Final Project

CSCI-GA.2590-001 Natural Language Processing, Spring 2012

Maor Leger, N12970347

ABSTRACT

A document describing the system implemented as part of the final project. The system is a relation tagging system implemented on the partitive corpus

Table of Contents

Simplifying Assumptions

I chose to slightly simplify the task as follows:

1. Only find ARG0, ARG1, ARG2, and ARG3. I chose to ignore ARGM due to the relatively low number of instances in both the training and development data. This is encoded in the scorer
2. Use the SUPPORT tag as a feature rather than predict it directly. This is also encoded in the scorer.

Instructions

Please see the attached README file for all relevant instructions, prerequisites, and folder structure.

Baseline System

My baseline system uses strictly MaxEnt to predict the ARG0, ARG1, ARG2, and ARG3 of a sentence given the PRED. The baseline system is actually a slightly modified version of the system used for the % task in homework 7.

The approach:
Using the training file, train the MaxEnt model using the following features:

1. relationClass = the name of the relation class that the PRED and ARGs belong to.
2. candToken = The actual token examined
3. candTokenPOS = The POS tag of candToken
4. tokenBeforeCand = The token immediately preceding candToken
5. posBeforeCand = The POS tag of tokenBeforeCand
6. tokenAfterCand = The token immediately following candToken
7. posAfterCand = The POS tag of tokenAfterCand
8. tokensBetweenCandPred = The _ joined list of tokens between but not including candToken and predicateToken
9. numberOfTokensBetween = The number of tokens between but not including candToken and predicateToken
10. possBetweenCandPred = The _ joined list of POS tags of the tokensBetweenCandPred
11. existVerbBetweenCandPred = Whether a VB_ POS tag exists between candToken and predicate
12. BIOChunkChain = The _ joined list of the BIO tags of the tokens between and including candToken and predicate
13. ChunkChain = The _ joined list of different phrase chunks between and including candToken and predicate
14. candPredInSameNP = whether the candToken and predicate are in the same NP or not
15. candPredInSamePP = whether the candToken and predicate are in the same PP or not
16. candPredInSameVP = whether the candToken and predicate are in the same VP or not
17. existSupportBetweenCandPred = whether a SUPPORT relation tag is found between candToken and predicate

In the next step, I tag each sentence individually as follows:

1. Extract the features above for each token
2. Call MaxEntPredict.jar to get a probability distribution among the different ARGS = {ARG0, ARG1, ARG2, ARG3, None}
3. Simply assign each ARGi to the token with the highest probability for the corresponding ARGi.

This system thus makes many simplifying assumptions but the first one that needs to be addressed in my project is the assumption that each sentence that contains a PRED will contain exactly one of each ARGi. Of course, we know that the relation class of the sentence can significantly constrain the types of ARGs the PRED token can have, and thus my first experiment would be to limit the type of ARGs that can be assigned for tokens in a sentence.

Results from the baseline system are not impressive, but they at least show me the task CAN be done:

Experiment	Precision	Recall	F-Score
Baseline MaxEnt	0.22	0.46	0.3

Description and analysis of experiments

Selectional Restrictions

My first experiment involved adding some restrictions on the type of ARGs a sentence can have. According to the guidelines:

1. ARG0 can only appear in the SHARE class
2. ARG1 can appear in all classes
3. ARG2 can appear in the following classes:
    a. SHARE
    b. GROUP

Therefore, my first experiment involved ignoring the probabilities of ARG0 and ARG1 if the relation class does not support it. For example, if the class is GROUP I will ignore any probabilities of ARG0.

The way I achieved this was by creating a separate model for each relation class such that each model will only contain the features that belong to that class. This happens automatically since each model will only contain outcomes that are pertinent to that class.

In addition, I experimented with forcing the system to find at least one ARG1 as well as setting thresholds on the minimum probability for each ARG. The results were poor enough that further experimentation was not needed.

