Labeling data-reuse statements through Active Learning

Gilbert Wong
Loyola University Maryland

April 26, 2020

Table of Contents

Introduction **3**

Data science management and research platform **4**

Related work **4**

Dataset **7**

Hypothesis **11**

Data Analysis **13**

Threats to Validity **41**

Future Work **41**

Reflections **42**

Conclusion **43**

References **43**

Appendix **44**

List of Figures

1 ML pipelines [5] 6
2 Pipeline for AL framework for natural language EMR de-identification [6] . . 7
3 Simple AL workflow [7] 12
4 Word Cloud 13
5 Count of labels 15
6 Distribution of labels by year 17
7 ROC graphs for LR for various vectorizers 23
8 ROC graphs for NB for various vectorizers 25
9 ROC graphs for SVM for various vectorizers 27
10 ROC graphs for RF for various vectorizers 28
11 Box plots for comparing models using various vectorizers 29
12 Performance plots for various vectorizer methods and ML algorithms 39
13 Screenshots of the UI of the mobile application 40

Listings

1 ingest.py 44
2 download.py 46

3 ingest.py 47
4 createDataset.py 49
5 analysis.py 55

Introduction

The National Institute of Mental Health (NIMH) is the lead federal agency for research on mental disorders. The National Institute of Health (NIH) is a large agency that consists of 27 institutes and the NIMH is one of them. In turn, the NIH is an agency of the United States Department of Health and Human Services and the primary agency of the United States government responsible for biomedical and health-related research. The heath field has been an evolving area of study and with the advancement of data analytics, more physicians, and medical researchers are leaning towards the area of data science to assist them in their diagnosis, treatment, and research. However, before all the data science magic can happen, researchers, would need a good sample size of training data and this project explores a technique that could be used to lessen the burden of manually labeling data.

The problem with data science at the NIH is figuring out what data is important for their research. Many researchers reuse data for multiple areas but knowing what data is being used is important so that the proper federal funds and grants are put in the right places. This project aims to solve that. If this type of labeling is done manually, someone would take a random sample of sentences in the research papers and then figuring out whether the information is important or not. It's like searching for a needle in a haystack. What the NIMH wants to do is let a classifier tell them which statements they are most likely to gain the most information from and manually label that particular record; think of it like a feedback loop where the model is constantly being retrained as new data comes into the system. This concept is called Active Learning (AL). According to [1], given a machine learning model and a pool of unlabeled data, the goal of AL is to select which data should be annotated in order to learn the model as quickly as possible. There are an abundance of unlabeled data and manually labeling them is expensive. By using this technique, learning algorithms can actively query the user/teacher for labels. For this project, it will serve as an interactive labeling tool to allow the client to obtain more labeled data for their research. This is what makes the problem interesting since we, as data scientists, know that labeling data is expensive. The client would like to run the classifier on a mobile device and hosted on a cloud-based infrastructure that they can serve up and take input from a mobile-friendly interface. The backend runs an initial pretrained AL model where given the input, it outputs the records to label and it retrains the model to increase its performance. The data is stored in a database for future access.

The two biggest challenges for this work are figuring out how to incorporate the concept of AL into the model and deployment. However, before any modeling can happen, we need to vectorize the input, also known as normalization, into something the ML algorithm could understand. Then once the model has been created, figuring out how to deploy the odel onto a mobile app for the client. Other challenges included exploring different tokenizers that would split the text into desired tokens (as will be explained later in more detail), as well as exploring different methods of natural language processing (NLP) to prepare the text data before it is fed into a ML model. By giving the client a way to label research papers in a more efficient manner, it allows them to tie the reuse data back to its original source and funding which gives them the ability to see which data is more useful so that more funding can be granted to that particular source. The NIH is at the forefront of many health-related discoveries and by having this particular knowledge at their disposure, they can justify the need for more funding/grants and make more medical breakthroughs that would save lives.

Data science management and research platform

The overall plan of analysis was building three programs; one to ingest three common separated value (CSV) files (describe in more detail in a later section) into a SQLite database and then a second program to use a simple JOIN SQL statement to pull only the IDs of interest and passing that ID to an API call so that the proper data is retrieved. The final program is the main analysis which included preprocessing of the data and building a series of ML models. The primary language used is Python and several key packages used were scikit-learn, nltk (Natural Language toolkit), spacy [2] (NLP library), and modAL [3] (AL wrapper around scikit-learn). In terms of what the client asked for about being able to label data on their phones, the application is written in React Native. The three programs described weren't complex (about 1000 lines of code) which involves downloading, filtering, preprocessing and modeling (different ML models were explored). The mobile application, which is still in the development stage, is a little more complex since it involves not only the frontend interface (look and feel of the app) but it also incorporates a backend for the UI to interact with. Some details of the mobile application will be touched upon at the end of this paper (but wouldn't go into much detail since it's not the major point of this project and it's something the client wanted as bonus).

There were no special hardware or software used in the project; only specific packages for NLP and AL. No interactive development environments (IDE) were used and everything was done via the command line and a basic text editor. In the proposal, one topic that wasn't touched on was how to turn text data into an array of normalized values that the ML models could understand. In the beggining, basic vectorizers were explored but as the project progressed and more research was done, a huge part of the preparation stage was missed and that was how these vectorizers worked in preprocessing and tokenizing of the data. The text data consists of email addresses, urls, etc. that weren't properly tokenized using the defaults of the vectorizers. Therefore, more research was done and it turns out that the vectorizers offered by scikit-learn allows the user to create custom preprocessors and tokenizers to pass to the initialization of the classes. Therefore, simple custom functions were created to account for this and some parts of the code had to be refactored.

To measure the effectiveness of the model, different metrics are looked at. The most common metrics is accuracy but we will see later that accuracy is not always the best metrics. The other metrics are recall, precision and F1 (which is a combination of recall and precision). Depending on the task and the data presented, different metrics are used to measure the effectiveness of the model.

Related work

Researchers, regardless of the domain, have done a lot of work in the areas of labeling. According to an article [4], the market for data labeling passed $500 million in 2018 and it will reach $1.2 billion by 2023. It accounts for 80 percent of the time spent building Artificial

Intelligence (A.I.) technology. It is ironic that ML, tool used for the automation of tasks and processes, often starts with the highly manual process of data labeling. The task of creating labels to teach computers new tasks is quickly becoming the blue collar job of the 21st century. There are work being done to create this ability to allow one to automate the process for creating data labels; this is highly desirable from a cost, time and from an ethical standpoint (although this is creating thousands of jobs, workers are often underpaid and exploited). Aside from AL, there is another python library called Snorkel. Difference between AL and Snorkel is that AL introduces human expertise into the loop to smartly label a small set of data where Snorkel removes humans from the labeling process. Snorkel is a really innovation concept since it creates a series of messy label functions and combine these in an intelligent way to build labels for a dataset. [4] The labels then could be used to train a ML model in the same way as a standard ML workflow.

Snorkel has been around since 2016 and continues to improve. It is used by many big names in the industry such as Google, IBM, and Intel. Version 0.9 of the library came out in 2019 which provided a more sophisticated way of building a label model, as well as a suite of well documented tutorials covering all of the key features. Although this library is an innovative concept but the process is very simple. It consists of mainly four steps.

- This first step is optional but is helpful for reviewing performance of the final model. Create a small subset of golden labels for items within the dataset.
- Write a series of label functions which define the different classes across the training data.
- Build a label model and apply this to the dataset to create a set of labels.
- Use labels in the normal ML pipelines.

The process is iterative and most likely, the user would evaluate the results and re-think and refine the label functions to improve the output.

This project is focused on the concept of AL and after some research, there have been a number of publications regarding AL published by the NIH. Two specific articles found talks about interactive ML (iML) for health informatics [5], and using AL for electronic medical record de-identification [6]. As a research institute, the NIH publishes many research papers (available at *https://www.ncbi.nlm.nih.gov/pmc/*; it is also where all the text data for this project comes from) that touches on a variety of health related topics. Interestingly, the source, [5], was cited 25 times by other publications and [6] was cited once.

Many ML researchers concentrate on automatic ML (aML) which works really well for speech recognition, recommender systems, or autonomous vehicles but these automatic approaches only works from big data with many training sets. However, according to [5], in the health domain, there are many instances where they have to deal with a small number of data sets or rare events, where aML isn't efficient due to insufficient training samples. Therefore, the concept of AL can help since it optimizes the learning behavior through interactions with agents (where agents could be a human). This human-in-the-loop can be extremely important in solving computationally hard problems such as subspace clustering, protein folding or k-anonymization of health data, where human expertise can help to reduce an exponential search space. This need of AL in the health domain is crucial because biomedical data sets are full of uncertainty, incompleteness, etc. and they can contain missing data, noisy data, dirty data, unwanted data, and more importantly, some problems in this domain are hard, which makes fully automated approaches difficult or even impossible. In the below figure, there is four ML workflows. A illustrates an unsupervised pipeline, B supervised, C semi-supervised, and D shows the iML approach where one input data, pre-process the data, human agent(s) interacting with the computational agent(s)), and final check done by the human expert. Scenario A illustrates the pipeline where learning is fully automatic and does not require a human to manually to label the data. Scenario B is where humans are providing labels for the training data then selecting features to feed the algorithm to learn (the more samples the better) and then the human expert can check results at the end of the pipeline.)Scenario C is kind of a mixture of A and B where one mixes labeled and unlabeled data, so that the algorithm can find labels according to a similarity measure to one of the given groups. Scenario D, as briefly mentioned above, is where the human expert is seen as an agent directly involved in the actual learning phase, step-by-step influencing measures, however, many questions remain open and needs further research, in terms of evaluation, robustness, etc.

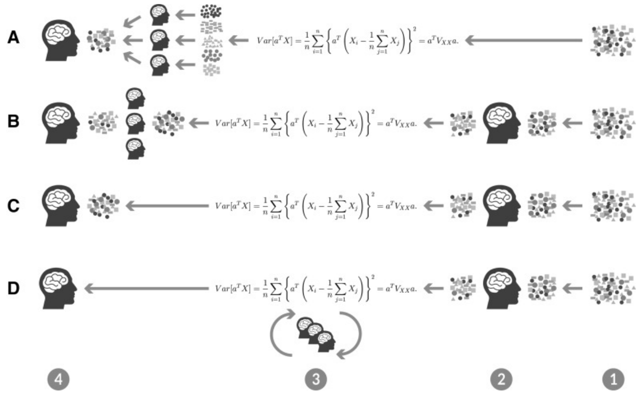

A
B
C
D
$Var[a^TX] = \frac{1}{n}\sum_{i=1}^{n}\left\{a^T\left(X_i - \frac{1}{n}\sum_{j=1}^{n}X_j\right)\right\}^2 = a^TV_{XX}a.$
4
3
2
1

Figure 1: ML pipelines [5]

There are still many evidence that humans sometimes still outperform ML algorithms (ex. diagnostic radiologic imaging). The goal is to integrate the physicians high-level expert knowledge into the retrieval process by acquiring his/her relevance judgments regarding a set of initial retrieval results. The reason for the popularity of aML approaches is that it is much better to evaludate and therefore, more publishable, as opposed to iML, where correct experiments and evaluations are not just more difficult and time-consuming but very difficult to replicate, due to the fact that human agents are subjective compared to data, algorithms, and computational agents.

Privacy is a huge issue in today's world where data is more widely assessible. Therefore, in the medical field, ensuring privacy for the patients remains one of the primary challenges to disseminating such data. [6] To protect someone's privacy, heath care organizations rely upon the de-identification standard of the Privacy Rule of the Health Insurance Portability and Accountability of 1996. It is straightforward to protect health information (PHI) (personal name, dates of birth, geocodes of residence) but it is more challenging to do so for clinical information where data is more free or semi-structured form. As mentioned earlier, ML can assist in such task, however, this ML approach requires the presense of sufficiently high-quality training data and it must be accomplished under limited budgets to informatics team running such systems. Therefore, the paper focuses on using AL in the process to reduce the overall cost for annotation and support the establishment of a more scable de-identification pipeline.

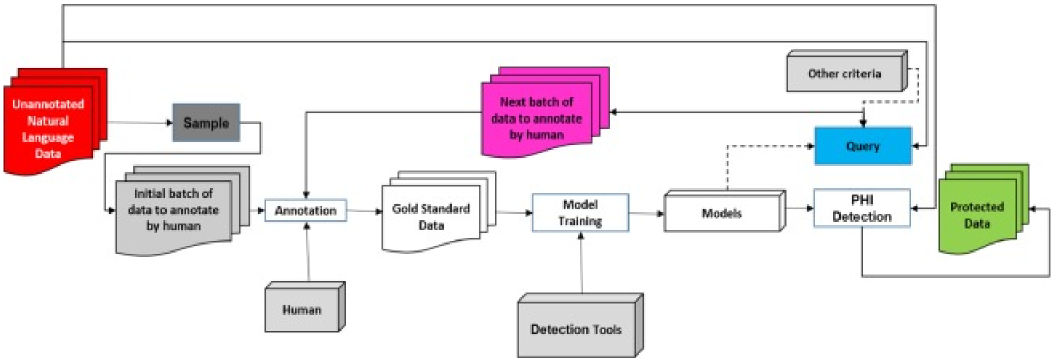


Figure 2: Pipeline for AL framework for natural language EMR de-identification [6]

Figure 2 above illustrates the overall pipeline of such tasks which is very similar to the pipeline for this project where, instead of tagging PHI, we're tagging reuse statements from research papers. Similarly, we use a small batch of data that is selected randomly from the dataset as the starting point of the AL then humans manually tag the data in the initial batch of data to create a gold standard for model traning. The concept of AL has proven to be an effective tool in named entity recognition tasks in clinical text and studies show that AL is more efficient than more passive learning. It also suggests that uncertainty sampling (which will be touched upon later) was the best strategy for reducing the annotation cost; therefore, it should be taken account when evaluating the performance of AL. There are still lots of research being done to see the full impact of what AL can do for the health related field but it has already proven useful in some research areas and this concept will continue to evolve.

Everything discussed in this paper will be available in my GIthub account (the scripts are shown in the Appendix), **https://github.com/gwong11/data-science-project**. There is a README.md file that briefly goes into the different stages of the project and what files are used for what. Therefore, anything discussed here can be reproduced with some slight modifications to the code, as some portions are tailored to my development environment.

Dataset

The client provided me an initial labeled data file of 1878 records, where 20 percent is reserved for testing/validation. Therefore, the initial training set consists of 1502 records and the testing/validation set consists of 376 records. The testing/validation set will be used to measure the performance of the model.)For the training set, there are 1401 records labeled ’0’, meaning it’s not a data reuse statement, and 101 records labeled ’1’, meaning it’s a data reuse statement. For the testing/validation set, there are 348 records labeled '0', and 28 records labeled '1'.

The client showed me how to obtain additional unlabeled data which will be used for the AL learning phase. The representational state transfer (REST) application programming interface ()API) documentation to retrieve additional data is found here: **https://www.ncbi.nlm.nih.gov/research/bionlp/APIs/BioC-PMC/**. The API call structure looks like this: **https://www.ncbi.nlm.nih.gov/research/bionlp/RESTful/pmcoa.cgi/ BioC_[format]/[ID]/[encoding]** where the format is either xml or json, the ID is either a PubMed ID (PMID) or a PMC ID (PMCID), and encoding is either Unicode or ASCII. For the purpose of this project, the format will be in json, ID will be the PMID (since after some preliminary analysis, not all research papers contain a PMCID), and the encoding will be unicode. [8]

In order to retrieve the PMIDs to pass to the REST API, the following link, **https://federalreporter.nih.gov/FileDownload**, is needed. In this link, there are three types of information: projects, publications, and link tables. The projects contain the project numbers and for this project, only the AGENCY='NIH' and IC_CENTER='NIMH' are of interest and that eliminates a lot of research papers and cuts down on the number of API calls. The publications contain the PMID and the link table is what links both the projects and publications. Looking at all the PMIDs, there are about three million research papers but not all the papers are related to NIH (or NIMH). Therefore, the projects are used to only find project numbers which are only associated with NIH (or NIMH) and with this information, we can pull back all the PMIDs that is specifically associated with NIH (or NIMH), as described above. Each of the three types of information is a separate common-separated file organized by year and for this project, research papers from 2008-2018 will be retrieved. Given the research questions that the NIMH is interested in, papers published before 2008 are unlikely to contain any data-sharing statements, since most of the data-sharing platforms did not exist at that time.

To simplify things, the three types of information, described above, were ingested into a SQLite3 database. Since this stage is just to retrieve the data, the database of choice needs to be fast and lightweight and didn't require a database management system with lots of features. This offers a fast way to retrieve all the PMIDs where the PROJECT_NUMBER from projects equals the PROJECT_NUMBER from link tables using SQL's INNER JOIN. The database schema consists of three tables (project, publication, and linktable). The schema is shown below:

- project
  - project_id PRIMARY KEY
  - project_terms
  - project_title
  - department
  - agency
  - ic_center

  - project_number NOT NULL
  - project_start_date
  - project_end_date
  - contact_pi_project_leader
  - other_pis
  - congressional_district
  - duns_number
  - organization_name
  - organization_city
  - organization_state
  - organization_zip
  - organization_country
  - budget_start_date
  - budget_end_date
  - cfda_code
  - fy
  - fy_total_cost
  - fy_total_cost_sub_projects

- publication
  - affiliation
  - author_list
  - country
  - issn
  - journal_issue
  - journal_title
  - journal_title_abbr
  - journal_volume
  - lang
  - page_number
  - pmc_id
  - pmid PRIMARY KEY NOT NULL
  - pub_date
  - pub_title

  - pub_year

- linktable
  - pmid PRIMARY KEY NOT NULL
  - project_number NOT NULL
  - FOREIGN KEY (pmid) REFERENCES publications (pmid)
  - FOREIGN KEY (project_number) REFERENCES projects (project_number)

As shown above, the schema isn't complicated; the only restriction is that the primary key needs to be defined so that it's unique and there are no duplicates. Other than that, everything else could be NULL since the other information isn't required to retrieve the data. However, before executing the below SQL statement to return all the PMIDs, some data cleaning with the project number in the project table was done since the format is not the same across the tables. For example, the following project number, 5R01MH069619-04, from the project table will not be present in the linktable. The actual project number that appears in the linktable only has 11 characters, which means the first character, 5, and anything after the dash needs to be ignored. However, there are some records which, instead of a dash, has a space and some number in parenthesis like this ' (01)' so that needs to be ignored too.

