CS 585 Homework 4: Coreference

November 22, 2015

Submission Instructions

Your submission will be a mix of a text writeup and code. Upload a zip file with:

- your writeup (pdf format)
- code — we will only look at *coref.py* unless directed otherwise. But you shouldn't have to change anything else (except for the last extra credit question)

In your text writeup, include your name and collaborators list at the top as per the usual course policy.

This homework is 50 points total. It also includes 35 extra credit points, which can be used to make up for scores on previous assignments.

It is due Friday, Dec 4 (at any time, or later night).

1 Introduction

1.1 Overview

You will implement and analyze a rule-based coreference system.

Specifically, it will be an antecedent selection system, similar to what was described in lecture. The algorithm loops through mentions in the document. For the nth mention, it makes an antecedent decision. There are n possible outcomes (n-way classification): either one of the $n-1$ previous mentions, or else it decides to not attach to anything mentioned previously.

1.2 Evaluating Coref Performance

There are a number of ways to evaluate the performance of a coreference system. The provided code computes 'pairwise' precision and recall, which doesn't look just at antecedent decisions, but instead looks at all pairs of mentions in the same cluster. You'll

see there's only one gold-standard link in small.json (the he-him link). There's 55,753 gold links in the test set.

At times, we will want to compare the precision vs. recall of a given approach. At other times, however, we'll want a single number to summarize performance. The code computes precision, recall, and the 'F1' measure (http://en.wikipedia.org/wiki/F1_score).

1.3 Code

We provide the files

- coref.py – the code that runs coref. modify this one.
- corefutil.py – do not modify (library support)
- view.py – a utility that converts coref data into an HTML format
- mentiontest.py – to help debug your code. may be useful to modify but do not submit.
- ml_coref.py, perc.py – only for extra credit ML-based coref
- samples/ – a few small documents for quick testing and viewing.

At the bottom of corefutil.py, we provide helper functions that may be useful when designing features and when debugging. To understand the document and mention datastructures, see how they are created in convertJsonDocIntoMyDoc, or see how they are used in coref.py. We also provided a script, mentiontest.py, which does not do any coreference, but instead just loops through the data structures and prints them out. This might help make it clear what these data structures look like. You can modify it to help debug your mention analysis functions. Do not submit your mentiontest.py. You can run it with something like

```
python mentiontest.py samples/small.json
```

coref.py is split into four parts. The first three are for code that you will complete in Sections 2 and 4 below. The final part of coref.py provides a main function and driver code for running different kinds of coreference. Each of these will train a model (if necessary), perform coreference on the test set, and print an F1 score. We provide a commandline interface. Here are some examples of how to use it (assuming you've downloaded and unzipped corefdata.zip as described further below):

```
# Test the rule-based coref system on a small example file (for debugging)
python coref.py samples/small.json
```

```
# Test the rule-based coref system on one doc
python coref.py samples/bobstone.json

# Test it on the full test set
python coref.py corefdata/test.jsons
```

Finally, note that the main function defines a 'verbose' flag at the top level. Toggle this to print out lots of output. This is useful for debugging, error analysis, and feature engineering.

2 Look at the Data

We provide a few small examples of data in the 'samples/' directory.

You should always test things first on a very small example like small.json. Look at the file to see what's in it: it's just two sentences, and there's only one non-singleton entity. small.html contains an HTML version of this document, produced by view.py.

The main dataset is from the file 'corefdata.zip', available on the Piazza resources page. It's a dataset of coref-annotated documents from the CoNLL-2012 competition (http://conll.cemantix.org/2012/). This data defines mentions as phrases (token spans), and for each mention has an entity ID. It also contains POS tags, NER tags, and parses. We'll only use the POS tags to keep things simple.

test.json contains the test data. There is no training data since this is a rule-based system. (Ignore train.json)

We ran the first 5 documents through view.py to create an HTML version viewable as 'samples/test_first_5.html' in the zip file. These are gold standard annotations.

Question 2.1. *[5 points]* A good thing to do with a new dataset is read it a little bit.

In test_first_5.html, go to the document wsj_1504 and read the first several sentences of the Bob Stone story (and learn how you improve your own corporation's governance procedures). Explain in English what the distinction you think the annotators were making when they said e12 and e22 are different entities. (e22 first appears in sentence S2.) Do you agree or disagree, and why?

3 Mention Analysis Implementation

The following code should be completed in coref.py.

Question 3.1. *[10 points]* Implement the isPronoun(m), isProper(m), and isPlural(m) mention attribute functions.

A good way to do this is to use POS tags that came from a POS tagger. (Actually the tags in this data might be gold-standard, so that's perhaps overly optimistic.) Implementing these will require using the headTokenPOSTag() function in corefutil.py.

Note that the POS tags are in the Penn Treebank format. Find the PTB tagset documentation online. You'll see which tags correspond to pronouns, proper nouns, and plural nouns. You might also have to hard-code some very small pronoun wordlists for plural pronouns, because the PTB tagset doesn't distinguish grammatical number for pronouns.

