I. Introduction

Newswires as subscription services (also called wire services, and so named because of the historical method of transmission via telegraph) have existed since the 19th century as a way for smaller newspapers to minimize costs in covering events that are distant both geographically and in terms of a newspaper's primary focus. In this model, the subscription allows a local news editor access to a stream of articles written by reporters from across the wire service's network of bureaus. A problem caused by the high volume of articles coming across the newswire is that an editor would have to spend a large amount of time reading each article to ascertain what each one covers. A solution that the wire services developed was to have standardized internal topic designations given to each article, from which an editor could easily filter the articles coming across the newswire. Initially, this document text classification was done by hand. In this project we will implement a multilabel classification algorithm to automate this document text classification process.

Multilabel text classification is a problem within Natural Language Processing (NLP) where, given the contents of a document, we try to label the document via an algorithm. Multilabel classification is a generalization of more basic problems in NLP. The simplest case is binary classification, where a document can have one of two labels. In the next level up, multiclass classification, more there are more than two options for labels, but each article can only have one label. In the most generalized case, multilabel classification, an article can have a variable amount of labels which must be determined by the algorithm. This is the theoretical framework that best describes our problem. From a practical point of view, an article about President Trump's trip to Mar-a-Lago can be labeled as both politics and Florida. As we shall see, the dataset reflects this reality. We also present the simpler cases because for multilabel problems, a common approach is to take an ensemble of binary or multiclass classifiers that each cover a portion of the topic space and in aggregate cover the entire space. A separate scoring algorithm is then implemented to aggregate the ensemble into a final result.

An important component of an NLP analysis is transforming a document into a form that can be entered into an algorithm. One solution is to impose a term vector model, where each document is represented as a vector in a vector space where each dimension corresponds to a particular word or term. This representation removes information about the punctuation and the word order in the document, but gives us a mathematical representation that can be entered into an algorithm. The intention in this model is that the set of word vectors for each topic will reside in well defined regions whose boundaries can be learned by our algorithm. Furthermore, a combination of topics, which happens when an article is assigned multiple topics, would correspond with a combination of characteristic word vectors from each of the individual topics.

The analysis will be executed in a Jupyter notebook running Python with the Anaconda package distribution. Additional libraries used in the analysis include OpenSP Unix library and scikit-multilearn Python library, installed with the homebrew and pip package managers, respectively.

The remainder of this report will be organized as such: Data Acquisition and Cleaning, Exploratory Data Analysis, Statistical Inference, Multilabel Classification, and Conclusion.

II. Data Acquisition and Cleaning

The dataset (a corpus in NLP literature) for this project is a sampling from the Reuters newswire in 1987. It was originally collected for academic NLP research, and is now housed at the UCI Machine Learning Repository as a flat .tar.gz file. Once retrieved and opened, the data consist of 22 .sgm files, a legacy data storage format that was the precursor to the currently used .xml format. To continue, we need to implement the Unix function osx from the OpenSP library to convert from .sgm to .xml. With the .xml file in hand, we can read the data into native python data structures via the BeautifulSoup library.

Inspecting the raw data, the most obvious quality is that the sampling of articles isn't from the general newswire, but rather a subset that consists of financial commodities. Also, half of the articles do not have a topic assigned to them, while the other half do indeed range in the number of topic labels. To do an initial cleaning of the data, we remove the articles that don't have a label and the topics that have an occurence in the corpus of 3 or less.

III. Exploratory Data Analysis

As stated in the introduction, an important step in an NLP analysis is giving a mathematical representation to our document data. We will be utilizing a term vector model where each dimension corresponds to a word or term. A decision we have to make upfront is what exactly the dimensions will be and what will be measured within each axis. As a first pass to explore the data, we will make the dimensions, called a vocabulary or lexicon in NLP literature, be the set of all occurring words in our corpus of documents. From this we will remove a set of stop words (predefined in our software), which occur in the English language with such ubiquity that their presence or absence in an article would provide no distinguishing information in our word space. As for the measurements, we will take the counts of each word in our lexicon for each document. Thus, each document will have its own word vector.

