CIS 530 Fall 2014 Homework 1

Instructor: Ani Nenkova
Head TA: Jessy Li

Released: September 10, 2014
Due: 11:59PM September 24, 2014

Overview

The goal of this assignment is gain first hand experience with VSM for computing document similarity and representing semantic information. The corpus for the assignment will be fixed. We will look at two different possibilities for determining the dimensions of the vector space: all words in a corpus and clusters of similar words. For the "all words representation", we will also compare binary representations, tf.idf weights and mutual information for determining the coordinates of an object. We will perform a simple evaluation to compare to what extent the different choices lead to different solutions and to decide which solution is best. The task is slightly artificial but useful for analysis of the concepts covered in class so far.

To run your code, please use the Biglab machines. Large jobs will be automatically terminated on Eniac. For instruction on how to use Biglab, please refer to the link provided below:

`http://www.seas.upenn.edu/cets/answers/biglab.html`

Submitting your work

The assignment should be done individually, so each student should make one submission with their own code for the assignment. The code for your assignment should be placed in a **single file** named `hw1_code_yourpennkey.py`, where `yourpennkey` is the first part of your Penn email address. For this assignment you will submit three other files: `startucks_tfidf_weights.txt`, `starbucks_mi_weights.txt` and `results.txt`. Put the files together in a zip file called `hw1_yourpennkey.zip`.

All submissions will be electronic, done from your Eniac account. You need to copy your files to Eniac if you choose to work elsewhere. To copy files you can use a SFTP client such as FileZilla, WinSCP or Cyberduck, or secure copy `% scp local_file yourpennkey@eniac.seas.upenn.edu:PATH` where `local_file` is the path to your homework on the local machine and PATH is the path to the location on Eniac where you wish to copy it.

Once you have the file in place in your Eniac account, connect via ssh to `seas.upenn.edu` and use the `turnin` command to submit your files for grading:

```
% turnin -c cis530 -p hw1 hw1_yourpennkey.zip
```

You will get a confirmation message. You can run `turnin` multiple times before the deadline. Each time you run `turnin`, it overwrites your previous submission for that assignment. You can check that the homework was submitted successfully:

```
% turnin -c cis530 -v
```

This will show you the list of files you have submitted.

Code Guidelines

In the assignment below we specify a number of functions. Your submission should include implementation for each of these required functions. In addition, you may write any extra functions that you find necessary. Avoid doing any preprocessing of the input text besides that explicitly described in the assignment. This will make grading easier.

Your code will be graded automatically. It is your responsibility to make sure your function can be called correctly and that they return exactly the type of values specified in the assignment. To make sure that your code will run correctly during grading, please include **only** function declarations in your source code or above the line `if __name__ == "__main__"`, and prefix **all** your global variables with `yourpennkey`.

Data

The data for our assignments was provided by the Wharton Research Data Services. It contains CIQ Key Development news summaries for Russell 1000 companies. For the first homework we will be using data from 10 companies (Starbucks, H.J.Heinz, Qualcomm, etc). The goal would be the create representations to characterize each company and compare new documents based on these characterizations.

You can find the training data for learning "frequency semantics" in the following directory:

`Corpus_root = '/home1/c/cis530/hw1/data/corpus'`

The goal is to learn from the corpus a representation for each company, i.e. companies will be point in multi-dimensional space of words. It is hard to evaluate how good such a representation is outside of an application so we will just ask you to interpret the representation and compare the differences in representations of different companies in order to check if these conform to your intuition.

Intuition however is never an acceptable evaluation. In order to make empirical evaluation possible, we will introduce a simple task: given a set of unseen documents and a target company, find the documents related to the target company. The test data consists of a mix of $1,000$ Key Development summaries for any of the 10 companies that also appear in the training data. The test data resides in:

`Mixed_root = '/home1/c/cis530/hw1/data/mixed'`

1 Loading the corpus (10 points)

All our representations will be learned from corpora, so we first need to read in the words contained in the training corpus files. You will write some simple functions that take a file directory as an argument and stores all words that appear in any of the documents.

a) Write a function `get_all_files(directory)` to list files of any depth within the given `directory`.

Return: a `list` of relative file paths for all files in the input directory.

b) Write a function `load_file_tokens(filepath)` which takes an absolute `filepath` and returns a list of all words that appear in this file. Convert them to lower-case.

Return: a `list` of words (converted to lower-case) in `filepath`.

c) Write a function `load_collection_tokens(directory)` to load the words from all files in `directory`.

Return: a `list` of all tokens (converted to lower-case) from all files in `directory`.

As you start implementing this function, you may notice that the definition of what is a word or token is rather unclear. To make your life simpler, assume a word is any alpha-numeric string surrounded by non alpha-numeric symbols. Take a look at the output to identify undesirable aspects of this representation.