- SELECT DISTINCT lt.pmid, lt.project_number FROM linktable AS lt INNER JOIN project AS p ON lt.project_number = p.project_number

After cleaning up the project number in the project table, the result of the SQL statement above returned 55,305 research papers. Not all PMIDs (from the list) are available; therefore, after going through 55,305 papers, the following results were obtained:

- Out of 55,305 papers, 40,808 papers were successfully retrieved.
- The results are written to a JSON file and to prevent the JSON being too large, each JSON contains about 2000-3000 JSON objects, which is about 120MB – 190MB, with 2-3 smaller JSON files. There are a total of 22 JSON files.
- The total data size is 3.5G.

After collecting the dataset, here are the fields and structure of the JSON object:

- date
- source
- infons
- documents
  - passages
    - text
    - offset

    - relations
    - infons
      - comment
      - source
      - name_#
      - type
      - section_type
      - volume
      - source
      - year
      - issue
      - (various other fields about the passage)
    - sentences: []
    - annotations: []
  - infons
    - license
  - id
  - relations: []
- key

From gathering the data, the research papers come from the U.S. National Library of Medicine, which is managed by the National Center for Biotechnology Information (NCBI) and available in BioC format (format used for biomedical text processing); It consists of a large number of full text research articles. All the articles from the API are available in the PMC Open Access Subset (**https://www.ncbi.nlm.nih.gov/pmc/tools/openftlist/**) and the PMC Author Manuscript Collection (**https://ncbi.nlm.nih.gov/pmc/about/mscollection/**). The articles are also available on an FTP service and articles in the API are updated within 24 hours of the files being modified.

The client provided initial requirements to filter text and extract sentences that are likely to contain data sharing information. The new retrieved data was run against a series of regular expressions (looking for different keywords) and out of 40,808 papers, 2674 records matched the criteria. Out of the 2674 records, there were 1261 unique papers. The papers were broken out into individual sentences, as it seems to result in better performance. According to the client, the series of regular rexpressions is a good rough approximation of finding statements that are likely targets for data sharing. The AL pretrained model will be used to go back and look at the 40,808 papers and see if there were something not captured by the regular expression. Before the papers were run against the regular expressions, a suitability test was performed to see if it makes sense to include that text in the JSON object. For example, looking at the JSON object structure, if the 'section_type' under 'infons' is equal to 'REF' or 'TITLE', it can be ignored. Or if title is in 'type' under 'infons', that can be ignored too. This is because things like tables, section titles, or references will probably not likely to contain any section of text relating to data reuse.

Hypothesis

Before AL, four different ML algorithms were ran against the dataset. They are Logistic Regression (LR), Naives Bayes (NB), Support Vector Machine (SVM) and Random Forest (RF). Since there are only two class labels, which makes this a binary classification problem, the theory is that LR and SVM would perform best. A binary classifier LR should be pretty powerful with a limited sample size and it has low parameters to tune, unlike the other more advanced algorithms. The purpose of running this experiment first is to see how the algorithms perform with the given dataset. As described in the previous section, the initial dataset is imbalanced with a majority of the records labeled as ’0’. With an imbalanced dataset like this, the hypothesis is that accuracy is going to be very high (over 90%). The reason being that the positive class (data reuse) is outnumbered by the negative class; therefore, accuracy is not a good measure for assessing model performance and other metrics are looked at to provide a complete assessment of model performance. There are ways to handle imbalanced data and one of them is a method called Synthetic Minority Oversampling Technique (SMOTE). Briefly, this method works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line. It will select a random example from the minority class and find its k nearest minority class neighbors. The synthetic instance is then created by choosing one of the k nearest neighbors b at random and connecting a and b to form a line segment in the feature space. In this project, SMOTE is not used since the goal is to use the concept of AL.

The main hypothesis in AL is that if a learning algorithm can choose the data it wants to learn from, it can perform better than traditional methods. As this particular method adds feedback to the workflow, it will constantly learn as new data comes in, which will increase the model performance.

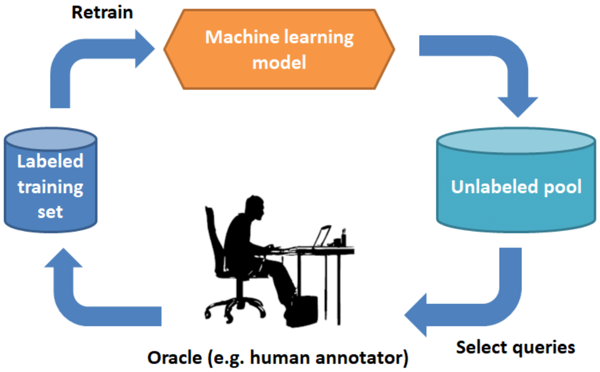


Figure 3: Simple AL workflow [7]

Figure 1 and 2, discussed earlier, explains in more detail of what an AL workflow looks like. Figure 3 above shows a simplified workflow of AL which is easier to understand to grasp the concept. The method works because it’s constantly taking an unlabeled pool of data, manually labeling them, and retraining the model. Therefore, this process, as it goes through multiple iterations of the unlabeled pool, the performance of the model increases. The oracle, shown in the figure, interacts with the model through a mobile application, as mentioned earlier, and explain in detail later.

A combination of metrics and graphs are used to test the theories. A classification report, consisting of all the metrics (accuracy, recall, precision, and F1) are analyzed. Also, a series of Receiver Operating Characteristic (ROC) graphs are analyzed and a boxplot is used to compare the different algorithms. The git repository, mentioned earlier, explains in detail the different files and ways to reproduce the data and results.

Data Analysis

Before creating the ML models, I did some preliminary analysis to get an idea of what the initial labeled data looks like. Before going into that, it's a good idea to see what kind of data did the regular expressions capture. Figure 4 below is a word cloud that captures the data sources that appeared in the new data, acquired using the method described earlier. As illustrated, a source called dbgap seems to be the most popular source. It's interesting to see that not every regular expression got a hit. You can check out the list of regular expressions in the analysis.py file in the git repository (also listed in the Appendix).

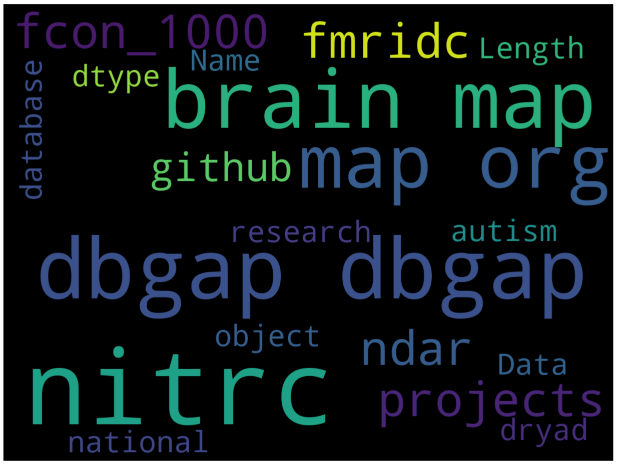


Figure 4: Word Cloud

The initial training set consists of 1893 records. The fields present are doi, data_reuse, paper_offset, pmcid, section, text, Journal Title, Year, DOI, and PMID. For the purpose of this analysis, the fields of interest are Year, PMID, data_reuse, and text. The only missing values appear in the 'text' field and since 'text' is the primary field used for this analysis, missing values need to be removed. Below shows that 15 records have missing text. After removing the missing values, there are 1878 records, in which, there are 684 unique papers. Since the 'text' field is a string, any basic imputations can't be used so they were removed. There's a table below that shows the year, PMID, and data reuse, where text is missing. Note that the records that are missing have data reuse labeled as '0'. There's already an imbalanced problem amongst the labels; therefore, losing a few dominant class records is fine. As shown below, there are no apparent patterns in the years where text is missing. The only thing I notice is that 2017 seems to have the most missing records but it's not an indication that data platforms, although started being used after 2008, are more heavily used in the later years, as these platforms advanced.

```
Count of missing values:
Year           0
PMID           0
data_reuse     0
text          15
dtype: int64

Rows where text is missing:
       Year     PMID  data_reuse text
915   2017  28569390           0  NaN
921   2017  27618273           0  NaN
1038  2016  26678596           0  NaN
1052  2013  23880391           0  NaN
1106  2010  28781389           0  NaN
1225  2017  28830029           0  NaN
1284  2013  24123049           0  NaN
1305  2017  28592562           0  NaN
1335  2017  29186694           0  NaN
1411  2015  26034955           0  NaN
1511  2009  19282707           0  NaN
1583  2014  25183549           0  NaN
1791  2014  25089330           0  NaN
1827  2015  25598502           0  NaN
1872  2013  23410851           0  NaN
```

Before splitting the data into training and testing/validaton, we can see below that 1749 records are in the ’not a data reuse’ category and only 129 records are in the ’data reuse’ category. Looking at the normalized percentage, ’0’ accounts for 93% of the data and ’1’ only accounts 7% of the data. Figure 5 below, visually, shows you the label distribution.

```
Count by data_reuse:

              Year  PMID  text
data_reuse
0            1749  1749  1749
1             129   129   129

Normalized percentage by data_reuse:

0    0.93131
1    0.06869
Name: data_reuse, dtype: float64
```

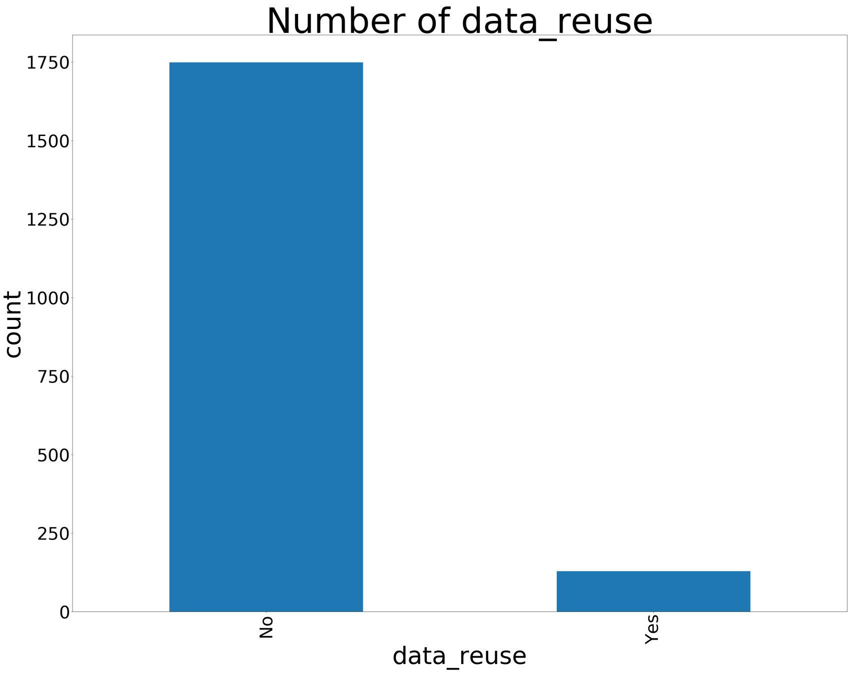


Figure 5: Count of labels

Further preliminary analysis shows the data by year, as well as a breakdown of the PMIDs with the top 10 most records. We can see a trend of which year sees a drastic change in the amount of data reuse statements present in the papers. There were no data reuse statements in 2008 or 2009; that's probably because the data platforms are just starting out and people are still getting used to it. It's interesting to see that 2017 accounts for 29% of the data and 25 records are marked as a data reuse statement, in which I would expect as the years increase, there are more data reuse statements. The year 2018 only accounts for the 8% of the data but you can see that the use of data platforms start getting traction after 2013. Another interesting data point to look at is what papers are contain the most records. There are a few papers that have more than 10 records. Besides the papers being written during the time that data platforms are being more utilized, other reasons could be that these particular papers entails researchers' writing style and the topic itself could be complex. Not all the recods in these papers is a data reuse statement but the probability of having one or more is higher than those papers with fewer records. It's just something worth mentioning, as the subject matter expert (SME) might find it interesting.

```
Count by Year:

:       PMID  data_reuse  text
Year
2008     6            6     6
2009    36           36    36
2010    86           86    86
```

```
2011    57          57    57
2012   152         152   152
2013   183         183   183
2014   159         159   159
2015   237         237   237
2016   270         270   270
2017   538         538   538
2018   154         154   154

Normalized percentage by Year:

2017    0.286475
2016    0.143770
2015    0.126198
2013    0.097444
2014    0.084665
2018    0.082002
2012    0.080937
2010    0.045793
2011    0.030351
2009    0.019169
2008    0.003195
Name: Year, dtype: float64

Count of data_reuse by Year:

                  PMID  text
Year data_reuse
2008 0               6     6
2009 0              36    36
2010 0              80    80
     1               6     6
2011 0              54    54
     1               3     3
2012 0             143   143
     1               9     9
2013 0             163   163
     1              20    20
2014 0             147   147
     1              12    12
2015 0             215   215
     1              22    22
2016 0             253   253
     1              17    17
2017 0             513   513
```

```
     1            25    25
2018 0           139   139
     1            15    15

Normalized percentage by pmid:

20697030    0.010650
25383518    0.010117
27933461    0.009052
28502781    0.009052
28506465    0.008520
23160490    0.008520
28695822    0.007987
30016334    0.007455
22438822    0.007455
28159617    0.006922
Name: PMID, dtype: float64
```

The figure below shows you, visually, the distribution of the labels by year. As illustrated by the tables above, it's just a way to visually see that 2017 contains the most records and the most data reuse statements present.

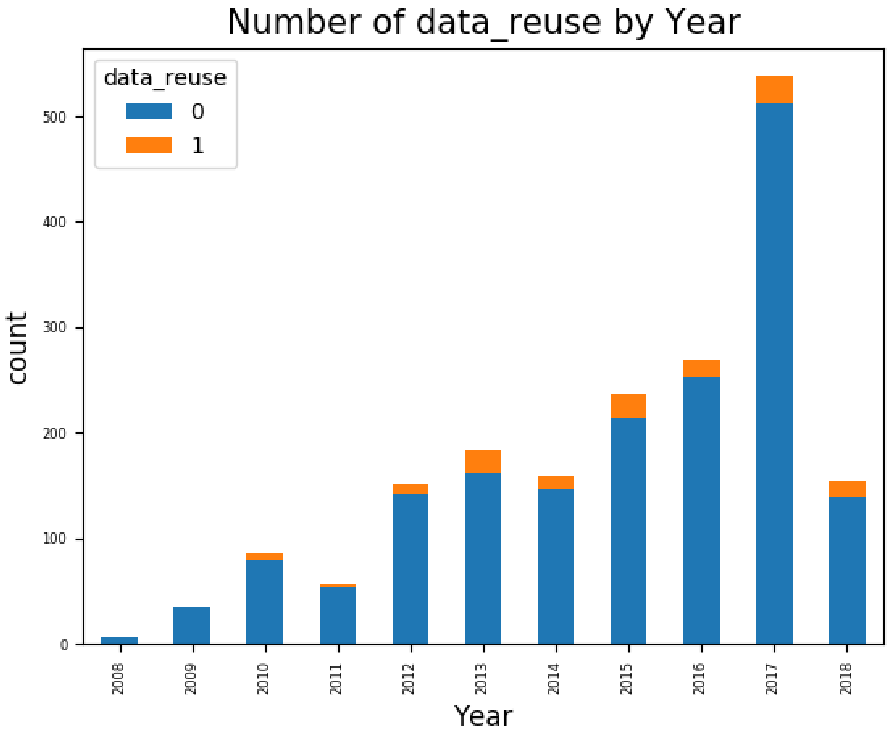


Figure 6: Distribution of labels by year

After understanding the initial training set a little more, it's time to split the data into training and testing/validation. As mentioned earlier, 20% of the data is used for testing/validation. The next step is to take the text and normalize it. This is also known as vectorization, which is the process of converting words into numbers. ML algorithms don't understand text and needs to be converted into something it could understand. There are numerous vectorizers and the two vectorizers explored is CountVectorizer and TfidfVectorizer. The CountVectorizer is a simple way to both tokenize a collection of text documents and build a vocabulary of known words. It also encodes new documents using that vocabulary. On the other hand, the TfidfVectorizer, which stands for Term frequency-inverse document frequency, is a numerical statistical method that is intended to reflect how important a word is to a document in a collection or corpus. Variations of the tf-idf weighting scheme are often used by seach engines in scoring and ranking a document's relevance given a query. Below shows the formula to calculate the tfidf score. The first term is the number of times a word appears in a document, divided by the total number of words in that document. The second term is computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

$$TF - IDF\ score = TF(i,j) * IDF(i),\ where$$

$$TF(i,j) = \frac{Term\ i\ frequency\ in\ document\ j}{Total\ words\ in\ docment\ j},\ and$$

$$IDF(i) = \log_2 \frac{Total\ documents}{documents\ with\ term\ i}$$

The vectorizer classes provide a fit, transform, and fit_transform method. The 'fit' method computes the mean and standard deviation (std) for the training data, then the 'transform' method is used to encode the data as a vector, to be passed on to the next stage in the pipeline. The 'fit_transform' method does both the fitting and the transforming in one optimized step. In this experiement, the training set is used to fit the data, while the 'transform' method is used for both the training and testing/validation set. We wouldn't want to fit the data using both the training and testing/validation set, as that would just recalculate the mean and std, which would mess up the training of the model. As you can see, the only difference between the two vectorizers is that TfidfVectorizer returns floats while the CountVectorizer returns ints. This is to be expected since TfidfVectorizer assigns a score while CountVectorizer counts.

