Paleontology Topic Trends

Final Report

CS 4624: Multimedia, Hypertext, and Information Access

Virginia Tech
Blacksburg, VA 24061
Instructor: Dr. Edward A. Fox
Client: Sterling Nesbitt

March 22, 2018

James Wilson, Rudy Cruz, Joe Martin, and Eric Weiler

Table of Contents

Table of Figures

Executive Summary

The purpose of the project was to run modern data analysis on abstracts created by the Society of Vertebrate Paleontology. The Society of Vertebrate Paleontology has a yearly convention in which members from all over the world gather together and present their studies from the appropriate year. Our client, Professor Sterling Nesbit, provided our group with a collection of abstracts dating back to 1987. Our job was to take all of the abstracts from each year and run analyses to see the trends and patterns spanning over all the years that the Society of Vertebrate Paleontology had been publishing abstracts in collections. The method the team has employed changed throughout the span of the project. In the beginning, the team planned on using Latent Dirichlet Allocation or LDA to summarize the abstracts. This would find the topics prevalent in the collection, and show the mix of those topics found in each of the abstracts. After further discussion with our client, the team decided on providing more straightforward analysis, based off graphing hierarchies in the abstracts. In order to properly run the graphing analysis on the abstracts our team had to scrape the abstracts to ensure the most useful data was not overlooked in the analysis. The process of scraping the abstracts began with removing all the metadata from the abstract text files (which were converted from PDF). Then the team eliminated any English stop words in the text files to remove words that are not commonly needed for analysis. The next step was to customize and add words to this list of stop words, based on yearly differences. For example, in some years the Society of Vertebrate Paleontology required its members to create their abstracts referencing the United States as “The United States of America” while in other years they were required to reference it as “United States.” These slight changes required our team to alter our method of stop word elimination to be specific to each year. Once the scraping was done, the team created graphing scripts to produce graph based off Vertebrate Paleontology hierarchies. After meeting with our client multiple times to further refine our analysis, we created the final analysis script version. These graphs helped our client visualize the patterns in findings made by the Society of Vertebrate Paleontology. The project should be further developed to automatically extract abstracts from the convention’s PDF collection, as well as some sort of update to stop words based off of the society’s yearly modifications.

1.0 Introduction

This report contains information regarding our Multimedia/Hypertext/Information Access Capstone Project, specifically the problem posed to us by Dr. Fox and Dr. Nesbitt. It contains a table of contents as well as a table of graphs and code samples. It has a developer manual so that any new developer can get information about our development process. This will allow new developers to start development quickly.

1.1 Problem Statement

From Dr. Nesbitt: I have been planning to analyze the abstracts (~700-800 per year) from our flagship society for vertebrate paleontology (the Society of Vertebrate Paleontology) for trends by pulling keywords from these abstracts over the length of our society (since 1981). I thought it would be fun to see which groups (e.g., dinosaurs vs. mammals) presentations focus on, depending on the year, and seeing if there are any other trends related to these (e.g., Movies like Jurassic Park coming out).[1] It would essentially be a data mining project to see how vertebrate paleontology has changed or stayed the same through time. I have been reading lots about social economics and I thought it would be fun to analyze vertebrate paleontology trends to see what has changed and what has remained the same (e.g., technology changes, number of people studying, integration with other sciences.....). As a start, this could chronicle the study of different groups (dinosaurs, reptiles, mammals) over at least the last 17 years and figure out trends in research (e.g., CT data usage, discovery rates.....). This could use natural language processing (NLP) and machine learning. I have at least 15 years of the abstract books from the annual meetings as searchable PDFs. The older abstract books are digitized also, but I don't think they are searchable PDFs; we can change that easily though.

1.2 Client Background

Dr. Nesbitt's holds a Ph.D. in Geosciences from Columbia University. His research area focus is in Paleobiology, which is the study of fossils and the biology behind them. He has very little background in Computer Sciences, nor does the Society of Vertebrate Paleontology, so he was very willing to work with us to get whatever information he could from us.

2.0 Project Design

Our project was designed in multiple steps. We broke up our project into three main parts: scraping, word clouds, and data analysis.

The process of scraping was required due to the fact that the provided PDF files containing the abstracts had useless information regarding our analysis. We removed all useless information based off of pre-determined stop words in combination with specific stop words provided to us by our client.

The next step in our project is to generate word clouds. With the scraped PDFs generated, we created word clouds with all the useful information in the original PDFs. The word clouds were created to give our client an idea of the most common words and phrases per year. Our client used the word clouds to give us appropriate feedback in terms of whether these word clouds were useful.

Once we have the word clouds that satisfied our client we moved onto the data analysis part of our project. Next the team researched Latent Dirichlet Allocation and quickly realized it was not the analysis our client was looking for. He was much more interested in the change in vocabulary of the Society of Vertebrate Paleontology abstract publications over almost three decades. After researching graphing frameworks our team settle on the PyPlot framework. With this framework the team designed a robust set of scripts to complete the analysis.

2.1 Implementation

We started the project by creating word clouds and word count documents for Dr. Nesbitt. He needed these easy-to-read documents in order to give us assistance in creating categorizations and stop word lists. Latent Dirichlet Allocation requires specific text corpora in order to generate categorizations, and if there are garbage words (e.g., session, technical, poster) then the categorizations will not be as accurate.

In Figure B.1 you can see our catFiles.py script. The purpose of this script is to concatenate each year's PDF files and create a different file containing every year's abstracts. The process to do this was to create a list of all the abstract files. Once this file is generated we created an output file where we write all the information of every PDF. We write a header indicating which file we are currently working on, then we write the content of that file in our output file. The resulting file is a combination of every year's PDFs.

In Figure B.2 you can see our htmlRemovalScript.py script. The purpose of this script is to remove unnecessary meta-text contained in our PDF files. The process to do this was as simple as removing meta-text in each line of our file. We iterate through each line in our file, and per line we replace meta-text with '' which results in the removal of the meta text. The text we removed consisted of the strings "<[^<]+?>" and "(^@)+". After removing these strings we wrote the resulting lines in an output file.

In Figure B.3 you can see our freqWords.py script. The purpose of this script is to provide our client with the top fifty most used words. The process in which we do this consists of three steps. The first step is to break up the words based off of spaces and blank space. Once we have an array of all the words in the abstract we use a list of stop words to extract unnecessary words from the list. We used a standard English language list of stop words in combination with a list created by our client. After this we create a list of the top fifty most used words and print them to a text file.

In Figure B.4 you can see our testwordcloud.py script. The purpose of this script is to generate word clouds for our client. The process to do this requires us to first input a file and using a word cloud framework we run the function WordCloud().generate() with the appropriate document and produce a word cloud. The word cloud provides our client with a view of all the words and phrases that are seen in the PDF. We create a word cloud for each year and present our findings to our client.

3.0 Project Evaluation

Our group began the project with the intentions of performing LDA on the given abstracts. Over time our client began to find interest in different aspects of the project. After generating word clouds for our client, he began to find patterns in the vocabulary used based on the years the abstracts came out. At the time the team was working on researching LDA as our main implementation goal. Our client however seemed more interested in the word clouds and the textual analysis the team was doing. Looking back, the team wishes they had more time to develop the graphing scripts and work on stretch goals. Even though the team made the transition from LDA to textual analysis, we wish we had realized this earlier. This would have allowed the team to finish the graph scripts earlier and possibly work on the association network problem our client was also interested in. He then asked us to format these word clouds in more useful visuals, so we created graphs of the given word counts per year. Working with our client, we generated a couple of categories and subcategories for these graphs. As a result we ended up with stack bar graphs that helped our client see trends in terms of what was being researched per year. Our client was very happy with these findings, as he could see trends relating to the categories we established. We were happy with the end result of our project, but we did run into some difficulties along the way. We were given categories to make visual data from, but in the future, our client will either only be able to run the analyses on the same categories or re-establish some categories. Another area of difficulty had to do with the formatting of the abstracts. The format changes so much from year to year that it was difficult to have a set code that can break up the PDF file into each of their abstracts.

4.0 User Manual

Running the dataset on a new year:

1. Convert the PDF to a text file
2. Place the text file in the subdirectory labelled SVPAbstractsOnly

    a. If there is no SVPAbstractsOnlyDirectory one must be made with every single year contained within, or at least the span of years you wish to analyze.
3. Inside of the scripts titled graphkeyword.py and graphStackedKeywords.py, you must edit the variable “abstract counts” to sequentially contain all of the years’ abstract counts.
    a. Unless you have a specific count for each year, estimating the counts is alright. This just gives a guide for the graphing data.
4. Then you must run the script by right clicking on it and selecting a Python launcher. This must be a Python 2 launcher.
5. Any figures that are created will be placed in a directory titled KeywordGraphFigs, otherwise the command line generated by your Python launcher will give details.