Experimenting with feature selection

The bulk of my time was spent experimenting with different features. Most features were taken from (ZHOU, SU and ZHANG) although I did experiment with my own features as well as features from (Sun) and from (Jurafsky and Martin)

Note: all concatenations are done using the underscore (_) character
Features were taken from the following groups:

1. Features of the words
    a. candToken, tokenBeforeCand, tokenAfterCand – the words immediately before, actual, and after the candidate token.
    b. predToken, tokenBeforePred, tokenAfterPred – the words immediately before, actual, and after the predicate token.
    c. predCand – predToken and candToken concatenated.
    d. tokensBetweenCandPred – a concatenated list of the bag-of-words between the candidate and predicate.
    e. numberOfTokensBetween – the number of tokens between the candidate and predicate.
    f. NoTokensBetween – evaluates to True if there are no tokens between candidate and predicate, false otherwise.
    g. WBFL – the only word between when only one word in between.
    h. WBF – the first word in between when at least two words in between.

i. WBL – the last word in between when at least two words in between.
j. WBO – other words in between except first and last words when at least three words in between.
k. bigramsBetweenCandPred – a concatenated list of the bag-of-bigrams between.
l. stemmedTokensBetween – using NLTK's built-in PorterStemmer a concatenated list of stemmed versions of the words between.
m. stemmedBigramsBetween – using NLTK's built-in PorterStemmer a concatenated list of the bag-of-bigrams of stemmed versions of the words between.

2. Features of the Named Entities using NLTK's built-in classifier
    a. candNETag – the named entity tag of the candidate if it exists, 'None' otherwise.
    b. predNETag – the named entity tag of the predicate if it exists, 'None' otherwise.
    c. candPredNETags – a concatenation of candNETag and predNETag.
3. Features of the BIO chunks
    a. BIOChunkChain – a concatenated list of the BIO chunk tags between.
    b. ChunkChain – a concatenated list of the base phrase chunks between.
    c. candPredInSameNP – true if candidate and predicate are in the same NP, false otherwise
    d. candPredInSamePP - true if candidate and predicate are in the same PP, false otherwise
    e. candPredInSameVP - true if candidate and predicate are in the same VP, false otherwise
4. Features of the POS tags
    a. posBeforeCand, candTokenPOS, posAfterCand – the POS tags of the words immediately preceding, actual token, and immediately following candidate token
    b. posBeforePred, predTokenPOS, posAfterPred – the POS tags of the words immediately preceding, actual token, and immediately following predicate token
    c. possBetweenCandPred – a concatenated list of the bag-of-POS-tags between.
    d. existVerbBetweenCandPred – whether a token whose POS tag begins with 'VB' exists between candidate and predicate
5. Features using semantic resources (WordNet)
    a. candSynsetsLexTypes – a concatenated list of the lexical types of all synsets of the candidate token whose POS matches the candidate's POS, or 'None if there are none.
        i. For example: a candidate token that is a Noun will have a concatenated list of all synsets of the candidate that are also Nouns
    b. predSynsetsLexTypes – a concatenated list of the lexical types of all synsets of the predicate token whose POS matches the candidate's POS, or 'None' if there are none.

c. predCandSynsetsLexTypes – a concatenated list of candSynsetsLexTypes and predSynsetsLexTypes
d. predCandPathSimilarity – the path similarity between predicate and candidate tokens
e. predCandwupSimilarity – the Wu & Palmer similarity score of the predicate and candidate.
f. lowerCommonHypernym – the lowest common hypernym to the predicate and candidate
g. Note: For d-f I needed to choose one synset from possibly many synsets for the candidate and predicate. Sense disambiguation is a complicated task and beyond the scope of this project. However, since WordNet naturally orders the synsets by common usage a simple heuristic was to use the first synset for each token, noting that the synsets were already filtered to those that have the same POS group as each token.

6. Miscellaneous features
   a. relationClass – the actual class of the relation is used as a feature
   b. existSupportBetweenCandPred – true if a token tagged as SUPPORT exists between candidate and predicate tokens, false otherwise

Making the case for relation extraction as a sequence labeling problem

For the next experiment, I wanted to see if I cast this problem as a sequence labeling problem. That is, rather than simply selecting the tag with the highest probability for each token, I wanted to see the impact of using a Viterbi decoder to find the most likely sequence of tags.

