626966	Relative span location in doc
4	0.12224689	Mean length of word in span
5	0.121799034	Span location
6	0.109669788	Header contains ‘diagnoses’
7	0.105392243	Span contains ‘history of’ etc
8	0.10532083	Length of longest word in span
9	0.038952827	Span length
10	0.033006751	Mean frequency of words

Table 2: Top 10 ranked features as illustrated through an information gain ranking using Weka’s InfoGainAttributeEval module and a Ranker Search method.

Ranker Search method. We summarize the top ten features in Table 2.

Using a successive application of features, we were able to go from F1 scores of 0.477 for the YES label (just using ‘diagnoses’ in the header) to an F1 score of 0.825 on the same label. These results are shown for both labels in Table 3.

Has Complication (YES)			
	Prec	Recall	F1
Header has ‘diagnoses’	0.916	0.322	0.477
With all our features	0.799	0.854	0.825
Does not have complication (NO)			
	Prec	Recall	F1
Header has ‘diagnoses’	0.965	0.699	0.477
With all our features	0.813	0.747	0.778

Table 3: Effect of feature engineering on precision, recall and F1 measures.

5 Results

Labeling with RegexMatcher

Since many complications handle only a few instances in our training data, it was not possible to separate the labeled files into a separate development and test set. We report here the F1 scores achieved on the development set, which is clearly not an accurate measure of performance. These results will be replaced with performance statistics on a new test set, once it’s available. For the interim, we have Table 8 in the Appendix, which shows F1 scores for each individual label as well as the overall performance of the RegexMatcher. Not surprisingly, the complications which are the most difficult to engineer regular expressions for, are those whose name consist of more common nouns in the medical vocabulary. Respiratory Distress Syndrome (RDS) and Arrhythmia (ARR)

were both challenging to engineer for this reason. Note that lexical frequency is not always correlated with incidence frequency. Though RDS is one of the most common complications in our data set, ARR is one of the least frequent.

In addition to negative and positive regular expressions, we also used the NegEx utility [1] and found that it was a doubled edged sword, typically improving the false positives, but also worsening the false negatives in many cases (SEPSIS and RDS, e.g.). There were, however, some labels (NEC, PPHN, CRF and PNA) that benefited from using NegEx, so we left this as a label-specific option in the code base. NegEx is able to catch negations that our regular expressions do not, particularly when the negation ranges over multiple clauses or is implied by embedding within various types of hypothetical statements.

1. He had no radiographic or clinical evidence otherwise of *necrotizing enterocolitis*.
2. There have been no further concerns with obstruction or *necrotizing enterocolitis*.
3. .. after she showed no clinical or radiological evidence of bacterial *pneumonia*.

Classification Results

We ran all experiments using the J48 WEKA classifier. Training data came from the following three sources:

A: The set of labeled files which were used to develop the regular expressions in RegexMatcher.

B: 15 randomly selected files of unlabeled data, from which complication mentions were removed. (As described in Sec. 4.)

C: 225 randomly selected files from the unlabeled pool, which were input to the RegexMatcher to generate a set of positive examples separate from A.

The first two experiments compare classifier performance when positive examples are drawn from A versus C. Results of this experiment are shown in Tables 4 and 5, where the scores generated are from 10-fold cross validation. Assuming that the distribution of known labels in the labeled files (A) is similar enough to the distribution in the unlabeled files (C), we would expect this comparison to largely reflect performance of *the RegexMatcher*. (Of course, it is confounded with the fitness of features between the two data sets, but since we did not strongly engineer features specific to A, we expect this confound to be relatively small.) What can be seen from Table 5, is that the number of false negatives increases slightly when we train with positive examples generated by the RegexMatcher on C. Given that the RegexMatcher was developed on A, this result suggests that the RegexMatcher likely has reasonable precision on unseen data, since it agrees with the classifier about what constitutes a complication-containing sentence around $\sim 75\%$ of the

time. Note that Table 4 shows the classifier agrees with output from the RegexMatcher on A around $\sim 80\%$ of the time. We can't estimate from this experiment what the recall measure might be for the RegexMatcher on unseen data, but it's likely to be considerably lower than precision. Going back to Table. 1 and noting that the majority of complications had fewer than 10 instances, we certainly expect that the generality of the RegexMatcher could be improved by developing on a larger data set. Table 4 on its own suggests that the feature engineering for the classification task was fairly successful, at least on this "makeshift" dataset.

