Dimensionality Reduction Strategies for Latent Semantic Analysis

Amanda Bertsch

May 2020

1 Introduction

Latent semantic analysis (LSA) is an unsupervised learning technique for topic modeling across large corpora of documents. It is widely used in the natural language processing community today and has been described as "the most prominent algebraic learning algorithm used for information retrieval" (Al-Sabahi et al. 2019). LSA is widely used both because of its strong performance across a variety of tasks and its status as an unsupervised learning algorithm. Proponents of unsupervised learning argue that a truly semantic model should produce similar-to-human performance after analyzing the text itself, without any added human-performed annotation (Lanaeur et al. 2007). This is in contrast to supervised learning algorithms, which train on pre-labeled data.

One task that LSA exhibits strong performance in is topic modeling. The goal of topic modeling is to take a corpus of documents and group them into some number of categories based on the topic each document discusses. This is a special case of document classification. Topic modeling is widely considered to be interconnected with other important tasks including document similarity analysis, document search, and document summarization (Allahyari et al. 2017).

While some topic modeling schemes attempt to assign multiple topics at various points within each document, LSA treats the entire document as a bag of words and assigns it a probability distribution across the topics (Allahyari et al. 2017). A primary topic for each document is assigned as the most probable topic. When latent semantic analysis is applied, the number of topics is determined not through the model, but through some other heuristic or prior knowledge and input as a parameter into the model.

Broadly, latent semantic analysis is performed by constructing a document term matrix containing the frequency of each word over the document list and then reducing the dimension of that matrix to draw conclusions about which topics are most prevalent in certain documents. Each topic is itself a probability

distribution over the words in the corpus (Allahyari et al. 2017).

The key step in latent semantic analysis is the dimensionality reduction. Traditionally, this is accomplished by computing the truncated SVD (Al-Sabahi). However, other dimensionality reduction methods have drawn interest in the last few years, including applying non-negative matrix factorization or CUR decomposition. The benefits of these methods include efficiency, better representation of the original data, and ease of interpretability of the resulting lower-dimension matrix (Mahoney et al. 2009).

2 Problem Formulation

There are two major parts of the latent semantic analysis algorithm: constructing a document-term matrix and reducing the dimensionality.

The most common – and most simple – way to construct the document-term matrix is as a matrix $M \in \mathbb{R}^{m \times n}$, where m is the number of documents and n is the number of words (sometimes, the top n most frequent words are chosen instead to reduce computation time). Then the entry in position (i, j) represents information about the frequency of word j in document i. Often, this information is represented by TF-IDF (term frequency–inverse document frequency), which counts frequency in the document but gives higher weights to words that are rarer across the corpus (Al-Sabahi et al. 2019). Although other methods can be used for this step, they are passed over here in favor of experimenting with dimensionality reduction methods.

The dimensionality reduction is typically performed through singular value decomposition (SVD). The singular value decomposition of a matrix $A \in \mathbb{R}^{m \times n}$ is written as

$$A = U \Sigma V^T$$

where $\Sigma \mathbb{R}^{m \times n}$ is a diagonal matrix consisting of the square roots of the eigenvalues of the matrix, called the *singular values*, which decrease in magnitude down the diagonal; $U \in \mathbb{R}^{m \times m}$ is an orthogonal matrix where the columns are called the *left singular vectors* of A; and $V \in \mathbb{R}^{n \times n}$ is an orthogonal matrix where the columns are called the *right singular vectors* of A. The first k singular vectors on each side along with the first k singular values can be used to construct a rank k approximation of A.

$$A_k = \sigma_1 u_1 v_1^T + \ldots + \sigma_k u_k v_k^T$$

The singular value decomposition is significant because this is the single best rank k approximation for A (Strang 2019). As a result, one would naively expect this to produce the best results when used for latent semantic analysis.