5.0 Developer Manual

Within the submission we have attached a multitude of different data sets. The key data set is within a folder titled SVPAbstracts. These contain text files for each of the related PDFs of abstracts. All of our scripts require this to be maintained, as well as making sure the titles of the text documents follow the same formatting. Our scripts require that from where they are run they contain the folder named in exactly the way that it is now. You can see short descriptions for each of our scripts in the Appendix, in section B. A more detailed description of our most complex scripts follows. All other scripts are automated, such that, as long as the directory structure is correct, the data analysis will be done automatically.

graphkeyword.py and graphStackedKeyword.py: Seen in Figures B5 and B6, these are our main scripts for our most detailed data analysis. The abstract counts are rough estimates given for each abstract sequentially. This estimate was made by gathering the total number of abstracts per section and multiplying that number by the number of sections that year. There are large inconsistencies between each abstract, which caused problems with abstract count automation.

B5: This uses the keywords given to us by Dr. Nesbitt, contained in the file KeyWordList.txt.
B6: This requests a name of a grouping, and looks in the same directory for the list of keywords that are contained. The grouping file must be called “groupname”.txt

6.0 Lessons Learned

The main issue we encountered at the beginning of the semester were issues with communication. We did not realize at the beginning of our project which type of LDA we were supposed to do. At first we believed it was Linear Discriminant Analysis, but we later learned the intention was Latent Dirichlet Analysis. In the end we ended up using neither.

A lot of our problems stemmed from the poor formatting of the PDFs. Since the format changed from year to year, it made automation very difficult. Our group was going to attempt to run a Latent Dirichlet Analysis on the data, but due to the difficult formatting of the PDFs we were not able to easily do this. The formatting prevented us from having a strong separation between abstracts that was consistent throughout the years; this made it impossible to pass our LDA model the different abstracts individually. Since the LDA model finds categories of interests throughout the different abstracts, it was crucial that we would pass in the abstracts individually. We tried many different methods of splitting up the abstracts but did not have good enough results as the format caused issues every time. Our client eventually liked the data visualizations that we created for him and decided to move forward with these. From this we learned that flexibility with the client was very important.

6.1 Timeline

Tues 1/30, Team will write code to create word clouds from abstracts.

Tues 2/6, Team will meet after class to prepare our presentation.

Tues 2/13, Team will present presentation during class time.

Thurs 2/15, Team will apply word cloud code to all abstracts.

Tues 2/20, Team will work with client to start distinguishing categories for our models.

Tues 3/13, Team will create initial models, as well as prepare for second presentation.

Thurs 3/22, Team will give presentation during class time.

Tues 4/3, Team will work on written report.

Thurs 4/5, Team will prepare for third presentation.

Tues 4/10, Team will present presentation during class time.

Tues 4/17, Team will complete final testing of models.

Thurs 4/19, Team will prepare for final presentation.

Tues 4/24, Team will complete written report.

Tues 5/1, Team will present final presentation during class time.

Thurs 4/19, Team will apply model to all abstracts.

6.2 Problems

The first problem we encountered was not a problem with our code at all but a misunderstanding about terminology. We believed for the first few weeks of our project design that LDA stood for Linear Discriminant Analysis, but was really referencing Latent Dirichlet Allocation.

The next problem that plagued the project was Python versioning as many of our scripts that we wrote or needed to use required different versions of Python.

Our next problem was the stopwords, as they were needed to create word clouds and to do text file analysis, but were not clear to us exactly, as some words were not typical English stop words, but required technical knowledge on the abstracts. For example, as the years went by many of the findings made were done so by using research methods and tools that other years hadn't seen before. Professor Nesbitt informed us that due to these changes in methods, some older methods were of no interest to him as they are no longer considered up to date methods.

Another problem was converting PDF to text files, which we needed to do for our scripts but we couldn't initially find a way to convert them via a script itself. So we used online resources to handle the conversions, such as online2pdf.com. There are options such as pdftotext, but we didn't want to modify our systems too much, when we could just use online resources that are doing it for us. The online versions of pdftotext also do not allow for files of the size of some of our abstract docs.

Finally the team realized that the LDA itself was a problem. We informed Dr. Nesbitt that the LDA model would return to us a list of the most important categories found per year. We discussed this with Dr. Nesbitt and were informed that he wanted something a little different from our project. He gave us a specific list of categories that he had in mind and told us to find the changes/trends in these categories over the different years of abstracts provided. Due to the fact that LDA would create its own categories per year, and Dr. Nesbitt wanted trends for specific categories over years; our team moved from LDA to more straightforward graphing analysis.

6.3 Solutions

It was only after we had already presented our plans to perform Linear Discriminant Analysis that we realized that we were using the wrong "LDA". This was thanks to the remarks our professor gave at the end of our presentation; we should have been using *Latent Dirichlet Allocation* instead of Linear Discriminant Analysis. Upon receiving this information, our team proceeded to look into how this method worked and try to figure out how we would use it in our project. After becoming familiar with the overview and terminology, LDA seemed that it would be remarkably useful for creating categories out of uncategorized data.[2] We planned on performing LDA with our word counts as input and let the analysis generate categories for the data. All we would have to do next is put a name to the categories. We then sought out to find a suitable implementation of LDA to use on our data. We settled upon the official Python LDA library with full support for MacOS, Windows, and Linux, and complete with thorough documentation. After working with LDA for a bit, it was apparent that the variability of the abstracts, and the lack of a standard formatting, would make it far too difficult to actually do LDA. After a meeting with our

client, we decided to self-define the categories and find trends throughout the years. It was then that the team realized that LDA was not actually what our client wanted. So the team moved to a graph based analysis using the PyPlot framework.

Another issue we ran into came from Python versioning. Scripts that we had written during the beginning of the project phase started giving us errors. We hadn't made any changes to the scripts and they certainly ran free of errors before. After running the scripts with different configurations in Python we learned that there were methods that were in the scripts that had been deprecated in Python 3, specifically the self.buffer_decode() method, in the version of Python that we were using. To solve this, we chose to run the scripts in Python 2. This returned the functionality of the deprecated methods and gave us error free results.

Finally, despite introducing a list of stop words to use when mining words from the abstracts, we were still coming across words that were redundant and meaningless and did not help us with determining topics. The stop words list that is default in Python contains the most common words in the English language, words like *I, you, is, a, an, etc*. This improved our results by filtering out those words, but we still found words like, *therefore, university, united states, consequently,* etc*.* We informed our client of this issue and he responded by giving us a list of additional words to include that are words that he is not interested in.

7.0 Future Work

Our solution requires the manual collection of abstracts from the Society of Vertebrate Paleontology on a year to year basis. One step we can take to improve our solution is to automate this process by building an online submission portal where abstracts can be submitted and saved in the repository where we keep all of the abstracts and run our code. Our solution also requires manual updating of the list of stop words and possibly even removal of stop words on particular abstracts that do not contain many of a certain word or where that word may be critical to the abstracts of that year. This customization of the code could be implemented with further work dedicated to building a fully packaged desktop application, complete with GUI where user could run tailored reports and modify certain parameters of the analysis and report. This would include making the stop word list editable by the end users. To do this the future team could develop the scripts to use stopwords from a text file instead of listing them in the script itself. Then the system could use a configuration script to run the graphing script with the correct stop word list. This would also allow non-technical end users to run the code as all of it will be hidden and presented in a UI.

Future work dedicated towards implementing our solution via a web application would include both previously mentioned solutions. The web app could include a submission portal for the new year's abstracts that the end user would like to analyze, and then the user could be taken to a secondary page where they could customize the parameters of the analysis. Parameters could include a list of default stop words or the option for the user to submit their own list of stop words. The benefit of this would be that those interested in viewing an analysis (like Dr. Nesbitt, or others interested in paleontology trends) would not need to submit the abstracts to us, for us to run and build the analysis, and then send back to them. Instead, the results could be handled by the user and obtained immediately.

In our last meeting with Dr. Nesbitt he mentioned that future work for the solution should include networking analysis: how the authors of the publications relate to one another. Many of the authors worked on multiple publications in different groups for each one. There is a level of information hidden in how different authors work with each other, the topics they may specialize in, and even the number and setup of these groups. Future work could include extracting this information from the title section of each abstract where the authors are listed and developing a way to represent the relationships of the data such as a graph data structure that links authors' connections to other authors. This graph could be expanded to include the subjects (high level) of the abstracts or even the trend words themselves (low level).

8.0 Acknowledgments

Dr. Sterling Nesbitt, Assistant Professor in Geosciences
Phone: 540-231-6330
Email: sjn2104@vt.edu
Office: 3061A Derring Hall

9.0 References

1. “Paleobiology.” Dictionary.com, Dictionary.com, www.dictionary.com/browse/paleobiology, accessed 5/7/18
2. Blei, David M. “Latent Dirichlet Allocation.” Journal of Machine Learning Research 3, vol. 3, Jan. 2003, pp. 994–1022.

