FInal Project for LING131A

Team members: Amy Wu, Jingdi Shen, Jiexin Cui, Kuan-Yu Chen

I. Goals and Motivations

This project aims to uncover trending topics in computational linguistics research by mining recently published academic papers in top-rated journals of the field. We built a corpus and extracted frequent words, phrases, and acronyms. The goal of this project is to apply what we have learned about NLP with Python and especially the NLTK packages to gain some insights into the field through text mining.

II. Data Collection (Jingdi Shen)

For the purpose of our project, we need to build a reasonably large corpus containing newly-published research papers in the NLP field. The intuitive place to look for them is online achieves of academic journals such as the ones run by the Association of Computational Linguistics.

In my search, we found the Association of Computational Linguistics Anthology (https://aclanthology.coli.uni-saarland.de/) and decided to use this resource because it seems well structured and comprehensive. It also provides bib files which contain metadata about each paper available with matching file names. The first step is to find a way to avoid manually downloading these files from the website. We used command lines in Firefox to do so and the process is conducted in three steps:

a) Save console log as pdf.bat, run it to download all pdf files.

b) Do the same thing with the bib files.

c) Move all pdf files to one folder and add .pdf extension to all files in Python.

III. Data Cleaning (Jiexin Cui)

PDF is like the wild west without any rules, because PDF is more like a graphic representation and the actually content is the instructions telling how to place the exact content in some place or say how to display the stuff.

There are several tools to deal with PDF, such as pypdf and pdfminer. The former one is a tool analyzing the pdf documents like getting the titles, created dates and page numbers. The latter one is to extract the informations. However, because PDF barely have any rules, what the tool can do is just attempts to reconstruct some of those structures by guessing from its positioning. Therefore, there is a great possibility to fail. And my effort is to try to deal with parsing PDF files into ASCII text documents.

Software implementation
First, the function convert make the converter link PDFResourceManager with a output file. And make a PDFInterpreter using the converter. Next, the function takes in a file and for each of its page, convert it into a PDFPage object and interpret/ parse it. Because of the link among output, the converter and interpreter, when the input file is done with the parsing, the output is written up. Second, the convertMultiple is taking every pdf document in a directory and convert it. Then write the output into a new directory.

What is important is that because of the environment limit of pdfminer, we have to run the code under python2 other than python3. Even though, there are pdfminer.six and pdfminer3k for python 3, attempting results are not as good as pdfminer under python 2.

And the way to use the code is to change the path in the bottom of the code according to need. And run the code in the prompt, some error in the file can be ignored.

And because pdf is normally evil, the convert process cannot always be that smooth. In the test cases, there are always files in the two-columns typesetting. The conversion result for these files end up being without blank spaces between tokens. Attempts in rewriting code in different ways have been made, however, without a satisfying result. Thus in order to get a proper set of converted files, the files without space should be picked out from the directory. Therefore, the function *pickout* is written, which remove the file containing words, the length of which is more than 100. In this way, the documents without space can be removed from the directory.

More progress should be made in the future to get a complete data of CL-related corpus on ACL in 2018.

Note
We also used the pdf2txt commands in pdfminer generated in a template to convert files that were not converted in the first trials. There were still some files that failed to be correctly formatted and thus were discarded. In total, we successfully converted 824 out of 1,279 files to be used as data for the current project. (Jingdi Shen)

