Final Project Report

Classification of Biomedical Abstracts: PubMed vs. BioRxiv

Tre Tomaszewski, Sayantan Dutta, Xiaoliang Jiang

Introduction

For the final project, we compared abstracts of both peer-reviewed articles from *PubMed Central Open Access* Subset and pre-prints found on *Biorxiv.org* (inspired by *ArXiv.org*) with the goal of classifying the source of the article abstract. As articles submitted to pre-print repositories like *Biorxiv.org* are not required to have gone through peer review, it is possible these articles differ from peer-reviewed articles from *PubMed* in research quality, depth, and rigor. While this text analysis experiment does not hope to determine these features, if classification accuracy is found to be above chance level (using a balanced dataset in both source and field), further analysis into how this difference is determined would be worthwhile.

Our project uses 52962 instances (balanced at 39074) of moderately unstructured text pulled from the two sources in XML and HTML. Beyond having to derive linguistic features and patterns in these texts, machine learning can assist with the analysis and comparison in an efficient manner.

Some works prior to 2019 used manual processes for text analysis as follows:

Klein, M., Broadwell, P., Farb, S. E., & Grappone, T. (2018). Comparing published scientific journal articles to their pre-print versions. *International Journal on Digital Libraries*. https://doi.org/10.1007/s00799-018-0234-1

Amaral, O. B. (March 20, 2018). Comparing the quality of reporting between preprints and peer-reviewed articles – a crowdsourced initiative. Retrieved March 8, 2019, from ASAPbio website: https://asapbio.org/amaral-quality

Carneiro, C. F. D., Queiroz, V. G. S., Moulin, T. C., Carvalho, C. A. M., Haas, C. B., Rayêe, D., … Amaral, O. B. (March 22, 2019). Comparing the quality of reporting between preprints and peer-reviewed articles in the biomedical literature [Preprint]. https://doi.org/10.1101/581892

Data

The *PubMed Central Open Access Subset* (PMC OA)[1] - offered by the National Institute of Health (NIH) - is a repository of medical and biomedical papers published across a number of journals. These texts are available for public access without subscription. Each article in the subset is available in HTML, XML, and (often) pdf data formats via the Entrez API. *[1]*

Biorxiv.org is a preprint service for biomedical articles, similar to the better known Cornell University's *ArXiv.org*. As preprints, the necessity of formal peer-review is not present and is indeed unlikely in many cases. The benefit of preprint services is twofold: Authors are able to make their findings immediately available to the scientific community and, due to the physical and monetary accessibility, feedback on draft manuscripts can be received before they are submitted to journals.

For this project, the corpora from either source were constrained so that each set comprised of the same set of topics. Additionally, number of articles *within* each topic was further constrained, providing the classes with equally representation. Due to the significantly more recent instantiation of the BiorXiv corpus (first papers were in 2013) relative to the PubMed corpus (at least pre 1970), BiorXiv is likely to be the constraining set.
These topics are:

- Bioinformatics (BiorXiv: ~4800)
- Neuroscience (BiorXiv: ~7880)
- Microbiology (BiorXiv: ~3390)
- Pharmacology and Toxicology (BiorXiv: ~340)
- Epidemiology (BiorXiv: ~1200)
- Genomics (BiorXiv: ~3320)
- Immunology (BiorXiv: ~960)

In sum, there are around 21890 desirable articles on BiorXiv. Dependent on the accessibility of articles on BioArXiv, achieving equal representation from PubMed provides a maximum corpus size of ~43780 articles. [2] In our project, equal instances of each topic from either source were collected. Topics were mapped via PubMed's MeSH thesaurus. Duplicate articles were removed, and the remained articles were randomly sampled for equal instance count for each topic. The final corpus after deduplication consisted of 39074 instances.

Method

Retrieval Process

The initial step for this project was data aggregation. This required the acquisition of the source materials, necessitating two separate processes for either source. PubMed articles

[1] https://www.ncbi.nlm.nih.gov/pmc/tools/openftlist/

were obtained through the Entrez API provided by the NIH's NCBI (National Institute of Health, National Center for Biotechnology Information). The Python package *BioPython* includes a premade wrapper for this API, which made a hand-made solution unnecessary. At the time of this paper, *BioArXiv* does not have an API. Due to this, a respectfully slow scrape of BioArXiv abstracts from each topic was conducted.

Preprocessing and Data cleaning

The Entrez API offered several data formats, XML was chosen due to familiarity with their relatively consistent use of resource document framework (RDF) standards. As such, the necessary fields (abstracts in this case) could be transformed into a raw text without much trouble.

Initially, Biorxiv.org was thought to be the more challenging of the two. First, only a limited, non-historical RSS feed of incoming papers is officially offered by the service. Thankfully, they push titles and a link to the article to several Twitter accounts, each specific to an overarching topic. Another issue is that all full-text articles are exclusively available in PDF format; abstracts are available on the linked page, thus standardized and retrievable through HTML processing tools. Because of the standard formatting required to submit to BiorXiv, this HTML was cleaner than PubMed's XML.

For standardization and ingestion of the two sources, a custom preprocessing pipeline was developed. Each abstract was:

1) Converted to lowercase
2) The following characters were replaced with spaces:
    a) tabs
    b) new lines
    c) carriage returns
    d) numbers
    e) email addresses
    f) URLs
    g) punctuation, and
    h) other associated characters (e.g., ^, ;, &, {}, etc.) (See Appendix).
3) Text was tokenized
4) Stopwords were removed
5) Words under a prespecified length were removed (4 for the final).
6) Tokens were lemmatized with WordNet Lemmatizer using the NLTK package for Python.

Feature Selection

Before being fed through the pipeline, the corpus was split 80/20 as 80% of the original corpus was used as the training set (28368 instances) and 20% was set aside for the final testing (7093 instances). The model pipeline was fit to the training data, then both the training and testing sets were transformed via this pipeline.

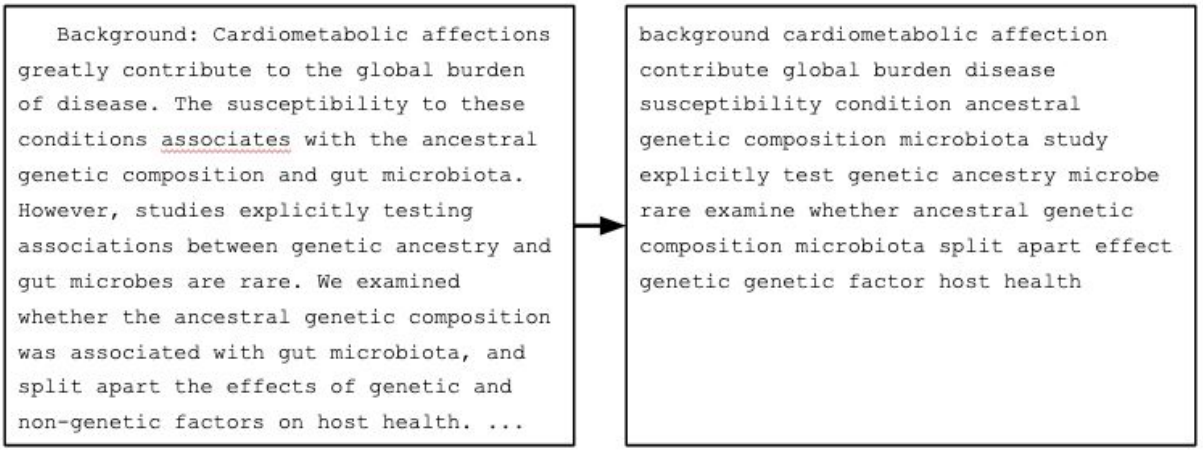


Fig 1. An example of text transform

Part 1: Latent Semantic Analysis

Initial vectorization of training instances was restricted to: terms with a document frequency less than 65%, with a total document occurrence of more than 25 and 10000 features returned (5847 unique terms returned).

Our initial pipeline was structured as a typical Latent Semantic Indexer/Analyzer (LSI/LSA). The count vector was then passed to a TF-IDF transformer and extracted singular values from the transformed matrix using Truncated SVD. Several component counts were tested as 5000 components returned a 97.95% total explained variance. The output was normalized and stored as a dataframe with appended article ID for reference back to the original document and label.

