Subjectivity Classification of Bengali News Articles

Zubair Abid, 20171076

Abstract: *Disambiguating acronyms is an important task in Computational Linguistics, often as a preprocessing step that needs to be run in order to get accurate results in further analysis on text. This isn't as trivially easy as maintaining a dictionary of expansions as Acronyms are often domain specific and have a high degree of polysemy, with an average of 9.7 [3] expansions per acronym. So, expansion of acronyms in most situations requires disambiguation. In this project we build a system to use the context around the occurence of an acronym and match it with known contexts for different expansions to make an estimation as to the correct expansion intended. Given some text, we get all possible expansions from an existing dictionary, and for each of the terms lookup associated wikipedia articles from an indexed dump for the known context, and then assign scores based on lexical matches. The accuracy of the system increases with larger text (about 100-200 words is sufficient for most cases.), although accurate disambiguation has been achieved with just single lines of text.*

1 Introduction

A constantly increasing amount of digitally available textual information online has opened up multiple opportunities for mining and analysing these texts for a multitude of commercial, governmental, and social applications. **Sentiment Analysis** or, as it is often referred to, **Opinion Mining** is an integral part of this process. By sorting out the target sentiment of a given text, we can minimise a lot of potentially wasted time when attempting analyses at deeper levels. Work in Sentiment Analysis is typically carried out as a *classification problem* at two levels:

1. **Subjectivity classification:** Classifying a text as subjective or objective
2. **Polarity Classification:** Classifying a text as having positive, neutral, or negative opinion

Unfortunately, while a lot of work in the field has been done in English, efforts in vernacular languages like Bengali are sorely lacking. Work that does exist is plagued by issues of much lower accuracies than their counterparts in English. From an extensive review of related work in the field, we concluded that the primary bottleneck is the lack of high-volume publicly available quality annotated data. While rule-based methods were not doing too well because of the rise of inconsistencies that came with the increasing number of rules to account for both better precision and recall, supervised and unsupervised learning methods were not doing too well either - due to the aforementioned lack of quality data publicly available in notable quantities.

In this project, we create a system to try and automatically annotate for sentiment analysis high quantities of Bengali data. We decided to focus on Subjectivity Classification, as it is a more pertinent question when considering newspaper data, as a bulk of newspaper data would be objective, and thus less likely to express polar opinions on content. Using a method found in *THAT PAPER*, we use a combination of

1. A high precision, low recall rule-based classifier to create and initial 'seed' data to be

used as the intial training set for further automated annotation

2. Extraction Pattern Learning to recognise patterns associated with subjective and objective texts
3. A Naive Bayes Classifier
4. Self training the sentence classifier

I have two options for the project. Either present just (1) as the contribution to be used by future researchers to implement (2) and beyond, or I can actually implement (2) and beyond. I'm trying to do the full implementation, but there's no guarantee that I'll succeed. In case I don't remove the other parts and add it to future work, and "why I'm doing this at all"

Acronym disambiguation is not as simple as storing a dictionary of possible expansions and getting output on a given input of acronyms. Ammar et al (2011) [3] mentioned that for english Wikipedia, articles contain an average of 9.7 abbreviations per article, and more than 63% of the articles contain at least one abbreviation. At sentence level, over 27% of sentences, from news articles, were found to contain abbreviations. Hence it is necessary to carry out some level of disambiguation in order to process them. Because many acronyms are domain specific, this requires at the least a basic world knowledge of the specific domain, one that readers (and by extension computers) may not have. However even acronyms that are not domain specific often have multiple expansions that humans are able to disambiguate based on their context of usage.

So we can hypothesize that in disambiguating acronyms, there are two important obvious factors that come into play - word context, and domain knowledge. Other contributors that might come into play are syntactic features.

2 Related Work

A lot of the work done in Acronym Disambiguation has been specific to the clinical and biomedical domain using the Unified Medical Language System (UMLS) (Bodenreider 2004), which is a knowledge-base providing semantic information and ontological relationships that can serve as features for machine learning (Li et al, 2015) [2]. A number of these approaches have focused on supervised machine learning algorithms where a corpus of text is annotated for acronyms and their sense is manually disambiguated, and then this data is used to train models for further prediction.

In more general contexts instead of using a domain specific knowledge base like UMLS, works have used **syntactic features** of languages; such as HaCohen-Kerner, Kass and Peretz (2008) [4] : working with abbreviations in Jewish law domains written in Hebrew. Similar work with the more general task of Word Sense Disambiguation is Martínez, et al (2002) [5]. Coming to other approaches to the problem, Pakhomov, et al (2006) [6] used a combination of **POS tags, unigrams, and bigrams** as training features for Machine Learning algorithms like the naïve Bayes classifier, decision trees and Support Vector Machines (SVMs). More recent work has focused on approaches with language modelling and entropy based approaches.