One area the SVD is weak in is efficiency. The naive approach for computing the SVD requires finding the eigenvalues of A, and the eigenvectors of

AA^T and A^TA, which is computationally expensive. As a result, many algorithms compute a close approximation in a more efficient way; for instance, the TruncatedSVD class used for the SVD in this project uses the RandomizedSVD algorithm, which closely approximates the true SVD in a more efficient way than computing the SVD directly.

Another dimensionality reduction technique is nonnegative matrix factorization (NMF). As the name implies, this carries the requirement that the original matrix A is nonnegative. NMF attempts to factor this matrix into two nonnegative matrices, producing a lower rank product. This is generally written as

$$A \approx UV$$

where $A, U, V \geq 0$. Ideally, U, V would be orthogonal, but this property is sacrificed in favor of non-negativity. Finding U and V is a minimization problem, using the Frobenius norm:

$$\text{argmin}_{U,V} ||A - UV||_F^2$$

There are many algorithms to estimate such U, V, though the problem of finding an optimized solution –or, even, a solution that always converges to the best U, V– is still open; indeed, this is in some sense a special case of the least squares problem. A common approach, and the one used by the scikit-learn implementation used in this project, is to alternate factorization by holding one factor constant, then optimize the other. This is repeated back and forth until the answer converges, usually (though not always) towards the best values of U and V (Strang 2019).

The advantage of nonnegative matrix factorization is interpretability of results. For feature values, such as the term frequencies of a document-term matrix, it often is not meaningful to consider negative values. As a result, NMF is also frequently used for latent semantic analysis and other text mining applications. A disadvantage is that there is no known tractable algorithm for computing the best U and V; NMF is considered an NP-hard problem. Despite this, the algorithms for approximating U and V tend to be relatively fast and yield reasonably good results.

Another dimensionality reduction technique that has drawn interest in recent years is CUR decomposition. The goal of CUR decomposition is to factor a matrix $A \in \mathbb{R}^{m \times n}$ as

$$A = CU^{\dagger}R$$

where C consists of columns of A corresponding to column indices J, R consists of rows of A corresponding to row indices I, and U is $A[I][J]$, a small, square submatrix of A. If $\text{rank}(U) = \text{rank}(A)$, then this is an exact equivalence; if $\text{rank}(U) < \text{rank}(A)$, then $CU^{\dagger}R$ is a rank k approximation of A. Finding the best subset of columns for a rank k approximation requires minimizing

$$||A - CC^{\dagger}A||_2$$

over all subsets of k columns C. Solving this minimization problem is NP-hard, but a key observation is that random sampling of the columns generally yields good results. Weighted random sampling over a probability distribution on the columns of A can yield more accurate results. Both approaches are generally quite fast.

The advantages of CUR decomposition are speed and interpretability. The CUR decomposition suggests which columns of A are most representative of the data, which is relevant to the document-term matrix since each column represents a term in the corpus. The CUR decomposition method applied to this data uses a probability distribution to randomly sample across the columns of the document-term matrix; a naive completely random sampling version was also run for comparison, though this produces a generally worse rank k approximation than using the probability distribution approach.

3 Implementation

The dataset used for this experiment is the 20 Newsgroups dataset, which consists of the raw text of 18,846 newsgroup posts. This set is commonly used as data for topic modeling because each raw text is associated with one of the 20 topic labels, which cover a wide variety of subjects. The task here was to sort the raw texts into 6 general categories, which the 20 canonical labels were divided into as follows:

Table 1: Categorization of Canonical Labels

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

This organization was proposed by Jason Rennie (2008), and it serves to reduce the number of topics while still leveraging the original labeling scheme.

The data was cleaned before analysis, largely following the methodology of Prateek Joshi's analysis of this dataset (2018). In particular, all non-alphabetic characters, words less than 3 characters long, and words on the Natural Language Toolkit's stopword list (representing common, low-information words such as "about" and "been") were removed. In addition to these steps recommended by Joshi's analysis, all words that were not in the English dictionary (as provided by the Natural Language Toolkit) were removed. Finally, all words were

made lowercase.

