Cathy Walsh
CSC 665, Final Project
Due Date: 05/02/2017

Coreference Resolution using Latent Semantic Analysis

Introduction

For my project, I attempted to use latent semantic analysis for coreference resolution. My assumption was that coreferent entity mentions would occur in similar contexts within the text data. I set up a matrix with entity mentions as columns and ten-word grams from the text files as the rows. Each cell in the matrix was a 0 or 1 value, 1 for "entity occurs in the ten-gram" and 0 for "entity does not occur in the ten-gram". I calculated the SVD of the count matrix and used these vectors to calculate similarities between entities with the expectation that coreferent entity mentions would have high similarity scores.

Data and Preprocessing

The data comes from the WikiCoref coreference data set. There are thirty text files that each contain the text from a different Wikipedia topic. Each document contains several groups of entities that refer to one another within the document. Each text file has a corresponding html file containing the entities and their coreferent mentions. I used the BeautifulSoup Python package to extract the groups of coreferent entities from each html file, and create a text file with a name that matched the corresponding text file.

A fundamental aspect of coreference resolution is distinguishing between mentions that are orthographically identical but refer to different entities. When parsing the groups, I needed to create a way to distinguish between identical entities. For example, "him" in the Barack Obama document could refer to Obama, or Joe Biden, or any other male person in the Barack Obama Wikipedia article. "Him" could refer to any entity in the entire corpus. One solution is to remove pronouns, or only use unique instances of entities, but this defeats the purpose and goal of coreference resolution. In order to identify unique entities, I created an entity id for each entity in the corpus. For each document, all entities had a "document name id" that matched the first four letter of the corresponding text file. Each group of entities had a unique numerical entity id for each group of coreferent mentions. This way, the only entities that have identical ids are those that corefer.

The context matrix will still count all instances of words that are orthographically identical. False positives will occur with orthographically identical words, but they will be easily identified as such by examining the entity ids. Only entities with matching ids should be counted as coreferent.

To create the ten-gram word contexts, I concatenated all thirty text files into one text file, and used the Python NLTK library to lowercase the text and create ten-word ngrams. As a result there are some "contexts" in the matrix that do not exist in the original texts, that are a result of ngrams that occur between documents.

Extracting Entities from the HTML Files

```
def get_entities(textf):
```

```
    """Uses beatiful soup to parse an html file that contains each entity set in a
        table
    finds all of the 'table' itmems and pulls the table data containing the entities
    and appends each entity set to a list writes the contents of each entity list to a
    file with the same topic name, with '_Table.html' removed and replaced with
        '_entities.txt'"""
    html = read_doc(textf)
    soup = BeautifulSoup(html, 'html.parser')
    entities = []
    for tab in list(soup.find_all('table')):
        td = tab.find_all('td')
        entities.append(td)
    efname = textf.strip('_Table.html')+'_entities.txt'
    o = open(efname, 'w')
    for i, entry in enumerate(entities):
        entity_list = []
        for item in list(entities[i]):
            if not (str(item.contents[0])).isdigit():
                entity_list.append((str(item.contents[0])))
        estr = '\t'.join(entity_list)
        o.write(estr +'\n\n')
    o.close()
```

Making the Entity ID Triples

```
def entity_triples(enfile):
    """reads an entity file and for each entity in a set of coreferent entities,
    creates a triple with (entity, docID, entityID) where the docID is the first
    four characters of the name of the document, and the entity ID is the number
    corresponding to the order in which that group of entities appears in the file
    this way, the only entities that have the same IDs are the ones that corefer to
    one another. for example if the first set of entities in document "Elephants.txt"
    is "elephant, he, the elephant" the triples will be: (elephant, Elep, 0),
    (he, Elep, 0), and (the elephant, Elep, 0)"""
    fname = path.basename(enfile).strip('.txt') + "_triples.txt"
    o = open(fname, 'w')
    docID = path.basename(enfile)[:4]
    entID = 0

    entities = [es for es in read_doc(enfile).split('\n\n') if es != '']
    doc_ents = []
    for ent_set in entities:
        co_ents = [e for e in ent_set.split('\t')]
        for co in co_ents:
            etrp = tuple([co.lower(), docID, entID])
            doc_ents.append(etrp)
            #o.write(co.lower() + '\t' + docID + '\t' + str(entID) + '\n')
        entID+=1
        o.write('\n')
    o.close()
    return doc_ents
```

Constructing the Co-occurrence Matrix

I constructed a sparse matrix matrix with the entities and their ids as the columns and the ten-gram contexts as the rows. The resulting matrix has shape (53692, 6791) and size 364,622,372. I used the NumPy linalg library to calculate the SVD of this matrix and represented it as a pandas dataframe so that I could retrieve columns by entity name and id.

Analysis

It would take more time and memory than I have available to compare every single entity to every single other entity, and this would be really difficult to evaluate. In order to cut down the comparisons and evaluation to a manageable size, I selected a list of "lead entities" from each group of coreferent mentions, and compared each of these leaders to all other vectors to find similar entity mentions. For each document, I selected the first mention from each group of entities as the leader entity for that group. I constructed a Python dictionary with the document names as the keys, and the list of leader entities as the values.