Context clues are a very important aspects in arriving at word meaning (A.S. Artley, 1943) and it is thus no surprise to see them being used extensively for a lot of current experiments in Acronym Disambiguation, or Word Sense Disambiguation in general. A lot of recent work in the field is based heavily on Neural Networks and vector representations like Word2Vec. Approaches using word embeddings like TF-IDF or SBE like Li et al (2015) [2] and Turtel, Shasha (2015) [1] essentially use them as a modelling tool to mathematically compute the similarity or "relatedness" of two texts, enabling matching against similar contexts for various known expansions of an acronym. Both papers use learning algorithms for the disambiguation method with varying degrees of success; [1] uses a multiclass linear Support Vector Machine (SVM) on TF-IDF word frequencies to compute similarity between possible expansions and their contexts (acquired from a Wikipedia dump) and chooses an optimal solution to get an accuracy

of 88.6%. [2] correlates word embeddings with a vector space model to represent a word in vector space, and then adds an additional output layer to the architecture based on (Mikolov et al. 2013), comparing results to get a best accuracy of 94.53% on the ScienceWISE dataset, and 93.10% on the MSH collection.
These two papers, particularly (Turtel, Shasha 2015)[1] serve as primary inspiration for our approach. We attempt to see if

1. Similar accuracy can be achieved without the use of a meaning representation system
2. What extending the context to the whole document, unbiased, does.

3 How our System works

3.1 Searchable Wiki Index

The first step in building the system was creating a searchable index of the Wikipedia dump. The dump in itself is over 50 gigabytes in size and running queries on that would be infeasible. Hence, the indexer uses the porter stemmer to stem the lexical items in titles of each Wikipedia article available in the dump and creates an easy to search for directory structure. To search for a particular query, each word in the query is stemmed and compared against a document map which points to the particular directory in the created index where the required information can be found.The indexed data creates a directory structure where each directory of the form n_ m where n and m are integers between 0 and 26 holds the results for stemmed queries starting with the nth followed by the mth English alphabet. 0 represents a blank case. The index takes about 12 hours to build from the main dump and further queries can be run on the indexed dump itself with a lookup into the created document map.For each item in the index, we have the associated tf-idf score based on the number of articles the item appears in and its frequency of appearance in each article. Now, when we search for something in the dump. We can order our results based on these scores and select a specified number of results.

3.2 Identifying Acronyms

Let's say, we want to process a particular text file. We require all the lexical items in the text file. For this, we tag each word using the averaged perceptron tagger available in the natural language toolkit. We disallow words having certain tags as they are simply functional. We also remove be-verbs and words falling in a predefined set of stop words. After this, the acronym finder module recognizes acronyms given in the text. We consider a capitalized sequence of alphabets as an acronym. Note that words such as "laser" which was once an acronym now exist in the English vocabulary as a common noun. Hence, these words are out of our scope. Also note that in order to redefine the definition of an acronym, only this particular module has to be modified and the entire system need not be affected. This module gives us a list of all acronyms recognized in the file.Note that, suppose an acronym, say ACM, has appeared twice in the text file, we consider it only once. Hence, in one particular file, we make an implicit assumption that a particular acronym which has made multiple appearances shall map to the same expansion.

3.3 Acronym Expansion and Disambiguation

Now, we pass each acronym into out disambiguator subsystem. The subsystem is further divided into modules. Firstly, one module sends a request to an API provided by 'The Acronym Server' hosted at 'http://acronyms.silmaril.ie/cgi-bin/xaa?'. The API provides us with an XML of the data which we parse using 'etree' available in the 'xml' package to extract the valid expansions of the acronym. Some error handling conditions such as checking for bracketed clauses and null values have been added here to make the system more robust and fault tolerant.Thus, we have a list of possible expansions for an acronym and we focus on a particular one. This is passed into the module for querying on the indexed dump. The expansion is broken up into word units and

stemmed. This collection of units is searched for in the index. The list of descriptions of titles of all Wikipedia pages which contains this collection is returned sorted according to the tf-idf scores. Now, we process the returned list and consider only the lexical items. We stem each lexical item using the Snowball Stemmer available in the Natural Language Toolkit and convert it into a set of comparable units. The lexical items in the text file which is presented by the user are also stemmed in the same manner and a comparable word set is created. We define the confidence score of the expansion as the number of matches in these two sets i.e the cardinality of the set intersection.Note that, in order to modify the parameters for the confidence score, only this module needs to be modified. Hence, if we want to try out some other heuristics, we can do so easily and the processing which happens following this need not be altered at all. Clearly, we can calculate the confidence score for every expansion of a particular acronym and choose the one which has the highest confidence score. We define a mapping from acronym to its selected expansion and create a new document in which we introduce the expansions for recognized acronyms and place them in brackets beside the corresponding acronyms. This document is made available to the user.The time taken by the system is directly proportional to the number of items whose confidence scores are to be found. We were able to speed up the process by loading the document map into the RAM just once instead of loading it before every query. The document map is about 260 megabytes in size and takes time to load into the RAM. Making this a one-time operation led to a huge increase in the speed of the system.