The document-term matrix was constructed using scikit-learn's TfidfVectorizer using the top 1000 most common words. The dimensionality of this matrix was reduced using scikit-learn's TruncatedSVD, scikit-learn's NMF, cur's cur_decomposition (a probabilistic CUR), and an inefficient but functional personal implementation of a randomized CUR. Each resulting lower-dimensional matrix was then used to transform the data to a matrix $M' \in \mathbb{R}^{18,846\times 6}$.

The results were taken by considering the column with the largest value to be the chosen label for that document. These labels were permuted and compared with the canonical labels, which were constructed according to Table 1. The accuracy was calculated for each permutation, and the permutation with the highest accuracy was chosen as correct. Precision and recall were calculated for each category.

Time was calculated purely as the time taken to construct and apply the dimensionality reduction matrix, as the data cleaning, term matrix construction, and validation steps were shared between all tasks.

4 Results

The accuracy, precision, and recall were considered by posing two questions: whether the correct label is the top label (accuracy@1), and whether the correct label is in the top 2 labels (accuracy@2).

Accuracy is simply calculated as the proportion of labels that were correct. Precision is the proportion of the items labeled with a given label that should have that label, while recall is the proportion of the instances that should have a given label that were labeled with that label. For instance, consider a model tasked with determining if a photo contains a cat. If there was one picture of a cat A and two other pictures B and C and the model classified the pictures A and B as cats, it would have a precision of $\frac{1}{2}$ and a recall of $\frac{1}{1} = 1$. To produce a single value for comparison, a weighted average of the precision and recall across the categories –based on the size of each category– was taken.

An overview of the results is in 2, below. A boxed value indicates that it is the best value for that criterion across algorithms, and a value in italics indicates that it is the worst value across algorithms. The results with a completely random scheme are also shown for comparison. In addition, note that the weighted average of recall across multiple classes is the same as accuracy across multiple classes; if the values here differ slightly, it is only a rounding error.

Table 2: Overview of Results of Dimensionality Reduction Methods

Evaluation Criteria	Algorithm Used				
	SVD	NMF	CUR	CUR(slow)	Random guessing
Accuracy@1	0.2905	0.3336	0.2623	0.2389	*0.1747*
Precision@1	0.4463	0.4104	*0.1568*	0.1723	0.2031
Recall@1	0.2896	0.3031	0.2214	0.2303	*0.1746*
Accuracy@2	0.5856	0.5332	0.4443	0.4590	*0.3690*
Precision@2	0.7614	0.6608	0.6844	*0.4534*	0.5029
Recall@2	0.5856	0.5332	0.4443	0.4590	*0.3698*
Time (s)	1.8180	1.0619	0.2381	*5.1528*	0.2082

The standard approach using the SVD has overall strong performance, although the NMF values are similar and in some cases better. The CUR approach is the fastest apart from random guessing. Across all algorithms except random guessing, the categorization seems to result in a large number of values being grouped into one topic, with few in the others. This means that the precision and recall, when considered for each topic individually, look very different: the precision for a few topics is very high, showing that the few texts sorted into those topics are almost always in that topic; however, the recall for those topics is generally low, meaning that many values in the class are missed. For other topics, the precision is very low, meaning that many of the texts sorted into that topic were sorted there incorrectly; however, the recall is very high, showing that all or almost all of the texts that belong in that group are present. For instance, consider the detailed precision data for the SVD and NMF algorithms in Table 3.