Class	# of examples	Prec	Recall	F1
YES	2086	0.799	0.854	0.825
NO	1772	0.813	0.747	0.778

Table 4: Experiment 1 - Trained using positive examples from A and B and negative examples from B. Tested using 10-fold cross validation on the training data.

Class	# of examples	Prec	Recall	F1
YES	1649	0.766	0.768	0.767
NO	1772	0.784	0.782	0.783

Table 5: Experiment 2 - Trained using positive examples from C and B and negative examples from B. Tested using 10-fold cross validation on the training data.

We also ran a third experiment, which trained the classifier on data that consisted of positive examples generated by RegexMatcher on C and negative examples from B. The classifier was then tested against all sentences from A identified as positive or negative by RegexMatcher. We expected this test to result in high recall and low precision, since the test set only has a sentence as positive if it matched one of the predetermined labels. All the sentences containing new complications should therefore wind up as false negatives, which is consistent with the results reported in Tables 6 and 7. Without overstated this result, it seems to indicate that the features we used are actually generalizing to novel complications and are not restricted to the complication types recognized by the RegexMatcher. We ran two trials of this experiment with different amounts of positive examples. Table 6 demonstrates that although the model improves when the distribution of positive to negative examples in the input is closer to the actual distribution in the Test set, roughly 1:10, the performance is not drastically different than when the ratio is $\sim 1:1$. Features (not priors) are doing most of the work.

Class	# of examples	Prec	Recall	F1
YES	700	0.217	0.624	0.322
NO	1772	0.954	0.778	0.857

Table 6: Experiment 3 - Trained using positive examples from C and B and negative examples from B. Tested using examples from A (1906 positive examples and 19281 negative examples).

Class	# of examples	Prec	Recall	F1
YES	1649	0.203	0.785	0.322
NO	1772	0.97	0.694	0.81

Table 7: Experiment 4 - Same as Table 6 with more positive sentences in the training data.

Clearly, collecting appropriate evaluation measures is significantly hindered by the lack of available training data. As mentioned above, we do intend to continuing working on this project in the future with more training data as it becomes available. The work done here is probably best considered a pilot study, which has been successful in showing that the problems posed here are in fact quite tractable and that the investment in time and energy to create a larger, high-quality training set would most likely have high returns.