9.1 Literature Review

1. This was just used for us to understand what paleobiology was, and give us a good way to explain it.

2. This is the paper on Latent Dirichlet Allocation, wherein David Blei explains the process of it, how it can be used, and why it is useful. This guided us in understanding how we might go about implementing it, as well as helping us understand why our projects formatting did not easily allow for it.

10.0 Appendices

10.1 Appendix A: Data Visualization

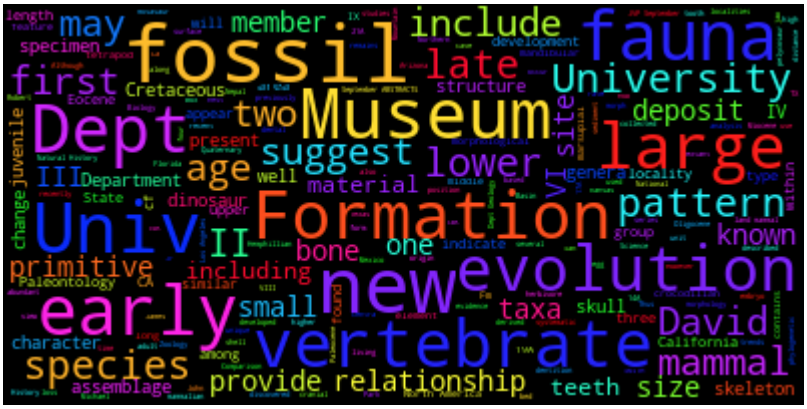

Figure A.1 1987 abstract word cloud

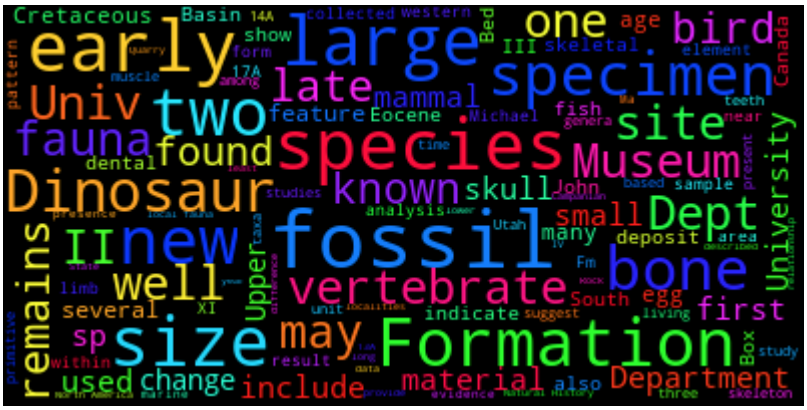

Figure A.2 1988 abstract word cloud

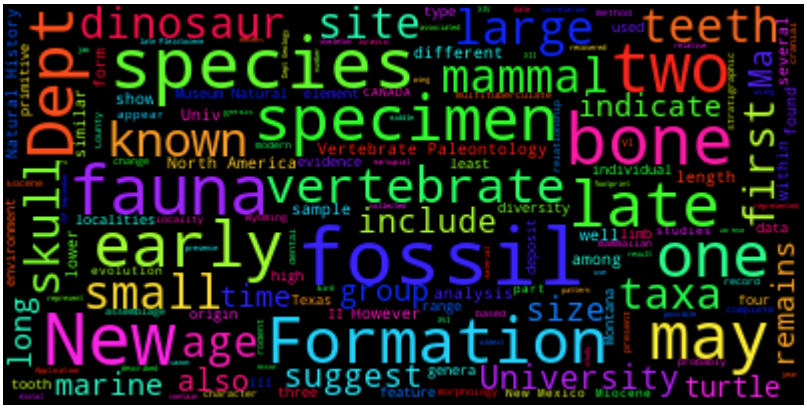

Figure A.3 1989 abstract word cloud