2 Frequency, TF-IDF and mutual information

Now we will use a corpus of summaries for ten different companies to learn a representation for each of the companies. The format of the file names shows the company which is discussed in the summary. In effect we have labeled data, with examples of texts referring to each of the companies we consider.

The goal is to find the words that are most descriptive of each company, so that the presence or absence of the word can be meaningfully interpreted when we analyze new documents from unseen collections, i.e. the presence of the word is more likely to indicate that the document is related to the company.

2.1 TF-IDF (16 points)

When given the task of finding words that characterize the articles about a target company, the first instinctive thought may be to find the words that occur most often in these articles and use them as descriptors. We will refer to this approach as *term frequency* (TF).

Often it is useful to normalize term frequency so that the weights fall in an easy to interpret range, for example [0,1]. Here a word with weight close to 1 is very descriptive.

It is also desirable to be able to compare the descriptive terms for different companies. In these cases if we have more or longer documents for one company than for another, the raw frequency for a word in the company with more textual data will be higher simply by virtue of the larger corpus, not necessarily because of the strength of association between company and term.

A simple normalization procedure addresses both potential issues: all raw counts are divided by the number of occurrences of the most common word in the data for each company. *Normalized term frequency is the times a word appears in a the training corpus divided by the times a most frequent word occurred in the came corpus.*

Inverse document frequency (IDF) is a simple way to introduce further information about words in the procedure for finding descriptive terms. It estimates how often a word appears in an arbitrary document. The idea descriptive words should be frequent in documents about the target company and infrequent in other texts. This intuition is captured in the TF-IDF weighting for words.

a) Write a function `get_tf(itemlist)` to get the normalized term frequency for all terms in `itemlist` (`itemlist` is a `list`).

   Return: a `dictionary` which has the terms in `itemlist` as keys and their corresponding normalized term frequency as values. All values will range between 0 and 1.

b) Document frequency (DF) is the number of documents which contain the term in the corpus for learning word properties. If we denote by N the number of documents in the corpus, inverse document frequency (IDF) is calculated as:

$$IDF(w) = \ln(N/DF(w))$$

   Notice that here we do not calculate how many times a word appears in each document, simply if the word appears in a document or not. Write a function `get_idf(itemlist)` to generate IDF for each term contained in `itemlist`. Here, `itemlist` is a 2-dimensional list, where each list contained in `itemlist` represents the terms in a document. For this problem, IDF values should be calculated from the given `itemlist`.

   Return: a `dictionary` with each term in `itemlist` as keys and the respective term IDF as values. Learning IDF values is a computationally expensive procedure because it requires reading a large corpus. This learning should be done once and the results stored for subsequent calculation of term weights. It is useful to augment this dictionary with one extra artificial word, "$< UNK >$". This would give the IDF for a word that doesn't appear in the corpus for calculating IDF. We can assume such a word appeared once, so its IDF will be equal to $\ln(N)$.

c) We are now ready to compute TF-IDF for each term in articles about a given company: terms with higher TF-IDF would be considered more representative of the company. The IDF of the term should

be calculated once from the larger general collection in `'/home1/c/cis530/hw1/data/all_data/'`. TF should be calculated only on the files for a specific company, with different values for each company in `Corpus_root = '/home1/c/cis530/hw1/data/corpus'`.

Write a function `get_tfidf_top(dict1, dict2, k)`, where `dict1` is a TF dictionary and `dict2` is an IDF dictionary. It should sort all terms by decreasing TF-IDF decreasing and returns the top k sorted terms.

Return: a `list` of `k` terms with highest TF-IDF values, sorted in *decreasing* order.

2.2 Mutual Information (10 points)

The second approach for identifying terms descriptive of a company is pointwise mutual information (MI). Mutual information between a term and a company topic is computed as:

$$MI(topic_i, w) = \ln \frac{p(w|topic_i)}{p(w)}$$

Our goal is to compute mutual information between terms and a company, where each company is represented by the news summaries related to that company. The probabilities will be computes as a ratio of occurrence counts. The conditional probability of a term, $p(w|topic_i)$ will be computed only from the summaries of the given company i. It is equal to the number of times the term occurs in a summary about the company divided by the number of total words in the summaries about the same company. The general probability of the term, $p(w)$ will be computed from all files in the directory `Corpus_root`. It is equal to the number of times the word appeared in any of the summaries, divided by the total number of words in all summaries.

As discussed in the Turney and Pantel paper, MI tends to be high for rare words. Instead of using the techniques for fixing this bias that are discussed in the paper, we will adopt the simple technique of assigning mutual information between a word and a company topic if the word appeared at least 5 times in the company data. Also as discussed in the paper, update the MI values so that any negative value is replaced by 0.

a) Write a function `get_mi_top(bg_terms, topic_terms, k)` to generate the top k terms from `topic_terms` (type `list`), with highest mutual information within `bg_terms` (type `list`).

