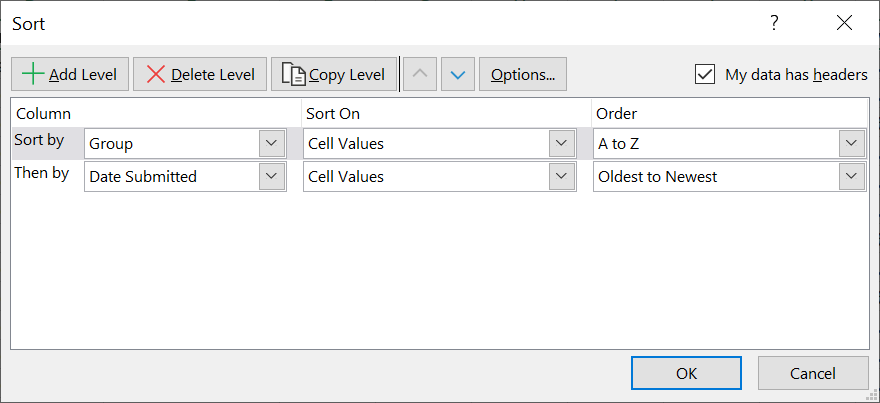
Overview

The eLearning office at Gies College of Business is going to do content analysis of two course called BADM 508 and BADM 590. Each of the course has undergone some changes on the course structure, platform and some other deliverables from year 17 to year 18. To see whether the changes actually work and get more insights from students’ feedback, the project, as part of the whole analysis project, aims at create a tool to extract keywords from the course review so as to have an overview of the feedback quickly.

Data Cleaning

The raw data file contains the survey reports for BADM 508 (SP 17 AND SP 18) Leadership (Loewenstein & Goncalo) and BADM 590 (Fa 17 and 18) Global Strategy (Buchelli) together and collects the students’ demographic information and their feedback on Course Overall Quality, Platform, Assignments, Live Session, Office Hour, Class Support, Sense of Belonging, and their time spent on each deliverables. Since the main goal of this project is to do a text analysis, column U (CombCourseraCompExplain), column AV (SuggestImprove for Class Q&A, Office Hour, Tech Deadline, Support), column BD (Describe Tech&Platform Issues), column CK (Explanation for Statistics) and column CM (ImproveIMBA).

First thing is to separate the spreadsheet into four sheets, each of which stands for one course in one term. To do that, I “sorted” the whole data first by group then by data submitted.



Considering I would like to use Python and NLTK to analyze the text, I extracted the useful columns and saved them as different .txt files. When extracting the texts, I found there are no data saved in columns for Suggestions for IMBA and Stat Explanation.

Methodology

The main language used is Python 3.7 and nltk, genism and sklearn.

1. Download packages

Since in the class we have downloaded Anaconda, I used “conda download nltk” in Anaconda Prompt. After importing nltk, I downloaded punkt, stopwords, wordnet and averaged\_perceptron\_tagger package by using nltk.download(). For the packagage which helps me find the dependency of words, I downloaded Standford CoreNLP 3.9.2 (<https://stanfordnlp.github.io/CoreNLP/download.html>) and the file used for English analysis (stanford-corenlp-3.9.2-models-english.jar).

1. Standford CoreNLP settings

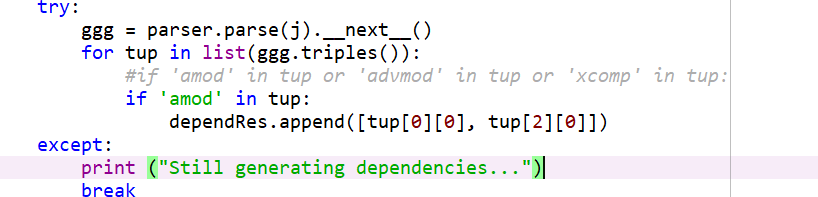
Stanford CoreNLP is an NLP analysis tool written in Java and provide interfaces used by Python. In order to use the dependency analysis function, we must first download java (you may also need to set the global path). We must set the path of Standford CoreNLP to where the file exists (code line 23).

1. Clean the sentences

To generate the tag for sentences, the overall idea is to identify the most frequent word combinations. To do that, extracting the meaningful words is the first step. In the functionpunctuations and symbols are removed from each sentence in the given txt file. With the clean line, I then tokenize the words by using word\_tokenize module in NLTK libriary. Tokenization is the process that divides big quantity of text into smaller parts. It is similar to the .split that we used in class, but I used tokenize here to remove stopwords and for stemming and word tokenization is better for text understanding in machine learning applications in that it converts text into numeric data (Guru99, 2019).