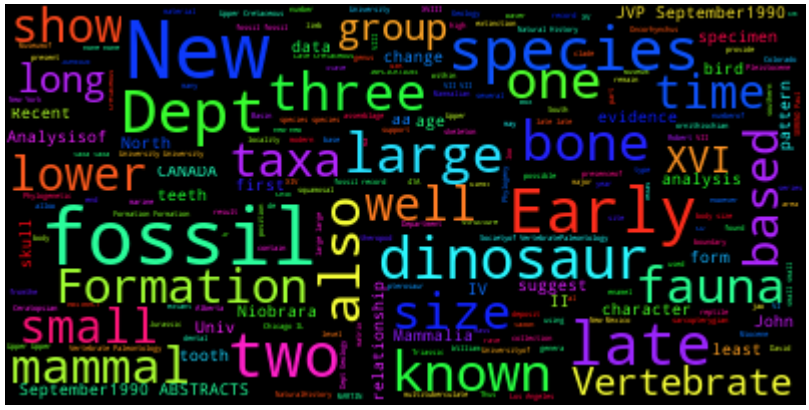

Figure A.4 1990 abstract word cloud

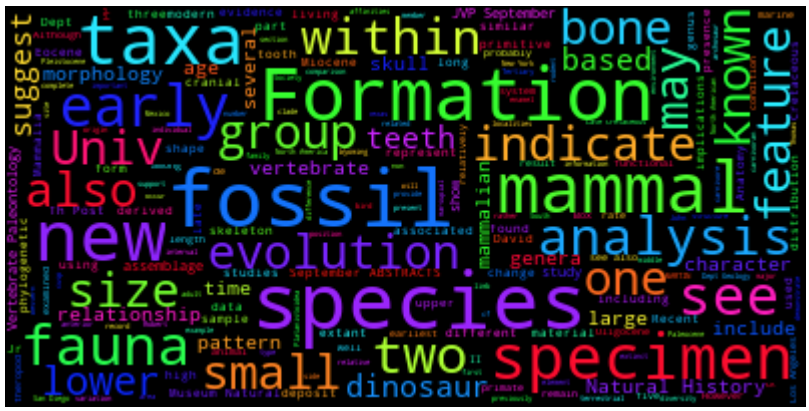

Figure A.5 1991 abstract word cloud

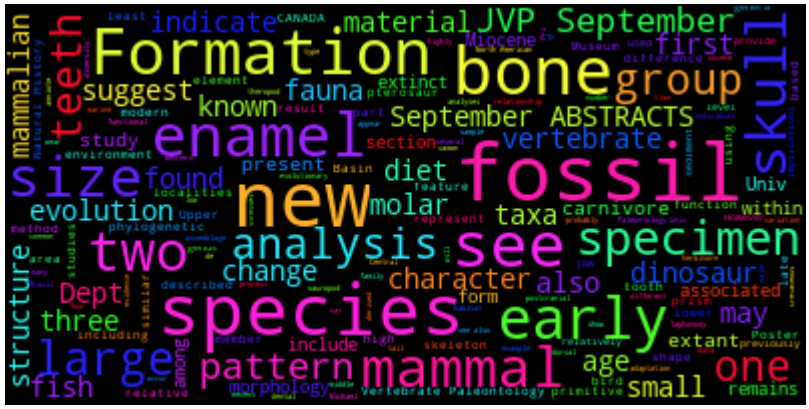

Figure A.6 1992 abstract word cloud

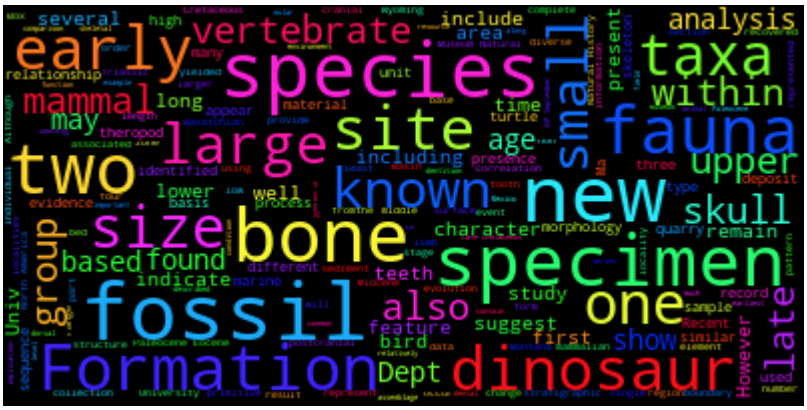

Figure A.7 1993 abstract word cloud

Figure A.8 1994 abstract word cloud

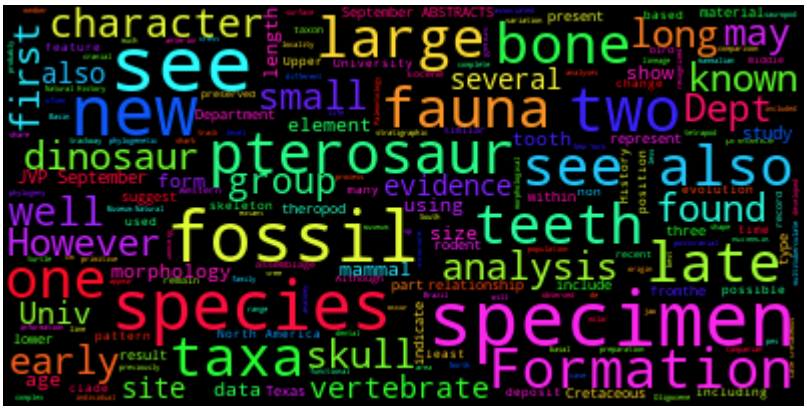

Figure A.9 1996 abstract word cloud

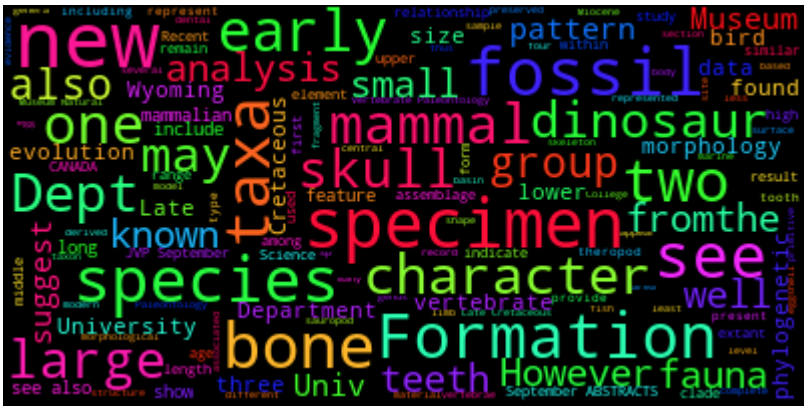

Figure A.10 1997 abstract word cloud

Figure A.11 1998 abstract word cloud

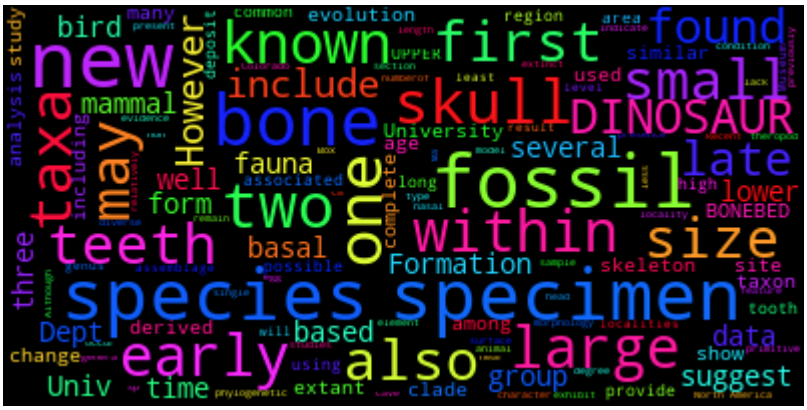

Figure A.12 1999 abstract word cloud

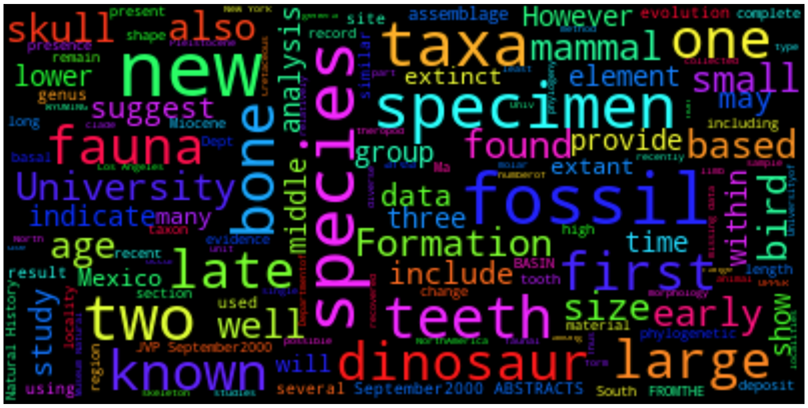

Figure A.13 2000 abstract word cloud

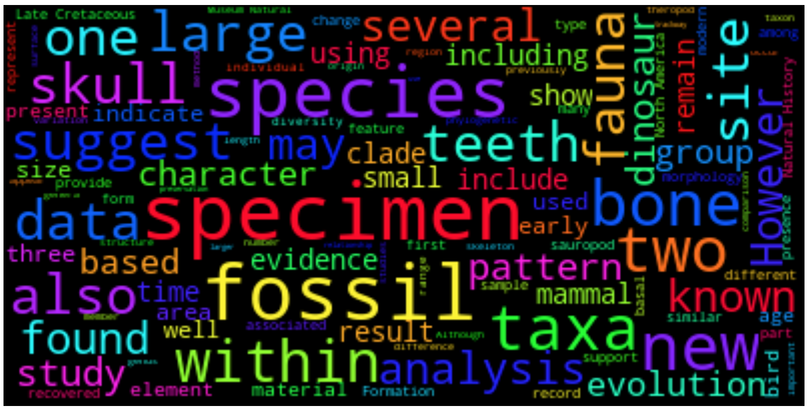