Table 3: Detailed Precision — Recall Values @ 2

Topic Number	Algorithm Used			
	SVD	NMF	CUR	CUR(slow)
Topic 0	1.00 — 0.70	1.00 — 0.75	0.35 — 0.76	1.0 — 0.98
Topic 1	0.97 — 0.43	0.87 — 0.48	1.00 — 0.52	0.30 — 0.42
Topic 2	0.34 — 0.95	0.51 — 0.27	1.00 — 0.34	0.33 — 0.28
Topic 3	0.36 — 0.08	0.33 — 0.65	0.15 — 0.47	0.00 — 0.00
Topic 4	0.63 — 0.17	0.29 — 0.81	1.00 — 0.19	0.27 — 0.15
Topic 5	0.94 — 0.66	0.41 — 0.26	0.19 — 0.10	0.18 — 0.28

This discussion of accuracy, precision, and recall makes a fairly large assumption: that the topics that the model finds correspond exactly to the topics labelling in the original documents. While this is an organization of the documents that would make sense, it is by no means the only organization possible. To validate that the topics seem to be splitting on subject, it is useful to look

at the top several words associated with each category in the model. Since the model constructs topics as probability distributions over the words of the corpus, the top words in each topic provide some insight into the kinds of articles that are categorized there. Consider the top words in each topic for the SVD model, below.

Table 4: Top Words by Topic: SVD Model

Topic 0	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
like	thanks	know	game	like	like
know	drive	thanks	thanks	drive	file
people	card	people	know	know	program
think	mail	advance	team	think	window
time	advance	government	year	hard	look
good	know	mail	like	bike	graphics
thanks	file	information	hockey	disk	image

While there are some commonalities across the topics such as "like," generally the topics are distinct. For instance, Topic 3 has words corresponding to sports, while Topic 5 features more words associated with computers. Similar analysis of the top words in each category for the other algorithms suggests that the dimensionality reduction is picking up topics in the text as features.

5 Conclusions

Perhaps unsurprisingly, the truncated SVD approach produces the most uniformly good results. The truncated SVD approach leads to identifying the correct label in the top 2 58.56% of the time, more than 5 percentage points better than the second best approach. It also has the highest precision for that metric. More interestingly, the NMF approach has a higher accuracy for the single top value.

One possible interpretation that is consistent with the results observed is that the NMF approach identifies a topic for each document, on average, with higher confidence and thus ranks all other options as lower confidence. This would explain why the NMF approach has a higher accuracy when only the top choice is considered but that doesn't scale to higher accuracy when the second choice is also taken into account. To examine this further, an analysis could construct a confusion matrix and also graph the ranked likelihood of the correct label for each approach. For instance, if the correct label had the third highest weight, it would be ranked the third most likely. This could expose differences in the error types between algorithms; however, this approach is limited slightly by the name of latent semantic analysis, because there is no guarantee that the chosen permutation of model topics to match them to label topics is accurate.

Overall, all of the accuracy values were fairly low; no method so much as doubled the accuracy of random guessing. This could suggest that more cleaning of the data is needed, which seems consistent with the presence of common but semantically uninformative words such as "like" in several topics' top words. This could also suggest that 6 topics is not the best number to choose for this dataset; a common analysis when the number of topics is unknown in latent semantic analysis is to graph performance over a range of numbers and choose the one that gives the best performance. However, since these tuning steps would be applied before or (in the case of a different number of topics) identically to the dimensionality reduction step, it is unlikely that they would impact the relative results of the dimensionality reduction methods. Since that is the focus of this work, and there is no intention to use the resulting model for any application, this was not explored.

While the SVD approach may seem to be the obvious best at a first glance, the trade-off for this accuracy is speed. The SVD approach was slower than the NMF and CUR approaches; the NMF approach took 58.4% of the time to produce results with similar accuracy, while the CUR approach took 13.1% of the time but produced much worse results. The best choice of dimensionality reduction, then, depends on the scenario: if getting the most accurate results is the only goal, then the SVD approach is likely the best – though further error analysis should be performed to examine why NMF performs better on accuracy for the top topic.