Our goal for this section was to create some visualizations for this inherently abstract data. (See Appendix for full visualizations) The most effective use of these visualizations would be as a guide to easily compare another corpus to the one used for this analysis. However, we can learn some insights from our dataset visualizations in isolation. Both our text matrix, the aggregate of our word vectors with rows corresponding to documents in our corpus and columns to words in our lexicon, and our topic matrix, a similar concept but using the topic labels instead of document text, are sparse and have no inherent structure. The lack of structure is due to the fact that both the rows and columns have no inherent order. As for the sparsity, this is indicative of a high dimensional dataset where most of the space is empty. Additionally, we have a histogram and two Kernel Density Estimation (KDE) plots. The histogram shows the distribution of topic counts per article with a log count scale, while the two KDE plots show the distributions of topic and word occurrence counts within the entire corpus with a log occurrence scale. All three of these plots are heavily skewed to the right, indicating that high topic count, topic occurrence, and word occurence are all rare events.

III. Statistical Inference

Before applying a classification algorithm onto our data, we wanted to test if the labeled topics create distinct clusters in our term vector space. This would serve two purposes: first, to guard against randomly assigned labels, and second, to be able to score different vector representations of our word data. In addition to the previously introduced count vectors, we will test binary vectors, which measure the occurrence or absence of a word in a document, and a Term Frequency-Inverse Document Frequency (tfidf) vectors. Tfidf vectors are count vectors (term frequency) that have been weighted by a function of the inverse document frequency, where the document frequency is the word occurrence in the entire corpus divided by the corpus length. In our case we will be using the log inverse document frequency plus 1 as our idf function.

With our three vector spaces determined, we next need to define our hypothesis test. We will be using a difference of means t test framework. We will first filter our dataset to articles that have exactly one topic. Then for each topic we will calculate the arithmetic mean or centroid of its corresponding data cluster. We will then use the L2 norm (euclidean metric) to calculate the pairwise distance between all topic centroids. Finding the pair with the smallest distance, we will then apply the t test onto these two topic clusters. Implicit in this dimension reduction to a single number distance is that there is negligible covariance and that the variance does not differ significantly between words. We will be using the t-distribution to calculate a test statistic and an associated p value. One test will be calculated for each vector space representation.

The results of the hypothesis test in each of our three vector space representation were all statistically significant given a standard significance level of α=0.05. However, the test conducted in the tfidf representation had a p-value an order of magnitude less than the other two representations at p=0.00195. In a crude but practical sense, this result implies that there is an even clearer separation in topic clusters in the tfidf space as compared with the other two, at least when it comes to the closest two topics in each representation. Therefore, we will use the tfidf vector space for all subsequent analysis.

IV. Multilabel Classification (Machine Learning Algorithms)

a. Setup

As stated earlier, we will be implementing a multilabel classification onto our dataset, with an input of vector representation of each article in our tfidf space and an output of a set of topic labels. To begin the machine learning analysis, we searched online for example solutions for similar types of problems and found the following article: Deep Dive Into Multi-label Classification. To expedite the computation time for the initial algorithm exploration, we created a simplified dataset filtering by topic that were either the top five single topics or within the top five topic pairs. With this simplified dataset in hand, we implemented five algorithms suggested in the previously linked article. Most of these algorithms consisted of a binary classification algorithm coupled with a problem transformation algorithm to generalize to our multilabel problem. The two binary classifiers implemented were Logistic Regression and Gaussian Naive Bayes.

For the problem transformation, we have 4 methods implemented, though two of them are very similar. The first two methods, Binary Relevance and One vs. Rest classifier, have the same theoretical underpinning. For each individual label in our label set, we create a classifier to see if the article/tfidf vector does or does not have that label/lie within the region defined by the label cluster. Binary Relevance comes directly from the multilabel literature, while One vs. Rest comes from multiclass work (more than two possible labels but only one label allowed per data point). However, if you allow multiple positive label classifications to be valid then we come back to Binary Relevance. Additionally, these methods have separate implementations in Python's sklearn/skmultilearn ecosystem, and both of these were implemented in this project. Next we have the Classifier Chain method, where we give an arbitrary ordering to the labels and classify sequentially, in each step utilizing a conditional probability of the previously classified labels in the chain. Ideally we would add a third ensembling layer on the label ordering to remove any artefact that it might cause, but for the sake of expediency we've implemented a single label ordering instance. Finally, we have the Label Powerset method where each possible combination of positive and negative label instances is recast as its own label and classified individually. We should add that we've described the problem transformation algorithms in increasing order of computational demand.