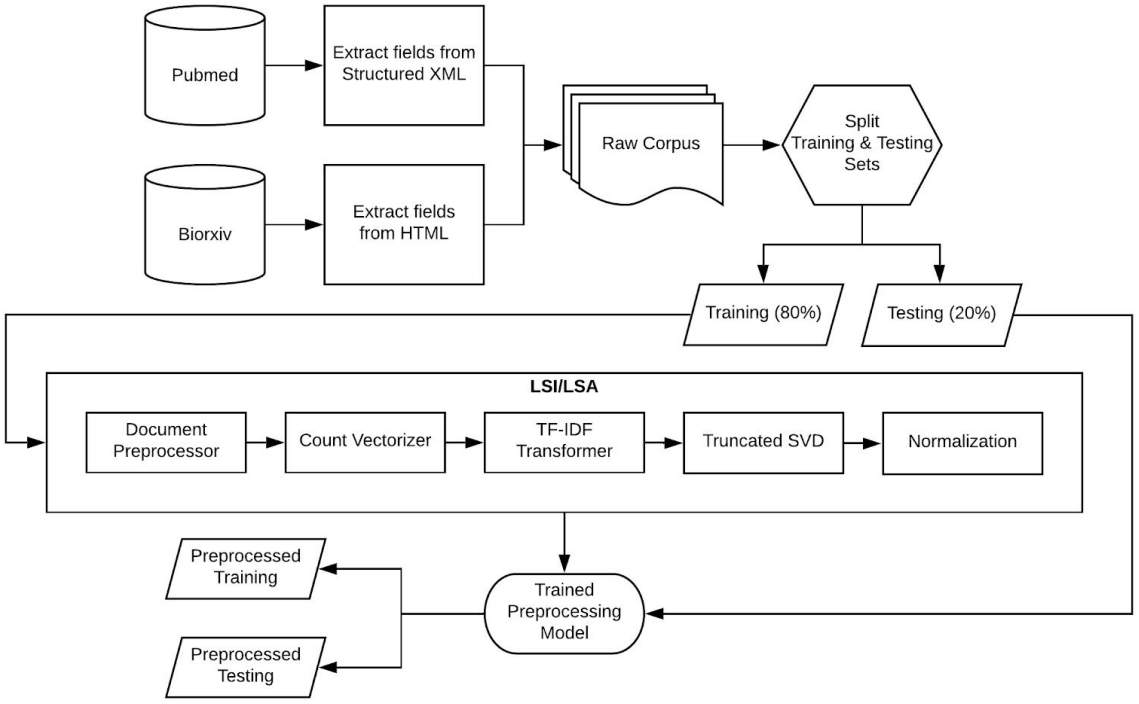


Fig 2. Original Feature Selection Pipeline

Part 2: Vectorization Only

Using feedback from the presentation, we switched to using only count vectorization for the pipeline. Here, two different maximum document frequency parameters were used for vectorization, 65% and 40%. Minimum document frequency was kept, ignoring terms which appeared in less than 25 documents. All terms in less than 65% of the documents and more than 25 documents provided 6138 terms, where as 40% only reduced the number by 1, to 6137. Falling to a maximum frequency of 20% only removed 16 more terms to 6121, and going down to 10% left 6049 terms.

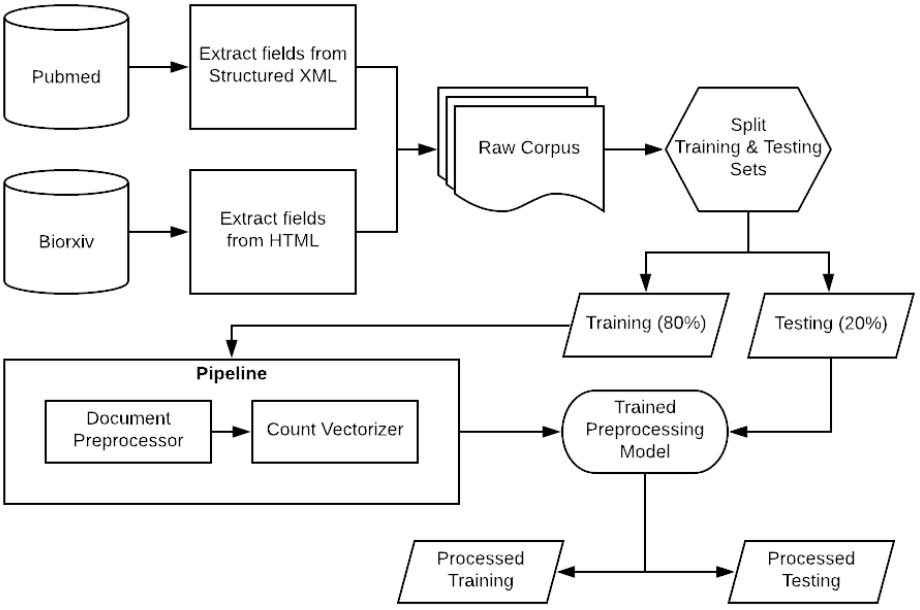


Fig 3. ModifiedFeature Selection Pipeline

Max. Document Frequency	Vocabulary Size (Term Count)
0.65	6138
0.4	6137
0.2	6121
0.1	6049

Table 1: Maximum document frequency and resulting vocabulary size

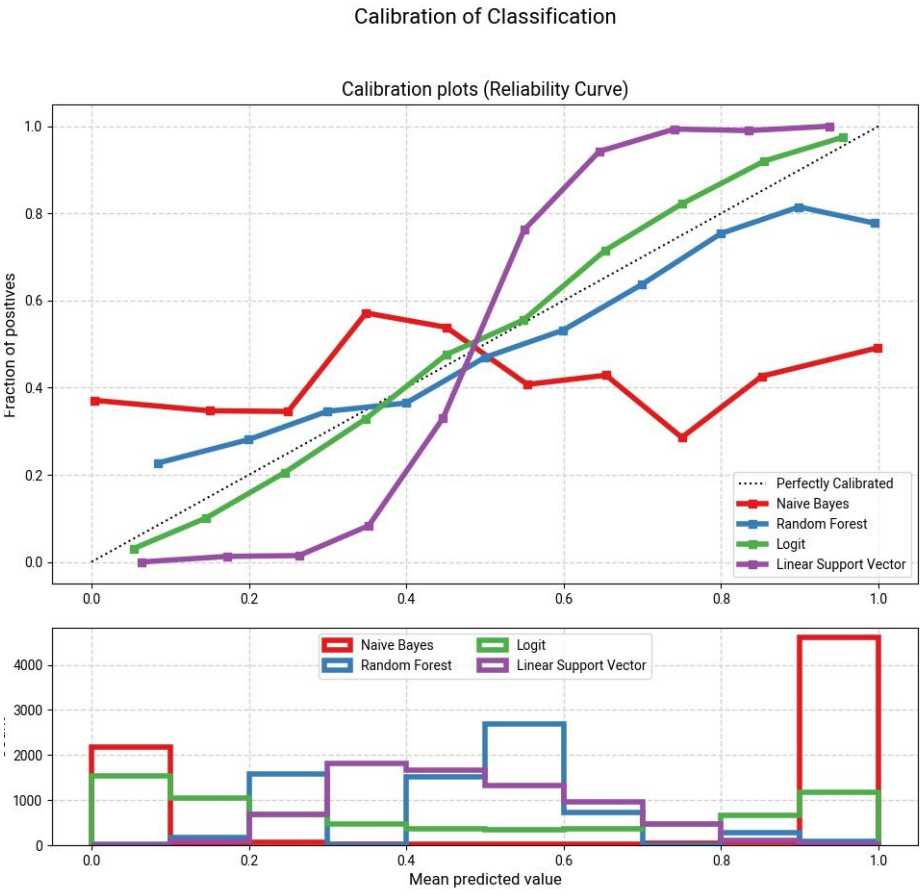
Algorithms

Part 1: Using Latent Semantic Analysis

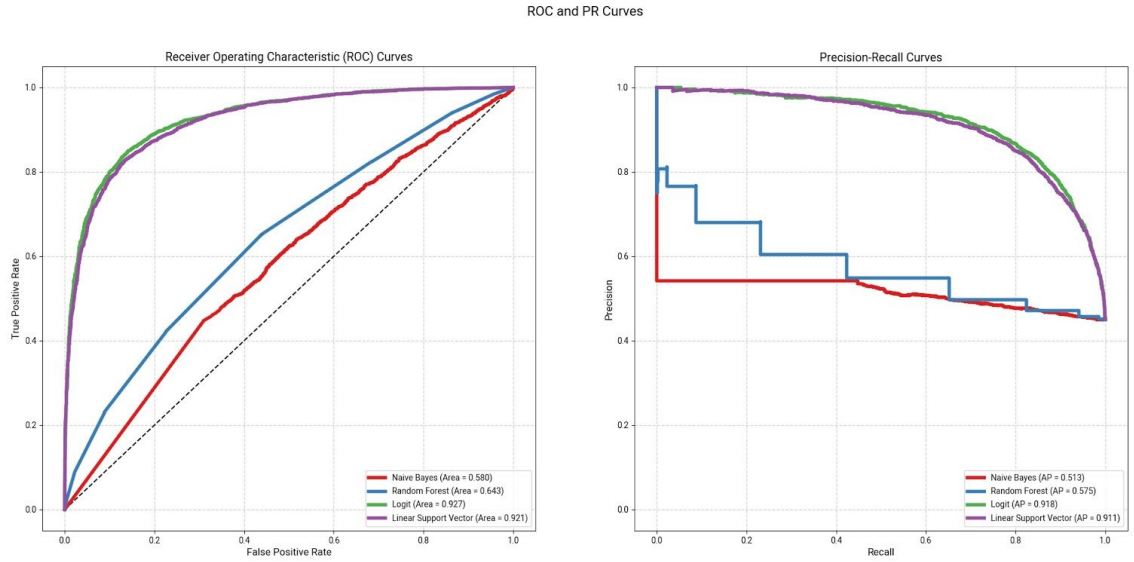
Naive Bayes classifier has the advantage that it requires a small amount of training data to estimate parameters necessary for classification. Naïve Bayes has the simplicity which

makes the framework attractive in various tasks and reasonable performances are obtained in the tasks although this learning is based on an unrealistic independence assumption. As a member of probabilistic classifiers based on Bayes' theorem, Naïve Bayes is one of the most popular methods for text categorization.