In NLP, there are some steps to prep the data before ML can happen. The first step is sentence segmentation. I did an experiment to include paragraphs per record but found that sentences worked better. The reason is with the regular expressions, a paragraph could consist of multiple data reuse statements and running the regular expression on a paragraph could miss some of that; therefore, sentences turned out to be more suitable. The NLTK library consists of a sent_tokenize method which takes in text and splits them up into sentences. This is known as tokenization, as it is splitting text into tokens. Tokens could be paragraphs, sentences, or individual words. The next step is to perform word tokenization so that the records could be normalized into vectors, as explained above with the vectorizer methods. There are methods to perform word tokenization, however, the vectorizer methods does that as part of its initiation. It includes the preprocessing and tokenization step. The default tokenizers did not properly create the vocabulary from the documents, as it didn't capture urls, email addresses, etc. It broke down those special strings into separate tokens which is not ideal. Therefore, a custom tokenizer was written to solve this issue. In a typical NLP workflow, during word tokenization, we would perform stemming/lemmatization on the words, to reduce the vocabulary and eliminate words that are similar in nature. Stemming, by definition, is the process of reducing inflection in words to their root forms such as mapping a group of words to the same stem even if the stem itself is not a valid word in the language. During the experiment, stemming has its issues. For example, it would take a word like ’languages’ and return ’languag’, which is not an actual word. On the other hand, lemmatization is the process of reducing the inflected words properly ensuring that the root word belongs to the language. The lemmatization root word is called lemma. It is the canonical form, dictionary form, or citation form of a set of words. Therefore, lemmatization was embedded in the custom tokenizer (also known as the analyzer in vectorizer method term). The custom tokenizer can also be used to perform some kind of feature selection. For example, we can pass in a predefined dictionary for words we’re interested in (and it will just normalize those and ignore the others) but for this project, we are not interested in that since we want to get a complete vocabulary of what is in the papers. Below shows the code for the custom tokenizer class, which is used to pass into the vectorizer class initialization as a parameter. As shown in the code, a library called spacy is used. Spacy is a free open-source library for NLP in Python. It features named entity recognition (NER), part-of-speech (POS) tagging, dependency parsing, word vectors and more. It is designed for production, however, many of the features that makes spacy such a robust library is not used. The library supports a variety of languages and in this project, it’s being used to define the english language so it knows what is a proper word in the english dictionary.

```
# create a custom analyzer class
class MyAnalyzer(object):

# load spaCy’s english model and define the tokenizer/lemmatizer
def __init__(self):
spacy.load(’en’)
self.lemmatizer_ = spacy.lang.en.English()

# allow the class instance to be called just like
# just like a function and applies the preprocessing and
# tokenize the document
def __call__(self, doc):
doc_clean = unescape(doc).lower()
tokens = self.lemmatizer_(doc_clean)
return([token.lemma_ for token in tokens])
```

The result is shown below after normalizing the data, where the vectorizer class is initialized with the custom tokenizer and fitted with the training data then transformed both the training and testing/validating data. The result is a sparse matrix, in which most of the elements are zero. This isn’t very intersting since what you see are zeros but the one thing to notice is the shape of the matrix (particularly, the number of columns). Since we didn’t create a predefined dictionary, the number of features will be equal to the vocabulary size found during vectorization. Looking at the matrix below, there are 8433 features in its vocabulary. However, as you will see later, not all 8433 features are actual words. The TfidfVectorizer consists of the same number of features; only difference is the numbers in the matrix are different.

```
Count Vectorizer (train):
[[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
...
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]]
(1502, 8433)
```

```
Count Vectorizer (test):
[[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
...
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]]
(376, 8433)
```

Below shows a few tokens in the vocabulary and you can see some interesting tokens. It’s a dictionary and the value of the key is an index. The dictionary shows, what looks like some data sources not captured by the regular expression. As shown in the word cloud above, ’nitrc’ is one of the more popular data sources but there are others such as www.targetvalidation.org, www.stari.org, etc., that also looks like sources of data. Although it captures the urls well, the custom tokenizer isn’t perfect and the papers contain a lot of special characters and math symbols which could be ignored. However, it will do for this project and in the AL portion, when we add the additional papers, retrieved from the REST API, it will show some more interesting tokens.

```
{..., youngest’: 8308, ’younger’: 8307, ’young’: 8306, ’you’: 8305,
’york’: 8304, ’yjm’: 8303, ’yields’: 8302, ’yielded’: 8301,
’ygtss’: 8300, ’yfp’: 8299, ’yet’: 8298, yes’: 8297, ’yeo’: 8296,
’yellow’: 8295, ’years’: 8294, ’yearly’: 8293, ’year’: 8292,
’ydeza’: 8291, ’ycentro’: 8290, ’yale’: 8289, ’y’: 8288, ’xy’: 8287,
’xxx’: 8286, ’xx’: 8285, ’xue’: 8284, ’xt’: 8283, ’xnat’: 8282, ’xiao’: 8281,
’xi(s’: 8280, ’xi’: 8279, ’xgen’: 8278, ’xenografts’: 8277, ’xenograft’: 8276,
’x8’: 8275, ’x4’: 8274, ’x-’: 8273, ’x’: 8272, ’w|y’: 8271, ’wyeth’: 8270,
’www.yeastgenome.org’: 8269, ’www.wtccc.org.uk’: 8268,
’www.tgen.org’: 8267, ’www.targetvalidation.org’: 8266, ’www.sfari.org’: 8265,
’www.scandb.org’: 8264, ’www.r-project.org’: 8263, ’www.qiagen.com’: 8262,
’www.python.org’: 8261, ’www.pubatlas.org’: 8260, ’
www.psychologicalscience.org’: 8259, ’www.phenowiki.org’: 8258, ’
www.oasis-brains.org’: 8257, ’www.nitrc.org’: 8256, ...}
```

After normalizing the data, let’s take a look at the ML models and see how well each one did. The first ML algorithm to explore is the LR. LR is a statistical model that uses a logistic function to model a binary dependent variable. In this type of analysis, LR is estimating the parameters of a logistic model, which is a form of binary regression. Below shows the output of running the LR and we can see that the accuracy (both for using the CountVectorizer and TfidfVectorizer) is very high, at 95% and 94%, respectively. As mentioned, accuracy is not a good metrics to measure performance because the data is extremely imbalanced. A confusion matrix and a classification report is shown below. A confusion matrix shows the actual and predicted labels; it is a performance measurement for ML classification problem where the output can be two or more classes. The diagonal are the true negative and true positive values. The top right is the false positive (also known as Type 1 error) and the bottom left is the false negative (also known as Type 2 error). False positive means the model predicted positive (data reuse statement) but it is false (not a date reuse statement). False negative means the model predicted negative (not a data reuse statement) but it is false (data reuse statement).

Looking at the confusion matrix below, it shows that using the CountVectorizer performed better with LR. There are 344 records correctly labeled as not a data reuse statement and 14 records labeled as a data reuse statement, as opposed to using TfidfVectorizer where 348 records are correctly labeled and 4 records incorrectly labeled. Note that the type 1 and 2 errors, where using CountVectorizer produced 4 Type 1 errors and 14 Type 2 errors, and using TfidfVectorizer produced 0 Type 1 errors and 24 Type 2 errors. To further look at the performance of this mode, the classification reports below are analyzed. The recall metric is the ability of the model to identify all relevant instances, the precision metric is the ability of a model to return only relevant instances, and the F1 score is a single metric that combines recall and precision using the harmonic mean. It is important to understand that as recall increases, precision decreases, or vice versa. The reason for the F1 score is because it is difficult to compare two models with low precision and high recall or vice versa, therefore, it helps to measure recall and precision at the same time, where it uses the harmonic mean in place of arithmetic mean by punishing the extreme values more. The formulas below shows how each metrics are calculated.

$$recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

The classification report also contains an extra column called ’support’, which is the number of occurance of the given class in the dataset; it’s the same numbers explained above, in the dataset session. There are also two additional metrics, macro avg and weighted avg. The macro-avg calculates the scores separated by class but not using weights for the aggregation, which results in a bigger penalization when the model does not perform well with the minority classes (which is exactly what we want when there is an imbalanced problem). On the other hand, weighted-avg caculates the scores for each class independently but when it adds them together, it uses a weight that depends on the number of true labels of each class. This will always be higher since the score to identify '0' is so high.

$$score_{macro-avg} = 0.5 * score_{class\ 0} + 0.5 * score_{class\ 1}$$

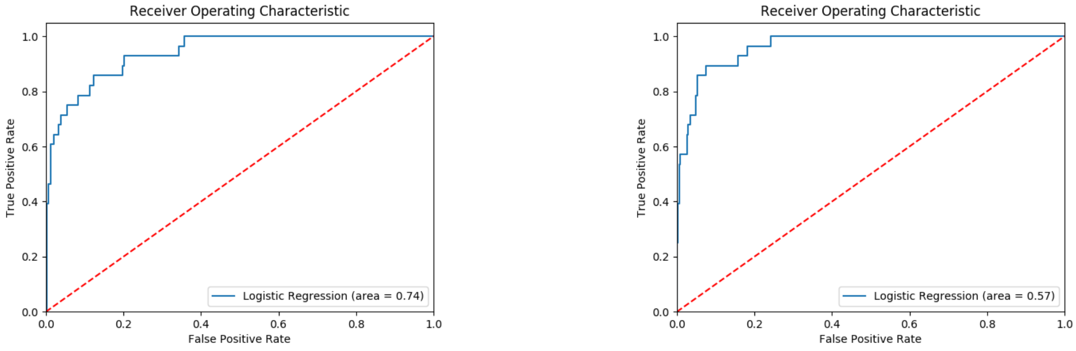
$$score_{weighted-avg} = 0.998 * score_{class\ 0} + 0.002 * score_{class\ 1}$$

For the classification report, numbers associated with '0' is less of an interest since we know the dataset is imbalanced and the numbers will be high; also, we care about identifying data reuse statements, not lack of. In the first classification report, it shows that the ability of predicting '1', recall is 0.5, precision is 0.78, and F1 score is 0.61. In this project, as shown in the metric formulas above, we want to try to maximize recall because we want to find all the sentences that contain data reuse sources. Knowing what to maximize is important because by maximizing one, we are sacrificing the others, or to find an optimal blend, the F1 score is a great metrics to look at. The second classification report shows lower numbers, where recall is only 0.14 and F1 is 0.25; therefore, that particular model isn't great at identifying reuse statements. Looking at the macro-avg, the first report gives 0.74 (for recall), as opposed to the second report, which only gives 0.57. Therefore, CountVectorizer is better than TfidfVectorizer, when doing LR.

```
Logistic Regression:
Accuracy (CountVectorizer): 0.9521276595744681
Confusion matrix :
[[344   4]
[ 14  14]]
Classification report (CountVectorizer):
                precision    recall  f1-score   support

        0       0.96          0.99      0.97        348
        1       0.78          0.50      0.61         28

accuracy                              0.95      376
macro avg          0.87       0.74      0.79      376
weighted avg       0.95       0.95      0.95      376

Graph saved:  /Users/G/Loyola/Spring2020/DS796/count_log_roc_graph.png

Model saved:  /Users/G/Loyola/Spring2020/DS796/finalized_model_count_log.sav

Accuracy (TfidfVectorizer): 0.9361702127659575
```

```
Confusion matrix :
[[348   0]
 [ 24   4]]
Classification report (TfidfVectorizer):
               precision    recall  f1-score   support

           0       0.94      1.00      0.97       348
           1       1.00      0.14      0.25        28

    accuracy                           0.94       376
   macro avg       0.97      0.57      0.61       376
weighted avg       0.94      0.94      0.91       376

Graph saved:  /Users/G/Loyola/Spring2020/DS796/tfidf_log_roc_graph.png

Model saved:  /Users/G/Loyola/Spring2020/DS796/finalized_model_tfidf_log.sav
```

In Figure 7 below, two ROC curves are shown. The ROC curve is used to plot the true positive rate (TPR) versus the false positive rate (FPR) as a function of the model's threshold for classifying a positive. The area under the curve (AUC), shown, is another metric to caculate the overall performance of a model based on area under the ROC curve. The ROC curve shows how the recall vs precision relationship changes as we vary the threshold for identifying a positive in the model. The AUC falls between 0 and 1 with a higher number indicating better classification performance. In the graphs below, the AUC on the left is 0.74, while AUC on the right is 0.57; therefore, this shows the first model achieves better performance.



(a) ROC graph for LR using CountVectorizer (b) ROC graph for LR using TfidfVectorizer

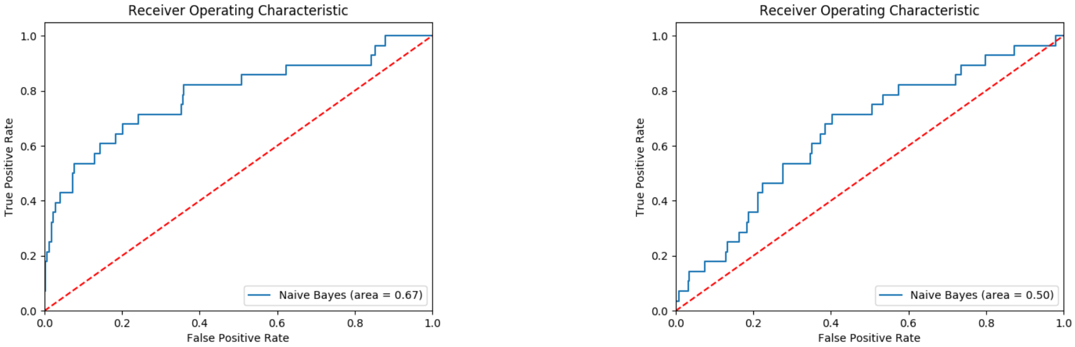
Figure 7: ROC graphs for LR for various vectorizers

The next ML algorithm ran is NB. NB uses the Bayes' Theorem, which predicts membership probabilities for each class such as the probability that given record belongs to a particular class; the class with the highest probability is considered as the most likely class. Bayes' theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event. There are five NB algorithms in scikit-learn and the one being used in the Multinomial NB. The multinomial NB is a specialized version of NB that is designed more for text documents; it explicitly models the word counts and adjusts the underlying calculations to deal within. A simple NB could be used to but it only models a document as the presence and absence of particular words, which isn't what we want. Looking the results below to see how well it predicts a data reuse statement, we can see that using CountVectorizer, the recall is only 0.36, with a F1 score of 0.43. Using TfidfVectorizer, we see that there are no correctly identified data reuse statements, thus you see the warning message. The recall and F1 score are both zero, therefore, this algorithm is not good for this problem. Although NB is mostly used in text classifications but it usually works best with problems having multiple classes. The main difference between NB and LR is that in NB, the model is specified so that both the data and the labels are dependent, while in LR, only the labels are dependent.

```
Naive Bayes:
Accuracy (CountVectorizer): 0.9308510638297872
Confusion matrix :
[[340   8]
 [ 18  10]]
Classification report (CountVectorizer):
            precision    recall  f1-score   support

       0       0.95      0.98      0.96       348
       1       0.56      0.36      0.43        28

accuracy                           0.93       376
macro avg      0.75      0.67      0.70       376
weighted avg   0.92      0.93      0.92       376

Graph saved:  /Users/G/Loyola/Spring2020/DS796/count_nb_roc_graph.png

Model saved:  /Users/G/Loyola/Spring2020/DS796/finalized_model_count_nb.sav

Accuracy (TfidfVectorizer): 0.925531914893617
Confusion matrix :
[[348   0]
 [ 28   0]]
Classification report (TfidfVectorizer):
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/
sklearn/metrics/_classification.py:1272: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use ‘zero_division‘ parameter to control this behavior.
```

```
             precision    recall  f1-score   support

           0       0.93      1.00      0.96       348
           1       0.00      0.00      0.00        28

accuracy                           0.93       376
macro avg          0.46      0.50      0.48       376
weighted avg       0.86      0.93      0.89       376

Graph saved:  /Users/G/Loyola/Spring2020/DS796/tfidf_nb_roc_graph.png

Model saved:  /Users/G/Loyola/Spring2020/DS796/finalized_model_tfidf_nb.sav
```

Although we determined that NB is not a good algorithm for this project, it's interesting to, visually, see why. Figure 8 below shows two ROC curves, where the curve on the left uses CountVectorizer and the curve on the right uses TfidfVectorizer. It clearly illustrates that the curve on the left has an AUC of 0.67, while the curve on the right has an AUC of 0.50. The curve on the right sees that the blue line is very close to the red line, which means the model is not better than guessing.



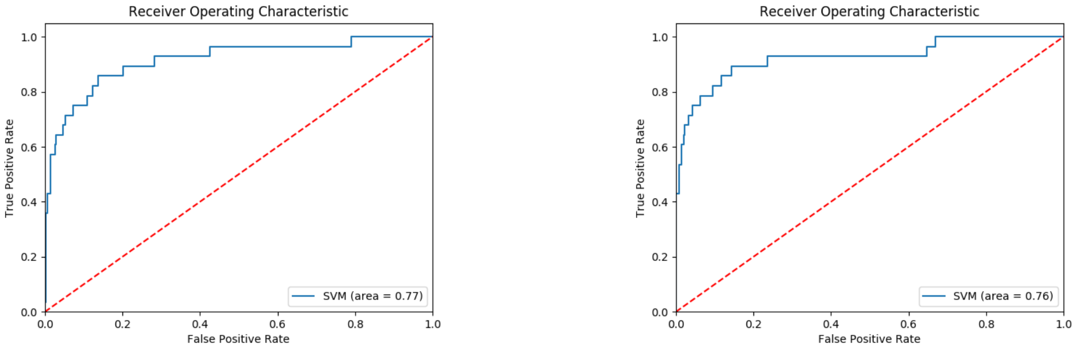
(a) ROC graph for NB using CountVectorizer (b) ROC graph for NB using TfidfVectorizer

Figure 8: ROC graphs for NB for various vectorizers

The third algorithm ran is SVM. SVM is a supervised ML model that uses classification algorithms for two-group classification problems. It is good for a text classification problem. This algorithm requires a kernel; the default is Radial Basis Function (RBF), which is not a parametric model and the complexity grows with the size of the training set). The RBF kernel is ideal for non-linear problems. The kernel that is used here is the linear kernel, which is a parametric model and ideal for linear problems. Looking at the results below, we can see for CountVectorizer, the recall is 0.57 and the F1 score is 0.62. The macro-avg is 0.77 (for recall). In the TfidfVectorizer experiment, the recall is 0.54 and the F1 score is 0.65. The macro-avg is 0.76 (for recall). If balancing both precision and recall, the F1 score is 0.03 higher for TfidfVectorizer. Therefore, for this algorithm, both vectorizer methods performed about the same, overall, as opposed to LR, where the CountVectorizer out performed the TfidfVectorizer.

```
Support Vector Machines:
Accuracy (CountVectorizer): 0.9468085106382979
Confusion matrix :
[[340   0]
 [ 12   16]]
Classification report (CountVectorizer):
            precision    recall  f1-score   support

       0       0.97      0.98      0.97       348
       1       0.67      0.57      0.62        28

accuracy                           0.95       376
macro avg      0.82      0.77      0.79       376
weighted avg   0.94      0.95      0.94       376

Graph saved:  /Users/G/Loyola/Spring2020/DS796/count_svm_roc_graph.png

Model saved:  /Users/G/Loyola/Spring2020/DS796/finalized_model_count_svm.sav

Accuracy (TfidfVectorizer): 0.9468085106382979
Confusion matrix :
[[345   3]
 [ 13   15]]
Classification report (TfidfVectorizer):
            precision    recall  f1-score   support

        0       0.96      0.99      0.98       348
        1       0.83      0.54      0.65        28

accuracy                            0.96       376
macro avg       0.90      0.76      0.81       376
weighted avg    0.95      0.96      0.95       376

Graph saved:  /Users/G/Loyola/Spring2020/DS796/tfidf_svm_roc_graph.png

Model saved:  /Users/G/Loyola/Spring2020/DS796/finalized_model_tfidf_svm.sav
```

Looking at Figure 9 below, we can see the ROC curves for both CountVectorizer and TfidfVectorizer. We can clearly see that both curves have similar AUCs, at 0.77 and 0.76, respectively. This is an acceptable model and there are no major differences between the vectorizer methods. A visual comparison of the different models is shown at the end of this session but just to compare both the SVM and LR; SVM works well with unstructured and semi-structured data like text and images, while LR works with already identified independent variables. SVM is based on geometrical properties of the data while LR is based on statistical approaches.