The second paradigm in multilabel classification is Algorithm Adaptation, where we take an existing binary/multiclass classifier and change its inner workings to output a multilabel result (there is some ambiguity here because one way to implement an Algorithm Adaptation is to apply a Problem Transformation onto an existing algorithm and repackage it as a "new" algorithm). In this project we have one implementation of an algoritm adaptation based on k-Nearest Neighbors and aptly named Multilabel k-Nearest Neighbors. In the traditional k-Nearest Neighbors algorithm, once we've calculated who the k-nearest neighbors are we implement majority voting to figure out what the output label should be. Whereas in the mutilabel extension we apply a Naive Bayes algorithm on our knn data subset, allowing for multiple positive label instances.

The choice of evaluation metrics for multilabel classification algorithms is nontrivial and should vary based upon the intended use cases. However, for this algorithm exploration we've chosen to implement two metrics. sklearn's built-in accuracy score evaluates set-wise coincidence between an algorithm's predictions and our source-of-truth data. It ranges from 0 to 1, with 1 being optimal. That is to say, we only count a success when all labels in our prediction match all labels in the source-of-truth data. The second metric used is the hamming loss, evaluating component-wise coincidence between positive or negative label instances between prediction and source-of-truth. It ranges from 0 to 1, this time 0 being optimal. While for obvious reasons the accuracy score could be argued to be overly strict, we could also argue that the hamming loss is overly lenient since coincidence between each negative label instance is individually counted as a success. In combination, these two metrics give a broad evaluation of a model algorithm's performance. Additionally, given the multiclass roots of One vs. Rest classification, we've implemented a different evaluation scheme. For each binary classifier we calculate its own accuracy score, which effectively becomes a component-wise metric.

b. Implementation and Results

We initially started with the following five algorithms: Binary Relevance with Gaussian Naive Bayes, One vs. Rest with Logistic Regression, Classifier Chain with Logistic Regression, Label Powerset with Logistic Regression, and Multilabel k-Nearest Neighbors. Timing the execution on the simplified dataset, the first four algorithms took between less than 1 and a few minutes to complete, while Multilabel k-Nearest Neighbors ran for over an hour without reaching a solution. We manually stopped the algorithm removed it from further consideration. Among the four remaining algorithms, Binary Relevance with Gaussian Naive Bayes performed significantly worse than the rest. We therefore reran Binary Relevance with Logistic Regression, and with improved performance inferred that it was the Gaussian Naive Bayes that was underperforming. At this stage we were getting similar performance amongst the algorithms, with aggregate accuracy scores of between 0.86 and 0.90, and individual accuracy scores of 0.97 to 0.99 for the One vs. Rest classifier. Proceeding to the full dataset, all but the One vs. Rest Classifier came up with errors. However, the One vs. Rest Classifier scheme had similar accuracy scores of 0.97 to 0.99 and took 48 seconds to compute. As a side note, the individual accuracy scores of the One vs. Rest Classifier isn't directly comparable to the aggregate accuracy scores from the other methods, as one minus the average of the individual accuracy scores is similar to the hamming loss in the agregate methods, except for the fact that the hamming loss is also counting true negatives as successes.