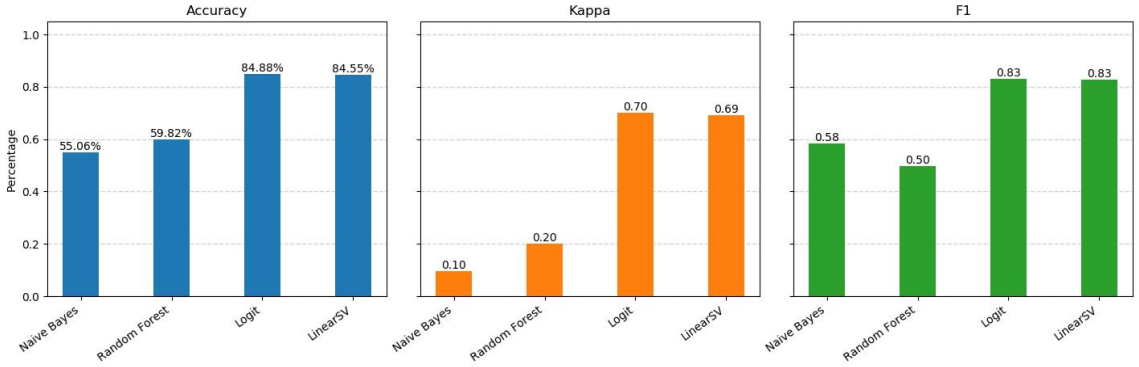
In addition, Random forests is an ensemble learning method for classification by constructing multiple decision trees at training time and outputting the class with a correction function to avoid overfitting caused by the decision tree.



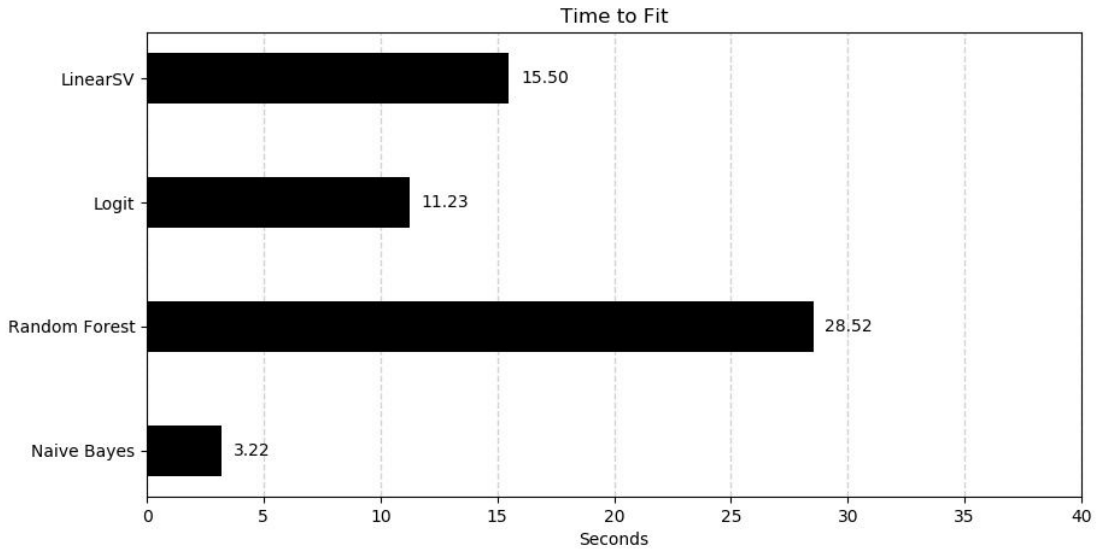
Graph 1.1: Calibration of Classification



Graph 1.2: ROC and PR curves



Graph 1.3: Accuracy, Kappa and F1 measure of different algorithms.



Graph 1.4: Time taken to fit model.

The model outputs showed some interesting results. From the reliability curve, we can see that Logit classifier has performed really well. For SVM we can see that it is under-estimating in lower probabilities and over-estimating in higher probabilities. This can be seen as some sign of overfitting but cannot be conclusively be said so. We can see that Naive Bayes has performed the worst. For Random Forest we can see that it started overestimating at lower frequencies and underestimating at higher frequencies.

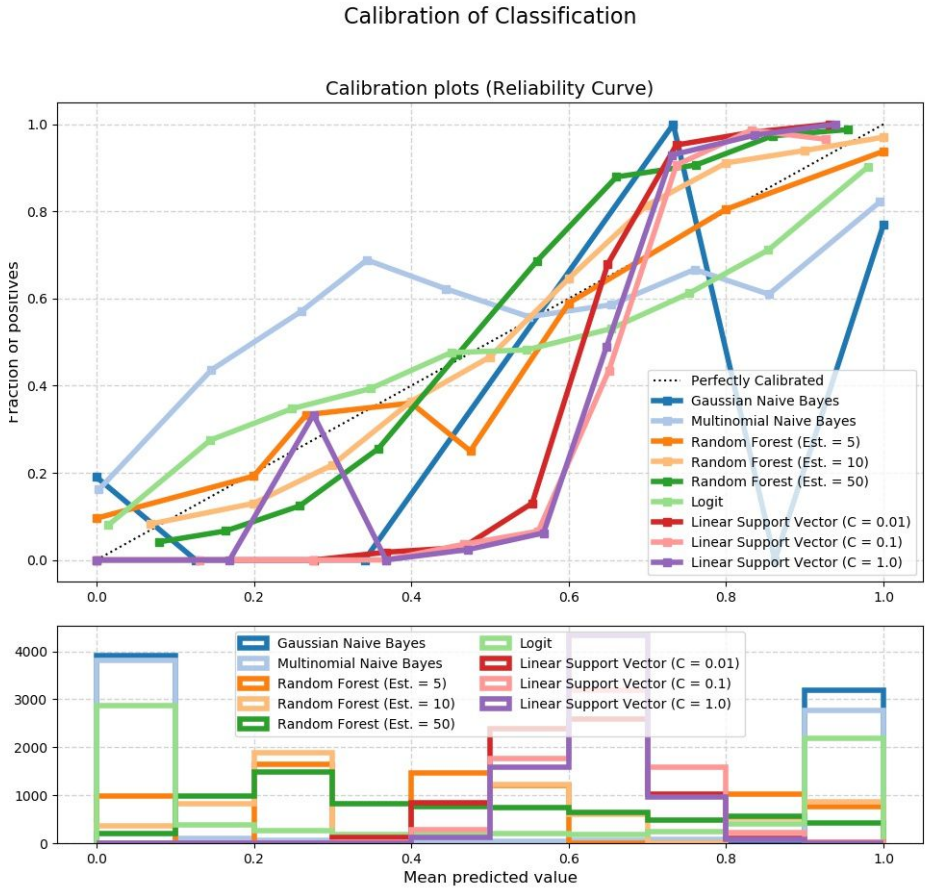
From the mean prediction plot, we can see that Logit has a very evenly paced curve compared to Naive Bayes which literally has only two values 0 and 1. We have the ROC curve and PR curves giving us a more clearer picture. Since our dataset doesn't have any class imbalance ROC curve will more accurately show the results rather than PR curves which generally gets more importance in case of imbalanced datasets. Even though we can see that both the plots clearly indicate that Logit and SVM classifiers have performed the best outperforming other classifiers. We can see the same trend in accuracy, kappa, and F1 scores. From the time-to-fit graph, we can see that Random Forest took maximum time to fit and as expected Naive Bayes took the least time followed by logit.

Part 2: Using Vectorization Only

Additional algorithms were included in this phase[2]: Gaussian Naive Bayes, Multinomial Naive Bayes, Random Forest, Logit, and Linear Support Vector Machine were used. While hyperparameter tuning was minimal, the three different estimator counts for Random Forest (5, 10, and 50), and three C values were used for the Linear Support Vector (0.01, 0.1, and 1.0).

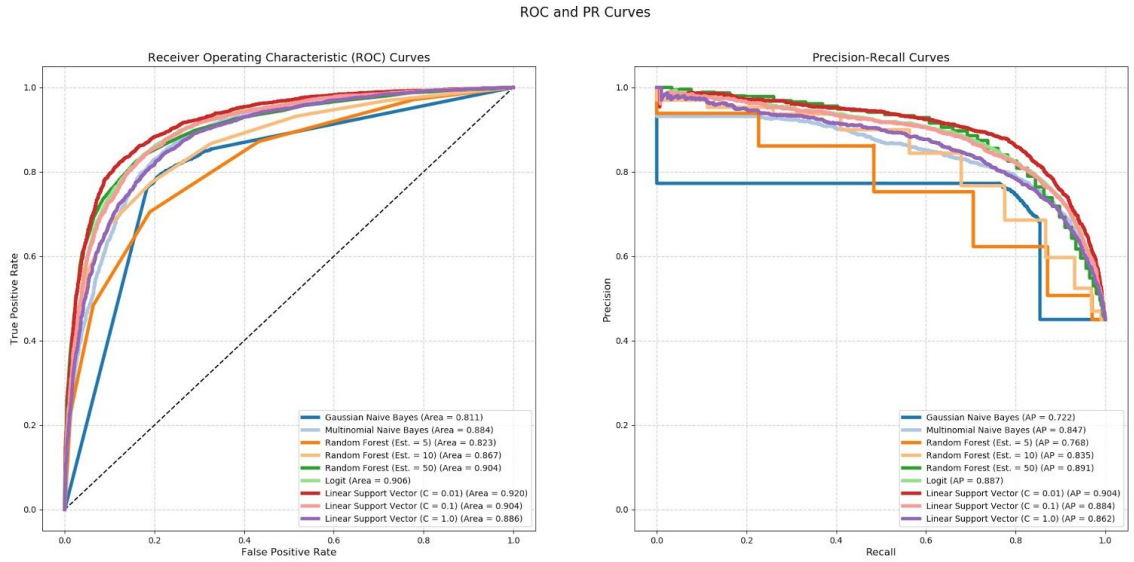
[2] Due to increased time and feedback

Analysis of the vectorized-only data[3] provided different results compared to the LSI pipeline, with a majority of models performing better across the board (except with respect to time). All tunings of the maximum document frequencies (65%, 40%, 20%, 10%) performed with nearly equal results, with only the calibration curves becoming more erratic as the percentage was lowered.