However, it may often be worth a slight drop in accuracy to cut computation time almost in half, especially in larger corpora – at almost 19,000 documents, Newsgroups is still considered quite small, and the texts are fairly short as well. The CUR decomposition produced significantly less accurate results, and so might not be worthwhile in this application despite its speed; however, this does not mean the approach should be disregarded in general. For instance, Mahoney and Drineas produced results with strong performance when performing CUR decompositions to cluster results into 2 categories (2009).

This work could be expanded by trying different variations on these dimensionality reductions, such as different algorithms for column choice in the CUR decomposition. In addition, different data cleaning and different numbers of topics could be chosen to potentially improve performance, though, as stated above, this is unlikely to change the relative results. Finally, the document-term matrix could be constructed differently, and it's possible that this would impact which dimensionality reduction technique yields the best results.

No method produced results that were indisputably better in all aspects, and so it would be irresponsible to conclude that one method should always be used. Instead, this analysis is an argument for trying multiple dimensionality methods when performing latent semantic analysis. A relatively short analysis of multiple methods could uncover significant differences in accuracy, depending

on the task. If nothing else, this allows for an informed decision on how much accuracy, if any, should be sacrificed for speed. There are tradeoffs with each method of dimensionality reduction, but no non-trivial scheme explored here is entirely without merit when applied for latent semantic analysis.

6 References

1. Al-Sabahi, Kamal, Zuping, Zhang, and Yang Kang. "Latent Semantic Analysis Approach for Document Summarization Based on Word Embeddings." *KSII Transactions on Internet and Information Systems*, 2019. https://arxiv.org/abs/1807.02748.

2. Allahyari, Mehdi, Pouriyeh, Seyedamin, Assefi, Mehdi, Safaei, Saied, Trippe, Elizabeth D., Gutierrez, Juan B., and Krys Kochut. "A Brief Survey of Text Mining: Classification, Clustering, and Extraction Techniques." *Cornell University: Computation and Language*, 2017. https://arxiv.org/abs/1707.02919.

3. Joshi, Prateek. "Text Mining 101: A Stepwise Introduction to Modeling using Latent Semantic Analysis (using Python)." *Analytics Vidhya*, 2018. https://www.analyticsvidhya.com/blog/2018/10/stepwise-guide-topic-modeling-latent-semantic-analysis/.

4. Kontostathis, April, and William M. Pottenger. "A framework for understanding Latent Semantic Indexing (LSI) performance." *Information Processing and Management*, 2006. https://www.sciencedirect.com/science/article/abs/pii/S0306457304001529.

5. Landauer, Thomas K., McNamara, Danielle S., Dennis, Simon, and Walter Kinstch. *Handbook of Latent Semantic Analysis*. Psychology Press, 2007. https://books.google.com/books?id=Jm_NgzzDntYC.

6. Mahoney, Michael W., and Drineas, Petros. "CUR matrix decompositions for improved data analysis." PNAS, 2009. https://www.pnas.org/content/106/3/697.