Figure A.14 2003 abstract word cloud

Figure A.15 2004 abstract word cloud

Figure A.16 2005 abstract word cloud

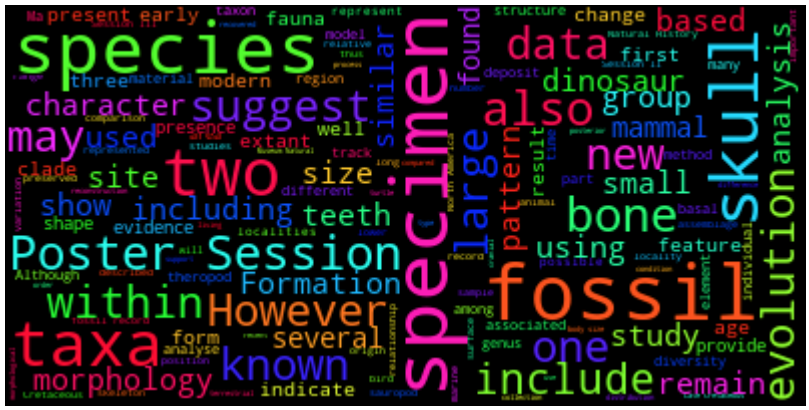

Figure A.17 2006 abstract word cloud

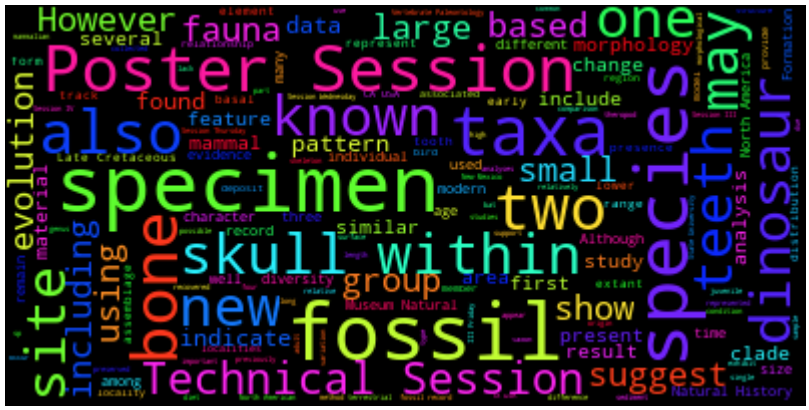

Figure A.18 2007 abstract word cloud

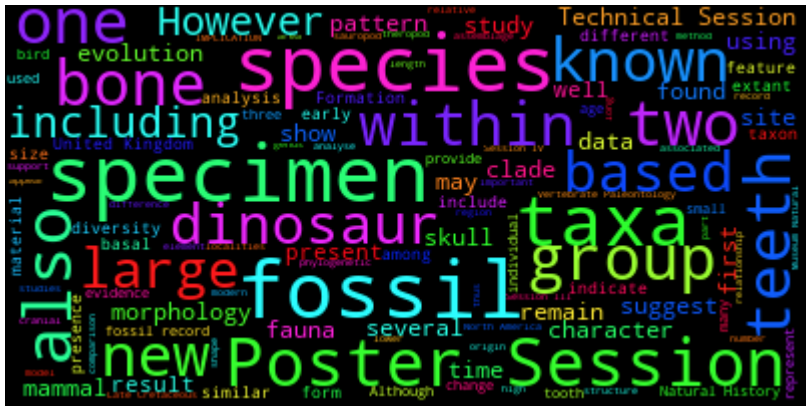

Figure A.19 2009 abstract word cloud

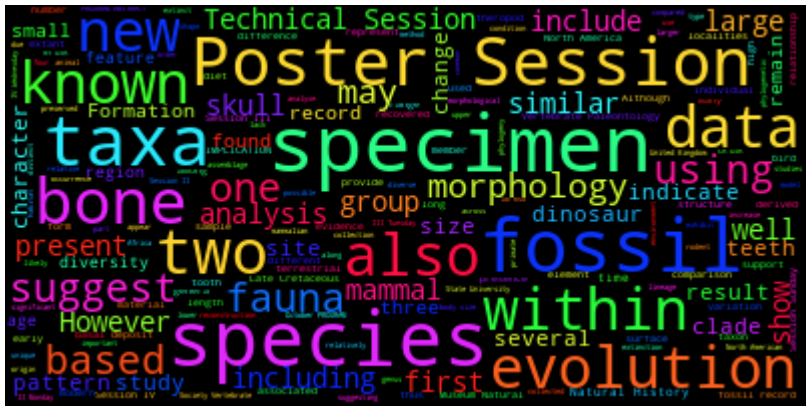

Figure A.20 2010 abstract word cloud

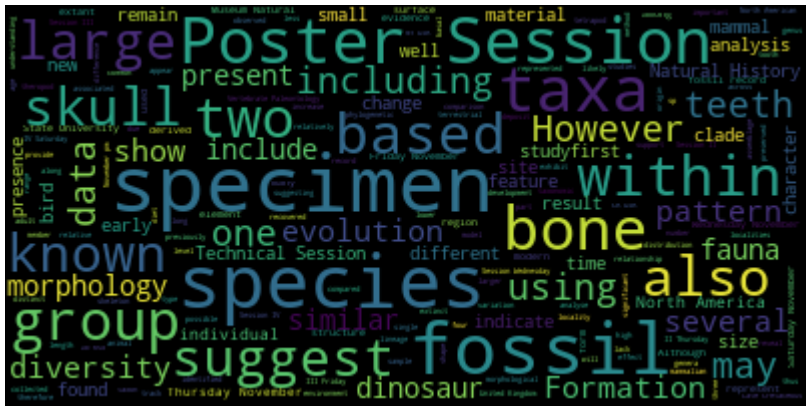

Figure A.21 2011 abstract word cloud

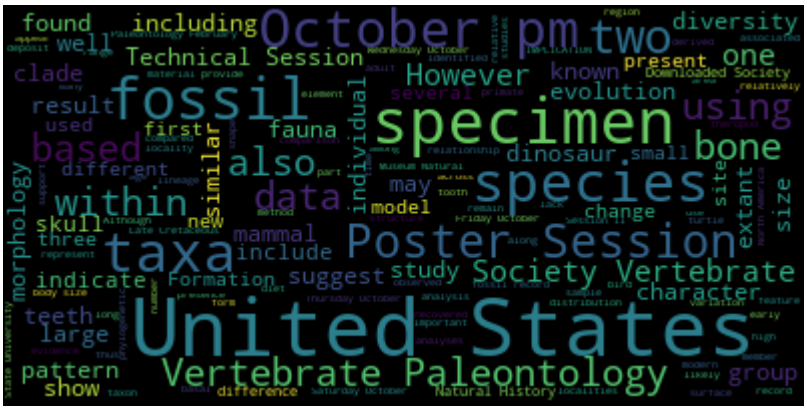

Figure A.22 2012 abstract word cloud

Figure A.23 2013 abstract word cloud

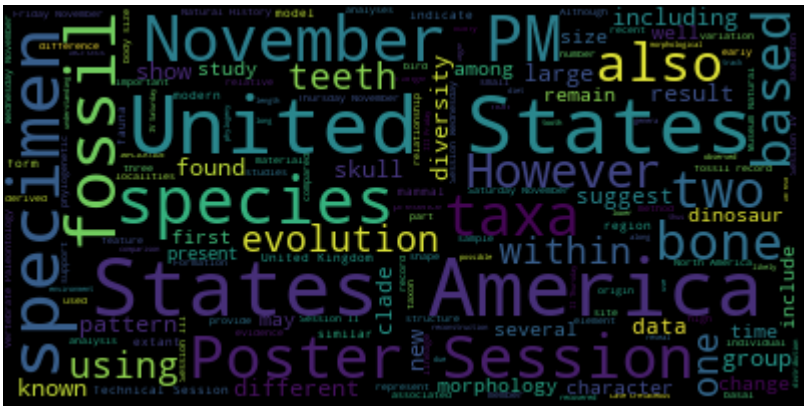

Figure A.24 2014 abstract word cloud