The list of states was: {<s>, ARG0, ARG1, ARG2, ARG3, NONE, PRED, SUPPORT, <qF>}

I then used the MEMM model to find the most likely sequence of tags (except for PRED and SUPPORT, which I gave a 1.0 Viterbi value for the corresponding tokens)

Before I began experimenting, I had to add one additional feature in order to convert my model to MEMM:
MEMMTagGuess – the chosen state of the previous token. When training, this was extracted directly from the input. When decoding, this was decided after decoding each token (and sub sequentially used in the next token)

First Attempt

The first attempt yielded generally negative results – while the precision was high, recall was very low. I speculate that this is due to the fact that most of the states were labeled as None thus making None the most likely next state. Note: when running my experiments again to generate the results table, I did not see the major drop-off in recall I observed the first time. Perhaps I added more features in between attempts.

Second Attempt

In order to improve my results, I decided to augment my MEMM decoder as follows: I assumed that each sentence that has a PRED has at least one ARG1. If the MEMM did not label a single token as ARG1, the algorithm "backs up" into a simple MaxEnt model and uses those results as the final tags. (Bird, Klein and Loper)

The results of the second attempt were very similar if not slightly worse than a simple MaxEnt model. Alas, my attempt to fit a round peg into a square hole failed and I returned to a simple MaxEnt model for my experiment.

Analysis of results

Experiment	Precision	Recall	F-Score
Baseline MaxEnt	0.22	0.46	0.30
+ Selectional Restrictions	0.38	0.59	0.46
+ additional word features*	0.76	0.48	0.59
+ additional POS features*	0.85	0.53	0.65
+ additional Base Phrase Chunks features*	0.87	0.56	0.68
+ Named Entity features	0.88	0.56	0.68
+ WordNet features	0.91	0.56	0.70
+ MEMM Tagger	0.91	0.54	0.67
+ MEMM Tagger backing up to MaxEnt	0.91	0.56	0.70

* features that were not part of the baseline system.

Final System Description

My final system consists of a MaxEnt model using Ang Sun's MaxEnt wrapper, NLTK's WordNet, PorterStemmer and ne_chunk.
The system is trained on the training file by extracting positive and negative samples.

Each relation class will have its own class.dat, and classModel.txt files which will be used as input parameters to Ang's wrapper.

The positive samples are extracted for all tokens that were labeled as ARG0, ARG1, ARG2, or ARG3.
The negative samples are extracted for all tokens that have one of the following POS labels:
'NN', 'NNS', 'NNP', 'VBD', 'PRP$', 'JJ', 'IN', 'VB', 'VBN', 'VBG', 'VBZ', 'CD', 'PRP', 'NNPS', 'VBP', 'DT', 'JJR', 'WP$', 'WP', 'RBR', 'RB', 'JJS', 'WDT', 'TO', '$', 'WRB', '``'
The reason of course is that these are the only POS tags that have a token labeled as one of the ARGs in the training file.

For each token, the system gets the MaxEnt prediction probabilities and selects the prediction with the highest probability.

Comparison with previous work

(ZHOU, SU and ZHANG) clearly form the basis for the feature selection and in fact, most of my feature ideas were derived from that.

(Jurafsky and Martin) also describes a set of features that proved useful and I used some of their features (e.g. bigrams).

My baseline system derived directly from homework 7 in which we used mostly Ang's features.

Clearly, there isn't much work on using MEMM's for relation extraction and the results of my experiments show that indeed, relation extraction is not a sequence labeling problem.

Overall, I felt my system performed fairly well, achieving very high precision but also showing modest recall, which I would have loved to improve further upon given extra time. Perhaps future experiments will involve balancing the precision and recall a bit by generating less negative samples and adding more features.

Finally, I used (Bird, Klein and Loper) to understand how to use NLTK for NLP.

Bibliography

Bird, Steven, Ewan Klein and Edward Loper. Natural Language Processing with Python. O'Reilly Media, 2009.

Jurafsky, Daniel and James H. Martin. Speech and Language Processing. 2nd Edition. n.d.

Sun, Ang. "Extracting Arguments for %." Natural Language Processing. <http://cs.nyu.edu/courses/spring12/CSCI-GA.2590-001/NLP_HW7_specs.pdf>.

ZHOU, GuoDong, et al. "Exploring Various Knowledge in Relation Extraction." <http://acl.ldc.upenn.edu/P/P05/P05-1053.pdf>.