7. Rennie, Jason. "20 Newsgroups." 2008. http://qwone.com/~jason/20Newsgroups/.

8. Strang, Gilbert. *Linear Algebra and Learning from Data*. Wellesley Cambridge Press, 2019.

7 Appendices

7.1 Appendix 1: Code

Please note that the most up-to-date version of the code is also accessible at `www.github.com/abertsch72/topic-modeling`.

```
"""
author: Amanda Bertsch
"""
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import NMF
from nltk.corpus import stopwords
from nltk.corpus import words
import cur
import numpy as np

import itertools
import time
import re
import random

NUM_TOPICS = 6


"""
Helper function to perform column selection for "slow CUR"
"""
def col_select(num, arr):
    cols_chosen = []
    cols_chosen.append(np.random.randint(0, arr.shape[1]))
    c = arr[:, cols_chosen[0]]
    while(len(cols_chosen) < num):
        next = np.random.randint(0, arr.shape[1])
        if(next not in cols_chosen):
            cols_chosen.append(next)
            col = arr[:, next]
            c = np.column_stack([c, col])
    return c


"""
Performs "slow CUR" and fits the data to reduce dimensionality
"""
def CUR_fit_transform(data, k):
```

```
    c = col_select(k, data)
    r = np.transpose(col_select(k, np.transpose(data)))
    u = r @ np.linalg.pinv(data) @ c
    a = c @ np.linalg.pinv(u) @ r
    return c, u, r


"""
For the canonical labels, sort into categories. Alter the category
descriptions here to change the topic groupings
"""
def find_num_in_each_cat(data):
    computers = [1, 2, 3, 4, 5]
    recreation = [7, 8, 9, 10]
    science = [11, 12, 13, 14]
    forsale = [6]
    politics = [16, 17, 18]
    religion = [0, 15, 19]
    num_each = [0, 0, 0, 0, 0, 0]

    correct = []
    for i in range(len(data)):
        curr = data[i]
        if curr in computers:
            num_each[0] += 1
            correct.append(0)
        elif curr in recreation:
            num_each[1] += 1
            correct.append(1)
        elif curr in science:
            num_each[2] += 1
            correct.append(2)
        elif curr in forsale:
            num_each[3] += 1
            correct.append(3)
        elif curr in politics:
            num_each[4] += 1
            correct.append(4)
        elif curr in religion:
            num_each[5] += 1
            correct.append(5)

    return num_each, correct


"""
```

```
Finds the top n weights for each document, returning a matrix
of the corresponding topic numbers
"""
def find_top_n(result, n):
    top = []

    for i in range(len(result)):
        curr_max = []
        for j in range(len(result[i])):
            if len(curr_max) < n:
                curr_max.append([j, result[i][j]])
            else:
                for k in range(len(curr_max)):
                    if result[i][j] > curr_max[k][1]:
                        curr_max[k] = [j, result[i][j]]
                        break
        top.append([c[0] for c in curr_max])
    return top


"""
Helper function to calulate a weighted average
"""
def weighted_avg(points, weights):
    num = 0
    denom = 0
    for i in range(len(points)):
        num += points[i] * weights[i]
        denom += weights[i]
    return num / denom


"""
Calculates and prints accuracy, precision, and recall information for the
case where the topic label is considered correct if it is the top label
by weight
"""
def validate_firm(results, correct, num_each):
    results = find_top_n(results, 1)
    results = [r[0] for r in results]
    accuracy = []
    precision = [[], [], [], [], [], []]
    recall = [[], [], [], [], [], []]
    permutations = itertools.permutations([0, 1, 2, 3, 4, 5])

    for per in permutations:
```

```
            true_pos = [0, 0, 0, 0, 0, 0]
            guessed = [0, 0, 0, 0, 0, 0]
            for j in range(len(results)):
                if correct[j] == per[results[j]]:
                    true_pos[correct[j]] += 1
                guessed[per[results[j]]] += 1
            accuracy.append(sum(true_pos) / len(results))

            precision.append([])
            recall.append([])

            i = len(precision) - 1