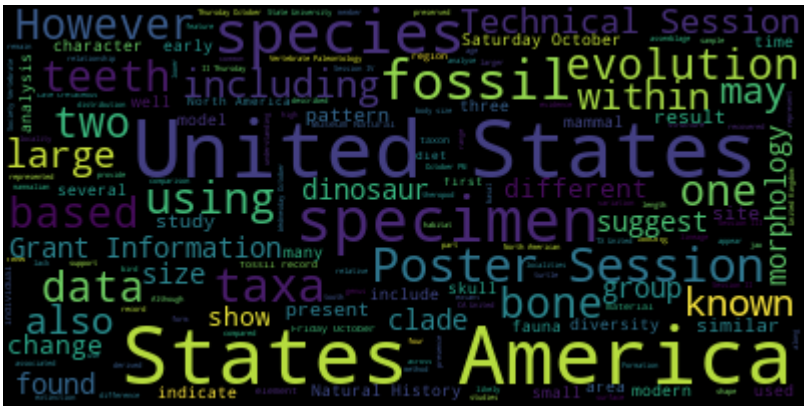

Figure A.25 2015 abstract word cloud

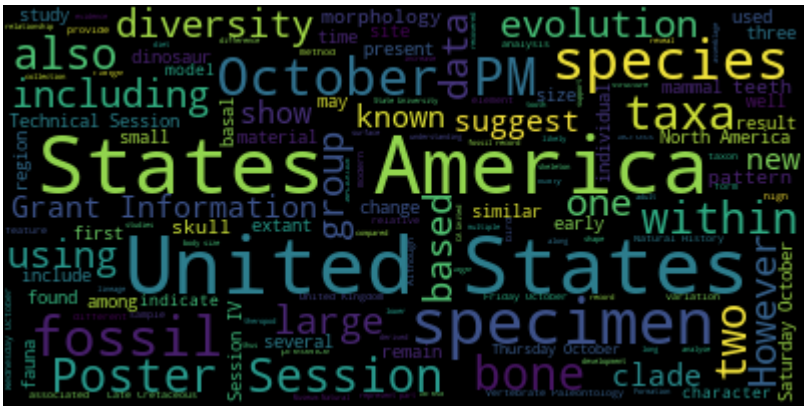

Figure A.26 2016 abstract word cloud

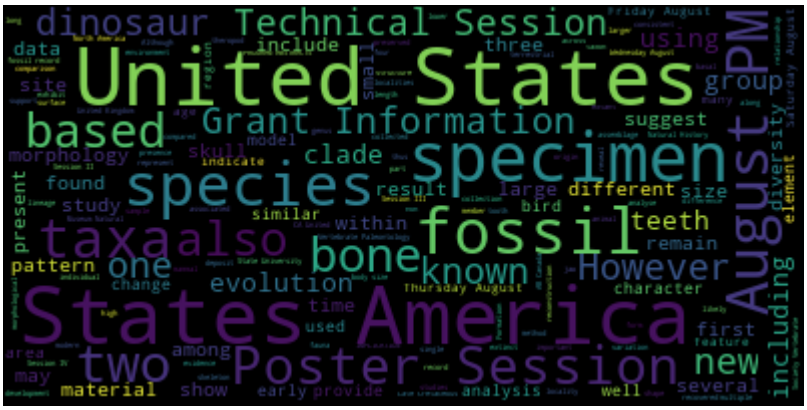

Figure A.27 2017 abstract word cloud

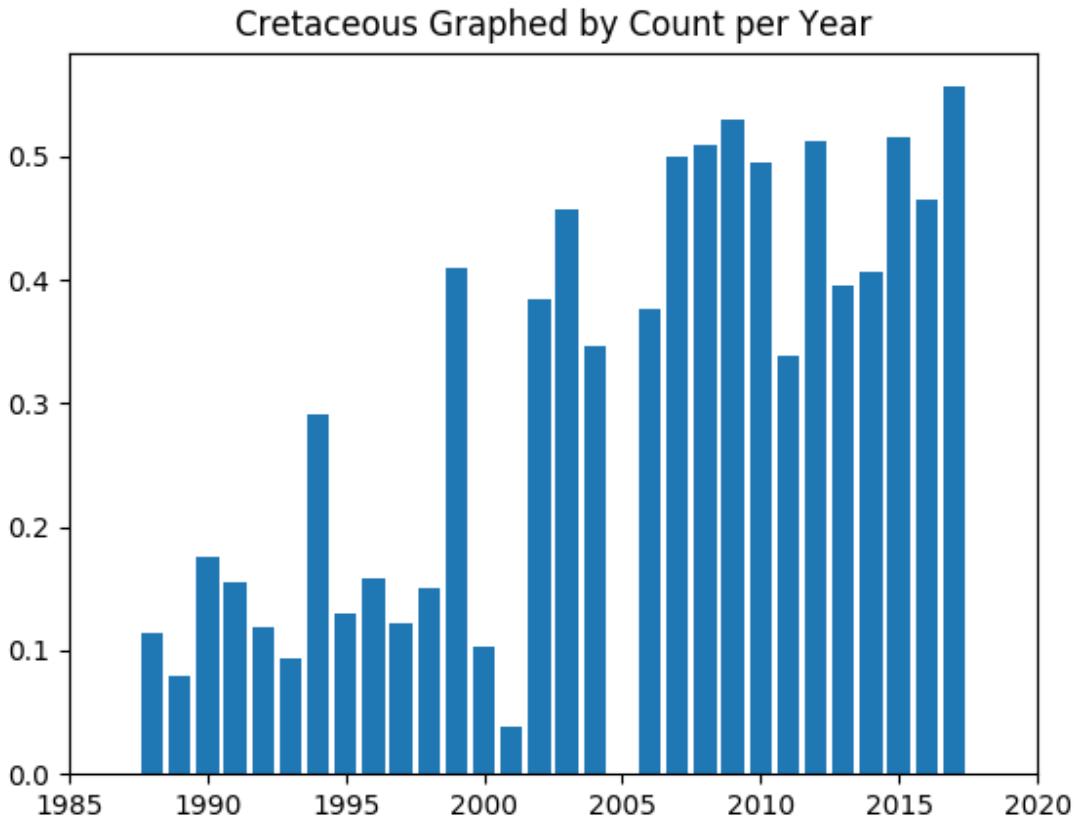

Figure A.28. Graph of the word Cretaceous, and its appearances in the abstracts by year

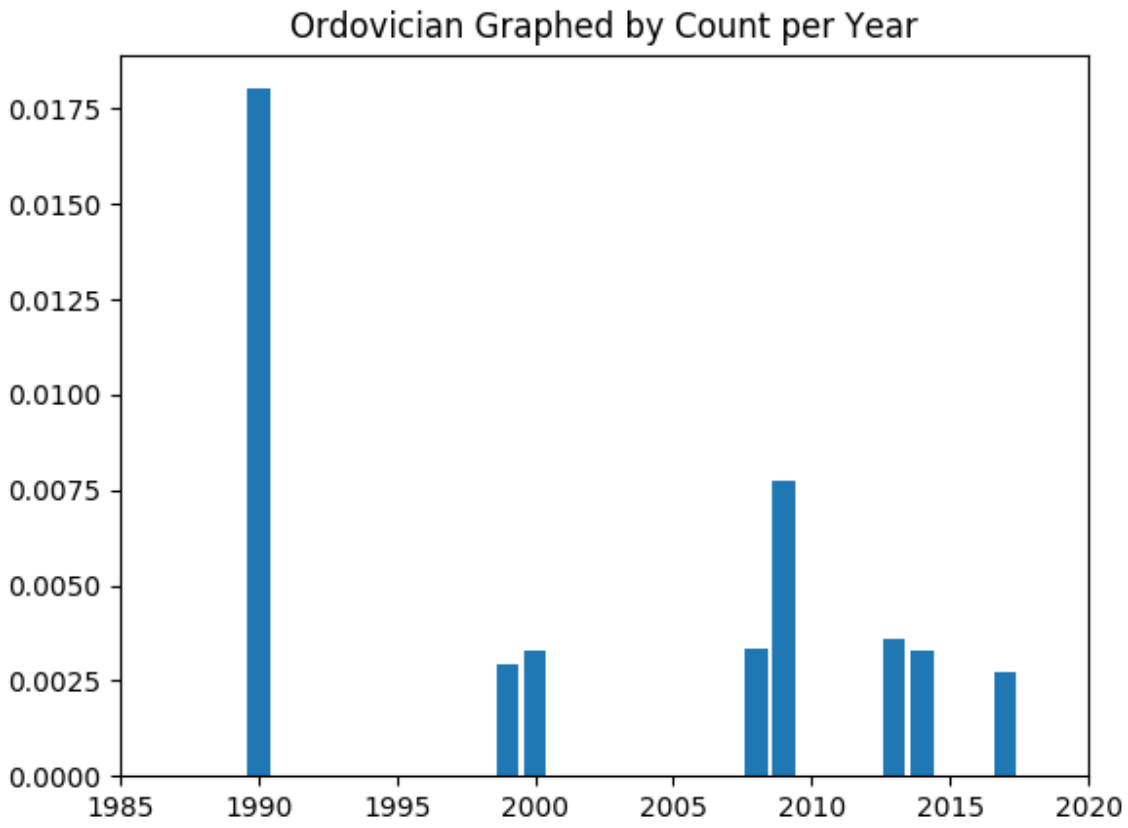

Figure A.29. Graph of the word Ordovician, and its occurrences. It does not appear in each abstract.

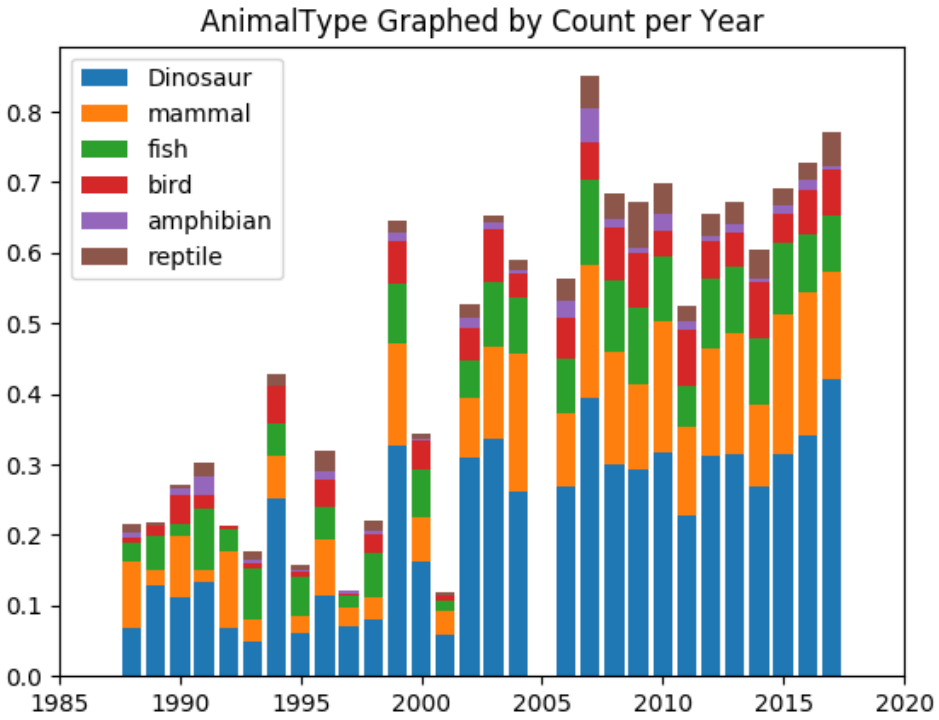

Figure A.30. Our most detailed graph: this graph shows the word count for different topics, but in a stacked graph form, in order to better compare similar topics.