(a) ROC graph for SVM using CountVectorizer (b) ROC graph for SVM using TfidfVectorizer

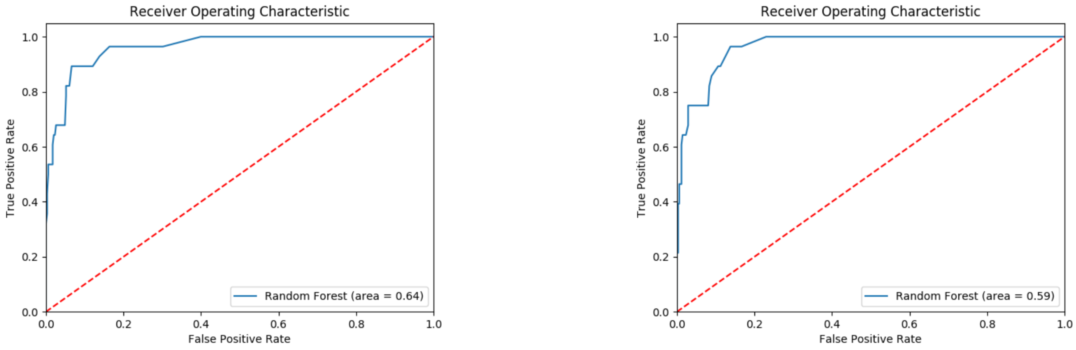
Figure 9: ROC graphs for SVM for various vectorizers

The final algorithm ran is RF. RF is an ensemble learning method for classification, regression, and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the record. There are quite a few parameters during the initialization of RF, however, for the purpose of this project, the defaults are used. One particular parameter worth mentioning is the criterion, which by default is set to gini. The criterion specifies the function to use to measure the quality of a split. Gini is the probability of a random sample being classified incorrectly if we randomly pick a label according to the distribution in a branch. The other criterion is called entropy, which is a measurement of information or lack of. The information gain is calculated by making a split. Overall, RF provides higher performance and this type of algorithm will handle missing values and maintain the accuracy of a large proportion of data (which is not an issue in this project). The advantage of RF is that if there are more trees, it won't allow overfitting trees in the model. However, looking at the results below, the performance of this algorithm isn't good (for either vectorizers). For CountVectorizer, the recall is 0.29 and the F1 score is 0.44, which makes the macro-avg, 0.64 (for recall). For TfidfVectorizer, the recall is 0.18 and the F1 score is 0.30, which makes the macro-avg, 0.59 (for recall). Therefore, these are not good models.

```
Random Forest:
Accuracy (CountVectorizer): 0.9468085106382979
Confusion matrix :
[[348   0]
 [ 20   8]]
Classification report (CountVectorizer):
               precision    recall  f1-score   support

         0      0.95       1.00       0.97        348
         1      1.00       0.29       0.44         28

accuracy                              0.95        376
macro avg       0.97       0.64       0.71        376
```

```
weighted avg    0.95      0.95      0.93      376

Graph saved:  /Users/G/Loyola/Spring2020/DS796/count_rf_roc_graph.png

Random Forest:
Model saved:  /Users/G/Loyola/Spring2020/DS796/finalized_model_count_rf.sav

Accuracy (TfidfVectorizer): 0.9441489361702128
Confusion matrix :
[[348   0]
 [ 23   5]]
Classification report (TfidfVectorizer):
             precision    recall  f1-score   support

         0      0.94      1.00      0.97      348
         1      1.00      0.18      0.30      28

accuracy                            0.94      376
macro avg       0.97      0.59      0.64      376
weighted avg    0.94      0.94      0.92      376

Graph saved:  /Users/G/Loyola/Spring2020/DS796/tfidf_rf_roc_graph.png

Model saved:  /Users/G/Loyola/Spring2020/DS796/finalized_model_tfidf_rf.sav
```

Looking at Figure 10 below, both curves look the same as they similar AUCs of 0.64 and 0.59, respectively. Comparing with the other poorly performed algorithm, NB is performed a little better, using CountVectorizer. However, overall, these curves show that although the blue lines are farther away from the red line, these aren't good models.

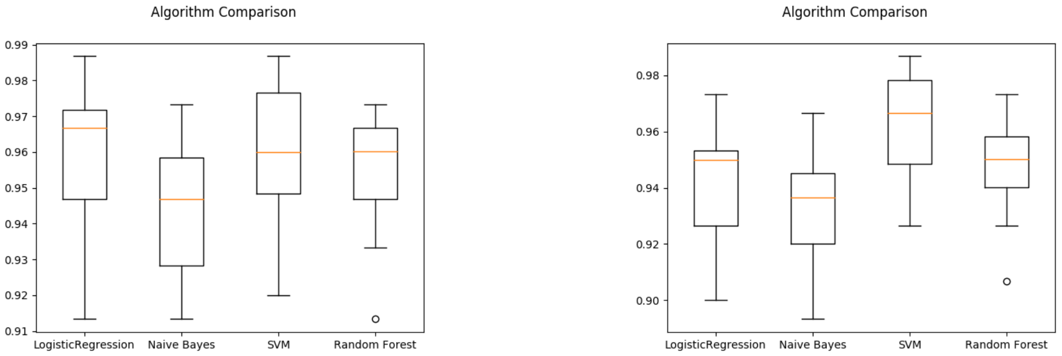


(a) ROC graph for RF using CountVectorizer (b) ROC graph for RF using TfidfVectorizer

Figure 10: ROC graphs for RF for various vectorizers

For the final part of this normal ML analysis part, let's compare the different algorithms used. The key to a fair comparison of ML algorithms is ensuring that each algorithm is evaluated in the same way on the same data. A method called 10-fold cross validation procedure is used to evaluate each algorithm, importantly configured with the same random seed to ensure that the same splits to the training data are performed and that each algorithms is evaludated in precisely the same way. Cross-validation is a technique to evaluate models by partitioning original sample into a training set to train the model, and a test set to evaluate it. The accuracy can be ignored below but it's interesting to see the std and widely spread the data is across algorithms. Figure 11 below clearly sees the spread across algorithms. We can see that SVM is the better algorithm, with LR coming in closeby, and it deserves further exploration. NB is the worst performer out of the four. This analysis is crucial as it will set the stage in AL, to determine which algorithm to use for training the model. The script allows the user to set the ML algorithm to use for AL but in this case, we know to use either LR or SVM and it should create a good enough model for labeling.

```
Comparing Models:
Using Count:
LogisticRegression: 0.958044 (0.021503)
Naive Bayes: 0.944088 (0.019076)
SVM: 0.958706 (0.021872)
Random Forest: 0.954062 (0.018240)
Graph saved: /Users/G/Loyola/Spring2020/DS796/compare_models_boxplot_count.png

Using TFIDF:
LogisticRegression: 0.940751 (0.020504)
Naive Bayes: 0.932759 (0.019176)
SVM: 0.962040 (0.020230)
Random Forest: 0.946742 (0.018605)
Graph saved: /Users/G/Loyola/Spring2020/DS796/compare_models_boxplot_tfidf.png
```



(a) Boxplot for comparing models using CountVectorizer

(b) Boxplot for comparing models using TfidfVectorizer

Figure 11: Box plots for comparing models using various vectorizers

Next is to explore AL and how it helps the client obtain more label data for their research. Before going into the AL experiment, let's look at the help menu, produced by the script, below. Not going to explain each option (you can check them out in git or the Appendix) but note option '4'. In option '4', you can choose 'regular', which is what was analyzed above. The option 'regular' runs through all the four ML algorithms, however, choosing 'active', it is more interactive and more flexible. The 'active' option not only allow the user to pick which vectorizer method and algorithm to use, but it also supports the various query strategies, specific to AL. Remember that AL is querying labels for selected unlabeled samples from a large pool of data so that the classification model can achieve the target performance with much less labeled training data. There are many query strategies (as shown below) and they are described in detail in [3]. The user can experiment with different query strategies, however, for this particular project, the query strategy most suited is uncertainty sampling (US), which is set as the default. According to [3], the way US works is when you present unlabelled examples to an AL, it finds you the most useful example and presents it for you to be labelled. This is done by first calculating the usefulness of prediction for each example and then selecting an instance based on the usefulness.

```
Enter option or h for help: h

 Options available:
 1:  Filter
 2:  Read
 3:  Write
 4:  Model
     regular (using conventional Machine Learning techniques)
     active   (Accepted vectorizers (case insensitive): COUNT (Default), TFIDF,
              Accepted models (case insensitive): LR, NB, SVM, RF,
              Accepted query strategy (case insensitive): CE, CM, CU, ES, MS, US)
      LR - Logistic Regression (Default)
      NB - Naives Bayes
      SVM - Support Vector Machine
      RF - Random Forest
      CE - Classifier Entropy
      CM - Classifier Margin
      CU - Classifier Uncertainty
      ES - Entropy Sampling
      MS - Margin Sampling
      US - Uncertainty Sampling (Default)
      Default # of queries: 10
  5:  WordCloud
  q:  Quit
  h:  Help

Enter option or h for help: q
```

Taking a look at the unlabeled data, obtained from the REST API, as explained in the dataset session, there are 2674 records (1261 unique papers). Remember from above that the number of features is 8433, where now, the number of features is 8508 after adding the new data to the vectorizer. This applies to the TfidfVectorizer as well.

```
Count Vectorizer (New):
[[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
...
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]]
(2674, 8508)
```

Similar to earlier, some new tokens in the vocabulary are shown below. Since the new data is everything captured by the regular expression, we expected a lot more data sources, which is apparent in the output below. As mentioned earlier, there are data sources not captured by the regular expression and the goal is to find those. Also, the regular expressions aren't perfect and it could give false positives, therefore, it's important to allow AL to do its magic and enable the user to teach the model what is or isn't a data reuse statement by interactively labeling them. After going through this new batch of unlabelled data, the user can, confidently, input any new papers into the model and obtain lots of labeled data. It's interesting to see that a large part of the data sources comes from the same base url, but different project. For example, nitrc and github seem to be a popular data source.

```
{..., ’year’: 8427, ’yb’: 8426, ’yaw’: 8425, ’yang’: 8424, ’yale’: 8423,
’y’: 8422, ’xq28’: 8421, ’xnat21’: 8420, ’xnat’: 8419, ’xml’: 8418,
’xlstat’: 8417, ’xia’: 8416, ’xclust’: 8415, ’x’: 8414, ’wójcik’: 8413,
’www.xnat.org’: 8412, ’www.wtccc.org.uk’: 8411,
’www.wholebrainsoftware.org’: 8410, ’www.stjudebgem.org’: 8409,
’www.rust-lang.org’: 8408, ’www.roadmapepigenomics.org’: 8407,
’www.r-project.org/’: 8406, ’www.patientslikeme.com’: 8405,
’www.osf.io/5zpve/’: 8404, ’www.openbrainmap.org’: 8403,
’www.nitrc.org/projects/uofm_jhu_atlas’: 8402,
’www.nitrc.org/projects/spharm-pdm’: 8401,
’www.nitrc.org/projects/nyu_trt’: 8400,
’www.nitrc.org/projects/mrtrix\u200e’: 8399,
’www.nitrc.org/projects/mrtrix’: 8398,
’www.nitrc.org/projects/ibvd’: 8397,
’www.nitrc.org/projects/fmricpca’: 8396,
’www.nitrc.org/projects/fcon_1000/.’: 8395,
’www.nitrc.org/projects/fcon_1000/’: 8394,
’www.nitrc.org/projects/fcon_1000’: 8393,
’www.nitrc.org/projects/dtiatlasbuilder/’: 8392,
’www.nitrc.org/projects/conn’: 8391,
’www.nitrc.org/projects/cmtk’: 8390,
’www.nitrc.org/projects/bnv/’: 8389,
’www.nitrc.org/projects/artifactdetect’: 8388,
’www.nitrc.org/projects/artifact_detect/’: 8387,
’www.nitrc.org/projects/artifact_detect’: 8386,
’www.nitrc.org/projects/art’: 8385, ’www.nitrc.org’: 8384,
’www.nimhgenetics.org’: 8383, ’www.neurosynth.org’: 8382,
’www.neuroexpresso.org’: 8381, ’www.ncbi.nlm.nih.gov/gap/’: 8380,
’www.ncbi.nlm.nih.gov/gap’: 8379, ’www.ncbi.nlm.nih.gov/dbgap’: 8378,
’www.mouseconnectome.org’: 8377,
’www.loni.usc.edu/adni’: 8376,
’www.informatics.jax.org/recombinase.shtml’: 8375,
’www.humanconnectome.org/connectome/connectome-workbench.html’: 8374,
’www.humanconnectome.org’: 8373, ’www.gtexportal.org/home/’: 8372,
’www.gtexportal.org/’: 8371, ’www.gtexportal.org’: 8370,
’www.github.com/welchr/swiss’: 8369,
’www.github.com/vistalab/vistasoft’: 8368,
’www.github.com/tractatus’: 8367,
’www.github.com/nellore/runs’: 8366, ’www.github.com/mhresearchnetwork’: 8365,
’www.github.com/mb3152/brain_graphs’: 8364,
’www.github.com/lypsychlab/rsa’: 8363,
’www.github.com/junqianxulab/habenula_segmentation’: 8362,
’www.github.com/hemberg-lab/mpranator/’: 8361,
’www.github.com/epurdom/rsecpaper’: 8360,
’www.github.com/cookpa/socialdefeat’: 8359, ...}
```

Below shows the power of interactive labeling. The series of text you see between the dotted lines are what's returned during querying of the model. This particular example below is set to query the dataset 10 times using the defaults, which the vectorizer is set to CountVectorizer, ML algorithm is set to LR, and the query strategy is set to use the uncertainty sampling method. I'm no expert at identifying what's considered data reuse and what's not (this is where the client comes in and label them with this specific domain knowledge), however, since this particular batch of data came from the filtering using regular expressions, I made an assumption that these are all data reuse statements. The next few pages explore four different models using different vectorizer methods and ML algorithms.

```
----------------------------------------------------------------------------------
As an example, took advantage of data from the Human Connectome Project
(https://www.humanconnectome.org/) to probe dFC during rest in a large sample
of participants.
Is this a data reuse statement or not (1=yes, 0=no)?
1

Microarray data were acquired from the Allen Brain Atlas
(http://human.brain-map.org/well_data_files), and included a total of 1340
microarray profiles from donors H0351.2001 and H0351.2002, encompassing
the different regions of the human brain.
Is this a data reuse statement or not (1=yes, 0=no)?
```

```
1

The current study utilized the Nathan Kline Institute (NKI) test-retest (TRT)
dataset publically available via the International Neuroimaging
Data-Sharing Initiative
(INDI: http://fcon_1000.projects.nitrc.org/indi/pro/eNKI_RS_TRT/FrontPage.html).
Is this a data reuse statement or not (1=yes, 0=no)?
1

The first source was an ongoing longitudinal study on early indicators of ASD
in high-risk populations (PI: Roberts; FXS, ASIBs, typically developing), the
second was an extant database from a study examining family adaptation
to FXS (PI: Bailey; FXS) and the third was the National Database for Autism
Research (NDAR), a collective national database for autism research funded
by the National Institutes of Health (FXS, ASIBs, typically developing).
Is this a data reuse statement or not (1=yes, 0=no)?
1

The control scans were acquired at these same five sites, as well as from
seven additional sites that have provided unrestricted public access to
their data through the 1000 Functional Connectomes Project
(http://fcon_1000.projects.nitrc.org).
Is this a data reuse statement or not (1=yes, 0=no)?
1

Data sharing statement: On completion of the trial, de-identified data
will be available through the International Initiative for Impact Evaluation
(3ie)'s public data access repository on Dataverse.
Is this a data reuse statement or not (1=yes, 0=no)?
1

Sequence data from the Simons Simplex Collection were obtained via
controlled access through the National Database for Autism
Research (http://ndar.nih.gov/study.html?id=334).
Is this a data reuse statement or not (1=yes, 0=no)?
1

Raw data from ASD family (Accession pending) and SAGE control
(Accession: phs000092.v1.p1) genotyping are at NCBI dbGAP.
Is this a data reuse statement or not (1=yes, 0=no)?
1

Other relevant cases of anterograde tracing from the IP that were less
specific or otherwise less suitable for analysis include Allen Mouse
Brain Connectivity Atlas experiments 184212995 (Ntrk1-Cre), 267538735
```

```
(Erbb4-Cre) and 171019710 (Slc32a1-Cre) which are available online
(http://connectivity.brain-map.org/), and one case that did not meet
quality control criteria for addition to the online database.
Is this a data reuse statement or not (1=yes, 0=no)?
1

dbGaP accession numbers for the publicly funded genotyping are:
NHS T2D (phs000091.v2.p1), NHS BrCa (phs000147.v1.p1), ,
HPFS T2D (phs000091.v2.p1), NHS/HPFS KS (phs000460.v1.p1).
Is this a data reuse statement or not (1=yes, 0=no)?
1

Graph saved: /Users/G/Loyola/Spring2020/DS796/active_model_count_LR_performance.png

Model saved:  /Users/G/Loyola/Spring2020/DS796/active_model_count_LR.sav
-------------------------------------------------------------------------------
```