IV. Corpus analysis (Jingdi Shen, Kuan-Yu Chen)

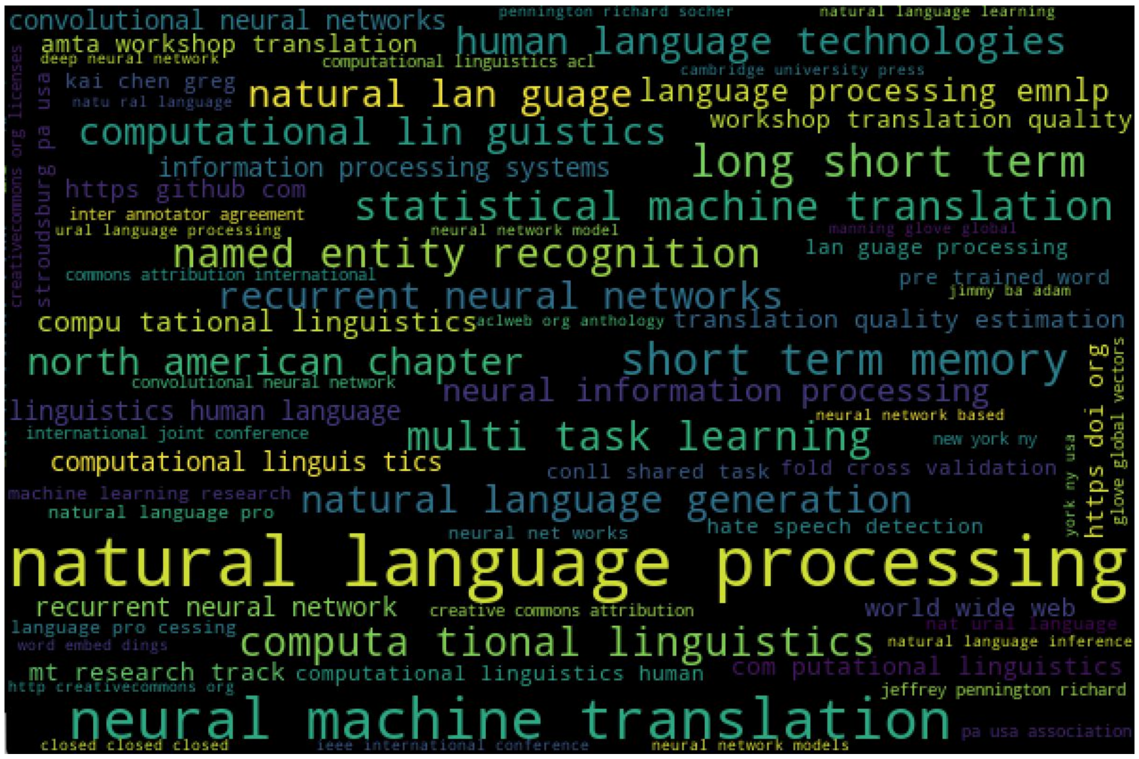
a) Building corpus

We designed the corpus to be a collection of all txt files indexed with file names. It has the basic functions including get files in the directory, tokenize words and sentences, find frequent words and phrases (bigrams and trigrams), concordance, and access and print metadata associated with each file. To achieve the last function, we used a bib package to parse the bib files and save all results in a json file. This file is then loaded in the corpus so user can access the metadata through keywords. The process of building the corpus was quite smooth.

b) Analysis and interpretation

In this part, we tried to make sense of the corpus data by applying some common text mining techniques.

First of all, based on the most common single words, bigrams, and trigrams found from the corpus, we obtained two lists of frequent topics, the frequent words and the frequent phrases (2-3 words). We generated word cloud images for these lists using the wordCloud package. After checking the images, we had to go back to the corpus class to do some additional cleaning in order to improve the the results. These include restricting the length of the word and creating a set of skip words that is combined with the English stop words so each frequent work or each word in a frequent phrase can be filtered.



After these improvements, we can see some interesting results in the word lists and word clouds. As expected, "Natural language processing", "language models", "computational linguistics" are highly frequent terms. Through the single word results we can see that the field is very much empirical-oriented. The most frequent words are 'model', 'data', ‘methods’, 'features', etc. Phrases describing common NLP tasks appear a lot, too, such as 'text classification', 'sentiment analysis', 'word embedding', and 'named entity extraction'. There is also a considerable amount of words associated with machine learning and deep learning techniques, such as 'recurrent neural networks', 'support vector machines', 'neural machine translation' and so on. A few interesting and unexpected words are 'code switching' and 'hate speech', which seem to be topics attracting much research attention right now. It also needs to be pointed out that there are still some problems with the current function, such as the splitting of a word into two nonsense parts (e.g. ('sen', 'tences')) and the inclusion of irrelevant words. Due to time constraint, these remain unsolved.

Second, we tried to do a geographical distribution of the papers using metadata in bib files. Unfortunately, we later found out that there are only very few locations included in these files. Also, these locations are likely places where the ACL conferences were held and in which these papers (conference proceedings) were presented, instead of locations of the authors as we originally thought. Still, it shows that Brussels, Belgium is the location where most of these papers are associated with. This is probably the case because multiple conferences and workshops took place there in the past year.

We also did two experimental analyses, one using the acronym dictionary described below, and one comparing the most frequent words and the most frequent bigrams in the corpus. For the acronym dictionary method, we take all the words in the frequent bigrams and look them up in the acronym dictionary. If there is a match, we simply add it to the result list.