Please write the implementations of these functions within coref.py where they're specified. However, for debugging, we suggest you call them from mentiontest.py (see the comments in there). You can run it on the sample file with:

```
python mentiontest.py small.json
```

Please remember:

- isProper should always be false for a pronoun.
- isPlural needs to work for both pronouns and nouns (POS tags for the latter might help).

4 Rule-Based Coreference

In this section, you will implement and test a rule-based system. The algorithm is as follows:

- Assume a window size K, which means you will look at the last K mentions as antecedent candidates. For example, $K = 5$ is the default.
- For each of these candidate mentions, use a filter to accept or reject them.
- Of the accepted candidates, choose the closest one as the antecedent. If none were accepted, choose a null antecedent.

We've implemented this in doRuleCoref. It calls isAcceptableAntecedent for the accept/reject filter. You only need to implement isAcceptableAntecedent.

4.1 Implementation

The version of isAcceptableAntecedent that we provide always returns False. Therefore, it refuses to ever link a mention to any candidate. Run it on the test data (or the smaller example files) and confirm that it should predict 0 links, yielding 100% precision but 0 recall. You should get the following results out of the box.

```
$ python coref.py samples/small.json
Pairwise Prec = 1.000 (0/0), Rec = 0.000 (0/1), F1 = 0.000

$ python coref.py corefdata/test.jsons
Pairwise Prec = 1.000 (0/0), Rec = 0.000 (0/55753), F1 = 0.000
```

Now play around with some different ways to implement isAcceptableAntecedent. For example, you could choose to: only resolve pronoun mentions; or only resolve pronouns to a candidate that is a non-pronoun; etc. For at least one of your versions, enforce some sort of grammatical agreement constraint: for example, only link mentions if their number (isPlural) attribute agrees. (The Extra credit section notes more things you can try.)

Question 4.1. *[20 points]* Implement three different versions of isAcceptableAntecedent, with different sets of rules.

In coref.py, provide each of them in its own function: rule1, rule2, rule3.

- One of the rules should only try to resolve pronouns to each other.
- One of the rules must try to resolve pronouns with non-pronouns.
- Over all the rules, use all of the mention analysis functions you just implemented at least once (you don't need to use all of them in every rule)

In your writeup, describe these three different rule choices. Provide the precision and recall for each, evaluated on the entire test set (corefdata/test.json). Discuss how different features help or don't help. Describe which rule system is best and give your reasoning.

5 Trading Off Precision and Recall

Our system evaluates the accuracy of *pairwise* linking decisions. This evaluation cares about correctly predicting coreferent pairs of mentions. It checks all pairs of predicted links (all pairs of mentions in the same predicted cluster), and also checks all pairs of ground-truth links (all pairs of mentions in the same ground-truth cluster). It calculates precision and recall for these pairwise linking decisions.

Question 5.1. *[2 points]* Explain in words and math, what precision and recall are.

Question 5.2. *[3 points]* Explain a context where you might prefer high-precision coreference decisions. When would you prefer high-recall coreference?

(Hint: think about different NLP applications like web search, machine translation, dialogue understanding, etc.)

Question 5.3. *[10 points]* In the top level main function, we have:

```
testSettings['windowWidth'] = 5
```

This parameter controls how many candidate antecedents the system looks at. It potentially can tradeoff precision against recall.

For each of your three rule systems, run an experiment of sweeping through 10 different windowWidth parameters. Create a precision-recall plot of the results. It will have three curves.

- X-axis: recall
- Y-axis: precision
- Each point is one run, from one rule system at one windowWidth setting.
- Please connect a line between points corresponding to runs of the same rule. Therefore there will be three separate lines.

6 Extra Credit

Question 6.1. EXTRA CREDIT *[5 points]*

You'll notice there's a bug in the head token finding algorithm. It's OK for "Adj Noun Noun", but it does "Noun Prep Noun" incorrectly. For extra credit, write a new head token finder and use it in your implementations. Please put it within coref.py so we can find it easily. Describe it in your writeup. Report its effect on coreference accuracy (if any).

Question 6.2. EXTRA CREDIT (up to *[10 points]*): Try to make a better rule system with more advanced features. Implement it in a function called "extraCreditRules". Describe it in your writeup and report its results. Ideas to try:

- Gender analysis, between pronouns (easy)
- Gender analysis, between a pronoun and a non-pronoun (harder: requires getting gendered word/name lists from the internet)
- String match between the mentions. String match of head token.

Question 6.3. EXTRA CREDIT (up to *[20 points]*): Implement a machine learning-based system, based on perceptron learning for antecedent selection. Add this to ml_coref.py (which uses perc.py) and submit your ml_coref.py, and describe the features you created and their results.

ml_coref.py includes both rule-based and ML-based systems and you run it on the commandline specifying which; see "python ml_coref.py -h". You will want to add features to the feature extraction function. Note that the perceptron implementation we give here works slightly differently than the HW2 one; in particular, you don't have to append the class label to the feature name. For example, to implement a grammatical number matching feature, it might look like

```
key = "Plural:%s-%s" % (isPlural(curMent),isPlural(candMent))
features[key] = 1
```

meaning "Plural:True-True" that both the current and candidate are plural, "Plural:True-False" meaning the current is plural but the candidate is not, etc.