   Return: a `list` of `k` terms with highest MI values, sorted in *decreasing* order.

b) Here we would want to compute and store representations for a company that we have learnt from `corpus_root`, which consists of words with greater than zero MI weights.

   Write a function `write_mi_weights(directory, outfilename)` to record the MI values for all words within a company directory in `outfilename`. Each line in the file should be formatted as `<word MI-value>` where the word and the MI values are separated by tab. Include the weights file for Starbucks in your final submission (please name it as `starbucks_mi_weights.txt`).

2.3 Comparing representations (16 points)

So far you have implemented three different procedures for finding terms that are descriptive of a company based on co-occurrence information in articles about the company and in a general corpus: normalized term frequency, TF-IDF and mutual information. We have not yet defined criteria with which to evaluate if the words we have identified are a good descriptor or not. We will develop an evaluation in the second half of the assignment. For the moment, we will just verify if the lists of descriptive words we obtain by different methods are actually different. If the descriptive words are the same, it is unlikely that the choice of representation will make any difference for whatever task we have, so this is a good sanity check.

We will use precision, recall and F-measure to compare the lists of the top $k = 100$ descriptive terms for a company derived using normalized TF, TF-IDF and MI. To do this, we will take the 50 words with

highest TF-IDF as the reference list of true descriptive terms, denoted by L_2. We will evaluate the top k terms returned by MI, denoted by L_1, with respect to that reference. Similarly we will compare the list of top k terms (L_3) returned by normalized term frequency. The reference is chosen arbitrarily to reduce the number of computations. In the "Viability of web-derived polarity lexicons" paper, the authors use each of the dictionaries they work with as a reference.

The precision of MI with respect to the TF-IDF reference is defined as the percentage of descriptive words correctly identified by MI in the retrieved result list. Its recall is defined as the percentage of descriptive words correctly identified by MI among the reference list. F-measure is defined as the harmonic mean of precision and recall and is often used to summarize the performance of a system into a single number. Precision, Recall and F-measure could then be calculated as below:

$$Precision = \frac{|L_1 \bigcap L_2|}{|L_1|}$$

$$Recall = \frac{|L_1 \bigcap L_2|}{|L_2|}$$

$$F\text{-}measure = \frac{2 * Precision * Recall}{Precision + Recall}$$

Specifically, write the following three functions, each taking three arguments—a list of words L_1, and a reference list L_2—and returning the precision, recall and F-measure:

`get_precision(L_1, L_2), get_recall(L_1, L_2), get_fmeasure(L_1, L_2)`

Now, compute the precision and recall for the list of top 100 words for Starbucks using MI and normalized term frequency, with the list of top 100 words using TF-IDF as reference. Record the values in the file `result.txt` as two lines, each line of the format `<precision-value>,<recall-value>`. The first line should be the values for MI, and the second line the values for normalized term frequency.

3 Brown clusters as word representation (12 points)

As we have seen in class, lexical frequency semantics can capture surprisingly rich information. However, because the learning is done via counting, the information for many words is not reliable. So far we consistently said that words that appear fewer than five times in the corpus will be ignored. It would be convenient to work with word categories instead. For example if we had a category *days of the week* or *calendar months* we could learn about the category by counting the occurrences each individual member of the category occurs. So we treat clusters of words as a new meta-word. The meta-word occurs in a text whenever any of its members occur. We can now compute TF-IDF for meta-words and use the meta-words for our representation.

Brown clustering is a clustering algorithm to group words together in a hierarchical fashion, such that it maximizes the mutual information between meta-words defined in the clustering over a large corpus. Right now, we will not go into details of how the clustering is done. We will revisit the topic later in class when we talk about language models. We have provided you with a pre-calculated brown cluster at `data/brownwc.txt`.

In this problem, we will replace words used as terms in the previous problem with meta-words corresponding to *word clusters*.

a) First, we need to construct a cluster lookup dictionary for each word type.

Write a function `read_brown_cluster()` that reads the Brown cluster file we have provided and constructs a `dictionary` with keys corresponding to words in the cluster file and cluster ids as values. You don't need to convert cluster ids to integers, just strings are fine.

Return: a `dictionary` with: words (`str`) as keys, and cluster ids (`str`) as values.

b) Write a function `load_file_clusters(filepath, bc_dict)` that takes a brown clustering dictionary `bc_dict` and a `filepath`, and returns a `list` of cluster ids corresponding to each token in `filepath`. Consider only words present in the cluster dictionary `bc_dict`.