For the other method, we check if each of the word in frequent bigram is also a frequent word by itself. Using this simple comparison, we were able to generate a list of meaningful bigrams that's highly likely a list of trending topics.

Our original plan was to do some further analysis of the geographical distribution of authors, the count of key terms by publishing time, and the organizations of the authors. However, it turned out that we did not have time to extract those information from the raw text. This could be something to work on for future projects now that we have a pretty good dataset to work with.

V. Acronym Dictionary (Kuan-Yu Chen & Amy Wu):

We attempted to create a dictionary of acronyms extracted from our entire corpus and their possible definitions to map the frequently occuring acronyms to the most frequent n-grams terms and vice-versa. We decided to do this so we can contextualize and extend our most frequent terms list to include acronyms which are ubiquitously used in technology-related research papers.

a) Acronym_processor.py
In **get_acronyms**, our ACL corpus is converted into a tokenized list of words before we loop through it and extract tokens that fit the criteria of being alphabetical, uppercase, and with a length greater than 1 which are stored in a set to remove duplicates. In **get_definitions**, English stopwords are then filtered out of our tokenized corpus leaving behind only content words. The acronym list is iterated through, checking every n number (n = length of acronym) of tokens in the corpus to find n words that fits the criteria of each first letter of each word matching the uppercase letters of each letter in the acronym, having a length of at least 2, and not entirely uppercase. These filter out words like 'MNT' (other acronyms), and {'A', 't', 'j', …} which give us no context. When n

words are found that match the criteria, the acronym and words are put into a dictionary respectively as key and value. If multiple and different combinations of n words that match the acronym criterias are found, they are added to the acronym value along with previous definitions found, giving us an acronym and a set of possible definitions in each dictionary key-value pair.

Sample entries:
'VSM': ['Vector Space Model', 'Variants Statistical Machine', 'Vector Space Modeling']
'NLP': ['Natural Language Process', 'Networks Lidia Pivovarova', 'Natural Language Pro', 'Natural Language Processing', 'Natural Language Pragmatic']

b) Process and Problems:

Our process of creating the dictionary went through a few different stages before we became mildly satisfied with the output. The biggest issue that came up was the processing speed of creating said dictionary. We first started off creating n-gram lists for each acronym, n depending on the size of the acronym. This meant that for each acronym, a list of all possible n-grams of the entire ACL corpus must be generated. Each gram would then be iterated through until it matches the criterias we listed in **acronym_processor.py**, and then finally will be added to the dictionary. This took a considerable amount of time and took about 1 hour for the process to end; this aggravated by the fact that the acronym dictionary we created is inherently flawed.

Firstly, the raw text data we have from our corpus is not perfect, as there are formatting issues that split and generate parts of non-acronym words as all uppercase tokens, which our algorithm captures and stores as an acronym - so false positives. However even regardless, just because some tokens are all caps, doesn’t automatically mean it is an acronym. For example, the word ‘noun’ is frequently displayed as “NOUN” or “NN” in the NLP field. “NN” is shortened for “noun”, but that does not necessarily make it an acronym. Another issue is that there are myriads of names of people in the corpus, and we have no real, quick, and easy way of filtering them out without finding a gigantic corpus of names (which also has non-English names) and filtering our corpus through it or train a classifier, which will take an unpredictable amount of time. Names are capitalized only in the first letter, so if the algorithm finds names and any other n number of words that match the acronym, it is still stored in the dictionary as a possible definition.