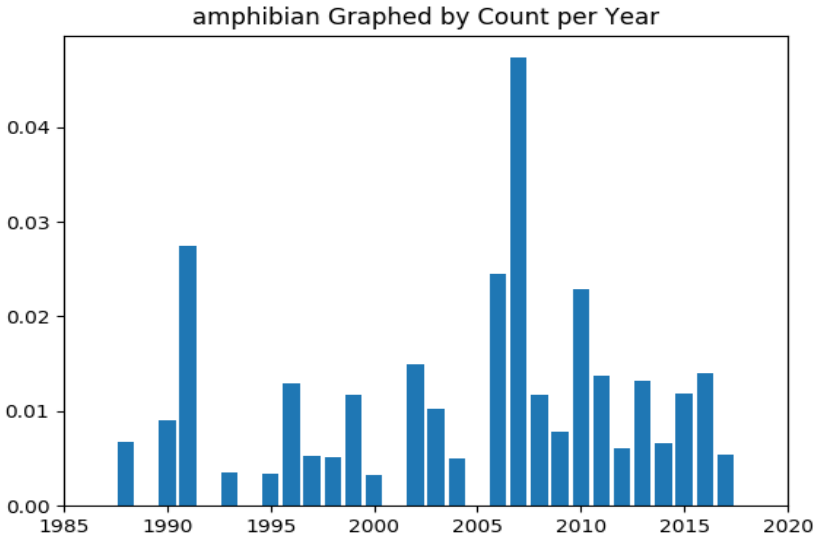

Figure A.31. Graph showing the appearance of “Amphibian” over the years in the abstract collections

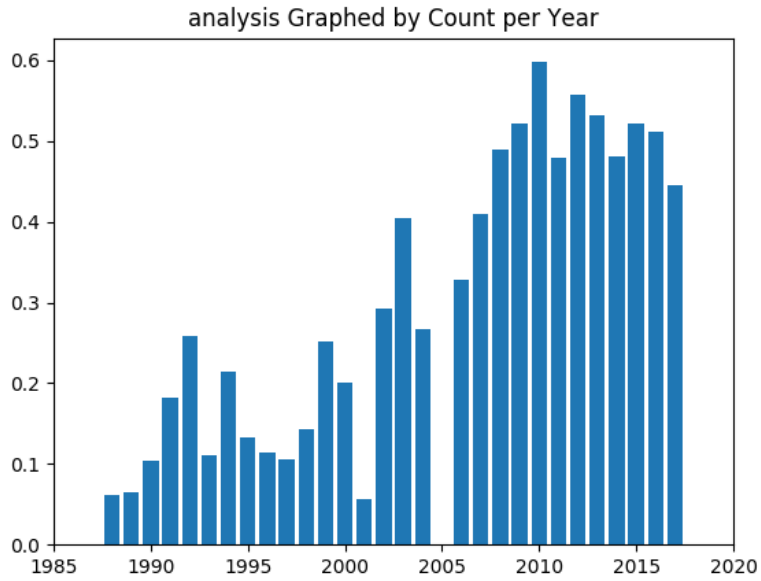

Figure A.32. Graph showing the appearance of “analysis” over the years in the abstract collections

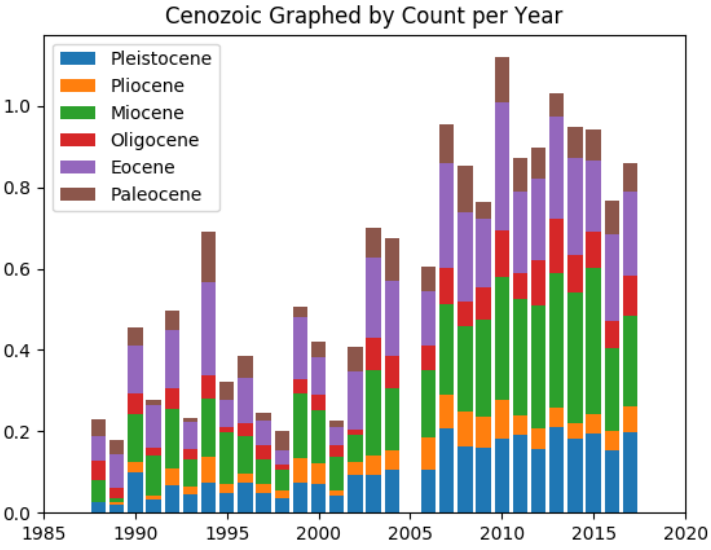

Figure A.33. Graph showing the appearance of Cenozoic sub-groups over the years in the abstract collections

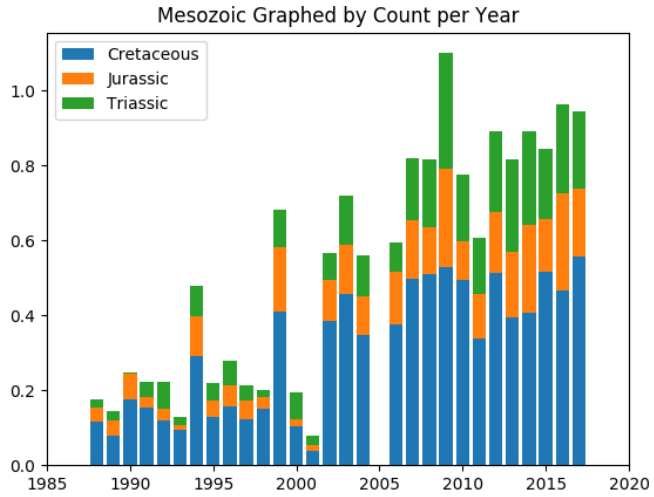

Figure A.34. Graph showing the appearance of Mesozoic sub-groups over the years in the abstract collections. Cretaceous seems to overshadow Jurassic and Triassic over the years.

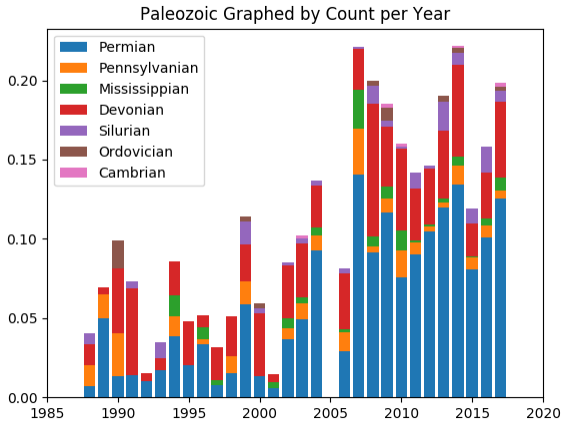

Figure A.35. Graph showing the appearance of Paleozoic sub-groups over the years in the abstract collections. Cretaceous seems to overshadow the other sub-groups over the years.

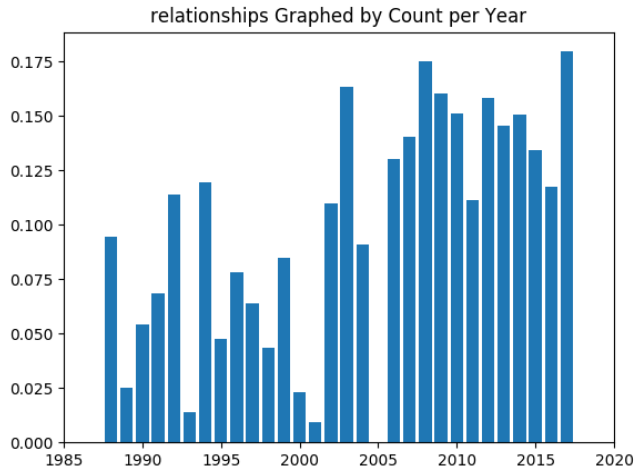

Figure A.36. Graph with a more common word; the range of the graph can be deceptive here.

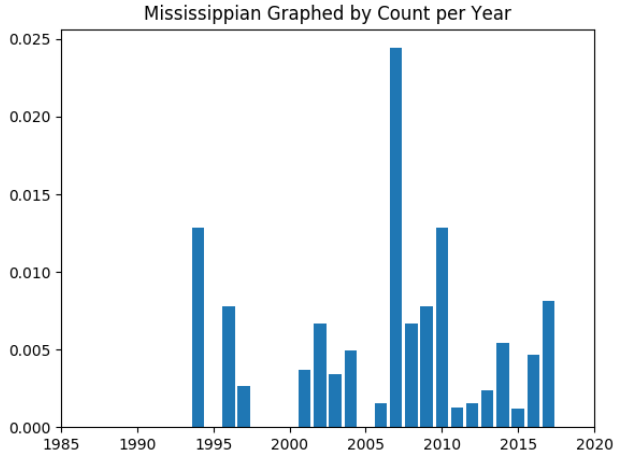

Figure A.37. Graph with an interesting spike in the middle of the data set

10.2 Appendix B: Code Samples

catFiles.py

catFiles.py 〉 No Selection

```
1  import os
2  filenames = os.listdir("SVPAbstracts/")
3  with open('catAbFile.txt', 'w') as outfile:
4      for fname in filenames:
5          print(fname)
6          with open('SVPAbstracts/' + fname) as infile:
7              for line in infile:
8                  outfile.write(line)
9
```

Figure B.1. Concatenation Script: It will read every file from the SVPAbstract directory and write the contents to the catABFile.txt in the current directory.

htmlRemovalScript.py

htmlRemovalScript.py 〉 No Selection

```
1  import os
2  import re
3  with open('htmlRemoveFile.txt', 'w') as outfile:
4      with open('catAbFile.txt', 'r') as infile:
5              for line in infile:
6              line = re.sub('<[^<]+?>', '', line)
7              line = re.sub('(^@)+', '', line)
8                      outfile.write(line)
9
10
```

Figure B.2. Metadata scrubber script. Because the text files came from PDF and have extra meta-text in them, we run scrubber script on the concatenated file.