References

- [1] Wendy W. Chapman, Will Bridewell, Paul Hanbury, Gregory F. Cooper, and Bruce G. Buchanan. A simple algorithm for identifying negated findings and diseases in discharge summaries. *Journal of Biomedical Informatics*, 5:301–310, 2001.
- [2] Cheryl Clark, Kathleen Good, Lesley Jezierny, Melissa Macpherson, Brian Wilson, and Urszula Chajewska. Identifying smokers with a medical extraction system. *J Am Med Inform Assoc*, 15(1):36–39, January 2008.
- [3] Aaron M. Cohen. Five-way smoking status classification using text hot-spot identification and error-correcting output codes. *J Am Med Inform Assoc*, 15(1):32–35, January 2008.
- [4] Miller RA Cooper GF. An experiment comparing lexical and statistical methods for extracting mesh terms from clinical free text. *Journal of the American Medical Informatics Association*, 1998.
- [5] C. Friedman, L. Shagina, Y. Lussier, and G. Hripcsak. Automated encoding of clinical documents based on natural language processing. *J Am Med Inform Assoc*, 11(5):392–402, 2004.
- [6] Yang Huang, Henry J. Lowe, Dan Klein, and Russell J. Cucina. Improved identification of noun phrases in clinical radiology reports using a high-performance statistical natural language parser augmented with the umls specialist lexicon. *J Am Med Inform Assoc*, 12(3):275–285, May 2005.
- [7] Genevieve B. Melton and George Hripcsak. Automated detection of adverse events using natural language processing of discharge summaries. *Journal of the American Medical Informatics Association*, 12(4):448 – 457, 2005.
- [8] Ishna Neamatullah, Margaret Douglass, Li-wei Lehman, Andrew Reisner, Mauricio Villarroel, William Long, Peter Szolovits, George Moody, Roger Mark, and Gari Clifford. Automated de-identification of free-text medical records. *BMC Medical Informatics and Decision Making*, 8(1):32, 2008.
- [9] Serguei V. Pakhomov, Penny L. Hanson, Susan S. Bjornsen, and Steven A. Smith. Automatic classification of foot examination findings using clinical notes and machine learning. *J Am Med Inform Assoc*, 15(2):198–202, March 2008.
- [10] Ozlem Uzuner, Ira Goldstein, Yuan Luo, and Isaac Kohane. Identifying patient smoking status from medical discharge records. *J Am Med Inform Assoc*, 15(1):14–24, January 2008.
- [11] Kim W and Wilbur WJ. Corpus-based statistical screening for phrase identification. *J Am Med Inform Assoc*, 2000.
- [12] Long W. Lessons extracting diseases from discharge summaries. *AMIA Symposium Proceedings*, 2007.
- [13] Richard Wicentowski and Matthew R. Sydes. Using implicit information to identify smoking status in smoke-blind medical discharge summaries. *J Am Med Inform Assoc*, 15(1):29–31, January 2008.

Appendix

Labels	FP	FN	TP	TN	Pos	Neg	Tot	Acc	Err	Prec	Recall	F1
SEPSIS	38	8	58	66	66	104	170	0.73	0.27	0.6	0.88	0.72
NEC	6	0	15	149	15	155	170	0.96	0.04	0.71	1	0.83
IVH	6	7	20	137	27	143	170	0.92	0.08	0.77	0.74	0.75
SEI	6	0	3	161	3	167	170	0.96	0.04	0.33	1	0.5
HYD	2	1	5	162	6	164	170	0.98	0.02	0.71	0.83	0.77
PDA	22	0	56	92	56	114	170	0.87	0.13	0.72	1	0.84
ARR	4	0	2	164	2	168	170	0.98	0.02	0.33	1	0.5
PPHN	2	0	1	167	1	169	170	0.99	0.01	0.33	1	0.5
RDS	34	0	121	15	121	49	170	0.8	0.2	0.78	1	0.88
BPD	12	0	27	131	27	143	170	0.93	0.07	0.69	1	0.82
PNE	2	0	12	156	12	158	170	0.99	0.01	0.86	1	0.92
PUL	0	0	6	164	6	164	170	1	0	1	1	1
ADR	2	0	8	160	8	162	170	0.99	0.01	0.8	1	0.89
ARF	0	1	4	165	5	165	170	0.99	0.01	1	0.8	0.89
ATN	0	0	1	169	1	169	170	1	0	1	1	1
UTI	6	0	5	159	5	165	170	0.96	0.04	0.45	1	0.63
BCS	1	0	8	161	8	162	170	0.99	0.01	0.89	1	0.94
BAC	0	0	3	167	3	167	170	1	0	1	1	1
PNA	1	0	2	167	2	168	170	0.99	0.01	0.67	1	0.8
DIED	2	0	10	158	10	160	170	0.99	0.01	0.83	1	0.91
Overall	146	17	367	2870	384	3016	3400	0.95	0.05	0.72	0.96	0.82

Table 8: Performance Measures generated by RegexMatcher on the Development Set