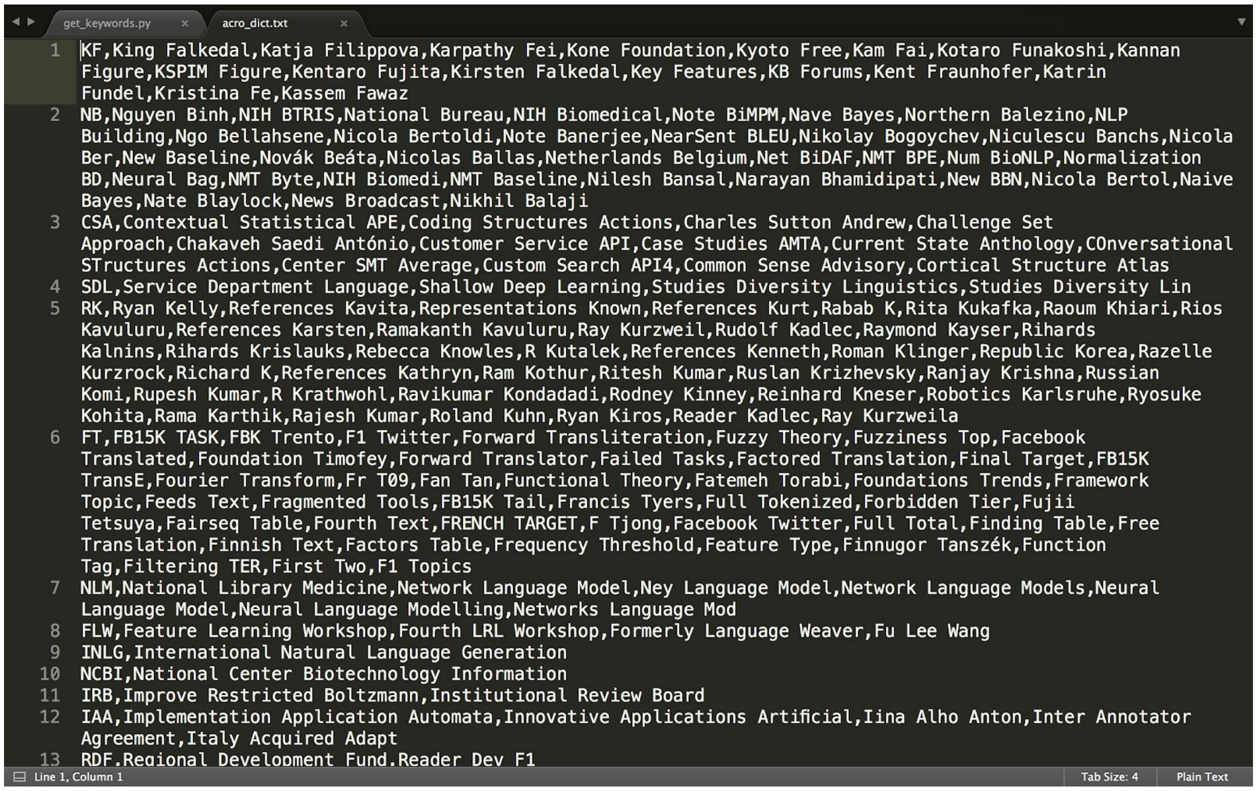
Finally, we created this dictionary with an assumption that most definitely doesn't hold true in the real word, and that is if we find an acronym, then its definition spelled out in the same text. We assumed that generally when an acronym is used, its definition will appear next to it, e.g. NLP (Natural Language Processing). We also assumed that the definition will have the initial letter of each word capitalized! When we tried allowing words whose initial letters are lowercase, we received a lot more meaningless definitions. There could still be valid definitions in the list, but the amount of invalid ones grew exponentially and finding the valid ones within wasn't feasible for us.

However, even with these problems, we are able to extract some accurate definitions. For commonly known acronyms in NLP such as LSA, NLU, NLP, the right definitions were found in the dictionary. There are definitions that are meaningless or probable, but if some of these definitions can be tied to the most frequent bigrams/terms, then we can say our dictionary is partly useable.

So, we fixed what we could - the running time. As mentioned before, we generated n-grams for each acronym, which caused much of the slowdown, so we decided get rid of that and indexed through the corpus each iteration to find matches. This also brings us to the usage of acro_dict.txt:

c) acro_dict.txt

In order to speed up the process of processing the meaningful keywords list, instead of creating the acronym dictionary every single time we run our code, we came up with the idea of generating the dictionary in advance and storing the result. So, in the end, we modified the **acronym_processor.py** program and made it write to an output file **acro_dict.txt** the dictionary that was built with the two previously mentioned functions. In the output file, each line specifies one single entry in the acronym dictionary, with the key of the entry (an acronym) being the first word. After that is the list of possible matches found in the corpus, each term separated by a comma. Some examples are shown in the picture on the next page.



```
KF,King Falkedal,Katja Filippova,Karpathy Fei,Kone Foundation,Kyoto Free,Kam Fai,Kotaro Funakoshi,Kannan Figure,KSPIM Figure,Kentaro Fujita,Kirsten Falkedal,Key Features,KB Forums,Kent Fraunhofer,Katrin Fundel,Kristina Fe,Kassem Fawaz
NB,Nguyen Binh,NIH BTRIS,National Bureau,NIH Biomedical,Note BiMPM,Nave Bayes,Northern Balezino,NLP Building,Ngo Bellahsene,Nicola Bertoldi,Note Banerjee,NearSent BLEU,Nikolay Bogoychev,Niculescu Banchs,Nicola Ber,New Baseline,Novák Beáta,Nicolas Ballas,Netherlands Belgium,Net BiDAF,NMT BPE,Num BioNLP,Normalization BD,Neural Bag,NMT Byte,NIH Biomedi,NMT Baseline,Nilesh Bansal,Narayan Bhamidipati,New BBN,Nicola Bertol,Naive Bayes,Nate Blaylock,News Broadcast,Nikhil Balaji
CSA,Contextual Statistical APE,Coding Structures Actions,Charles Sutton Andrew,Challenge Set Approach,Chakaveh Saedi António,Customer Service API,Case Studies AMTA,Current State Anthology,COnversational STructures Actions,Center SMT Average,Custom Search API4,Common Sense Advisory,Cortical Structure Atlas
SDL,Service Department Language,Shallow Deep Learning,Studies Diversity Linguistics,Studies Diversity Lin
RK,Ryan Kelly,References Kavita,Representations Known,References Kurt,Rabab K,Rita Kukafka,Raoum Khiari,Rios Kavuluru,References Karsten,Ramakanth Kavuluru,Ray Kurzweil,Rudolf Kadlec,Raymond Kayser,Rihards Kalnins,Rihards Krislauks,Rebecca Knowles,R Kutalek,References Kenneth,Roman Klinger,Republic Korea,Razelle Kurzrock,Richard K,References Kathryn,Ram Kothur,Ritesh Kumar,Ruslan Krizhevsky,Ranjay Krishna,Russian Komi,Rupesh Kumar,R Krathwohl,Ravikumar Kondadadi,Rodney Kinney,Reinhard Kneser,Robotics Karlsruhe,Ryosuke Kohita,Rama Karthik,Rajesh Kumar,Roland Kuhn,Ryan Kiros,Reader Kadlec,Ray Kurzweila
FT,FB15K TASK,FBK Trento,F1 Twitter,Forward Transliteration,Fuzzy Theory,Fuzziness Top,Facebook Translated,Foundation Timofey,Forward Translator,Failed Tasks,Factored Translation,Final Target,FB15K TransE,Fourier Transform,Fr T09,Fan Tan,Functional Theory,Fatemeh Torabi,Foundations Trends,Framework Topic,Feeds Text,Fragmented Tools,FB15K Tail,Francis Tyers,Full Tokenized,Forbidden Tier,Fujii Tetsuya,Fairseq Table,Fourth Text,FRENCH TARGET,F Tjong,Facebook Twitter,Full Total,Finding Table,Free Translation,Finnish Text,Factors Table,Frequency Threshold,Feature Type,Finnugor Tanszék,Function Tag,Filtering TER,First Two,F1 Topics
NLM,National Library Medicine,Network Language Model,Ney Language Model,Network Language Models,Neural Language Model,Neural Language Modelling,Networks Language Mod
FLW,Feature Learning Workshop,Fourth LRL Workshop,Formerly Language Weaver,Fu Lee Wang
INLG,International Natural Language Generation
NCBI,National Center Biotechnology Information
IRB,Improve Restricted Boltzmann,Institutional Review Board
IAA,Implementation Application Automata,Innovative Applications Artificial,Iina Alho Anton,Inter Annotator Agreement,Italy Acquired Adapt
RDF,Regional Development Fund,Reader Dev F1
```

Figure: acro_dict.txt before definitions with acronyms were eliminated from definition choices.

VI. Further Improvements on Acronym Dictionary:

Problems identified with our acronym dictionary were issues filtering what is and what isn't an acronym, what definitions counts as probable and improbable, not being able to know for sure if a definition is correct in the context of how the acronym is used in each research paper, and finding and filtering out names (especially non-English names). Acronym identification and mapping is a very complicated task and we doubt the criterias we listed barely scratches the surface. However, something that we could try to make name identification less of an impossible task is to create a name classifier and train it on multitudes of data. This may work for English names, but other languages may require a seperate classifier or training set. Another idea is to find POS patterns that acronym definitions may follow and filter them out even more from there. For example, they can only be nouns, proper nouns, verbs, etc - or they follow similar POS structures, e.g. JJ NN VBG, NNP NN NN, …. .