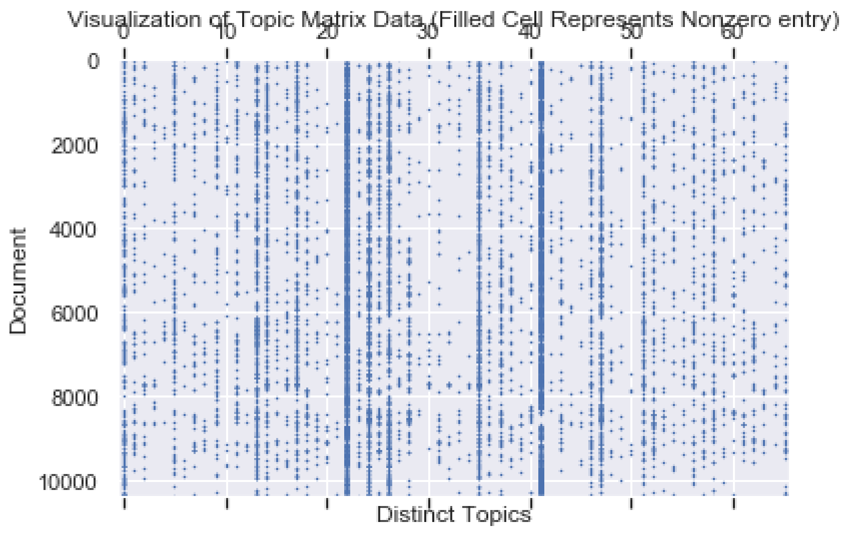
V. Conclusion

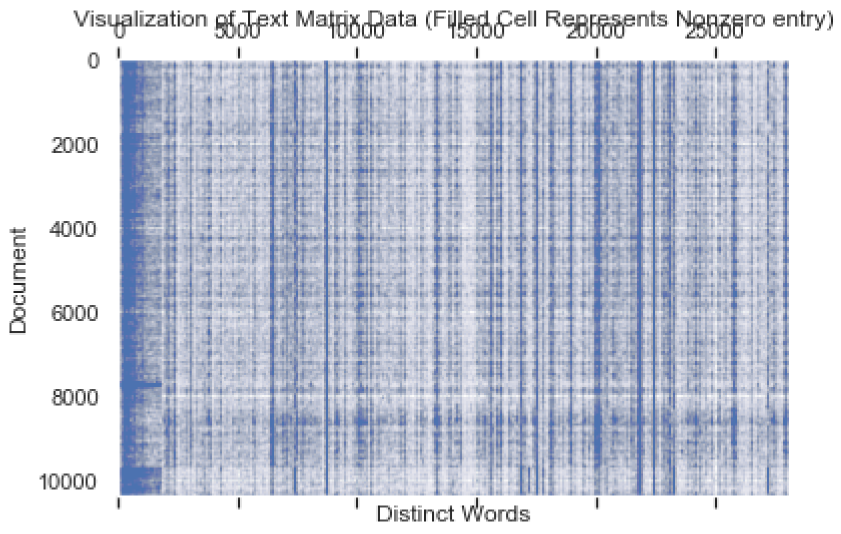
From a flat file of sampled (and dated) Reuters newswire articles, we have read the data into a native Python format, generated a term vector model, visualized the inherently abstract dataset, performed a hypothesis test on the separation between topics within the vector space, and conducted an initial algorithm exploration for the multilabel classification of our data. The data was formatted in a antiquated file standard that required running a unix subprocess to parse. The hypothesis test, in addition to lending evidence of adequate separation between topic clusters, also discriminated which term vector model should be used in the subsequent analysis. The algorithm exploration yielded one algorithm that functions over the full dataset.

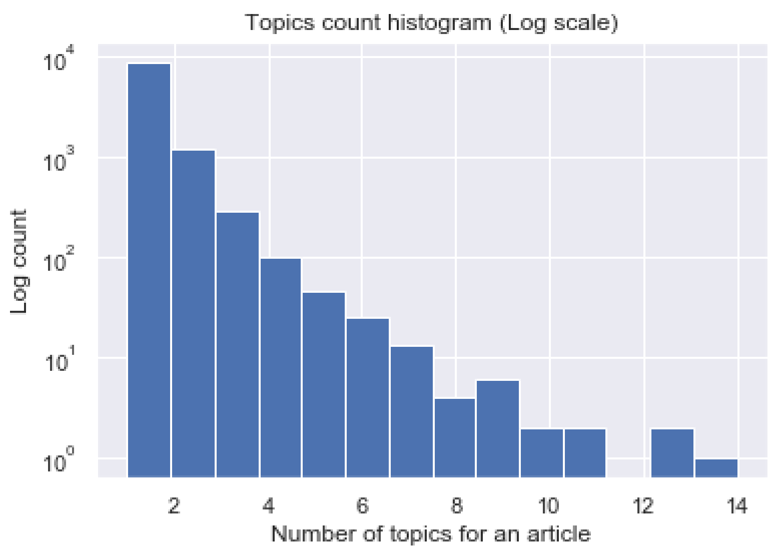
The results from the One vs. Rest Logistic Regression Classification look good on paper, but much would depend on the use case for the algorithm. In most reasonable use cases, topics are searched for one at a time rather than in aggregate, suggesting that the individual accuracy scores and hamming loss are better metrics than the aggregate accuracy score. In this context, an individual accuracy score between 0.97 and 0.99 means that in the search results for a particular topic, 97 to 99% of the articles will actually correspond the the search topic. For the algorithms with a hamming loss score, we can also argue that it places undue weight on true negatives. With a clearly defined use case, the optimal solution would be to create a custom scoring metric. We would also be able to say if the execution time is of any importance.