There are 1408 lead entities and 6791 entities in the matrix, so the entire prediction process makes 9,561,728 calculations. For each leader entity in a document, I calculated a similarity score with all other entity vectors in the co-occurrence matrix. I used inverse euclidean distance to calculate the similarity score. For a leader entity vector *leader* calculate the similarity to each vector in the matrix $evect_0, ...evect_n$

$$similarity = 1/(1 + euclidean\_distance(leader, evect_i))$$

If the similarity score was greater than or equal to 0.995, I added the to the "is coreferent list" for the leader entity. I chose a high similarity score to keep the number of similar entities for each leader to a manageable minimum. With similarity thresholds of 0.85 or 0.9 each leader entity had too many similar entities to make evaluation feasible. I only added similar entities if it did not already appear in the list, as entities with identical ids will always occur in the same contexts. I repeated this process for each document in the corpus so that I had coreference predictions for each leader entity in each document.

Finding Similarities and Corefs for each Document

```
triples = glob(gold_triples)
def get_leaders(trpf):
    leaders = []
    tf = open(trpf, 'r')
    ent_groups = tf.read().split('\n\n')
    tf.close()
    for group in ent_groups:
        group = group.split('\n')
        leader = group[0].split('\t')
        leader_fmt = str(tuple([str(leader[0]), str(leader[1]), int(leader[2])]))
        leaders.append(leader_fmt)
    return leaders

def all_leaders(edirectory):
    leader_dict = {}
    for f in edirectory:
        fleads = get_leaders(f)
```

```
        docname = path.basename(f).strip('_entities_triples.txt')
        leader_dict.update({docname:fleads})
    return leader_dict

lead_entities = all_leaders(triples)
```

Evaluation

In order to evaluate my predictions, I looked at each document with predicted entities and looked at the groups of predicted entities. If the document and entity IDs of the predictions in each matched the document and entity ID of the leader entity, it was a match and counted toward the overall coreference success score. I did not detect any coreferential entities.

Counting Correct Coreferences

```
"""
Look at each of the prediction files, if the predictions in a group match the
    document and entity ID of the lead
entity, this is a corectly identified coreference, print: "match!"
"""
output = glob(eval_files)
total_correct = 0
for filepath in output:
    o = open(filepath, 'r')
    groups = o.read().split('\n\n')
    o.close()
    print(path.basename(filepath).strip("_predict.txt"))
    for group in groups:
        guesses = group.split('\n')
        lead = guesses[0]
        if len(lead.split('\t'))==3:
            entID = lead.split('\t')[2]
            docID = lead.split('\t')[1]
            print(lead)
            if len(guesses) > 1:
                for guess in guesses[1:]:
                    if len(guess.split('\t')) == 3:
                        guessENT = guess.split('\t')[2]
                        guessDOC = guess.split('\t')[1]
                        if guessID==guessENT and docID==guessDOC:
                            print(guess)
                            print("match!")
                            print('\n')
print('\n')
```

Conclusion

This method was unsuccessful in identifying entity mentions that corefer. For most of the "lead entities" the entity vectors with high similarity were not coreferential. Many of the entities identified as similar to their leads began with stop words, especially "the" and "these". If I were to try coreference resolution using context similarity again, I would remove stop words

from the entity mentions. It is possible that the presence of stop words skewed the similarities so that entities had high similarity scores because they began with "the..." or "these...".

Another problem was that even with a very high similarity threshold there were more predicted coreferential entities for each entity group than actually occurred in the data. I also did not attempt to evaluate cross-document entity coreferences. And this was technically not necessary because each document was about a different topic. However, the co-occurrence matrix included information on all entities and all context, both within document and cross-document. I only evaluated with-in document entities, but the source documents were Wikipedia articles, which makes it unlikely that there could be enough repetition for the entities to appear in similar contexts.

References

[A. and Langlais(2016)] Ghaddar A. and P. Langlais. 2016. WikiCoref: An English Coreference-annotated Corpus of Wikipedia Articles. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. European Language Resources Association (ELRA).

[Bird et al.(2009)Bird, Klein, and Loper] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media.

[Pedregosa et al.(2011)Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

[Plank and Moschitti(2013)] Barbara Plank and Alessandro Moschitti. 2013. Embedding semantic similarity in tree kernels for domain adaptation of relation extraction. ACL, ACL Anthology.

[Sahlgren(2006)] Magnus Sahlgren. 2006. *The Word-Space Model, Using distributional analysis to represent syntagmatic and paradigmatic relations between words in high-dimensional vector spaces*. Ph.D. thesis, Stockholm University, Drottning Kristinas vg 53B 114 28 Stockholm.

[Zelaia et al.(2015)Zelaia, Arregi, and Sierra] Ana Zelaia, Olatz Arregi, and Basilio Sierra. 2015. Combining Singular Value Decomposition and a multi-classifier: A new approach to support coreference resolution. *Engineering Applications of Artificial Intelligence* 46:279–286.

,