            for k in range(NUM_TOPICS):
                if guessed[k] != 0:
                    precision[i].append(true_pos[k] / guessed[k])
                else:
                    precision[i].append(0)
                recall[i].append(true_pos[k] / num_each[k])

    print("STRICT VALIDATION: RESULTS")
    print("Accuracy: " + str(max(accuracy)))
    m = accuracy.index(max(accuracy))
    print("Average Precision (detailed on next line): " +
          str(weighted_avg(precision[m], num_each)))
    print(precision[m])
    print("Average Recall (detailed on next line): " +
          str(weighted_avg(recall[m], num_each)))
    print(recall[m])


"""
Calculates and prints accuracy, precision, and recall information for the
case where the topic label is considered correct if it is in the top 2
labels by weight
"""
def validate_lax(results, correct, num_each):
    results = find_top_n(results, 2)

    accuracy = []
    precision = []
    recall = []
    permutations = itertools.permutations([0, 1, 2, 3, 4, 5])

    for per in permutations:
        true_pos = [0, 0, 0, 0, 0, 0]
        guessed = [0, 0, 0, 0, 0, 0]
```

```
        for j in range(len(results)):
            if correct[j] == per[results[j][0]] or correct[j] == per[results[j][1]]:
                true_pos[correct[j]] += 1
                guessed[correct[j]] += 1
            else:
                guessed[per[results[j][0]]] += 1
        accuracy.append(sum(true_pos) / len(results))

        precision.append([])
        recall.append([])

        i = len(precision) - 1
        for k in range(NUM_TOPICS):
            if guessed[k] != 0:
                precision[i].append(true_pos[k] / guessed[k])
            else:
                precision[i].append(0)
            recall[i].append(true_pos[k] / num_each[k])
    print("LAX (TOP 2) VALIDATION: RESULTS")
    print("Accuracy: " + str(max(accuracy)))
    m = accuracy.index(max(accuracy))
    print("Average Precision (detailed on next line): " +
          str(weighted_avg(precision[m], num_each)))
    print(precision[m])
    print("Average Recall (detailed on next line): " +
          str(weighted_avg(recall[m], num_each)))
    print(recall[m])


"""
Uses SVD to reduce dimensionality of matrix and prints validation and timing data
"""
def run_validate_SVD(vectorizer, data, correct_labels, num_each):
    time_start = time.time()
    svd_model = TruncatedSVD(n_components=6, algorithm='randomized', n_iter=100,
                             random_state=122)
    results = svd_model.fit_transform(data)
    time_end = time.time()

    print("SVD RESULTS: \n" + "-" * 70)
    validate_lax(results, correct_labels, num_each)
    validate_firm(results, correct_labels, num_each)
    print("*" * 70)
    print("TIME TAKEN: " + str(time_end - time_start) + " milliseconds")
    print("*" * 70)
    print("TOP TERMS:")
```

```
    terms = vectorizer.get_feature_names()
    for i, comp in enumerate(svd_model.components_):
        terms_comp = zip(terms, comp)
        sorted_terms = sorted(terms_comp, key=lambda x: x[1], reverse=True)[:7]
        print("Topic " + str(i) + ": ")
        for t in sorted_terms:
            print(t[0], end=' ')
        print("")
    print("\n\n\n")


"""
Uses NMF to reduce dimensionality of matrix and prints validation and timing data
"""
def run_validate_NMF(vectorizer, data, correct_labels, num_each):
    time_start = time.time()
    nmf_model = NMF(n_components=6, max_iter=100, random_state=122)
    nmf_results = nmf_model.fit_transform(data)
    time_end = time.time()

    print("NMF RESULTS: \n" + "-" * 70)
    validate_lax(nmf_results, correct_labels, num_each)
    validate_firm(nmf_results, correct_labels, num_each)
    print("*" * 70)
    print("TIME TAKEN: " + str(time_end - time_start) + " milliseconds")
    print("*" * 70)
    print("TOP TERMS:")
    terms = vectorizer.get_feature_names()
    for i, comp in enumerate(nmf_model.components_):
        terms_comp = zip(terms, comp)
        sorted_terms = sorted(terms_comp, key=lambda x: x[1], reverse=True)[:7]