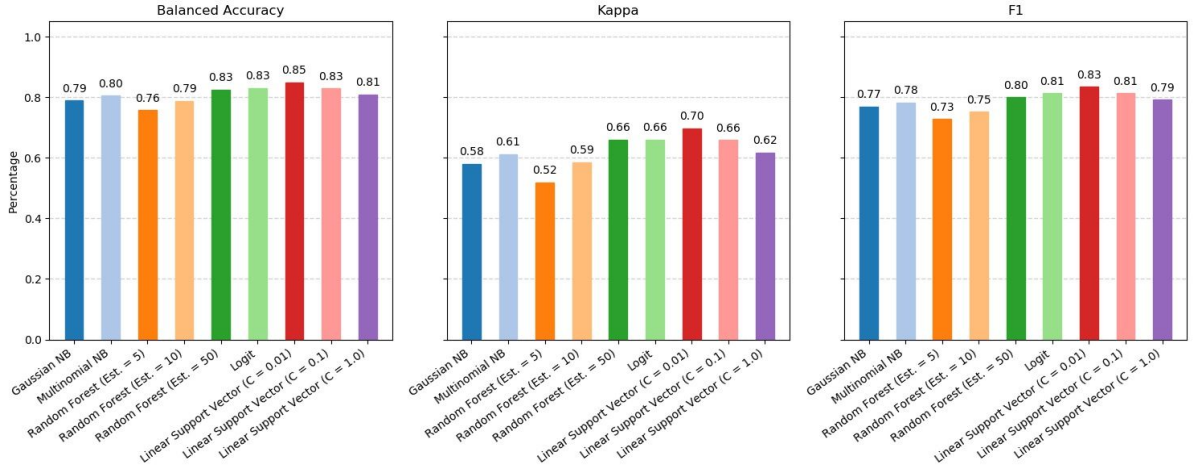


Graph 2.1: Calibration of Classification

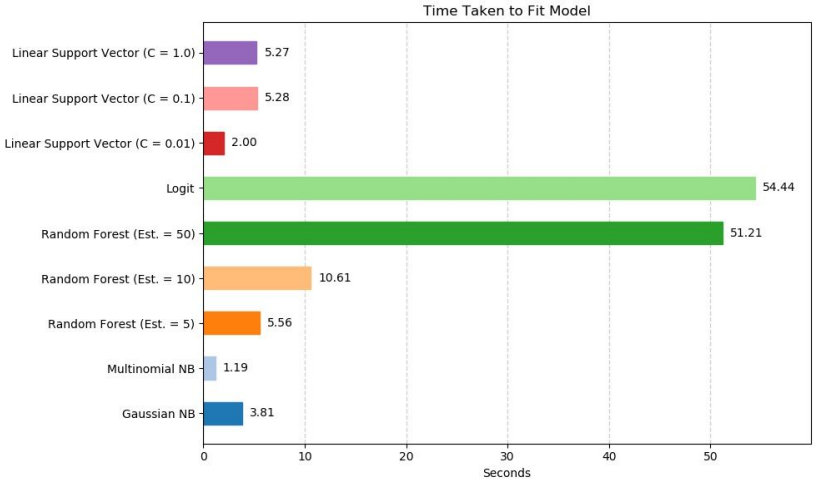
[3] Unfortunately completed after the final presentation



Graph 2.2: ROC and PR curves



Graph 2.3: Accuracy, Kappa and F1 measure of different algorithms.



Graph 1.4: Time taken to fit model.

Discussion

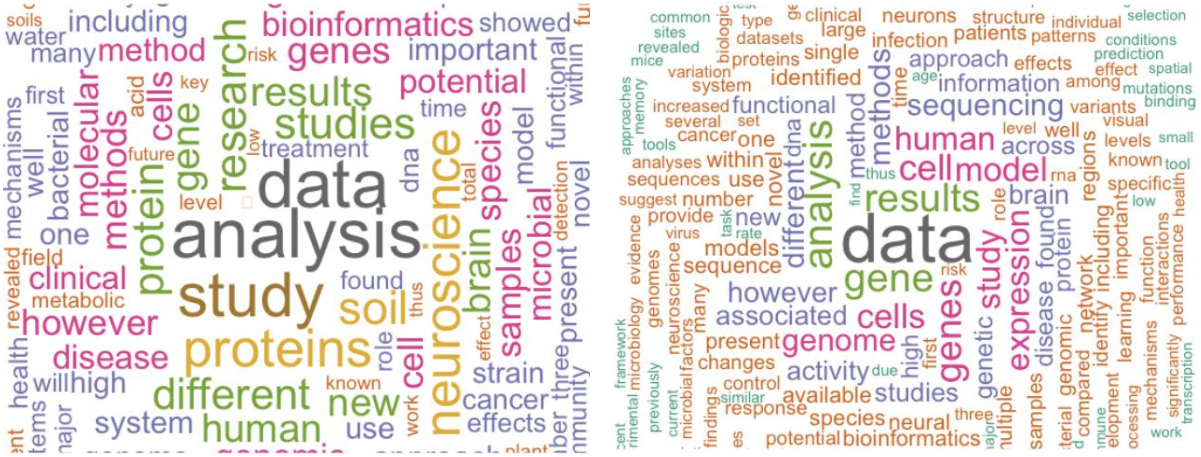


Fig 2. Word Cloud of PubMed (left) and Biorxiv (right) before pre-processing.

Above are word cloud graph of PubMed and Biorxiv before pre-processing which shows basic patterns of these two databases. According to them, we can found that they shared similar word patterns. Since a given color and size shows a particular range of word frequency, we can find that for words in PubMed, the word frequency is descending gradually; however, for the words in Biorxiv, only several green words around black word data, which lack of brown words and yellow ones so that the word frequency in Biorxiv is descending suddenly, or say the words in Biorxiv show more diversity. It makes sense that it is a pre-published corpus which may contain broader topics, but for the PubMed which only contains published paper which may focus on some popular topic.

Contrary to our hypothesis, we found there exists a marked difference between the two repositories. Due to the goal of the project and time constraints, no reason can be provided at this time. The possibility that there is information leakage cannot be ruled out. It was hoped, post-presentation, that limiting the maximum document frequency might show an imbalance, but changing the value significantly had minimal effect on the results. As mentioned in the presentation feedback, the S curves on the reliability plots may indicate overfitting. Increasing or randomizing the test set may alleviate this. Another option would be to introduce unrelated documents from both sources, such as articles from different topics.

We had originally considered adding more algorithms on our data (e.g., Bagging, non-linear SVC, k-Neighbors, NN, etc.) and comparing their performance with the other methods. However, given the results, a better path would be to determine methods which might provide more insight into the qualitative nature of the differences. Possible avenues include using word vectors instead of bag-of-words approach or analyzing part-of-speech tags as features attached terms (further distinguishing between word usage or even part of speech preference).

Our work was conducted largely online. We discussed and shared progress via Slack or email instead of a face-to-face meeting. This permitted a more fluid workflow, without having

to negotiate and strategize meeting times. We work separately and focuses on different parts of the work but discussed the problems we encountered together. Since different group member has different skills and background, we tried our best to let all group members involved in this project, but the most difficult parts were solved by particular people based on their specialties.

Appendix

Preprocessing Regex

numbers	`[0-9]+?`
url	`(https?:?\\?\\?)?(www\.)?[a-zA-Z0-9_\-\%]+?\.[a-z]{2,3}(\.[a-z]{2,3})\s`
email	`([a-zA-Z0-9]+?\(?\s?\{?(at|\@)\}?\)?\s?.+?\s)`
hyphens & underscores	`[\_\-]`
punctuation	`[\[\]\'\.,\/\#\!\?\$\%\^\&\*;\:{}=\_`~\(\)\n\r\<\>\@\\\]+?`
multiple spaces	`\s{2,}`