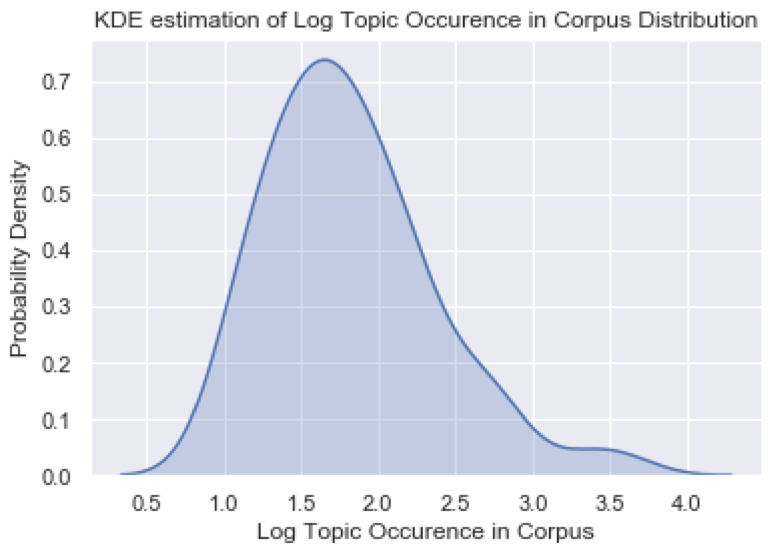
There are some improvements that could be implemented for this project. We could do an exhaustive literature search to account for the breadth of possible methods availble for multilabel classification. Further exploration of the skmultilearn documentation is warranted to ascertain why we have code failures in the full dataset. Where appropriate, we should do a proper parameter tuning, at least on the most promising algorithms. And implementing a ranking algorithm at the end of our pipeline would make the final results more useful. However, we've completed an initial algorithm exploration and have a functioning classification algorithm in hand. Any such improvements should be informed by the particular use case we wish to support.

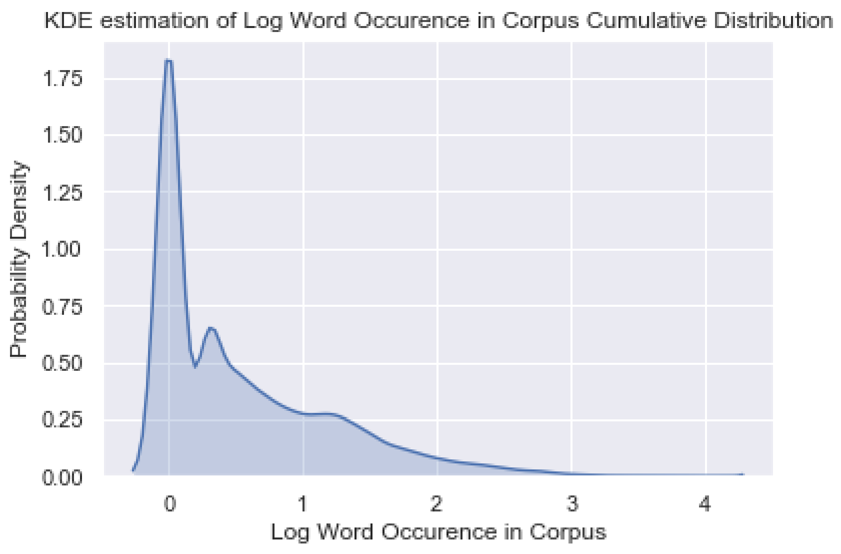
Appendix: Data Visualizations






Topics count histogram (Log scale)
Log count
10^4
10^3
10^2
10^1
10^0
2
4
6
8
10
12
14
Number of topics for an article


KDE estimation of Log Topic Occurence in Corpus Distribution
Probability Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Log Topic Occurence in Corpus


KDE estimation of Log Word Occurence in Corpus Cumulative Distribution
Probability Density
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0
1
2
3
4
Log Word Occurence in Corpus