1. Generate dependencies

To find meaningful pattern of words, dependency parsing is crucial as it dives into the grammar structure of the sentence and defines the relationships between “head” words and words which modify those heads (NLP Progress, 2019). Since the data used in this project are all comments and feedbacks, I am looking for patterns like “amod”, “advmod” or “xcomp” according to Standford Dependency (2008), which separately stands for “adjectival modifier”, “adverb modifier” and “open clausal complement”.



After testing, I found that more dependencies lead to more unreasonable tags, so I finally decided to use only ‘amod’ in this situation. The parser will generate a graph data structure including the word combination and relationship (every vertex represents a word and each edge between words represents their relationship) and we will apply the **\_next\_()** and **triples()** methods to generate the word combinations which have ‘amod’ as their relationship. Then we extract 2 words of each return tuple to be our candidate tags.

1. Generate Word2Vec model

As we have a respiratory of several tags from step 4, and the meaning of some tags may be similar. I have to apply some methods to cluster those tags. And I chose to use Word2Vec as it can turn words into vectors, so that I can use vectors of words to do clustering algorithms. The Word2Vec is a great model to generate word vectors. Those words who have similar meaning will finally have less difference in vectors after training. And that is what we want for the next step.

1. Tag Generator (Clustering tags)

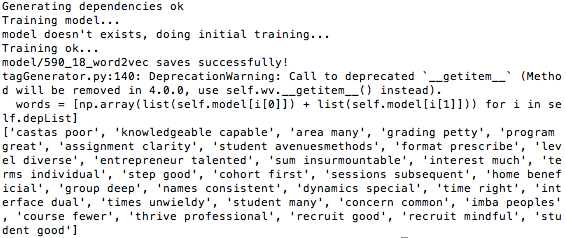
With the word2vec models in hand, the last step is to use DBSCAN as our clustering method to aggregate tags. The goal of this method is to eliminate the duplicate tags and return the tag which can best represents the attributes of the class.

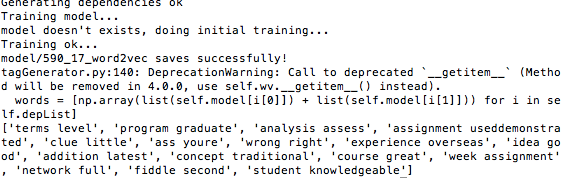
Furthermore, I set a baseline to control the quality of generating tags. The chosen tags must contain words which ranks in top 10 of appearing frequency. In that way, we eliminate most unreasonable tags and make sure the tags have the majority votings.

Results

I only ran the suggestion\_IMBA for both BADM 590 FA18 and FA17, the tags showed hard to understand so I stopped here for further analyzing.

Here are the screenshots of the tag generated from the two courses:

590\_18\_suggestion\_IMBA.txt

590\_17\_suggestion\_IMBA.txt

# Explanation

Although here we used a lot of NLP methods to make sure that our tags will make sense, there may still be some situation that tags are not making sense.

1. DBSCAN parameter

The parameter of DBSCAN decides the clustering of our tags (maybe soft, maybe hard). We have tried a lot of combinations of parameters in order to have the best clustering results. However, the test sample is always changing. So the fixed parameter will perform differently on them and that’s why sometimes unreasonable tags appear.

1. More chosen dependencies in Dependency Generator

In my case, I have only chosen ‘amod’ as my main dependency. However, the true situation is that we may need more dependencies to be able to generate more reasonable tags.

1. Tag format

In my function, tags are always generated in two words, which may have a limitation on expression.

# Future

In the future, I concieve several ways to optimize the model:

1. Adaptive methods of learning parameter

We may apply some dynamic algorithms to choose different parameters for DBSCAN.

1. More testing on dependencies

To see what dependencies and result in more reasonable results, we have to try and

1. Figure out a way for multiple formats of tags

References and Citations

Guru99 (2019). Tokenize Words and Sentences with NLTK. Retrieved May 4, 2019, from <https://www.guru99.com/tokenize-words-sentences-nltk.html>

Dependency Grammars. Retrieved May 4, 2019, from <http://www.nltk.org/howto/dependency.html>

Source code for nltk.parse.dependencygraph. Retrieved May 4, 2019, from <http://www.nltk.org/_modules/nltk/parse/dependencygraph.html>

NLP Progress (2019). Retrieved May 4, 2019, from <http://nlpprogress.com/english/dependency_parsing.html>

De Marneffe, M. C., & Manning, C. D. (2008). *Stanford typed dependencies manual* (pp. 338-345). Technical report, Stanford University.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of machine learning research*, *12*(Oct), 2825-2830.