Code

main.py

```
#/usr/bin/env python

"""
   IS590ML - Final Project
   Preprocessing for comparison of 'Pubmed' against 'BiorXiv' article abstracts
"""
import csv
import time
import pickle

import pandas as pd
```

```
from src.helpers import PATHS, get_raw_data, clean_tmp_data,
get_interim_corpus_path, get_processed_corpus_path
from src.data.extraction import CorpusExtractor
from src.data.preprocess import Preprocessor
from src.features.lsa import LSA
from src.models.classification import Classifier
from src.visualization.classification import ScorePlotter, TimePlotter

def preprocess():
   csv.field_size_limit(10000000)

   preprocessor = Preprocessor()

   outpath = (PATHS.PROCESSED / 'corpus.tsv')
   outpath.touch()

   with get_interim_corpus_path().open('r') as infd, outpath.open('w',
errors='ignore', newline='\n') as outfd:
       infile = csv.DictReader(infd, delimiter='\t', quoting=csv.QUOTE_NONE)
       outfile = csv.DictWriter(outfd, infile.fieldnames, delimiter='\t')

       outfile.writeheader()
       count = 0
       current_id = None
       start_time = time.time()
       try:
           for row in infile:
               count += 1
               if count % 100 == 0:
                   print(count, ':', time.time() - start_time)
               new_row = row.copy()
               current_id = new_row['id']
               new_row['abstract'] = preprocessor.preprocess(row.get('abstract'))
               outfile.writerow(new_row)
       except Exception as e:
           print(count, current_id, e)


def run_vectorizer(max_features, max_df, file_suffix=None):
   lsa = LSA(PATHS.PROCESSED / 'corpus.tsv', max_tfidf_features=max_features,
max_df=max_df)
   training = {
       'X': lsa.training_corpus['abstract'],
       'y': lsa.training_corpus['source']
   }
   testing = {
       'X': lsa.testing_corpus['abstract'],
       'y': lsa.testing_corpus['source'],
   }

   X = lsa.vectorizer.fit_transform(training.get('X'), training.get('y'))
   X_test = lsa.vectorizer.transform(testing.get('X'))
```

```
    X_df = pd.DataFrame(X.toarray(), columns=lsa.vectorizer.get_feature_names(),
index=lsa.training_corpus.index)
    print(X_df.head())
    X_df.insert(0, '_label_', training.get('y'))

    X_test_df = pd.DataFrame(X_test.toarray(),
columns=lsa.vectorizer.get_feature_names(), index=lsa.testing_corpus.index)
    X_test_df.insert(0, '_label_', testing.get('y'))

    training_filepath = 'vec_training_set'
    testing_filepath = 'vec_testing_set'

    if file_suffix:
        training_filepath += '_{}'.format(file_suffix)
        testing_filepath += '_{}'.format(file_suffix)

    training_filepath += '.pkl'
    testing_filepath += '.pkl'

    X_df.to_pickle(PATHS.PROCESSED / training_filepath)
    X_test_df.to_pickle(PATHS.PROCESSED / testing_filepath)

    print("Number of features:", len(lsa.vectorizer.get_feature_names()))

def run_classifier(file_suffix):
    classifier = Classifier(file_suffix)

    training_filepath = 'vec_training_set'
    testing_filepath = 'vec_testing_set'

    if file_suffix:
        training_filepath += '_{}'.format(file_suffix)
        testing_filepath += '_{}'.format(file_suffix)

    training_filepath += '.pkl'
    testing_filepath += '.pkl'

    training = pd.read_pickle(PATHS.PROCESSED / training_filepath)
    testing = pd.read_pickle(PATHS.PROCESSED / testing_filepath)

    y = training['_label_']
    X = training.drop('_label_', axis=1)

    y_test = testing['_label_']
    X_test = testing.drop('_label_', axis=1)

    stats = classifier.run(X, y, X_test, y_test)

    filename = 'classification_scores'

    if file_suffix:
        filename += '_{}'.format(file_suffix)
```

```
    filename += '.pkl'

    with (PATHS.REPORTS / filename).open('wb') as fd:
        pickle.dump(stats, fd)


def run_models(max_df=0.65, file_suffix=None):
    run_vectorizer(10000, max_df=max_df, file_suffix=file_suffix)
    run_classifier(file_suffix)

    filename = 'classification_scores'

    if file_suffix:
        filename += '_{}'.format(file_suffix)

    filename += '.pkl'

    stats = None
    with (PATHS.REPORTS / filename).open('rb') as fd:
        stats = pickle.load(fd)

    score_plotter = ScorePlotter(file_suffix=file_suffix)
    score_plotter.run(stats)
    score_plotter.save()

    time_plotter = TimePlotter(file_suffix=file_suffix)
    time_plotter.run(stats)
    time_plotter.save()

if __name__ == "__main__":
    corpus_extractor = CorpusExtractor(get_raw_data())
    corpus_extractor.run()

    preprocess()

    run_models(max_df=0.65, file_suffix='max_df_65')
    run_models(max_df=0.4, file_suffix='max_df_40')
    run_models(max_df=0.1, file_suffix='max_df_10')
```

src/data/extraction.py

```
#/usr/bin/env python

"""
    IS590ML - Final Project
    Preprocessing for comparison of 'Pubmed' against 'BiorXiv' article abstracts
"""
import re
import csv
from xml.etree import ElementTree as ET
```

```
from src.helpers import PATHS, get_raw_data, clean_tmp_data


class CorpusExtractor():
    def __init__(self, raw_data):
        self.raw_data = raw_data
        self.headers = [
            'pmid',
            'pmc',
            'doi',
            'title',
            'abstract',
            'body',
            'year',
            'journal',
            'publisher'
        ]

        self.spacings_regex = re.compile(r'(\t|\n+?|\r+?|\s{2,})')

    def run(self):
        with (PATHS.INTERIM / 'corpus.tsv').open('w', errors='xmlcharrefreplace',
newline="\n") as fd:
            csv_file = csv.DictWriter(fd, self.headers, delimiter='\t',
extrasaction='ignore', lineterminator='\n')
            csv_file.writeheader()

            for instance in self.raw_data:
                row = self.get_article_data(instance)
                try:
                    csv_file.writerow(row)
                except Exception as e:
                    print(instance)
                    raise(e)


    def clean_text(self, text):
        try:
            return  ' '.join([self.spacings_regex.sub(' ', s.strip()) for s in
text.itertext()]).strip()
        except:
            return


    def get_ids(self, parent_tag, instance):
```

```
        id_texts = {}
        try:
            id_list = parent_tag.findall('article-id')

            for article_id in id_list:
                id_type = article_id.get('pub-id-type')
                id_text = self.clean_text(article_id)

                if len(id_text) > 64:
                    print(id_type, id_text)
                    raise(Exception("There was an error with an id"))

                if id_type == 'doi':
                    id_text = 'https://doi.org/' + str(id_text)


                id_texts[id_type] = id_text

        except Exception as e:
            print(e)
            raise(e)

        instance.update(id_texts)
        return instance


    def get_title(self, parent_tag, instance):
        title = None
        try:
            title = parent_tag.find('article-title')
            if not title:
                title = parent_tag.find('title-group').find('article-title')
            title = self.clean_text(title)
        except Exception as e:
            print(instance.get('pmc'), 'title', e)

        instance[title] = title
        return instance


    def get_abstract(self, parent_tag, instance):
        abstract = None
        try:
            abstract = self.clean_text(parent_tag.find('abstract'))
        except Exception as e:
```

```
            print(instance.get('pmc'), 'abstracat', e)

        instance['abstract'] = abstract
        return instance


    def get_journal(self, parent_tag, instance):
        journal = None
        try:
            journal = self.clean_text(parent_tag.find('journal-title'))
        except Exception as e:
            print(instance.get('pmc'), 'journal', e)

        instance['journal'] = journal
        return instance


    def get_publisher(self, parent_tag, instance):
        publisher = None
        try:
            publisher = parent_tag.find('publisher-name')
            if not publisher:
                publisher = parent_tag.find('publisher').find('publisher-name')
            publisher = self.clean_text(publisher)
        except Exception as e:
            print(instance.get('pmc'), 'publisher', e)
            publisher = ''

        instance['publisher'] = publisher
        return instance


    def get_year(self, parent_tag, instance):
        year = None
        try:
            pub_dates = parent_tag.findall('pub-date')

            for pub_date in pub_dates:
                year = pub_date.find('year')
                if year:
                    break

            year = self.clean_text(year)
        except Exception as e:
            print(instance.get('pmc'), 'date', e)
```

```
        instance['year'] = year
        return instance



    def get_article_data(self, xml_file):

        tree = ET.parse(xml_file)
        root = tree.getroot()
        instance_tmpl = {header: None for header in self.headers}

        instance = instance_tmpl.copy()

        try:
            front_tag = root.find('front')
            article_meta = front_tag.find('article-meta')
            journal_meta = front_tag.find('journal-meta')

            instance = self.get_ids(article_meta, instance)
            instance = self.get_title(article_meta, instance)
            instance = self.get_year(article_meta, instance)
            instance = self.get_abstract(article_meta, instance)
            instance = self.get_journal(journal_meta, instance)
            instance = self.get_publisher(journal_meta, instance)
            instance['body'] = self.clean_text(root.find('body'))


        except Exception as e:
            print(xml_file, e)
            pass

        return instance
```