freqWords.py

freqWords.py 〉 No Selection

```
# Program will display a welcome message to the user
print("Welcome! This program will analyze your file to provide a word count, the top 50 words and remove the following stopwords.")

s = open('htmlRemoveFile.txt','r').read()  # Open the input file

# Program will count the characters in text file
num_chars = len(s)

# Program will count the lines in the text file
num_lines = s.count('\n')

# Program will call split with no arguments
words = s.split()
words = [x.lower() for x in words]
d = {}
for w in words:
    if w in d:
        d[w] += 1
    else:
        d[w] = 1

num_words = sum(d[w] for w in d)
from nltk.corpus import stopwords # Import the stop word list
from nltk.tokenize import wordpunct_tokenize

stop_words = set(stopwords.words('english'))
nbslist = [u'.', u'-', u'j.', u'a.', u'm.', u'session', u'technical', u'poster', u'grant', u'information', u'august', u'calgary', u'united', u'states', u'america', u'october',
    u'november', u'pm', u'may', u'jstor', u'mesa', u'society', u'dept', u'univ', u'jvp', u'september', u'1990', u'dept', u'university', u'department', u'univ' ]
stop_words = stop_words.union(nbslist)
print(stop_words)
keysofdict = list(d.keys())
for w in keysofdict:
    if w in stop_words:
        keysofdict.remove(w)
lst = []
for w in keysofdict:
    if w in d:
        lst.append((d[w],w))
lst.sort()
lst.reverse()
print('Your input file has characters = '+str(num_chars))
print('Your input file has lines = '+str(num_lines))
print('Your input file has lines = '+str(num_lines))
print('Your input file has the following words = '+str(num_words))

print('\n The 50 most frequent words are /n')

i = 1
for count, word in lst[:50]:
    print('%2s. %4s %s' %(i,count,word))
    i+= 1

```

Figure B.3. Script to count the most frequent words in our text file. It also contains the custom stopwords and tokenizing.

testwordcloud.py

testwordcloud.py 〉 No Selection

```
#!/usr/bin/env python
#Simply run this from your python launcher to generate a wordcloud.

from os import path
from wordcloud import WordCloud
from Tkinter import Tk
from tkFileDialog import askopenfilename

Tk().withdraw() # we don't want a full GUI, so keep the root window from appearing
filename = askopenfilename()
print(filename)
d = path.dirname(__file__)

# Read the whole text.
text = open(path.join(d, filename)).read()

# Generate a word cloud image
wordcloud = WordCloud().generate(text)

# Display the generated image:
# the matplotlib way:
#import matplotlib.pyplot as plt
#plt.imshow(wordcloud, interpolation='bilinear')
#plt.axis("off")

# lower max_font_size
wordcloud = WordCloud(max_font_size=40).generate(text)
"""
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
"""
# The pil way (if you don't have matplotlib)
image = wordcloud.to_image()
pngfilename = filename.split('.')
print(pngfilename[0])
image.save(pngfilename[0] + ".png")
image.show()
```

Figure B.4. Script taking textfiles and creating word cloud out of the most popular words

```
import os
import re
import numpy as np
import matplotlib.pyplot as plt
#ask for input for word to count
#initialize double array
abstractcounts = [114,148,201,222,219,193,287,234,293,385,377,390,342,303,540,600,588,606,551,654,655,600,773,701,798,651,837,916,841,860,735]
with open('KeyWordList.txt') as KeyWordList:
    for lineKey in KeyWordList.readlines():#linebreakdown
        for keyword in lineKey.split():
            #keyword = raw_input("What is the keyword?")
            #print(keyword)
            graph_data = []
            year = []
            yearcount = 1986
            filenames = os.listdir("SVPAbstractsOnly/")
            for fname in filenames:
                wordcount = 0.0
                yearcount+=1
                totalwords = 0.0 #count total words
                year.append(yearcount)
                with open('SVPAbstractsOnly/' + fname) as infile:
                    for line in infile.readlines():
                        for word in line.split():
                            if word.lower() == keyword.lower():
                                wordcount = wordcount + 1
                            totalwords = totalwords + 1 #count all the words
                graph_data.append((fname,wordcount/(abstractcounts[yearcount-1987])))
            #print(graph_data)
            yearthis = zip(*graph_data)[0]
            count = zip(*graph_data)[1]
            axes = plt.gca()
            axes.set_xlim([1985,2020])
            #axes.set_ylim([0,.75])
            plt.title(keyword+" Graphed by Count per Year")
            plt.bar(year, count, align='center')
            plt.savefig("KeywordGraphFigs/"+keyword+".png")
            plt.clf()
            plt.cla()
            plt.close()
```

Figure B.5. Script taking the text files as input, as well as a list of keywords, generating a new graph for each keyword, and putting them all into the same folder

```
import os
import re
import numpy as np
import operator
import matplotlib.pyplot as plt
#ask for input for word to count
#initialize double array
totalcount = 0
bottomcount = [0]*31
#Number of abstracts
abstractcounts = [114,148,201,222,219,193,287,234,293,385,377,390,342,303,540,600,588,606,551,654,655,600,773,701,798,651,837,916,841,860,735]
graphtitle = raw_input("What is the Grouping?")
with open(graphtitle+'.txt') as KeyWordList:
    for lineKey in KeyWordList.readlines():#linebreakdown
        for keyword in lineKey.split():
            graph_data = []
            year = []
            yearcount = 1986
            count = 0
            filenames = os.listdir("SVPAbstractsOnly/")
            for fname in filenames:
                wordcount = 0.0
                yearcount = yearcount+1
                totalwords = 0.0 #count total words
                year.append(yearcount)
                #print(fname)
                with open('SVPAbstractsOnly/' + fname) as infile:
                    for line in infile.readlines():
                        for word in line.split():
                            if word.lower() == keyword.lower():
                                wordcount = wordcount + 1
                            totalwords = totalwords + 1 #count all the words

                    infile.close()
                graph_data.append((fname,wordcount/(abstractcounts[yearcount-1987])))

            yearthis = zip(*graph_data)[0]
            count = zip(*graph_data)[1]

            plt.title(graphtitle+" Graphed by Count per Year")
            plt.bar(year, count, bottom = bottomcount, align='center', label = keyword)
            bottomcount = tuple(map(sum, zip(bottomcount, count)))
axes = plt.gca()
axes.set_xlim([1985,2020])
plt.legend()
plt.savefig("KeywordGraphFigs/"+graphtitle+".png")
```

Figure B.6. Script acting like in B.5. However, it uses a predetermined list to generate a stacked bar graph, instead of individual graphs. To generate the stacked graphs a count of the heights of each bar is kept for the stacking procedure.

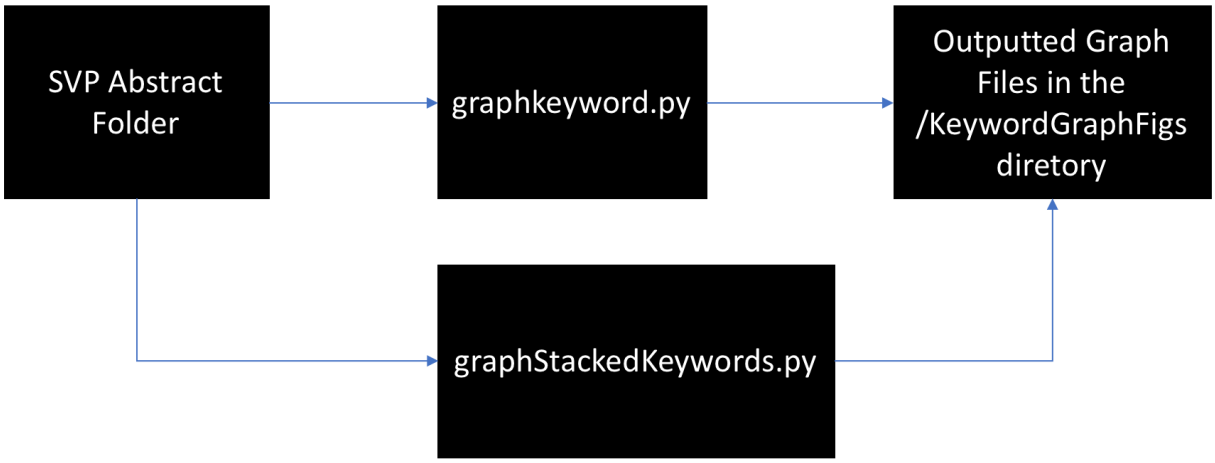

Figure B.7. Structure of our graphing code. The text files in the SVPAbstract folder will be accessed by either the graphkeyword.py script or the graphStackedKeywords.py script. Both scripts output the graph PNGs into the KeywordGraphFigs folder. If the folder does not exist, the scripts will create it.