The next model, shown below, uses a CountVectorizer, but this time, it uses SVM. Notice that compared to the previous model, the records returned is different from the ones above.

```
-------------------------------------------------------------------------------
Diagrams in d and g are modified from the Allen Mouse Brain Atlas, Allen
Institute for Brain Science; available from http://mouse.brain-map.org/.
Is this a data reuse statement or not (1=yes, 0=no)?
1

Data from mouse connectivity map of Allen Brain Atlas: id 263242463,
http://connectivity.brain-map.org/).
Is this a data reuse statement or not (1=yes, 0=no)?
1

A population standard map of functional connectivity for the human brain
underlies large scale imaging initiatives such as the NIH's Human Connectome
Project and the INDI/NITRC 1000 Functional Connectome database.
Is this a data reuse statement or not (1=yes, 0=no)?
1

For RNA-seq-based eQTLs from DLPFC (Brodmann area 9, n samples = 92)
that are part of the Genotype-Tissue Expression (GTEx) Project, we utilized
those eQTLs significant after permutation (as performed by the GTEx
Consortium); these data were downloaded from the GTEx Portal
(www.gtexportal.org), corresponding to dbGaP accession number phs000424.v6.p1.
Is this a data reuse statement or not (1=yes, 0=no)?
1
```

```
This dataset were obtained from the WU-Minn Human Connectome Project database
(http://humanconnectome.org/).
Is this a data reuse statement or not (1=yes, 0=no)?
1

We confirmed the neuron-enriched expression of these transcripts using
the Allen Brain Atlas (http://mouse.brain-map.org/); of the 30 DE
neuron-enriched transcripts that had detectable expression by the Allen
Brain Atlas, only 1 was excluded for being non neuron-enriched.
Is this a data reuse statement or not (1=yes, 0=no)?
1

We downloaded the structural brain MRI data and corresponding clinical
and genetic data from baseline and follow-up from the ADNI publicly
available database (http://adni.loni.usc.edu/).
Is this a data reuse statement or not (1=yes, 0=no)?
1

Regarding the 1,185 MR data of living humans, the data from IXI Dataset
(http://brain-development.org/ixi-dataset/) and the Human Connectome Project
(http://www.humanconnectome.org/) are available from the websites.
Is this a data reuse statement or not (1=yes, 0=no)?
1

At the time the data were accessed (October, 2011), the Allen Brain Institute
(ABI, website: http://www.brain-map.org) had complete microarray gene
expression results available from two, previously healthy human male
brains (age 24 and 39 yr).
Is this a data reuse statement or not (1=yes, 0=no)?
1

fMRI data was extracted from the open-access ‘1000 Functional Connectomes Project’
(http://fcon_1000.projects.nitrc.org/) in which resting-state fMRI
scans have been aggregated from 28 sites.
Is this a data reuse statement or not (1=yes, 0=no)?
1

Graph saved: /Users/G/Loyola/Spring2020/DS796/active_model_count_SVM_performance.png

Model saved:  /Users/G/Loyola/Spring2020/DS796/active_model_count_SVM.sav
-------------------------------------------------------------------------------
```

The third model uses TfidfVectorizer and LR. The model below presents some interesting results. There are duplicate records returned. We would have to understand the algorithms behind AL to see why this particular model returns a few duplicate records. Looks like as the model learns of new data (more queries), it starts to produce different records.

```
-------------------------------------------------------------------------------
Data used in this study were obtained from the human connectome
project (HCP) database
(http://www.humanconnectome.org/) (Van Essen et al., 2013).
Is this a data reuse statement or not (1=yes, 0=no)?
1

Data used in the preparation of this article were obtained from the Alzheimer’s
Disease Neuroimaging Initiative (ADNI) database (adni.loni.usc.edu).
Is this a data reuse statement or not (1=yes, 0=no)?
1

\Data used in the preparation of this article were obtained from the Alzheimer’s
Disease Neuroimaging Initiative (ADNI) database (adni.loni.usc.edu).
Is this a data reuse statement or not (1=yes, 0=no)?
1

\Data used in the preparation of this article were obtained from the Alzheimer’s
Disease Neuroimaging Initiative (ADNI) database (adni.loni.usc.edu).
Is this a data reuse statement or not (1=yes, 0=no)?
1

Data used in the preparation of this article were obtained from the Alzheimers
Disease Neuroimaging Initiative (ADNI) database (adni.loni.usc.edu).
Is this a data reuse statement or not (1=yes, 0=no)?
1

Data used in this study were obtained from the Alzheimer’s Disease Neuroimaging
Initiative (ADNI) database (http://adni.loni.usc.edu) and approval for
this project was granted.
Is this a data reuse statement or not (1=yes, 0=no)?
1

All data for these analyses were obtained from the database of Genotypes
and Phenotypes (dbGaP) (http://www.ncbi.nlm.nih.gov/gap).
Is this a data reuse statement or not (1=yes, 0=no)?
1

The genotype data for cases and controls was obtained from Database of Genotypes
and Phenotypes (dbGaP) (http://www.ncbi.nlm.nih.gov/sites/entrez?db=gap).
Is this a data reuse statement or not (1=yes, 0=no)?
1
```

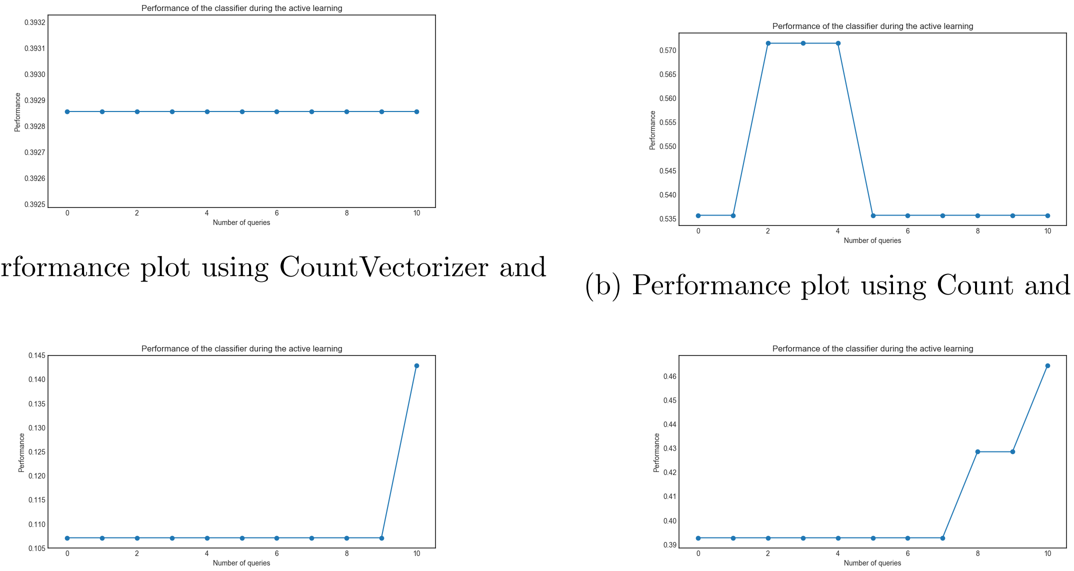
```
The 1st dataset used in this study was obtained from the Alzheimer’s Disease
Neuroimaging Initiative (ADNI) database (http://adni.loni.usc.edu).
Is this a data reuse statement or not (1=yes, 0=no)?
1

Data used in this article were obtained from the Alzheimer’s Disease
Neuroimaging Initiative database (http://adni.loni.usc.edu/).
Is this a data reuse statement or not (1=yes, 0=no)?
1

Graph saved: /Users/G/Loyola/Spring2020/DS796/active_model_tfidf_LR_performance.png

Model saved:  /Users/G/Loyola/Spring2020/DS796/active_model_tfidf_LR.sav
-------------------------------------------------------------------------------
```

The last model uses TfidfVectorizer and SVM. Any of these models presented is different and returns different results. There are many options the user can explore, including using different query strategies, to see how they differ. Next, we will look at some graphs and see the performance of the models.

```
-------------------------------------------------------------------------------
The HCP is generating a detailed in vivo mapping of functional connectivity
in a large cohort (over 1000 subjects), and is making these datasets freely
available for use by the neuroimaging community (over 200 subjects’
datasets are already acquired and publicly released { available
via humanconnectome.org).
Is this a data reuse statement or not (1=yes, 0=no)?
1

The RSFC and SC analyses used data from the Nathan Kline Institute \Rockland"
cohort, available online via the International Neuroimaging Data sharing initiative
(http://fcon_1000.projects.nitrc.org/indi/pro/nki.html).
Is this a data reuse statement or not (1=yes, 0=no)?
1

We used FSL (http://www.fmrib.ox.ac.uk) and AFNI (http://afni.nimh.nih.gov/afni)
based script libraries from the 1000 Functional Connectomes Project
(http://www.nitrc.org/projects/fcon_1000 35) for preprocessing of
resting-state scans (\fcon scripts").
Is this a data reuse statement or not (1=yes, 0=no)?
1

Genotyped data for HRS participants was downloaded from dbGAP.
Is this a data reuse statement or not (1=yes, 0=no)?
1
RS fMRI images of 153 healthy volunteers (mean age 41.1 +/- 18.0 [SD]
years; 92 male) from the NKI/Rockland sample were obtained through
the 1,000 functional connectomes project
(www.nitrc.org/projects/fcon_1000/).
Is this a data reuse statement or not (1=yes, 0=no)?
1

Data for the entire study were downloaded from
http://aging.brain-map.org/download/index, and the methods for this RNA
sequencing study can be found here:
http://help.brain-map.org/display/aging/Documentation).
Is this a data reuse statement or not (1=yes, 0=no)?
1

Diagrams in a, g and i were modified from the Allen Mouse Brain Atlas, Allen
Institute for Brain Science; available from http://mouse.brain-map.org/.
Is this a data reuse statement or not (1=yes, 0=no)?
1

A recent study explores relationships between gene expression and distributed
spatial patterns of synchronous brain activity consistently observed in resting
state (RS) fMRI (Richiardi et al.,) using microarray data from the
Allen Brain Atlas (http://human.brain-map.org; Hawrylycz et al.,).
Is this a data reuse statement or not (1=yes, 0=no)?
1

Riboprobes were designed to overlap probe designs for homologous genes in
mouse and human used in the Allen Mouse Brain Atlas
(http://mouse.brain-map.org) and Allen Human Brain Atlas
 (http://human.brain-map.org/), and cross-species comparisons were made
 to data publicly available in those databases.
Is this a data reuse statement or not (1=yes, 0=no)?
1

Data access for several cohorts used in this study was provided by the National
Center for Biotechnology Information (NCBI) database of Genotypes and
Phenotypes (dbGaP).
Is this a data reuse statement or not (1=yes, 0=no)?
1

Graph saved: /Users/G/Loyola/Spring2020/DS796/active_model_tfidf_SVM_performance.png

Model saved:  /Users/G/Loyola/Spring2020/DS796/active_model_tfidf_SVM.sav
--------------------------------------------------------------------------------
```

Figures 12(a) and 12(b) show the performance of the various AL models using CountVectorizer. As mentioned, recall works best in our case so that's what is being plotted in the figures below. We can see some interesting results with the graphs. As we know that the great advantage of AL is the ability to provide unlabelled data and label the ones that would increase the performance of the model. Figure 12 below shows some interesting things. In the first graph, the performance did not improve at all and stayed stagnant but keep in mind that the number of queries it ran is only 10. Looks like for this particular model, a large number of queries need to be set, in order to see an upward trajectory of the recall value. The second graph, which I didn't expect, is that during the first query, we start to see an upward trajectory but then several queries later, performance starts decreasing. For these two models, looks like the number of queries need to hit a particular threshold before the model start seeing improvements. The hypothesis is that as we increase the number of queries for both models and label more informative records, the performance will increase.

Figrures 12(c) and 12(d) show two graphs using the TfidfVectorizer. This is what I would expect, as the number of queries increase, we see the performance improve. For the LR model, it starts to improve after the 9th query which I would think would take more than that, but also take note of the recall value on the y-axis; it's very low. On the other hand, for the SVM model, the recall value starts at 0.39 which isn't great but as the number of queries are performed, the recall value went up to 0.46. In other words, it proves that using SVM produces the better model.



(a) Performance plot using CountVectorizer and LR

(b) Performance plot using Count and SVM

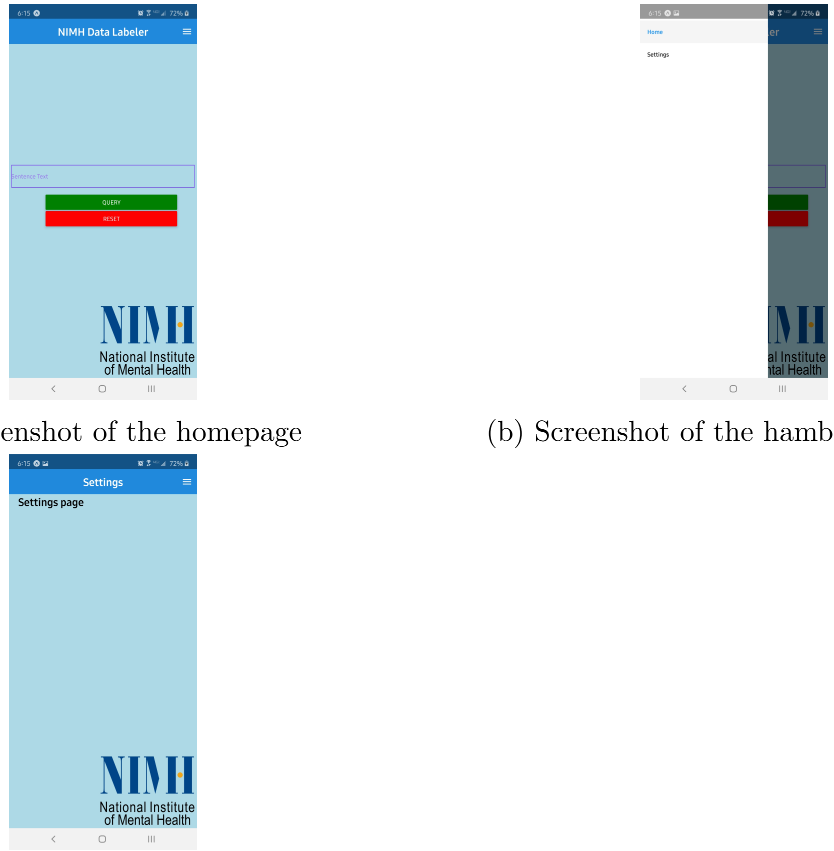
(c) Performance plot using TfidfVectorizer and LR

(d) Performance plot using Tfidf and SVM

Figure 12: Performance plots for various vectorizer methods and ML algorithms

From running a few AL models and doing the analysis on the various methods, we can clearly see what kind of vectorizer you use in the preparation of the data is important and affects the model greatly. From the two figures above, figure 12(b) looks to be the best model, even though it doesn't clearly show the AL working yet (probably due to not having enough queries). Figure 12(d) isn't bad as the recall value stopped at 0.46 and if this particular model was run with a higher query number, it would probably do just as well or better than 12(b). Overall, it seems that unlike running the regular ML workflows above, the AL models performed quite well using TfidfVectorizer. As we can see, as described above, that as the number of queries increases, the performance of the model increases. I, personally, like 12(d) as you can see the benefit of AL, but any of the models would see the benefit of AL as the model is constantly teaching itself, as it sees new data.

The last task for this project is the deployment of one of these models onto a mobile application. This won't be touched on in detail as it's still in progress and not the main focus of this project but I want to briefly talk about the idea behind it. As data scientists, we talk a lot about how to analyze data and how to make pretty graphs that we can present to leadership to drive the decision-making process. However, when talking about ML modeling, we need to talk about how to productize it and deploy it. As mentioned, the client would like to be able to use the concept of AL and be able to label records, on a mobile device (similar to what was presented above, which was ran in a terminal). The code is not listed in the Appendix due to the complexity but can be found in the git repositroy.



(a) Screenshot of the homepage

(b) Screenshot of the hamburger menu

(c) Screenshot of the settings page

Figure 13: Screenshots of the UI of the mobile application

The screenshots shown in Figure 13 above is what the mobile user interface looks like. There are only two pages, a homepage and settings page. The homepage is where the client can upload new data and then it will talk with the backend to return records for the frontend for the client to label. The labeled data is then stored in a backend database for future retrieval so that the client can keep track of all the labeled data in one spot. The plan for the settings page is to allow the client to upload a model to run, as well as be able to define the parameters such as the vectorizer method, ML algorithm, etc. (similar to option '4' described above).

Threats to Validity

The only potential weakness of this analysis is during the preprocessing and tokenizing step. As noted earlier, the custom tokenizer isn't perfect. We saw in the vocabulary that there are lots of words that could be ignored or reduced and looks like the lemmatization needs some tweaking. As concluded in the analysis, the type of vectorizer method use largely affects the performance of the model since the sentences are represented in numbers so that the ML algorithm could understand. The only assumption made during the conclusion of the data analysis is during interactive labeling of the AL model and all the records returned were considered as 'data reuse' statement, which created the performance plots shown above. It made sense to mark them as 'data reuse' statements since they were records that came out of the regular expression filtering process but as mentioned, some records could be false positives. Therefore, it's good to have the SME manually label the informative records in a large dataset.

Future Work

There are still lots of work that are interesting to explore. The first one is to explore other vectorizer methods such as HashVectorizer or Word2vec. HashVectorizer is designed for memory efficiency, therefore, instead of storing the tokens as strings, the vectorizer applies a hash function to encode them as numerical indexes. The disadvantage is that once vectorized, the features' names can no longer be retrieved. Memory isn't a big issue here as the vocabulary isn't very large but still worth some exploring. However, these vectorizers are very basic and requires lots of work to productize, therefore, Word2vec is definitely useful in helping to create the AL model. The reason is because Word2vec consists of a vast majority of models that would group the vectors of similar words together in vectorspace. On the other hand, it detects similarities mathematically. Word2vec creates vectors that are distributed numerical representations of word features; features such as the context of individual words. As we saw above that the vocabularies and the way it was preprocessed and tokenized need some work and Word2vec consists of models that would bring context to the text, which might be helpful in identifying what is data reuse and what is not. The last thing on vectorization is to read more about the spacy library. The library seems to have a lot of potential in helping improve this project since what was used is just the tip of the iceberg.