src/data/preprocess.py

```
#/usr/bin/env python

"""
   IS590ML - Final Project
   Preprocessing for comparison of 'Pubmed' against 'BiorXiv' article abstracts
"""
from pathlib import Path
import re
```

```python
import pickle

from src.helpers import PATHS, get_stopwords

import nltk
# download necessary libraries
nltk.download('punkt', quiet=True)
nltk.download('wordnet', quiet=True)
nltk.download('averaged_perceptron_tagger', quiet=True)

from nltk import PorterStemmer, word_tokenize, pos_tag
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer

from sklearn.preprocessing import Normalizer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer,
TfidfVectorizer
from sklearn.decomposition import TruncatedSVD

import numpy as np
import pandas as pd



# Only look at terms which occur in MAX_DF or fewer documents (ignoring extremely
common words)
# 65% tends to be a good starting point

class Preprocessor():
   def __init__(self,
       min_word_len=3,
       lemmatize=True,
       append_pos_tags=False,
       new_stopwords=[]
   ):
       self.min_word_len = min_word_len
       self.append_pos_tags = append_pos_tags

       self.lemmatize = lemmatize
       self.stemmer = PorterStemmer()
       self.lemmatizer = WordNetLemmatizer()
       self.stopwords = get_stopwords(new_stopwords)

       self.regex = [
```

```
            ('number',  re.compile(r'[0-9]+'), ' '),
            ('url',
re.compile(r'(https?:?\\?\\?)?(www\.)?[a-zA-Z0-9_\-\%]+?\.[a-z]{2,3}(\.[a-z]{2,3}
)\s'), ' '),
            ('email',
re.compile(r'([a-zA-Z0-9]+?\(?\s?\{?(at|\@)\}?\)?\s?.+?\s)'), ' '),
            ('hyscore', re.compile(r'[\_\-]'), ' '), # add spacing
            ('punct',
re.compile(r'[\[\]\'\.,\/\#\!\?\$\%\^\&\*;\:{}=\_`~\(\)\n\r\<\>\@\\\]+?'), '
'),
            ('spaces',  re.compile(r'\s{2,}'), ' '),
        ]

    def update_stopwords(self, new_stopwords=[]):
        self.stopwords += new_stopwords

    def get_wordnet_pos(self, treebank_tag):
        """ Annoying conversions to wordnet abbreviations. """
        if treebank_tag.startswith('J'):
            return wordnet.ADJ
        elif treebank_tag.startswith('V'):
            return wordnet.VERB
        elif treebank_tag.startswith('N'):
            return wordnet.NOUN
        elif treebank_tag.startswith('R'):
            return wordnet.ADV
        else:
            return ''

    def preprocess(self, doc):
        """ Convert doc to lowercase, remove apostrophes and numbers. """
        new_words = []
        # print("PREPROCESSING.")

        doc = str(doc)

        doc = doc.lower()
        for name, edit, rep in self.regex:
            doc = edit.sub(rep, doc)
        doc = pos_tag(word_tokenize(doc))

        for word, tag in doc:
            tag = self.get_wordnet_pos(tag)

            if word in self.stopwords or len(word) < self.min_word_len:
```

```
                # ignore words with 2 or fewer characters
                continue

            try:
                float(word) # ignore numbers
            except ValueError:
                pass
            else:
                continue

            if tag and tag != '' and self.lemmatize:
                word = self.lemmatizer.lemmatize(word, tag)

            # remove lemmatized terms
            if word in self.stopwords:
                # ignore words with 2 or fewer characters
                continue

            if self.append_pos_tags:
                word = word + '(' + tag + ')'

            new_words.append(word)

        doc = ' '.join(new_words)
        return doc
```

src/features/lsa.py

```
#/usr/bin/env python

"""
   IS590ML - Final Project
   Preprocessing for comparison of 'Pubmed' against 'BiorXiv' article abstracts
"""
from pathlib import Path
import re
import pickle

from src.helpers import PATHS

import nltk
# download necessary libraries
nltk.download('punkt', quiet=True)
nltk.download('wordnet', quiet=True)
```

```
nltk.download('averaged_perceptron_tagger', quiet=True)

from nltk import PorterStemmer, word_tokenize, pos_tag
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer

from sklearn.preprocessing import Normalizer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer,
TfidfVectorizer
from sklearn.decomposition import TruncatedSVD

import numpy as np
import pandas as pd

class LSA():
   def __init__(self,
       corpus_path=None,
       corpus_frac=None,
       max_df=0.65,
       min_df=25,
       max_tfidf_features=None,
       n_components=100,
       label_col=None,
       id_col='id',
   ):
       self.tokenizer = str.split

       self.corpus = self.load_corpus(corpus_path, label_col=label_col,
corpus_frac=corpus_frac)
       self.training_corpus, self.testing_corpus = self.split_corpus()

       self.vectorizer_params = {
           # 'lowercase': True,  # Covered by preprocessor
           # 'stop_words': self.stopwords,  # Covered by preprocessor
           'analyzer': 'word',
           'tokenizer': self.tokenizer,
           'max_df': max_df,
           'min_df': min_df,
           'max_features': max_tfidf_features,
       }

       self.transformer_params = {
           'sublinear_tf': True
```

```
        }

        self.decomposer_params = {
            'n_components': n_components,
            'n_iter': 5
        }

        self.pipeline = None

        self.vectorizer = self.update_vectorizer()
        self.transformer = self.update_transformer()
        self.decomposer =  self.update_decomposer()

        self.pipeline_steps = [
            ('vectorizer', self.vectorizer),
            ('transformer', self.transformer),
            ('decomposer', self.decomposer),
            ('normalizer', Normalizer(copy=False))
        ]

        self.pipeline = Pipeline(steps=self.pipeline_steps)

    def update_vectorizer(self, **kwargs):
        """ Update vectorization function (CountVectorizer). """
        if kwargs:
            self.vectorizer_params.update(**kwargs)

            if 'max_df' in kwargs:
                self.max_df = kwargs.get('max_df')

            if 'max_features' in kwargs:
                self.max_features = kwargs.get('max_features')

        self.vectorizer = CountVectorizer(**self.vectorizer_params)

        # set LSA params if needed
        if self.pipeline:
            self.pipeline.set_params(**{'vectorizer__{}'.format(k): v for k,v in
self.vectorizer_params.items()})

        return self.vectorizer

    def update_transformer(self, **kwargs):
        """ Update transformation function (TFIDFTransformer). """
        if kwargs:
```

```
            self.transformer_params.update(**kwargs)

        self.transformer = TfidfTransformer(**self.transformer_params)

        if self.pipeline:
            self.pipeline.set_params(**{'transformer__{}'.format(k): v for k,v in
self.vectorizer_params.items()})

        return self.transformer

    def update_decomposer(self, **kwargs):
        """ Update decomposition function (TruncatedSVD). """
        if kwargs:
            self.decomposer_params.update(**kwargs)

        self.decomposer = TruncatedSVD(**self.decomposer_params)

        # set LSA params if needed
        if self.pipeline:
            self.pipeline.set_params(**{'decomposer__{}'.format(k): v for k,v in
self.decomposer_params.items()})

        return self.decomposer

    def update_pipeline(self):
        """ Update pipeline to capture any changes to internal models """
        self.pipeline = Pipeline(steps=self.pipeline_steps)
        return self.pipeline

    def load_corpus(self, filepath, label_col=None, id_col='id',
corpus_frac=None):
        self.corpus = pd.read_csv(
            filepath,
            sep='\t',
            header=0,
            na_values=['None', ''],
            keep_default_na=False,
            dtype=str)

        self.corpus.drop_duplicates(id_col, keep='first', inplace=True)
        self.corpus.dropna(inplace=True)

        self.corpus.set_index(id_col, drop=True, inplace=True)

        if label_col:
```

```
            self.corpus.rename(columns={label_col: 'label'}, inplace=True)

        if corpus_frac:
            original_size = self.corpus.shape[0]
            self.corpus = self.corpus.sample(frac=corpus_frac, random_state=17)
            print('Reducing corpus to {:02%} of original size: {} to
{}'.format(corpus_frac, original_size, self.corpus.shape[0]))

        return self.corpus

    def split_corpus(self, test_size=0.2):
        self.training_corpus, self.testing_corpus = train_test_split(self.corpus,
test_size=test_size, random_state=17)
        return self.training_corpus, self.testing_corpus

    def store(self):
        with (PATHS.MODELS / 'lsa.pkl').open('wb') as fd:
            pickle.dump(self, fd)
```