Return: a `list` of word cluster ids.

c) Write a function `load_collection_clusters(directory, bc_dict)` that takes a brown clustering dictionary `bc_dict` in addition to `directory`, and returns a `list` of cluster ids corresponding to each word in each file inside `filepath`. Consider only words present in the cluster dictionary `bc_dict`.

Return: a `list` of word cluster ids.

d) Write a function `get_idf_clusters(bc_dict)` that takes a Brown clustering dictionary and returns a dictionary of IDF values from each cluster id under `all_data`.

e) To store this representation of each company, which consists of clusters and their TF-IDF weights, write a function `write_tfidf_weights(directory, outfilename, bc_dict)` to record the TF-IDF values for all clusters within a company directory in `outfilename`. The format should be the same as specified in `write_mi_weights()`. Include the weights file for Starbucks in your final submission (please name it as `starbucks_tfidf_weights.txt`).

4 Computing similarity

Now we will address a task in which the learned representation for each company may be helpful in a specific task. The task is to take a list of new documents and a company and find which documents discuss the company. To solve the task, we will compute the similarity between the company representation and the new documents, and will assume that the documents most similar to the company representation are the documents talking about the company.

4.1 Binary representation (12 points)

First we will represent each document in the collection of so far unseen documents. We will use the meta-word Brown clusters as basis for the representation space. For each document, a component v_i will be equal to 1 only if a word from Brown cluster i appears in the document. Create functions to do the following:

(a) Write a function `create_feature_space(list)` to create a `dictionary` mapping each unique cluster in all documents to a unique integer starting from zero (order doesn't matter). This creates a mapping between each cluster and a vector component corresponding to this cluster.

Return: a `dictionary` whose keys are cluster ids and values are its corresponding vector component.

(b) Write a function `vectorize(feature_space, lst)` which takes a **list** of cluster ids and a feature space as input, and returns a vector $\vec{v}$ where each dimension v_i of that vector is set to 1 if list contains the i^{th} cluster from the feature space, 0 otherwise. Clusters that appear in the list but not in the feature space are simply ignored (you can include them in the representation with value zero, but they will never contribute to any similarity computation).

Return: a list of 0s and 1s.

4.2 Cosine similarity (4 points)

There are a variety of similarity and distance metrics. For detailed discussion, see page 299 of Manning and Schütze, `http://cognet.mit.edu/library/books/view?isbn=0262133601`.

Here we use cosine similarity, one of the similarity metrics most widely used in NLP applications.

$$\text{Cosine:} \frac{|X \cap Y|}{\sqrt{|X| \times |Y|}} = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i \times w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

Write a function `cosine_similarity(X, Y)` to compute the cosine similarity between vector `X` (type: `list`) and vector `Y` (type: `list`). Return 0 if either of the vectors is $\vec{0}$.

Return: a float.

4.3 Most similar documents (20 points)

So far we have implement functionality to represent information that we have learned from corpora about a company and to represent new documents. We will now compute the similarity between a company and a set of so far unseen documents, some of which are about that company and some of which are not.

In the folder `/home1/c/cis530/hw1/data/mixed`, we have put together 1,000 documents which do not appear in the data for individual companies on which we built the representations. Each document is a summary of an event related to one of the 10 corporations. The name of the file shows which is the company discussed in the file. There are 100 summaries about each company. Some of the summaries are actual reports on ongoing events, others are one-line announcements for earnings calls for example.

We calculate the similarity between each document in this test folder to the company representations learned which we learned in the first half of the homework. For example we can calculate the similarity between all new documents and the representation for Starbucks learned from `/home1/c/cis530/hw1/data/corpus/starbucks`. The goal is to identify the most similar documents for a given company.

We will calculate the precision and recall of a method which predicts that the 100 new documents that are most similar to the company are about the company and that all other documents are not about the company. (We choose 100 because we know that in our test data we have exactly that many documents about each of the company. In realistic situation that cut-off would need to be tuned on held out data.)

We will use the two representations that you have saved in problems 2 and 3 for describing what was learned about a company: words weighted with mutual information and Brown clusters weighted by cluster TF-IDF. We will use binary representation in the same space for the documents in the test folder.

a) Write a function `rank_doc_sim(rep_file, method, test_path, bc_dict)` to generate an ordered list from most to least similar files within `test_path` to the document representation description file in `rep_file`, using the cluster dictionary `bc_dict`. (Usually, we wouldn't want to include `bc_dict` as a parameter but rather a global variable. Here we use it this way to facilitate grading). The `method` parameter here is either `'mi'` or `'tfidf'` that tells you which representation of the company you should use for comparisons.

   Return: a `list` of (document-base-name, similarity) tuples, ordered in *decreasing* similarity.

b) Now, using Starbucks as the reference company, record the precision for the top 100 documents from `Mixed_root` for each representation in `results.txt`. The first line should be the precision for MI, and the second line results for TF-IDF on word clusters.