As described above, the mobile application is still in development, as it's more complex than I first imagined. It has lots of moving pieces which involves the frontend and the backend, therefore, it will take longer to complete. As mentioned, the mobile application is a nice tool to have which deploys an AL model for interactive labeling. As talked about early on, data is everywhere but obtaining labeled data is labor intensive and this will give any data scientist a way to label data efficiently and used for future research/projects.

Another interesting area worth exploring is topic modeling. Topic modeling is an unsupervised ML technique that's capable of scanning a set of documents, detecting word and phrase patterns within them, and automatically clustering word groups and similar expressions that best characterize a set of documents. I'm not an expert in the domain of this project but reading a few papers, a lot of them seems to be very similar. What topic modeling can do is to cluster similar papers together; the idea is that a single cluster would more or less have some data sources in common. Also, what I lack but the SME would have is once clustered, they can look at it and they would be able to identify data sources common to that topic.

Reflections

There were many lessons learned from doing this data science research project. When I took my first ML class and started learning about the various ML algorithms; the thing I've always been curious about is how labeled data is generated. From the school projects, we already have well defined datasets to work with, in order to learn the subject being taught but never understood how it works in real life. Now I understand the nature of the labeling problem and this project gave me a chance to experience firsthand what that problem looks like. It confirmed by original suspicion that labeling data is labor intensive and people actually get paid to label data. The other thing I have learned about this project is deploment. As briefly mentioned above, we talk a lot about using ML to aid with a particular task and how to write scripts to analyze the performance of a ML model but little is covered in terms of what deploying to a production environment looks like. How do you input new data into a pretrained model? After doing this data science project, it is more clear to what that process looks like.

The most rewarding in doing this project is learning the concept of AL. It was very rewarding when the AL was implemented and the query engine returned records back to the terminal for the user to label. There are still lots to learn about how the query strategies work and how passing in different ML algorithms return different results. A lot of work is still being done in this area and I got to see firsthand how this new technology helps in the labeling problem. I have always been interested in the health field and doing a project for the NIMH was fun and rewarding, although, the project is not directly related to any health topic. According to the client, if the labeling problem is solved, they have lots of interesting research projects to explore and that is very satisfying to hear. The labeling problem might not sound like an interesting problem but to solve it, brings a lot of opportunities.

The most challenging part was the normalization step. We only explored two basic vectorizers (Count and TFIDF) but there are others and for a production environment, it might be more suited for the client. I haven't taken any NLP classes so it took longer than expected to figure out what to do in that preparation stage. After that was figured out, although not perfect in anyway, the modeling wasn't difficult and straight forward.

Anyone doing a future project, I would suggest that communication is key. In the beginning, I communicated with the client back and forth every week, to understand the requirement and the process of getting data. However, I misunderstood the goal of the project right after I did the proposal. We had a short phone call to make sure everyone is on the same page and that's when I was given a small batch of labeled data to train the model with. In the early stages, the idea I had was when I finish running the papers against the regular expressions, the resulting records would have to be manually labeled so I have something to work with. I thought the client was going to label those for me since they're the SME and know which is data reuse and which is not. However, the client and I spoke and I told the client that I needed some initial dataset to work with and that's when I was given that initial batch. If I had voiced my ideas in the beginning, I would have saved some time. It was a learning process and doing the initial data retrieval work wasn't a total waste since I had some extra data to run it against the AL model but it wasn't necessary to get started.

Conclusion

Working with unstructured data like text has its challenges. This project shows that the vectorizer methods affect the performance of the model. With this text data, CountVectorizer resulted in better performance when performing the regular ML workflow. Both LR and SVM are the better ML algorithms compared to NB and RF. When creating the AL model, 10 queries were performed for retraining and we saw that the performance of the model started to improve, especially with the TfidfVectorizer and SVM combination. The other combinations didn't do as well according to the plots but we also didn't perform a lot of queries. The theory is that with AL modeling, as more queries are performed, the model will start to see an upward improvement of performance. AL is an interesting concept and will definitely help with the labeling problem as this technology continues to evolve, especially if the client wants to explore their research with neural networks, which requires lots of labeled data.

References

[1] Konyushkova, K, Raphael, S, Fua, P. *Learning Active Learning from Data*, ArXiv, 2017

[2] Industrial-Strength Natural Language Processing. Available at *https://spacy.io/*

[3] modAL: A modular active learning framework for Python3. Available at *https://modal-python.readthedocs.io/en/latest/index.html*

[4] Tayor, Josh (2020, March 5). No labels? No problem! towards data science. *https://towardsdatascience.com/no-labels-no-problem-30024984681d*

[5] Holzinger, A. *Interactive machine learning for health informatics: when do we need the human-in-the-loop?*, Brain Informations. 2016

[6] Li M, Scaiano M, El Emam K, Malin B. *Efficient active learning for electronic medical record de-identification.* AMIA Jt Summits Transl Sci Proc. 2019

[7] Li Y, Xie X, Shen L, Liu S. *Reversed Active Learning based Atrous DenseNet for Pathological Image Classification* ResearchGate, 2018

[8] Comeau DC, Wei CH, Islamaj Doğan R, and Lu Z. *PMC text mining subset in BioC: about 3 million full text articles and growing*, Bioinformatics, btz070, 2019

Appendix

```python
import sqlite3
from sqlite3 import Error
import pandas as pd

class DatabaseIngest:
    '''
    Initialize the database file
    '''
    def __init__(self, db_file):
        self.db_file = db_file
        self.conn = None

    '''
    Create a database connection
    '''
    def create_connection(self):
        try:
            self.conn = sqlite3.connect(self.db_file)
        except Error as e:
            print(e)

    '''
    Create table
    '''
    def create_table(self, table, schema):
        try:
            sql = 'CREATE TABLE IF NOT EXISTS ' + table +
                    ' (' + schema + ');'
            curr = self.conn.cursor()
            curr.execute(sql)
            self.conn.commit()
        except Error as e:
            print(e)

    '''
    Insert record
    '''
    def insert_record(self, table, exist_strategy, df):
        try:
            df.to_sql(table,
                    self.conn,
                    if_exists=exist_strategy,
                    index=False)
        except Error as e:
            print(e)

    '''
    Delete all record
    '''
    def delete_record(self, table):
        try:
            sql = 'DELETE FROM ' + table
            cur = self.conn.cursor()
            cur.execute(sql)
            #cur.execute('vacuum')
            self.conn.commit()
        except Error as e:
            print(e)

    '''
    Drop the table
    '''
    def drop_table(self, table):
        try:
            sql = 'DROP TABLE ' + table
            cur = self.conn.cursor()
            cur.execute(sql)
            #cur.execute('vacuum')
            self.conn.commit()
        except Error as e:
            print(e)

    '''
    Query database
    '''
    def query_record(self, sql_statement):
        try:
            df = pd.read_sql_query(sql_statement, self.conn)
            return df
        except Error as e:
            print(e)

    '''
    Commit transaction
    '''
    def commit(self):
        self.conn.commit()

    '''
    Vacuum databse
    '''
    def vac(self):
        self.conn.execute("VACUUM")

    '''
    Close database
    '''
    def close(self):
        self.conn.close()
```

Listing 1: ingest.py

```
import csv
import shutil
import os
import json
from urllib.request import urlopen
from collections import defaultdict

'''
This module retrieves files from a REST API
'''
class DownloadFiles:

    '''
    Initialize the class by providing a URL to download files
    with an optional filter date parameter
    '''
    def __init__(self, base_url):

        self.base_url = base_url
        self.res_format = 'json'
        self.papers = defaultdict(list)

    '''
    Method to do a GET to retrieve the response, given the URL
    '''
    def retrieve(self, pmid):
        paper = None
        found = True
        url = self.base_url + self.res_format + '/' + pmid + '/unicode'
        try:
            with urlopen(url) as response:
                paper = response.read().decode('utf-8')

            print(pmid + " downloaded!")
            self.papers['papers'].append(json.loads(paper))
        except:
            found = False
            print(pmid + " doesn't exist!")

        return found

    '''
    Getter to retrieve the papers dict
    '''
    def getPapersDict(self):
        return self.papers

    '''
    Clear the papers dict
    '''
    def clearPapersDict(self):
        self.papers = defaultdict(list)

    '''
    Method to write JSON string to disk
    '''
    def write(self, json_data, directory, filename):
        with open(os.path.join(directory, filename), 'w') as outfile:
            json.dump(json_data, outfile)

    '''
    Method to pretty print the JSON
    '''
    def pprint_JSON(self, json_string):
        parsed = json.loads(json_string)
        print(json.dumps(parsed, indent=4, sort_keys=True))
```

Listing 2: download.py

```
import sqlite3
from sqlite3 import Error
import pandas as pd

class DatabaseIngest:
    '''
    Initialize the database file
    '''
    def __init__(self, db_file):
        self.db_file = db_file
        self.conn = None

    '''
    Create a database connection
    '''
    def create_connection(self):
        try:
            self.conn = sqlite3.connect(self.db_file)
        except Error as e:
            print(e)

    '''
    Create table
    '''
    def create_table(self, table, schema):
        try:
            sql = 'CREATE TABLE IF NOT EXISTS ' + table +
                  ' (' + schema + ');'
            curr = self.conn.cursor()
            curr.execute(sql)
            self.conn.commit()
        except Error as e:
            print(e)

    '''
```

```python
    Insert record
    '''
    def insert_record(self, table, exist_strategy, df):
        try:
            df.to_sql(table,
                        self.conn,
                        if_exists=exist_strategy,
                        index=False)
        except Error as e:
            print(e)

    '''
    Delete all record
    '''
    def delete_record(self, table):
        try:
            sql = 'DELETE FROM ' + table
            cur = self.conn.cursor()
            cur.execute(sql)
            #cur.execute('vacuum')
            self.conn.commit()
        except Error as e:
            print(e)

    '''
    Drop the table
    '''
    def drop_table(self, table):
        try:
            sql = 'DROP TABLE ' + table
            cur = self.conn.cursor()
            cur.execute(sql)
            #cur.execute('vacuum')
            self.conn.commit()
        except Error as e:
            print(e)

    '''
    Query database
    '''
    def query_record(self, sql_statement):
        try:
            df = pd.read_sql_query(sql_statement, self.conn)
            return df
        except Error as e:
            print(e)

    '''
    Commit transaction
    '''
    def commit(self):
        self.conn.commit()

    '''
    Vacuum databse
    '''
    def vac(self):
        self.conn.execute("VACUUM")

    '''
    Close database
    '''
    def close(self):
        self.conn.close()
```

Listing 3: ingest.py

```
from download import DownloadFiles
import ingest
import time
import os
import pandas as pd

'''
The following consists of the headers required to retrieve the papers that
are funded by the NIH:

Index(['PROJECT_ID', 'PROJECT_TERMS', 'PROJECT_TITLE', 'DEPARTMENT', 'AGENCY',
       'IC_CENTER', 'PROJECT_NUMBER', 'PROJECT_START_DATE', 'PROJECT_END_DATE',
       'CONTACT_PI_PROJECT_LEADER', 'OTHER_PIS', 'CONGRESSIONAL_DISTRICT',
       'DUNS_NUMBER', 'ORGANIZATION_NAME', 'ORGANIZATION_CITY',
       'ORGANIZATION_STATE', 'ORGANIZATION_ZIP', 'ORGANIZATION_COUNTRY',
       'BUDGET_START_DATE', 'BUDGET_END_DATE', 'CFDA_CODE', 'FY',
       'FY_TOTAL_COST', 'FY_TOTAL_COST_SUB_PROJECTS'],
      dtype='object')
Index(['AFFILIATION', 'AUTHOR_LIST', 'COUNTRY', 'ISSN', 'JOURNAL_ISSUE',
       'JOURNAL_TITLE', 'JOURNAL_TITLE_ABBR', 'JOURNAL_VOLUME', 'LANG',
       'PAGE_NUMBER', 'PMC_ID', 'PMID', 'PUB_DATE', 'PUB_TITLE', 'PUB_YEAR'],
      dtype='object')
Index(['PMID', 'PROJECT_NUMBER'], dtype='object')
'''

def create(conn):

    conn.create_table('project', """
                                  project_id PRIMARY KEY,
                                  project_terms,
                                  project_title,
                                  department, agency,
                                  ic_center,
                                  project_number NOT NULL,
                                  project_start_date,
                                  project_end_date,
                                  contact_pi_project_leader,
                                  other_pis,
                                  congressional_district,
                                  duns_number,
                                  organization_name,
                                  organization_city,
                                  organization_state,
                                  organization_zip,
                                  organization_country,
                                  budget_start_date,
                                  budget_end_date,
                                  cfda_code,
                                  fy,
                                  fy_total_cost,
                                  fy_total_cost_sub_projects
                                  """)
    conn.create_table('publication', """
                                      affiliation,
                                      author_list,
                                      country,
                                      issn,
                                      journal_issue,
                                      journal_title,
                                      journal_title_abbr,
                                      journal_volume,
                                      lang,
                                      page_number,
                                      pmc_id,
                                      pmid PRIMARY KEY NOT NULL,
                                      pub_date,
                                      pub_title,
                                      pub_year
                                      """)
    conn.create_table('linktable', """
                                      pmid PRIMARY KEY NOT NULL,
                                      project_number NOT NULL,
                                      FOREIGN KEY (pmid) REFERENCES publications (pmid),
                                      FOREIGN KEY (project_number) REFERENCES projects (project_number)
                                      """)

def insert(conn, year):
    #start = time.time()

    proj_dir = '/Users/G/git/repository/data-science-project/data/projects'
    pub_dir = '/Users/G/git/repository/data-science-project/data/publications'
    link_dir = '/Users/G/git/repository/data-science-project/data/link_tables'

    # Get NIH related papers from projects
    proj_csv_file = os.path.join(proj_dir, 'FedRePORTER_PRJ_C_FY' + str(year) + '.csv')
    pub_csv_file = os.path.join(pub_dir, 'RePORTER_PUB_C_' + str(year) + '
.csv')
    link_csv_file = os.path.join(link_dir, 'FedRePORTER_PUBLNK_C_FY' + str(year) + '.csv')

    proj_data = pd.read_csv(proj_csv_file, dtype='unicode')
    pub_data = pd.read_csv(pub_csv_file, dtype='unicode', encoding='latin-1')
    link_data = pd.read_csv(link_csv_file, dtype='unicode')

    proj_data.rename(columns=lambda x: x.strip(), inplace=True)
    pub_data.rename(columns=lambda x: x.strip(), inplace=True)
    link_data.rename(columns=lambda x: x.strip(), inplace=True)

    #print(proj_data.shape)
    # Only care about projects where IC_CENTER = 'NIMH' and 'AGENCY'
    proj_data = proj_data[proj_data['AGENCY'] == 'NIH']
    proj_data = proj_data[proj_data['IC_CENTER'] == 'NIMH']

    # Remove first character of the number and anything after a dash or space
    proj_data['PROJECT_NUMBER'] = proj_data['PROJECT_NUMBER'].str[1:]
    proj_data['PROJECT_NUMBER'].replace(to_replace='[- ][()a-zA-Z0-9]*', value='', regex=True, inplace=True)

    # drop duplicates
    proj_data.drop_duplicates(subset='PROJECT_NUMBER', keep='last', inplace=True)
    pub_data.drop_duplicates(subset='PMID', keep='last', inplace=True)
    link_data.drop_duplicates(subset='PMID', keep='last', inplace=True)

    #print(proj_data.shape)
    #print(proj_data[['PROJECT_NUMBER', 'IC_CENTER', 'AGENCY']].head(20))
    #print(pub_data[['AFFILIATION', 'PMC_ID', 'PMID']].head())
    #print(link_data[['PMID', 'PROJECT_NUMBER']].head())
    #print(proj_data.columns)
    #print(pub_data.columns)
    #print(link_data.columns)

    # Create the database
    print("Creating the database: project")
    conn.insert_record('project', 'append', proj_data)
    print("Done")

    print("Creating the database: publication")
    conn.insert_record('publication', 'append', pub_data)
    print("Done")

    print("Creating the database: linktable")
    conn.insert_record('linktable', 'append', link_data)
    print("Done")

    conn.commit()

    #print('It took ', time.time()-start, 'seconds.')
```

```

def check_PMID(pmid, status_file, action, found=False):

    present = False
    content = None
    fd = open(status_file, action)
    if 'a' in action:
        if found is False:
           fd.write(pmid + " *\n")
        else:
           fd.write(pmid + "\n")
    elif 'r' in action:
        content = fd.read()
        #print("Content: \n" + content)
        if pmid in content:
            present = True

    fd.close()
    return present

def download(conn, paper_index, number_of_pmids):

    status_file = '/Users/G/git/repository/data-science-project/data/NIMH_pmids_downloads.txt'
    base_url = 'https://www.ncbi.nlm.nih.gov/research/bionlp/RESTful/pmcoa.cgi/BioC_'
    output_dir = '/Users/G/git/repository/data-science-project/data/papers'

    sql_statement = "SELECT DISTINCT lt.pmid, lt.project_number FROM linktable AS lt INNER JOIN project AS p ON lt.project_number = p.project_number"

    df = conn.query_record(sql_statement)
    count = 0

    downloadObj = DownloadFiles(base_url)
    for pmid in df['pmid'].values:
        if os.path.exists(status_file) == False:
            print(str(count) + " - Attemtping to Download: " + pmid)
            found = downloadObj.retrieve(pmid)
            print("Creating status file: " + status_file)
            check_PMID(pmid, status_file, 'a+', found)
        elif check_PMID(pmid, status_file, 'r') is True:
            print(str(count) + " - Skipping (already attempted): " + pmid)
        elif check_PMID(pmid, status_file, 'r') is False:
            print(str(count) + " - Attempting to Download: " + pmid)
            found = downloadObj.retrieve(pmid)
            check_PMID(pmid, status_file, 'a+', found)

        if count == number_of_pmids:
            json_data = downloadObj.getPapersDict()
            if json_data:
               print("Writing output file: " + os.path.join(output_dir, '
NIMH_research_papers_' + str(paper_index) + '.json'))
               downloadObj.write(downloadObj.getPapersDict(), output_dir, 'NIMH_research_papers_' + str(paper_index) + '.json')
               downloadObj.clearPapersDict()
               paper_index += 1
               count = 0
        elif count == (len(df['pmid']) - 1):
            json_data = downloadObj.getPapersDict()
            if json_data:
               print("Writing final output file: " + os.path.join(output_dir, 'NIMH_research_papers_' + str(paper_index) + '.json'))
               downloadObj.write(downloadObj.getPapersDict(), output_dir, 'NIMH_research_papers_' + str(paper_index) + '.json')
               downloadObj.clearPapersDict()
        else:
            count += 1

def reprocess(conn, status_file, paper_index, number_of_pmids):

    reprocess_status_file = '/Users/G/git/repository/data-science-project/data/NIMH_pmids_reprocess_downloads.txt'
    reprocess_pmids = []

    fd = open(status_file, 'r')

    for line in fd.readlines():
        if "*" in line:
            reprocess_pmids.append(line.split(" ")[0])

    fd.close()

    base_url = 'https://www.ncbi.nlm.nih.gov/research/bionlp/RESTful/pmcoa.cgi/BioC_'
    output_dir = '/Users/G/git/repository/data-science-project/data/papers'
    count = 0

    reprocessDownloadObj = DownloadFiles(base_url)
    for pmid in reprocess_pmids:
        if os.path.exists(reprocess_status_file) == False:
            print(str(count) + " - Attemtping to Reprocess: " + pmid)
            found = reprocessDownloadObj.retrieve(pmid)
            print("Creating reprocess status file: " + reprocess_status_file)
            check_PMID(pmid, reprocess_status_file, 'a+', found)
        elif check_PMID(pmid, reprocess_status_file, 'r') is True:
            print(str(count) + " - Skipping (already reprocessed): " + pmid)
        elif check_PMID(pmid, reprocess_status_file, 'r') is False:
            print(str(count) + " - Attempting to Reprocess: " + pmid)
            found = reprocessDownloadObj.retrieve(pmid)
            check_PMID(pmid, reprocess_status_file, 'a+', found)

        if count == number_of_pmids:
            json_data = reprocessDownloadObj.getPapersDict()
            if json_data:
                print("Writing reprocessed output file: " + os.path.join(output_dir, 'NIMH_research_papers_' + str(paper_index) + '.json'))
                reprocessDownloadObj.write(reprocessDownloadObj.getPapersDict(), output_dir, 'NIMH_research_papers_' + str(paper_index) + '.json')
                reprocessDownloadObj.clearPapersDict()
                paper_index += 1
                count = 0
        elif count == (len(reprocess_pmids) - 1):
            json_data = reprocessDownloadObj.getPapersDict()
            if json_data:
                print("Writing final reprocessed output file: " + os.path.join(output_dir, 'NIMH_research_papers_' + str(paper_index) + '.json'))
                reprocessDownloadObj.write(reprocessDownloadObj.getPapersDict(), output_dir, 'NIMH_research_papers_' + str(paper_index) + '.json')
                reprocessDownloadObj.clearPapersDict()
        else:
            count += 1

def options(argument):

    switcher = {
        '1':  'Create',
        '2':  'Append',
        '3':  'Query',
        '4':  'Delete',
        '5':  'Drop',
        '6':  'Download',
        '7':  'Reprocess',
        'q':  'Quit',
        'h':  'Help'
    }

    return switcher.get(argument, "Invalid argument")

def help():

    print("""
          Options available:
          1:  Create
          2:  Append
          3:  Query
          4:  Delete
          5:  Drop
          6:  Download
          7:  Reprocess
          q:  Quit
          h:  Help
          """)

if __name__ == '__main__':

    database = '/Users/G/git/repository/data-science-project/data/nimhresearch.db'
    conn = ingest.DatabaseIngest(database)
    conn.create_connection()

    arg = input("Enter option or h for help: ")
    while(arg != 'q'):
        if options(arg) == 'Help':
            help()
        elif options(arg) == 'Create':
            create(conn)
        elif options(arg) == 'Append':
            year = input("Enter year: ")
            insert(conn, year)
        elif options(arg) == 'Query':
            count = input("Enter number of results to display: ")
            sql = input("Enter sql statement: ")
            conn.query_record(int(count), sql)
        elif options(arg) == 'Delete':
            conn.delete_record('project')
            conn.delete_record('publication')
            conn.delete_record('linktable')
        elif options(arg) == 'Drop':
            conn.drop_table('project')
            conn.drop_table('publication')
            conn.drop_table('linktable')
        elif options(arg) == 'Download':
            paper_index = input("Enter starting index for creating files: ")
            number_of_pmids = input("Enter number of pmids to attempt: ")
            download(conn, int(paper_index), int(number_of_pmids))
        elif options(arg) == 'Reprocess':
            status_file = input("Enter status file (reprocessing): ")
            while os.path.exists(status_file) == False:
                status_file = input("File does not exist. Enter status file (reprocessing): ")
            paper_index = input("Enter starting index for creating files (reprocessing): ")
            number_of_pmids = input("Enter number of pmids to attempt (reprocessing): ")
            reprocess(conn, status_file, int(paper_index), int(number_of_pmids))

        arg = input("Enter option or h for help: ")

    conn.vac()
    conn.close()
```