src/models/classification.py

```
#/usr/bin/env python

"""
   IS590ML - Final Project
   Preprocessing for comparison of 'Pubmed' against 'BiorXiv' article abstracts
"""
from pathlib import Path
import time

from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
from sklearn.svm import LinearSVC

from sklearn.preprocessing import LabelBinarizer
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import RepeatedStratifiedKFold, RepeatedKFold,
StratifiedKFold, GridSearchCV, train_test_split
from sklearn.metrics import classification_report, cohen_kappa_score as kappa,
confusion_matrix, auc, roc_auc_score, roc_curve, precision_recall_curve,
average_precision_score, f1_score, balanced_accuracy_score, recall_score
from sklearn.calibration import calibration_curve
```

```python
from src.helpers import PATHS
import pandas as pd

from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
from sklearn.svm import LinearSVC

from src.visualization.calibration import CalibrationPlotter
from src.visualization.classification import ScorePlotter
from src.visualization.rocauc import RocAucPlotter

class Classifier():
    def __init__(self, file_suffix):
        self.file_suffix = file_suffix
        self.calib_plotter = CalibrationPlotter(file_suffix=file_suffix)
        self.roc_auc_plotter = RocAucPlotter(file_suffix=file_suffix)
        self.score_plotter = ScorePlotter(file_suffix=file_suffix)

        self.classifiers = {
            'Gaussian Naive Bayes': GaussianNB(),
            'Multinomial Naive Bayes': MultinomialNB(),
            'Random Forest (Est. = 5)': RandomForestClassifier(n_estimators=5),
            'Random Forest (Est. = 10)': RandomForestClassifier(n_estimators=10),
            'Random Forest (Est. = 50)': RandomForestClassifier(n_estimators=50),
            'Logit': LogisticRegression(solver='lbfgs', max_iter=1000),
            'Linear Support Vector (C = 0.01)': LinearSVC(C=0.01),
            'Linear Support Vector (C = 0.1)': LinearSVC(C=0.1),
            'Linear Support Vector (C = 1.0)': LinearSVC(C=1.0),
        }

    def run(self, X, y, X_test, y_test, file_suffix=None):
        filename = 'classification_report'
        if self.file_suffix or file_suffix:
            filename += '_{}'.format(self.file_suffix or file_suffix)

        filename += '.txt'

        print(filename)

        total_stats = {}

        with (PATHS.REPORTS / filename).open('w') as report_file:
            for name, clf in self.classifiers.items():
                print("Classifying with", name)
```

```
                try:
                    stats = self.classify(clf, X, y, X_test, y_test, name=name,
report_file=report_file)
                    total_stats[name] = stats
                except Exception as e:
                    print(e)

        self.score_plotter.run(total_stats)
        self.calib_plotter.save()
        self.roc_auc_plotter.save()
        self.score_plotter.save()

        return total_stats

    def classify(self, clf, X, y, X_test_final, y_test_final, name='', folds=4,
report_file=None):
        skf = StratifiedKFold(folds)

        fit_times = {}
        stats = {}

        if report_file:
            report_file.write('\n--- Classifying with {} ---\n'.format(name))

            for i, (train_index, test_index) in enumerate(skf.split(X, y)):
                X_train, X_test = X.iloc[train_index], X.iloc[test_index]
                y_train, y_test = y.iloc[train_index], y.iloc[test_index]

                start_time = time.time()
                clf.fit(X_train, y_train)
                fit_time = time.time() - start_time

                y_pred = clf.predict(X_test)
                score = clf.score(X_test, y_test)

                if hasattr(clf, 'predict_proba'):
                    y_score = clf.predict_proba(X_test)[:, 1]
                else: # use decision function
                    y_score = clf.decision_function(X_test)
                    y_score = (y_score - y_score.min()) / (y_score.max() -
y_score.min()) # normalize

                bas = balanced_accuracy_score(y_test, y_pred)
                average_precision = average_precision_score(y_test, y_score,
pos_label='pubmed')
```

```
                f1 = f1_score(y_test, y_pred, pos_label='pubmed')
                k = kappa(y_test, y_pred)

                stats['Fold {}'.format(i)] = {
                    'time': fit_time,
                    'mean accuracy': score,
                    'balanced accuracy': bas,
                    'avg precision': average_precision,
                    'f1': f1,
                    'kappa': k
                }

                if report_file:
                    report_file.write('\n--- Fold #{} ---\n\n'.format(i))
                    report_file.write('Fit Time: {} seconds\n'.format(fit_time))
                    report_file.write('Kappa: {}\n'.format(score))
                    report_file.write('Mean Accuracy: {}\n'.format(score))
                    report_file.write('Average Precision (Pubmed):
{}\n'.format(average_precision))
                    report_file.write('Classification report:\n'.format(i))
                    report_file.write(classification_report(y_test, y_pred))
                    report_file.write('\n---\n\n')


        # Final testing
        start_time = time.time()
        clf.fit(X, y)
        fit_time = time.time() - start_time

        fit_times['Final'] = fit_time

        y_pred_final = clf.predict(X_test_final)
        score = clf.score(X_test_final, y_test_final)


            ##
https://scikit-learn.org/stable/auto_examples/calibration/plot_compare_calibratio
n.html
        if hasattr(clf, 'predict_proba'):
            y_score = clf.predict_proba(X_test_final)[:, 1]
        else: # use decision function
            y_score = clf.decision_function(X_test_final)
            y_score = (y_score - y_score.min()) / (y_score.max() - y_score.min())
# normalize
```

```
        bas = balanced_accuracy_score(y_test_final, y_pred_final)
        average_precision = average_precision_score(y_test_final, y_score,
pos_label='pubmed')
        f1 = f1_score(y_test_final, y_pred_final, pos_label='pubmed')
        k = kappa(y_test_final, y_pred_final)

        stats['Final'] = {
            'time': fit_time,
            'mean accuracy': score,
            'balanced accuracy': bas,
            'avg precision': average_precision,
            'f1': f1,
            'kappa': k
        }

        positives_frac, mean_pred_val = calibration_curve(y_test_final, y_score,
n_bins=10)
        self.calib_plotter.plot(mean_pred_val, positives_frac, name)
        self.calib_plotter.hist(y_score, name)

        fp_rate, tp_rate, threshold = roc_curve(y_test_final, y_score,
pos_label='pubmed')
        roc_auc = auc(fp_rate, tp_rate)
        self.roc_auc_plotter.roc_auc_curve(fp_rate, tp_rate, name, roc_auc)

        precision, recall, threshold = precision_recall_curve(y_test_final,
y_score, pos_label='pubmed')
        self.roc_auc_plotter.pr_curve(recall, precision,name, average_precision)


        if report_file:
            report_file.write('\n--- Test (Final) ---\n\n'.format(i))
            report_file.write('Fit Time: {} seconds\n'.format(fit_time))
            report_file.write('Kappa: {}\n'.format(score))
            report_file.write('Mean Accuracy: {}\n'.format(score))
            report_file.write('Average Precision (Pubmed):
{}\n'.format(average_precision))
            report_file.write('Classification Report:\n')
            report_file.write(classification_report(y_test_final, y_pred_final))
            report_file.write('\n--- End {} Classification ---\n'.format(name))

        return stats
```

src/visualization/calibration.py

```
#/usr/bin/env python

"""
   IS590ML - Final Project
   Preprocessing for comparison of 'Pubmed' against 'BiorXiv' article abstracts
"""
import matplotlib as mpl
from matplotlib import cm
import matplotlib.pyplot as plt

from src.helpers import PATHS

colors = plt.get_cmap('tab20').colors
plt.rcParams['axes.prop_cycle'] = plt.cycler(color=colors)

class CalibrationPlotter():
   def __init__(self, file_suffix):
       self.file_suffix = file_suffix
       # Calibration Plot
       self.figure = plt.figure(figsize=(10, 10))
       self.figure.subplots_adjust(left=0.05, right=1-0.05)
       self.figure.suptitle('Calibration of Classification', fontsize=16)
       self.axs = [
           plt.subplot2grid((3, 1), (0, 0), rowspan=2, fig=self.figure),
           plt.subplot2grid((3, 1), (2, 0), fig=self.figure)
       ]
       self.axs[0].plot([0, 1], [0, 1], "k:", label="Perfectly Calibrated")

       self.axs[0].grid(color='lightgray', linestyle='dashed', linewidth=1)
       self.axs[0].set_axisbelow(True)
       self.axs[0].set_ylabel("Fraction of positives", fontsize=12)
       self.axs[0].set_ylim([-0.05, 1.05])

       self.axs[0].set_title('Calibration plots (Reliability Curve)',
fontsize=14)
       for tick in (self.axs[0].get_xticklabels() +
self.axs[0].get_yticklabels()):
           tick.set_fontsize(10)

       self.axs[1].grid(color='lightgray', linestyle='dashed', linewidth=1)
       self.axs[1].set_axisbelow(True)
       self.axs[1].set_xlabel("Mean predicted value", fontsize=12)
       self.axs[1].set_ylabel("Count", fontsize=12)
```

```
        for tick in (self.axs[1].get_xticklabels() +
self.axs[1].get_yticklabels()):
            tick.set_fontsize(10)


    def plot(self, mean_pred_val, positives_frac, clf_name):
        self.axs[0].plot(mean_pred_val, positives_frac, "s-",
label="{}".format(clf_name), linewidth=4)

    def hist(self, y_score, clf_name):
        self.axs[1].hist(y_score, range=(0, 1), bins=10, label=clf_name,
histtype="step", lw=4)

    def save(self, file_suffix=None):
        self.axs[0].legend(loc="upper left")
        self.axs[1].legend(loc="upper center", ncol=2)

        filename = 'calibration_plot'
        if self.file_suffix or file_suffix:
            filename += '_{}'.format(self.file_suffix or file_suffix)
        filename += '.jpg'

        self.figure.savefig((PATHS.FIGURES / filename))
```