3.4 Future Modifications

In addition to trying out other heuristics for the confidence scores, one can consider entire Wikipedia articles for disambiguation as opposed to just the title descriptions. Note that this would be an extremely heavy operation and would require a lot of computing power.To increase the system speed, one can make operations run in parallel using a pool of threads. The multiprogramming library in Python allows this. We can disambiguate each acronym in parallel and for each disambiguation, the confidence scores for each expansion can also the calculated in parallel. One would need a powerful GPU to see good improvements in speed using this approach.

The Acronym Server in itself can be made more robust. It mainly has acronyms for day to day conversation used in social media platforms and is primarily restricted to the United States. You may not get expansions for a particular political party in India or something which is very domain specific, say CL mapping to Computational Linguistics. The Acronym Server is evolving and one can help them out by submitting acronyms. Hence, in the near future, this system will have much more base data than it does today.

4 Evaluating the system

4.1 Evaluation metric

We define an evaluation metric based on existing literature [2], with some modifications to accomodate for granularity in testing if needed. Evaluation was done at two levels of the pipeline:

1. Acronym Identification Accuracy
2. Acronym Expansion (Disambiguation) Accuracy

This is done to get more accurate measures of disambiguation where the correct option is not possible at all due to issues with identification, helping isolate the results of the disambiguation algorithm from the identification one if need be.

4.1.1 Evaluating Acronym Identification

A test to measure how correctly the Find Acronym stage of the pipeline works, it classifies its results into three categories:

1. **Acronym recognised correctly (AR):** When an acronym exists in the text and

the finder tags it correctly. This includes acronyms that are later not accepted by the expansion API, as it is independent of that stage

2. **Acronym not recognised correctly (ANR):** When an acronym exists in the text and the finder is unable to recognise it (e.g: PwC), or gets the wrong acronym

3. **Non acronym recognised (NAR):** When an acronym does not exist in the text but the finder still tags it (like the 'II' in World War II)

We define the acronym recognition accuracy as

$$\text{Recognition Accuracy} = \frac{AR - (\frac{NAR}{NAR+AR+ANR})}{AR + ANR}$$

4.1.2 Evaluating Acronym Expansion (Disambiguation)

We can calculate accuracy of acronym expansion in different ways, based on factors we wish to consider or eliminate at various stages. In absolute terms,

1. If the acronym was properly identified, the correct expansion was in the list, and the disambiguation gave the correct result, then the result is correct

2. All other cases are wrong

This is not used to get the accuracy of the system as a whole as a large chunk of acronyms do not have correct expansions. To get the actual calculation of the system, we use the counts of correct outputs, incorrect outputs, lack of output for unrecognised acronyms and output for uncalled for acronyms.
As such, the formula to calculate it is:

$$\text{Disambiguation Accuracy} = \frac{correct - \frac{extra}{total+extra}}{total},$$

$$Where$$

$$total = correct + incorrect + unrecognised$$

If we wish to measure just the results of the disambiguator assuming all other steps go fine (or are ignored where necessary), then we can do away with the extra/uncovered acronyms and work with only correct and incorrectness of that stage alone.

$$= \frac{correct}{correct + incorrect}$$

References

[1] Turtel, Benjamin D., Dennis Shasha. *Acronym Disambiguation.* [*2015*].

[2] Li, Chao, Lei Ji, and Jun Yan. *Acronym Disambiguation Using Word Embedding.* [*AAAI. 2015*].

[3] Ahmed, Akram Gaballah, Mohamed Farouk Abdel Hady, Emad Nabil, Amr Badr. *A Language Modeling Approach for Acronym Expansion Disambiguation* . [*LNCS, volume 9041*].

[4] HaCohen-Kerner, Kass, Peretz *Combined one sense disambiguation of abbreviations.* [*ACM. 2008*].

[5] Martínez, David, Eneko Agirre, and Lluís Màrquez. *Syntactic features for high precision word sense disambiguation..* [*Proceedings of the 19th international conference on Computational linguistics-Volume 1. Association for Computational Linguistics, 2002*].

[6] Pakhomov, Serguei, Mahesh Joshi, Ted Pedersen, Christopher G. Chute, . *A Comparative Study of Supervised Learning as Applied to Acronym Expansion in Clinical Reports.* [*AMIA ASP. 2006*].