Listing 4: createDataset.py

```
import json
import os
import re
import string
import pandas as pd
import numpy as np
import sys
import time
import pickle
import spacy
import collections
import plotly.express as px
import matplotlib.pyplot as plt
import statsmodels.api as sm
from spacy.tokenizer import Tokenizer
from nltk.tokenize import sent_tokenize
from PIL import Image
from html import unescape
from wordcloud import WordCloud, STOPWORDS
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
#from sklearn.pipeline import Pipeline
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import recall_score
from scipy.sparse import csr_matrix
from modAL.models import ActiveLearner
from modAL.uncertainty import classifier_entropy
from modAL.uncertainty import classifier_margin
from modAL.uncertainty import classifier_uncertainty
from modAL.uncertainty import entropy_sampling
from modAL.uncertainty import margin_sampling
from modAL.uncertainty import uncertainty_sampling

#pd.options.display.max_columns = None
#pd.set_option(’display.max_colwidth’, -1)
#pd.set_option(’display.max_rows’, None)

FILE_REGEX = re.compile(r’NIMH_research_papers_[0-9]+\.json’, re.VERBOSE)
DATA_REGEX1 = re.compile(r’(github)’, re.VERBOSE)
DATA_REGEX2 = re.compile(r’(osf\.io)’, re.VERBOSE)
DATA_REGEX3 = re.compile(r’(nda\.nih\.gov)’, re.VERBOSE)
DATA_REGEX4 = re.compile(r’(openneuro)’, re.VERBOSE)
DATA_REGEX5 = re.compile(r’[\s\’\"]+(ndar)[\s\’\"]+[?.!]*’, re.VERBOSE)
DATA_REGEX6 = re.compile(r’(national\sdatabase\sfor\sautism\sresearch)’,
   re.VERBOSE)
DATA_REGEX7 = re.compile(r’(brain-map\.org)’, re.VERBOSE)
DATA_REGEX8 = re.compile(r’(humanconnectome\.org)’, re.VERBOSE)
DATA_REGEX9 = re.compile(r’(balsa\.wustl\.edu)’, re.VERBOSE)
```

```
DATA_REGEX10 = re.compile(r'(loni\.usc\.edu)', re.VERBOSE)
DATA_REGEX11 = re.compile(r'(ida\.loni\.usc\.edu)', re.VERBOSE)
DATA_REGEX12 = re.compile(r'(fmridc)', re.VERBOSE)
DATA_REGEX13 = re.compile(r'(ccrns)', re.VERBOSE)
DATA_REGEX14 = re.compile(r'(datalad)', re.VERBOSE)
DATA_REGEX15 = re.compile(r'(dataverse)', re.VERBOSE)
DATA_REGEX16 = re.compile(r'(dbgap)', re.VERBOSE)
DATA_REGEX17 = re.compile(r'(nih\.gov\/gap)', re.VERBOSE)
DATA_REGEX18 = re.compile(r'(dryad)', re.VERBOSE)
DATA_REGEX19 = re.compile(r'(figshare)', re.VERBOSE)
DATA_REGEX20 = re.compile(r'(fcon_1000\.projects)', re.VERBOSE)
DATA_REGEX21 = re.compile(r'(nitrc)', re.VERBOSE)
DATA_REGEX22 = re.compile(r'(mcgill\.ca\/bic\/resources\/omega)', re.
    VERBOSE)
DATA_REGEX23 = re.compile(r'(xnat\.org)', re.VERBOSE)
DATA_REGEX24 = re.compile(r'(zenodo)', re.VERBOSE)
DATA_REGEX25 = re.compile(r'(opendata\.aws)', re.VERBOSE)

#PREFIX_RE = re.compile(r'''^[\[\("']''', re.VERBOSE)
#SUFFIX_RE = re.compile(r'''[\]\)"']$''', re.VERBOSE)
#INFIX_RE = re.compile(r'''[-~]''', re.VERBOSE)
#SIMPLE_URL_RE = re.compile(r'''^(https?:/?/? ? ?)?(www)?''', re.VERBOSE)

def remove_punc(text):
    text_nopunct = "".join([char for char in text if char not in string.
    punctuation])
    return text_nopunct

# create a custom analyzer class
class MyAnalyzer(object):

    # load spaCy's english model and define the tokenizer/lemmatizer
    def __init__(self):
        spacy.load('en')
        self.lemmatizer_ = spacy.lang.en.English()

    # allow the class instance to be called just like
    # just like a function and applies the preprocessing and
    # tokenize the document
    def __call__(self, doc):
        doc_clean = unescape(doc).lower()
        tokens = self.lemmatizer_(doc_clean)
        return([token.lemma_ for token in tokens])

def test_suitability(passage):
    #figures out whether a passage is suitable for what we're after.
    #probably still needs some fiddling.
    suitability = True
    try:
        if passage['infons']['section_type'].upper() == 'REF':
            suitability = False

        if passage['infons']['section_type'].upper() == 'TITLE':
            suitability = False

        if passage['infons']['type'].upper() == 'TABLE':
            suitability = False

        if 'TITLE' in passage['infons']['type'].upper():
            suitability = False
    except:
        suitability = False

    return(suitability)

'''
def custom_tokenizer(nlp):
    return Tokenizer(nlp.vocab, prefix_search=PREFIX_RE.search,
                                suffix_search=SUFFIX_RE.search,
                                infix_finditer=INFIX_RE.finditer,
                                token_match=SIMPLE_URL_RE.match)
                                '''

# create a dataframe from a word matrix
def wm2df(wm, feat_names):

    # create an index for each row
    doc_names = ['Doc{:d}'.format(idx) for idx, _ in enumerate(wm)]
    df = pd.DataFrame(data=wm.toarray(), index=doc_names,
                      columns=feat_names)
    return(df)

def filter_text():
    data_dir = '/Users/G/git/repository/data-science-project/data/papers'
    files = sorted(os.listdir(data_dir))
    papers = pd.DataFrame(columns=['Date', 'PMID', 'Passage', 'Data'])
    #papers = papers.append(pd.DataFrame({'Date': [20191214], 'PMID': ['0000000000'], 'Passage': ['This is a test']}), ignore_index = True)

    #no_match_filename = "/Users/G/git/repository/data-science-project/data/NIMH_pmids_noMatchRegex.txt"

    count = 0
    paper_count = 0
    for f in files:
        if re.match(FILE_REGEX, f) is not None:
            with open(os.path.join(data_dir, f)) as json_file:
                data = json.load(json_file)

            df = pd.io.json.json_normalize(data['papers'], record_path='documents', meta=['date'])
            print("Processed filename: " + f)

            for row in df.itertuples():
                pmid = ''
                first = True
                #print("Paper " + str(paper_count))
                for passage in row.passages:
```

```
                try:
                    if first is True:
                        pmid = passage['infons']['article-id_pmid']
                        first = False

                    suitability = test_suitability(passage)
                    if suitability is True:
                        sentences = sent_tokenize(passage['text'])
                        for s in sentences:
                            #s = remove_punc(s)
                            m = (re.search(DATA_REGEX1, s.lower()) or re.search(DATA_REGEX2, s.lower()) or re.search(DATA_REGEX3, s.lower()) or
                                    re.search(DATA_REGEX4, s.lower()) or re.search(DATA_REGEX5, s.lower()) or re.search(DATA_REGEX6, s.lower()) or
                                    re.search(DATA_REGEX7, s.lower()) or re.search(DATA_REGEX8, s.lower()) or re.search(DATA_REGEX9, s.lower()) or
                                    re.search(DATA_REGEX10, s.lower()) or re.search(DATA_REGEX11, s.lower()) or re.search(DATA_REGEX12, s.lower()) or
                                    re.search(DATA_REGEX13, s.lower()) or re.search(DATA_REGEX14, s.lower()) or re.search(DATA_REGEX15, s.lower()) or
                                    re.search(DATA_REGEX16, s.lower()) or re.search(DATA_REGEX17, s.lower()) or re.search(DATA_REGEX18, s.lower()) or
                                    re.search(DATA_REGEX19, s.lower()) or re.search(DATA_REGEX20, s.lower()) or re.search(DATA_REGEX21, s.lower()) or
                                    re.search(DATA_REGEX22, s.lower()) or re.search(DATA_REGEX23, s.lower()) or re.search(DATA_REGEX24, s.lower()) or
                                    re.search(DATA_REGEX25, s.lower()))
                            if m is not None:
                                papers = papers.append(pd.DataFrame({'Date': [row.date], 'PMID': [pmid], 'Passage': [s.strip()], 'Data': [m.group(1)]}),
                                        ignore_index = True)
                    #else:
                    #    print(passage['infons']['section_type'] + ',' + passage['infons']['type'] + ',' + passage['text'])
                except:
                    continue
            paper_count += 1
        print("   - " + str(paper_count))
        paper_count = 0
        #if count == 4:
        #    print(papers.head())
        #    #print(papers.groupby(['PMID']).count())
        #    print(papers['Passage'][2])
        #    exit(1)
```

```
            count += 1
    #print(papers.head())
    #print(len(papers))
    #print(papers.groupby(['PMID']).count())
    #print(papers['Passage'][0])

    # pickle dataframe
    papers.to_pickle("/Users/G/git/repository/data-science-project/data/NIMH_pmids_matched.pkl")

def read_df(filename="/Users/G/git/repository/data-science-project/data/NIMH_pmids_matched.pkl"):
    unpickled_df = pd.read_pickle(filename)
    print(unpickled_df.head())
    print(unpickled_df.tail())
    print(len(unpickled_df['PMID']))
    print(unpickled_df.groupby(['PMID']).count())
    print(unpickled_df['Passage'][4])

def write_df_to_csv(output_filename, filename="/Users/G/git/repository/data-science-project/data/NIMH_pmids_matched.pkl"):
    unpickled_df = pd.read_pickle(filename)
    unpickled_df.to_csv(output_filename, index=False)

def method_ML(model, ML_method, vectorizer_method, X_train, X_test, y_train, y_test, vis_filename, model_filename):
    # Fit the model on training set
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    score = model.score(X_test, y_test)
    cm = confusion_matrix(y_test, y_pred)

    print("Accuracy (" + vectorizer_method + "):" , score)
    print("Confusion matrix : ")
    print(cm)
    print("Classification report (" + vectorizer_method + "):")
    print(classification_report(y_test, y_pred))

    roc_auc = roc_auc_score(y_test, y_pred)
    fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test)[:,1])
    plt.figure()
    plt.plot(fpr, tpr, label = '%s (area = %0.2f)' % (ML_method, roc_auc))
    plt.plot([0,1], [0,1], 'r--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc='lower right')
    plt.savefig(vis_filename)
    print("Graph saved: ", vis_filename)
    print()
    #plt.show()
```

```python
    plt.close()

    pickle.dump(model, open(model_filename, 'wb'))
    print("Model saved: ", model_filename)
    print()

def compare_models(models, seed, X, y, vectorizer):
    results = []
    names = []
    scoring = 'accuracy'
    for name, model in models:
        kfold = model_selection.KFold(n_splits=10, random_state=seed, shuffle=True)
        cv_results = model_selection.cross_val_score(model, X, y, cv=kfold, scoring=scoring)
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)

    # boxplot algorithm comparison
    fig = plt.figure()
    fig.suptitle('Algorithm Comparison')
    ax = fig.add_subplot(111)
    plt.boxplot(results)
    ax.set_xticklabels(names)
    plt.savefig('/Users/G/Loyola/Spring2020/DS796/compare_models_boxplot_' + vectorizer + '.png')
    print('Graph saved: /Users/G/Loyola/Spring2020/DS796/compare_models_boxplot_' + vectorizer + '.png')
    print()
    #plt.show()
    plt.close()

def prepare_data(filename, verbose=False):

    #fd = open(filename, 'r')
    #fd.close()
    df = pd.read_csv(filename, header=0)
    new_df = df[['Year', 'PMID', 'data_reuse', 'text']]
    #print(new_df.head(10))
    #print(new_df.tail(10))

    # Total number of records (with nulls)
    print("Count of records (with nulls): \n" + str(new_df.data_reuse.count()))
    print()

    # Checking missing values
    print("Count of missing values: \n" + str(new_df.isnull().sum()))
    print()
    print("Rows where text is missing: ")
    print(new_df[new_df['text'].isnull()])
    print()
    # Total number of records (without nulls)
    new_df2 = new_df.dropna(inplace=False)
    print("Count of records (without nulls): \n" + str(new_df2.data_reuse.count()))
    print()

    if verbose:
        # Balance of data
        print("Count by data_reuse: \n\n" + str(new_df2.groupby('data_reuse').count()))
        print()
        print("Normalized percentage by data_reuse: \n\n" + str(new_df2['data_reuse'].value_counts(normalize=True)))
        print()
        print("Count by pmid: \n\n" + str(new_df2.groupby('PMID').count()))
        print()
        print("Normalized percentage by pmid: \n\n" + str(new_df2['PMID'].value_counts(normalize=True).head(10)))
        print()
        print("Count by Year: \n\n: " + str(new_df2.groupby('Year').count()))
        print()
        print("Normalized percentage by Year: \n\n" + str(new_df2['Year'].value_counts(normalize=True)))
        print()
        print("Count of data_reuse by Year: \n\n" + str(new_df2.groupby(['Year', 'data_reuse']).count()))
        print()

        # Visualize
        plt.figure(figsize=(40,30))
        new_df2['data_reuse'].value_counts().plot(kind='bar')
        plt.xticks([0, 1], ('No', 'Yes'))
        plt.title("Number of data_reuse", fontsize=100)
        plt.xticks(fontsize=50)
        plt.yticks(fontsize=50)
        plt.xlabel("data_reuse", fontsize=70)
        plt.ylabel("count", fontsize=70)
        plt.savefig('/Users/G/Loyola/Spring2020/DS796/count_data_reuse_graph.png')
        print("Graph saved: /Users/G/Loyola/Spring2020/DS796/count_data_reuse_graph.png")
        print()
        #plt.show()

        plt.figure(figsize=(40, 30))
        new_df2.groupby(['Year', 'data_reuse'])['text'].count().unstack().plot(kind='bar', stacked=True)
        plt.title("Number of data_reuse by Year", fontsize=15)
        plt.xticks(fontsize=6)
        plt.yticks(fontsize=6)
        plt.xlabel("Year", fontsize=12)
        plt.ylabel("count", fontsize=12)
        plt.savefig('/Users/G/Loyola/Spring2020/DS796/
   count_data_reuse_year_graph.png')
        print("Graph saved: /Users/G/Loyola/Spring2020/DS796/
   count_data_reuse_year_graph.png")
        print()
        #plt.show()
        plt.close()

    return new_df2

def model(filename, model_type):
    try:

        # Initialize custom analyzer
        analyzer = MyAnalyzer()

        new_df2 = prepare_data(filename, True)

        # Split the data into train and test dataset
        X = np.array(new_df2['text'])
        y = np.array(new_df2['data_reuse'])
        print("X shape: " + str(X.shape))
        print("y shape: " + str(y.shape))
        print()

        sentences_train, sentences_test, y_train, y_test =
   train_test_split(X, y, test_size = 0.2, random_state = 0)
        print("Reuse (train): ")
        print(collections.Counter(y_train))
        print()
        print("Reuse (test): ")
        print(collections.Counter(y_test))
        print()

        # Tokenization/Normalization/Word Embedding
        # CountVectorizer
        # Shape without Custom tokenizer - train (1502, 7103)
        # Shape without Custom tokenizer - test (376, 7103)

        # Shape with Custom tokenizer - train (1502, 7553)
        # Shape with Custom tokenizer - test (376, 7553)

        #vectorizer_count = CountVectorizer()
        vectorizer_count = CountVectorizer(analyzer=analyzer)
        #sentence_vectors_count = vectorizer_count.fit_transform(new_df2['
   text'])
        #vectorizer_count.fit(sentences_train)
        vectorizer_count.fit(X)
        count_X_train = vectorizer_count.transform(sentences_train)
        count_X_test = vectorizer_count.transform(sentences_test)
        print("Count Vectorizer (train): ")
        print(count_X_train.toarray())
        print(str(count_X_train.shape))
        print()
        print("Count Vectorizer (test): ")
        print(count_X_test.toarray())
        print(str(count_X_test.shape))
        print()

        print("Vocabulary: ")
        ordered_count = dict(sorted(vectorizer_count.vocabulary_.items(), key=lambda x: x[1], reverse=True)[:1000])
        print(str(ordered_count))
        print()

        #print(len(vectorizer_count.get_feature_names()))
        #print()

        # TFIDF
        #vectorizer_tfidf = TfidfVectorizer(norm = False, smooth_idf = False)
        vectorizer_tfidf = TfidfVectorizer(analyzer=analyzer)
        #sentence_vectors_tfidf = vectorizer_tfidf.fit_transform(new_df2['text'])
        #vectorizer_tfidf.fit(sentences_train)
        vectorizer_tfidf.fit(X)
        tfidf_X_train = vectorizer_tfidf.transform(sentences_train)
        tfidf_X_test = vectorizer_tfidf.transform(sentences_test)
        print("TFIDF Vectorizer (train): ")
        print(tfidf_X_train.toarray())
        print(str(tfidf_X_train.shape))
        print()
        print("TFIDF Vectorizer (test): ")
        print(tfidf_X_test.toarray())
        print(str(tfidf_X_test.shape))
        print()

        # TODO: Dimensionality Reduction?

        if model_type == "regular":
            # Machine learning
            models = []
            print("Logistic Regression: ")
            #logit_model = sm.Logit(y_train, count_X_train)
            #result = logit_model.fit()
            #print(result.summary2())
            #print()

            classifier_logistic = LogisticRegression()
            method_ML(classifier_logistic, "Logistic Regression", "CountVectorizer", count_X_train,
                      count_X_test, y_train, y_test, "/Users/G/Loyola/Spring2020/DS796/count_log_roc_graph.png",
                        "/Users/G/Loyola/Spring2020/DS796/finalized_model_count_log.sav")
            method_ML(classifier_logistic, "Logistic Regression", "TfidfVectorizer", tfidf_X_train,
                  tfidf_X_test, y_train, y_test, "/Users/G/Loyola/
Spring2020/DS796/tfidf_log_roc_graph.png",
                     "/Users/G/Loyola/Spring2020/DS796/
finalized_model_tfidf_log.sav")

        print("Naive Bayes: ")
        classifier_nb = MultinomialNB()
        method_ML(classifier_nb, "Naive Bayes", "CountVectorizer",
count_X_train,
                    count_X_test, y_train, y_test, "/Users/G/Loyola/
Spring2020/DS796/count_nb_roc_graph.png",
                     "/Users/G/Loyola/Spring2020/DS796/
finalized_model_count_nb.sav")
        method_ML(classifier_nb, "Naive Bayes", "TfidfVectorizer",
tfidf_X_train,
                     tfidf_X_test, y_train, y_test, "/Users/G/Loyola/
Spring2020/DS796/tfidf_nb_roc_graph.png",
                     "/Users/G/Loyola/Spring2020/DS796/
finalized_model_tfidf_nb.sav")

        print("Support Vector Machines: ")
        classifier_svm = SVC(kernel='linear', probability=True)
        method_ML(classifier_svm, "SVM", "CountVectorizer",
count_X_train,
                     count_X_test, y_train, y_test, "/Users/G/Loyola/
Spring2020/DS796/count_svm_roc_graph.png",
                  "/Users/G/Loyola/Spring2020/DS796/
finalized_model_count_svm.sav")
        method_ML(classifier_svm, "SVM", "TfidfVectorizer",
tfidf_X_train,
                     tfidf_X_test, y_train, y_test, "/Users/G/Loyola/
Spring2020/DS796/tfidf_svm_roc_graph.png",
                  "/Users/G/Loyola/Spring2020/DS796/
finalized_model_tfidf_svm.sav")

        print("Random Forest: ")
        classifier_rf = RandomForestClassifier()
        method_ML(classifier_rf, "Random Forest", "CountVectorizer",
count_X_train,
                  count_X_test, y_train, y_test, "/Users/G/Loyola/
Spring2020/DS796/count_rf_roc_graph.png",
                  "/Users/G/Loyola/Spring2020/DS796/
finalized_model_count_rf.sav")
        method_ML(classifier_rf, "Random Forest", "TfidfVectorizer",
tfidf_X_train,
                     tfidf_X_test, y_train, y_test, "/Users/G/Loyola/
Spring2020/DS796/tfidf_rf_roc_graph.png",
                  "/Users/G/Loyola/Spring2020/DS796/
finalized_model_tfidf_rf.sav")

        print("Comparing Models: ")
        models = []
        models.append(('LogisticRegression', classifier_logistic))
        models.append(('Naive Bayes', classifier_nb))
        models.append(('SVM', classifier_svm))
        models.append(('Random Forest', classifier_rf))
        print("Using Count:")
        compare_models(models, 7, count_X_train, y_train, 'count')
        print("Using TFIDF:")
        compare_models(models, 7, tfidf_X_train, y_train, 'tfidf')

    elif model_type == "active":
        print("Active Learning: ")
        accepted_vectorizers = ['COUNT', 'TFIDF']
        accepted_models = ['LR', 'NB', 'SVM', 'RF']
        accepted_query_strategies = ['CE', 'CM', 'CU', 'ES', 'MS', 'US']

        new_data_filename = input("Enter new data file: ")
        vectorizer = input("Enter the vectorizer to use (Default: COUNT): ").upper()
        model = input("Enter machine learning model to use (Default: LR): ").upper()
        query_strategy = input("Enter query_strategy to use (Default: US): ").upper()
        n_queries = input("Enter number of queries (Default: 10): ")
        print()

        if not vectorizer:
            vectorizer = 'COUNT'

        if not model:
            model = 'LR'

        if not query_strategy:
            query_strategy = 'US'

        if not n_queries:
            n_queries = int(10)

        if filename and new_data_filename and vectorizer.upper() in accepted_vectorizers and model.upper() in accepted_models and query_strategy.upper() in accepted_query_strategies:

            # Date,PMID,Passage,Data
            df = pd.read_csv(new_data_filename, header=0)
            df['Data_Reuse'] = None
            print(df['Passage'].head(10))
            #print(df.tail(10))
            print()

            # Total number of records (with nulls)
            print("Count of records: \n" + str(df.Passage.count()))
            print()
            print("Count of unique records: \n" + str(df.groupby(['PMID']).count()))
            print()