src/data/classification.py

```
#/usr/bin/env python

"""
   IS590ML - Final Project
   Preprocessing for comparison of 'Pubmed' against 'BiorXiv' article abstracts
"""
from src.helpers import PATHS

import pandas as pd
import numpy as np

import matplotlib as mpl
from matplotlib import cm
import matplotlib.pyplot as plt


colors = plt.get_cmap('tab20').colors
plt.rcParams['axes.prop_cycle'] = plt.cycler(color=colors)
```

```
TERM_TRANSLATIONS = {
   'Linear Support Vector': 'LinearSV',
   'Gaussian Naive Bayes': 'Gaussian NB',
   'Multinomial Naive Bayes': 'Multinomial NB',
}

class ClassificationPlotter():
   def __init__(self, filename_root=None, horizontal=False, file_suffix=None):
       self.filename_root = filename_root
       self.horizontal = horizontal
       self.file_suffix = file_suffix

       self.figure = None
       self.axs = {}

   def setup(self):
       self.bar_thickness = 0.5

       for name, ax in self.axs.items():
           ax.grid(color='lightgray', linestyle='dashed', linewidth=1, axis='x'
if self.horizontal else 'y')
           ax.set_axisbelow(True)
           ax.set_title(name)
           # ax.set_ylim([-0.0, 1.05])


   def run(self, clf_stats):
       self.stats = {}

       for model, v in clf_stats.items():
           if model in TERM_TRANSLATIONS:
               model = TERM_TRANSLATIONS[model]
           self.stats[model] = {
               k: v for k, v in v['Final'].items()
           }

       self.stats = pd.DataFrame(self.stats)
       self.clf_names = self.stats.columns
       self.index = np.arange(len(self.clf_names))


   def make_barplot(self, ax, scores, label=None, hide_dep_axis=False,
adjust_ticks=True, adjust_scale=False):
```

```
        barplot_fn = ax.barh if self.horizontal else ax.bar
        barplot = barplot_fn(self.index, scores, self.bar_thickness, label=label)
        self.format_barplot(ax, barplot, adjust_ticks=adjust_ticks,
adjust_scale=adjust_scale)

        if self.horizontal:
            ax.set_yticks(self.index)
            ax.set_yticklabels(self.clf_names)
            if hide_dep_axis:
                ax.set_xticklabels([])
        else:
            ax.set_xticks(self.index)
            ax.set_xticklabels(self.clf_names)
            if hide_dep_axis:
                ax.set_yticklabels([])


    def format_barplot(self, ax, barplot, adjust_ticks=False, adjust_scale=False):
        max_length = 0
        for i, rect in enumerate(barplot):
            rect.set_color(colors[i])

            # bar_thickness= rect.get_height() if self.horizontal else
rect.get_width()
            bar_length = rect.get_width() if self.horizontal else
rect.get_height()
            if bar_length > max_length:
                max_length = bar_length

            amount = '{:.02f}'.format(float(bar_length))

            if self.horizontal:
                ax.text(bar_length + 0.75, rect.get_y() + self.bar_thickness/2.,
amount, ha='left', va='center', fontsize=10)
                if adjust_scale:
                    ax.set_xlim([0.0, max_length * 1.1])

                if adjust_ticks:
                    for tick in ax.get_yticklabels():
                        tick.set_rotation(36)
                        tick.set_va('center')
            else:
                ax.text(rect.get_x() + self.bar_thickness/2., bar_length + 0.02,
amount, ha='center', va='bottom', fontsize=10)
                if adjust_scale:
```

```
                    ax.set_ylim([0.0, max_length * 1.1])

                if adjust_ticks:
                    for tick in ax.get_xticklabels():
                        tick.set_rotation(36)
                        tick.set_ha('right')



    def save(self, file_suffix=None):
        filename = self.filename_root
        if self.file_suffix or file_suffix:
            filename += '_{}'.format(self.file_suffix or file_suffix)
        filename += '.jpg'

        if self.figure:
            self.figure.tight_layout()
            self.figure.savefig((PATHS.FIGURES / filename))


class ScorePlotter(ClassificationPlotter):
    def __init__(self, file_suffix=None):
        super(ScorePlotter, self).__init__('classification_scores', False,
file_suffix=file_suffix)

        self.figure = plt.figure(figsize=(15, 6))
        self.figure.subplots_adjust(left=0.05, right=1-0.05, bottom=0.2,
wspace=0.1)

        self.axs = {
            'Balanced Accuracy': plt.subplot2grid((1,3), (0,0), fig=self.figure),
            'Kappa': plt.subplot2grid((1,3), (0,1), fig=self.figure),
            'F1': plt.subplot2grid((1,3), (0,2), fig=self.figure)
        }

        self.setup()
        self.axs['Balanced Accuracy'].set_ylabel('Percentage')

        for ax in self.axs.values():
            ax.set_ylim([-0.0, 1.05])


    def run(self, clf_stats):
        super(ScorePlotter, self).run(clf_stats)
        hide_dep_axis = False
```

```
        for score_name, ax in self.axs.items():
            stat = self.stats.loc[score_name.lower()]
            self.make_barplot(ax, stat, score_name, hide_dep_axis=hide_dep_axis,
adjust_ticks=True)
            if not hide_dep_axis:
                hide_dep_axis = True


class TimePlotter(ClassificationPlotter):
    def __init__(self, file_suffix=None):
        super(TimePlotter, self).__init__('classification_times', horizontal=True,
file_suffix=file_suffix)
        self.figure = plt.figure(figsize=(10, 6))

        self.axs = {
            'Time Taken to Fit Model': self.figure.add_subplot(111)
        }


        self.setup()
        self.axs['Time Taken to Fit Model'].set_xlabel('Seconds')

    def run(self, clf_stats):
        super(TimePlotter, self).run(clf_stats)
        self.make_barplot(self.axs['Time Taken to Fit Model'],
self.stats.loc['time'], adjust_ticks=False, adjust_scale=True)
```

src/visualization/rocauc.py

```
#/usr/bin/env python

"""
    IS590ML - Final Project
    Preprocessing for comparison of 'Pubmed' against 'BiorXiv' article abstracts
"""
import matplotlib as mpl
from matplotlib import cm
import matplotlib.pyplot as plt

from src.helpers import PATHS

colors = plt.get_cmap('tab20').colors
plt.rcParams['axes.prop_cycle'] = plt.cycler(color=colors)
```

```
class RocAucPlotter():
   def __init__(self, file_suffix):

       self.file_suffix = file_suffix

       self.figure = plt.figure(figsize=(20,10))
       self.figure.subplots_adjust(left=0.05, right=1-0.05)
       self.figure.suptitle('ROC and PR Curves', fontsize=16)
       self.axs = [
           plt.subplot2grid((1,2), (0,0), fig=self.figure),
           plt.subplot2grid((1,2), (0,1), fig=self.figure)
       ]
       self.axs[0].plot([0, 1], [0, 1], "k--")

       self.axs[0].grid(color='lightgray', linestyle='dashed', linewidth=1)
       self.axs[0].set_axisbelow(True)
       self.axs[0].set_ylabel('True Positive Rate', fontsize=12)
       self.axs[0].set_xlabel('False Positive Rate', fontsize=12)
       self.axs[0].set_ylim([-0.05, 1.05])

       self.axs[0].set_title('Receiver Operating Characteristic (ROC) Curves',
fontsize=14)
       for tick in (self.axs[0].get_xticklabels() +
self.axs[0].get_yticklabels()):
           tick.set_fontsize(10)

       self.axs[1].grid(color='lightgray', linestyle='dashed', linewidth=1)
       self.axs[1].set_axisbelow(True)
       self.axs[1].set_ylabel('Precision', fontsize=12)
       self.axs[1].set_xlabel('Recall', fontsize=12)
       self.axs[1].set_ylim([-0.05, 1.05])

       self.axs[1].set_title('Precision-Recall Curves', fontsize=14)
       for tick in (self.axs[1].get_xticklabels() +
self.axs[1].get_yticklabels()):
           tick.set_fontsize(10)

   def roc_auc_curve(self, fp_rate, tp_rate, clf_name, roc_auc_score):
       label = '{} (Area = {:.03f})'.format(clf_name, roc_auc_score)
       self.axs[0].plot(fp_rate, tp_rate, label=label, linewidth=4)

   def pr_curve(self, recall, precision, clf_name, avg_precision_score):
       label = '{} (AP = {:.03f})'.format(clf_name, avg_precision_score)
       self.axs[1].step(recall, precision, where='post', label=label,
```

```
linewidth=4)

    def save(self, file_suffix=None):
        self.axs[0].legend(loc='lower right')
        self.axs[1].legend(loc='lower right')

        filename = 'roc_auc_curve'
        if self.file_suffix or file_suffix:
            filename += '_{}'.format(self.file_suffix or file_suffix)
        filename += '.jpg'

        self.figure.savefig((PATHS.FIGURES / filename))
```

src/helpers.py

```
#/usr/bin/env python

"""
    IS590ML - Final Project
    Preprocessing for comparison of 'Pubmed' against 'BiorXiv' article abstracts
"""
from pathlib import Path
import os
import pickle
import tarfile
import lzma

class PATHS():
    ROOT = Path(os.path.realpath(__file__)).parent.parent

    SRC = ROOT / './src'

    DATA = ROOT / 'data'
    INTERIM = DATA / 'interim'
    CORPUS = INTERIM / 'corpus'
    OBJECTS = INTERIM / 'objects'
    PROCESSED = DATA / 'processed'
    RAW = DATA / 'raw'
    RAWCORPUS = RAW / 'xml'

    MODELS = ROOT / 'models'
    REFERENCES = ROOT / 'references'
    REPORTS = ROOT / 'reports'
    FIGURES = REPORTS / 'figures'
```

```
def get_raw_data():
    """ Extract raw corpus data to .tmp directory and return a file descriptor
iterator. """
    print('Extracting Corpus.')
    if not PATHS.RAWCORPUS.exists():
        with tarfile.open(PATHS.RAW / 'corpus.tar.xz', 'r:xz') as tar:
            tar.extractall(PATHS.RAWCORPUS)
    return PATHS.RAWCORPUS.glob('*.xml')

def clean_tmp_data():
    """ Remove uncompressed files from the .tmp directory. """
    for fd in PATHS.RAWCORPUS.glob('*.xml'):
        fd.unlink()
    PATHS.RAWCORPUS.rmdir()


def get_interim_corpus_path():
    """ Get formatted csv. """
    corpus_path = (PATHS.INTERIM / 'corpus.tsv')
    if not corpus_path.exists():
        print("No interim corpus found")
    return corpus_path

def get_processed_corpus_path():
    """ Get formatted csv. """
    corpus_path = (PATHS.PROCESSED / 'corpus.tsv')
    if not corpus_path.exists():
        print("No process corpus found. Creating file.")
        corpus_path.touch()
    return corpus_path

def get_stopwords(new_stopwords = []):
    with (PATHS.REFERENCES / 'stopwords.txt').open('r') as fd:
        return [line.strip() for line in fd] + new_stopwords
```

Please contact Tre Tomaszewski at jtomasz2(at)illinois.edu for entire code base