```

```python
            # Checking missing values
            print("Count of missing values: \n" + str(df.isnull().sum()))
            print()
            print("Rows where text is missing: ")
            print(df[df['Passage'].isnull()])
            print()

            if vectorizer == "COUNT":
                # Using count vectorizer
                print("##################################")
                print("Using count vectorizer:")
                print("##################################")
                print()

                # Tokenize, add vocabulary, and encode new data
                count_X_new = vectorizer_count.fit_transform(df['Passage'])

                # Re-encode training/test documents with the new vocabulary
                count_X_train = vectorizer_count.transform(sentences_train)
                count_X_test = vectorizer_count.transform(sentences_test)

                print("Count Vectorizer (New): ")
                print(count_X_new.toarray())
                print(str(count_X_new.shape))
                print()

                print("Vocabulary (New): ")
                ordered_new_count = dict(sorted(vectorizer_count.vocabulary_.items(), key=lambda x: x[1], reverse=True)[:1000])
                print(str(ordered_new_count))
                print()

                active_learning("count", count_X_train, y_train, count_X_test, y_test, df['Passage'], count_X_new, model, query_strategy, int(n_queries),
                                      "/Users/G/Loyola/Spring2020/DS796/active_model_count_" + model + ".sav")
                print()
            elif vectorizer == "TFIDF":
                # Using tfidf vectorizer
                print("##################################")
                print("Using TFIDF vectorizer:")
                print("##################################")
                print()

                # Tokenize, add vocabulary, and encode new data
                tfidf_X_new = vectorizer_tfidf.fit_transform(df['Passage'])

```

```
                    # Re-encode training/test documents with the new vocabulary
                    tfidf_X_train = vectorizer_tfidf.transform(sentences_train)
                    tfidf_X_test = vectorizer_tfidf.transform(sentences_test)

                    print("Tfidf Vectorizer (New): ")
                    print(tfidf_X_new.toarray())
                    print(str(tfidf_X_new.shape))
                    print()

                    print("Vocabulary (New): ")
                    ordered_new_tfidf = dict(sorted(vectorizer_tfidf.vocabulary_.items(), key=lambda x: x[1], reverse=True)[:1000])
                    print(str(ordered_new_tfidf))
                    print()

                    active_learning("tfidf", tfidf_X_train, y_train, tfidf_X_test, y_test, df['Passage'], tfidf_X_new, model, query_strategy, int(n_queries),
                                        "/Users/G/Loyola/Spring2020/DS796/active_model_tfidf_" + model + ".sav")
                    print()
                else:
                    print("Vectorizer not supported. Please see help for more info.")
                    print()

            else:
                print("Something went wrong with your input(s). Please see help for more info and check your inputs.")
                print()

    except:
        print("Encountered Error: ", sys.exc_info())
        raise

def active_learning(vectorizer_method, X_train, y_train, X_test, y_test, orig_text, X_new, model, qstrategy, n_queries, model_filename):

    classifier = None
    strategy = None

    if model == 'LR':
        classifier = LogisticRegression()
    elif model == 'NB':
        classifier = MultinomialNB()
    elif model == 'SVM':
        classifier = SVC(kernel='linear', probability=True)
    elif model == 'RF':
        classifier = RandomForestClassifier()

    if qstrategy == 'CE':
        strategy = classifier_entropy
    elif qstrategy == ’CM’:
        strategy = classifier_margin
    elif qstrategy == ’CU’:
        strategy = classifier_uncertainty
    elif qstrategy == ’ES’:
        strategy = entropy_sampling
    elif qstrategy == ’MS’:
        strategy = margin_sampling
    elif qstrategy == ’US’:
        strategy = uncertainty_sampling

    learner = ActiveLearner(
                 estimator=classifier,
                 query_strategy=strategy,
                 X_training=X_train, y_training=y_train
             )

    accuracy_scores = [learner.score(X_test, y_test)]
    recall_scores = [recall_score(y_test, learner.predict(X_test))]

    for i in range(n_queries):
        #print(X_train.shape)
        #print(X_new.shape)
        #print(orig_text.iloc[0])
        query_idx, query_inst = learner.query(X_new)
        #print(query_inst)
        #print(query_idx)
        print(orig_text.iloc[query_idx[0]])
        print("Is this a data reuse statement or not (1=yes, 0=no)?")
        try:
            y_new = np.array([int(input())], dtype=int)
            if y_new in [0,1]:
                learner.teach(query_inst.reshape(1, -1), y_new)

                X_new = csr_matrix(np.delete(X_new.toarray(), query_idx, axis=0))
                orig_text = pd.Series(np.delete(orig_text.to_numpy(), query_idx, axis=0))
                accuracy_scores.append(learner.score(X_test, y_test))
                recall_scores.append(recall_score(y_test, learner.predict(X_test)))
                #print(accuracy_scores)
                #print(recall_scores)
                print()
            else:
                print("Input not accepted. Type ’1’ for yes or ’0’ for no")
                print()
                return
        except:
            print("Incorrect input. Skipping.")
            print()

```

```python
    # Performance of classier
    with plt.style.context('seaborn-white'):
        plt.figure(figsize=(10, 5))
        plt.title('Performance of the classifier during the active learning')
        #plt.plot(range(n_queries+1), accuracy_scores)
        #plt.scatter(range(n_queries+1), accuracy_scores)
        plt.plot(range(n_queries+1), recall_scores)
        plt.scatter(range(n_queries+1), recall_scores)
        plt.xlabel('Number of queries')
        plt.ylabel('Performance')
        plt.savefig('/Users/G/Loyola/Spring2020/DS796/active_model_' + vectorizer_method + '_' + model + '_performance.png')
        print("Graph saved: /Users/G/Loyola/Spring2020/DS796/active_model_" + vectorizer_method + '_' + model + "_performance.png")
        print()
        #plt.show()
        plt.close()

    pickle.dump(learner, open(model_filename, 'wb'))
    print("Model saved: ", model_filename)
    print()

def generateWordCloud(filename):
    df = pd.read_csv(filename, header=0)
    mask = np.array(Image.open('/Users/G/Loyola/Spring2020/DS796/Brain.jpg'))
    wordcloud = WordCloud(
            width = 3000,
            height = 2000,
            background_color = 'black',
            stopwords = STOPWORDS,
            mask = mask).generate(str(df['Data']))
    fig = plt.figure(
            figsize = (40, 30),
            facecolor = 'k',
            edgecolor = 'k')
    plt.imshow(wordcloud, interpolation = 'bilinear')
    plt.axis('off')
    plt.tight_layout(pad=0)
    plt.savefig('/Users/G/Loyola/Spring2020/DS796/wordcloud.png')
    print("Wordcloud saved: /Users/G/Loyola/Spring2020/DS796/wordcloud.png")
    print()
    #plt.show()
    plt.close(fig)

'''
def loadModel(filename, vectorizer_method):
    # load the model from disk
    loaded_model = pickle.load(open(filename, 'rb'))
    #result = loaded_model.score(X_test, Y_test)
    yPredict = loaded_model.predict(Xtest)
    print(yPredict)
    print()
    '''

def options(argument):

    switcher = {
        '1':  'Filter',
        '2':  'Read',
        '3':  'Write',
        '4':  'Model',
        '5':  'WordCloud',
        'q':  'Quit',
        'h':  'Help'
    }

    return switcher.get(argument, "Invalid argument")

def help():

    print("""
          Options available:
          1:  Filter
          2:  Read
          3:  Write
          4:  Model
              regular (using conventional Machine Learning techniques)
              active  (Accepted vectorizers (case insensitive): COUNT (Default), TFIDF,
                       Accepted models (case insensitive): LR, NB, SVM, RF,
                       Accepted query strategy (case insensitive): CE, CM, CU, ES, MS, US)
              LR - Logistic Regression (Default)
              NB - Naives Bayes
              SVM - Support Vector Machine
              RF - Random Forest
              CE - Classifier Entropy
              CM - Classifier Margin
              CU - Classifier Uncertainty
              ES - Entropy Sampling
              MS - Margin Sampling
              US - Uncertainty Sampling (Default)
              Default # of queries: 10
          5:  WordCloud
          q:  Quit
          h:  Help
          """)

if __name__ == "__main__":

    arg = input("Enter option or h for help: ")
    while(arg != 'q'):
        if options(arg) == 'Help':
            help()
        elif options(arg) == 'Filter':
            start = time.time()
            filter_text()
            print('It took', round((time.time()-start)/60, 2), 'minutes.')
            print()
        elif options(arg) == 'Read':
            filename = input("Enter pickled file: ")
            if filename:
                read_df(filename.strip())
            else:
                read_df()
        elif options(arg) == 'Write':
            output = input("Enter output file: ")
            if output:
                print("Writing dataframe to csv.")
                write_df_to_csv(output)
                print("Done.")
                print()
            else:
                print("Please provide an output file.")
                print()
        elif options(arg) == 'Model':
            build_model = input("Build (regular or active): ")
            filename = input("Enter input file to build model: ")

            try:
                if filename and build_model:
                    model(filename.strip(), build_model.lower())
                else:
                    print("Please enter a filename/model type to build model.")
                    print()
            except:
                pass
        elif options(arg) == 'WordCloud':
            filename = input("Enter input file to build wordcloud: ")
            try:
                if filename:
                    generateWordCloud(filename.strip())
                else:
                    print("Please enter a valid filename to begin.")
                    print()
            except:
                pass

        arg = input("Enter option or h for help: ")
```

Listing 5: analysis.py