        print("Topic " + str(i) + ": ")
        for t in sorted_terms:
            print(t[0], end=' ')
        print("")
    print("\n\n\n")


"""
Uses "slow CUR" to reduce dimensionality of matrix and prints validation and
timing data
"""
def run_validate_slowCUR(vectorizer, data, correct_labels, num_each):
    time_start = time.time()
    c, u, r = CUR_fit_transform(data.toarray(), NUM_TOPICS)
    cur1_results = (data.toarray() @ np.linalg.pinv(r) @ u)
```

```
    time_end = time.time()

    print("CUR RESULTS (SLOW VERSION): \n" + "-" * 70)
    validate_lax(cur1_results, correct_labels, num_each)
    validate_firm(cur1_results, correct_labels, num_each)
    print("*" * 70)
    print("TIME TAKEN: " + str(time_end - time_start) + " milliseconds")
    print("*" * 70)
    print("TOP TERMS:")
    terms = vectorizer.get_feature_names()
    for i, comp in enumerate(u @ np.linalg.pinv(c)):
        terms_comp = zip(terms, comp)
        sorted_terms = sorted(terms_comp, key=lambda x: x[1], reverse=True)[:7]
        print("Topic " + str(i) + ": ")
        for t in sorted_terms:
            print(t[0], end=' ')
        print("")
    print("\n\n\n")


"""
Uses CUR to reduce dimensionality of matrix and prints validation and timing data
"""
def run_validate_CUR(vectorizer, data, correct_labels, num_each):
    time_start = time.time()
    C, U, R = cur.cur_decomposition(data.toarray(), NUM_TOPICS)
    cur2_result = (data.toarray() @ np.linalg.pinv(R) @ U)
    time_end = time.time()

    print("CUR RESULTS (LIBRARY VERSION): \n" + "-" * 70)
    validate_lax(cur2_result, correct_labels, num_each)
    validate_firm(cur2_result, correct_labels, num_each)
    print("*" * 70)
    print("TIME TAKEN: " + str(time_end - time_start) + " milliseconds")
    print("*" * 70)
    print("TOP TERMS:")
    terms = vectorizer.get_feature_names()
    for i, comp in enumerate(U @ np.linalg.pinv(C)):
        terms_comp = zip(terms, comp)
        sorted_terms = sorted(terms_comp, key=lambda x: x[1], reverse=True)[:7]
        print("Topic " + str(i) + ": ")
        for t in sorted_terms:
            print(t[0], end=' ')
        print("")
    print("\n\n\n")
```

```
"""
Chooses random values for each weight and prints validation and timing data
"""
def run_validate_random(vectorizer, data, correct_labels, num_each):
    random.seed(122)
    time_start = time.time()
    rand_result = [[random.random(), random.random(), random.random(), random.random(),
                    random.random(), random.random()] for i in range(data.shape[0])]
    print(rand_result)
    time_end = time.time()

    print("RANDOM GUESSING RESULTS: \n" + "-" * 70)
    validate_lax(rand_result, correct_labels, num_each)
    validate_firm(rand_result, correct_labels, num_each)
    print("*" * 70)
    print("TIME TAKEN: " + str(time_end - time_start) + " milliseconds")
    print("\n\n\n")


"""
Gets and prepares data, then passes it to each method to run and analyze in turn
"""
def entrypoint():
    newsgroups = fetch_20newsgroups(subset='all', shuffle=True, remove=('headers',
                                        'footers', 'quotes'), random_state=1)
    num_each, correct_labels = find_num_in_each_cat(newsgroups.target)

    # clean data
    clean = clean_data(newsgroups.data)
    print(len(clean))

    # construct document-term matrix
    vectorizer = TfidfVectorizer(stop_words='english', max_df=0.5, smooth_idf=True,
                                 max_features=1000)
    X = vectorizer.fit_transform(clean)

    # LSA with each dimesionality reduction strategy
    run_validate_SVD(vectorizer, X, correct_labels, num_each)
    run_validate_NMF(vectorizer, X, correct_labels, num_each)
    run_validate_CUR(vectorizer, X, correct_labels, num_each)
    run_validate_slowCUR(vectorizer, X, correct_labels, num_each)
    run_validate_random(vectorizer, X, correct_labels, num_each)

    # uncomment this to produce a graph of topics, color-coded
    """
    import umap
```

```
    import matplotlib.pyplot as plt

    X_topics = svd_model.fit_transform(X)
    embedding =
    umap.UMAP(n_neighbors=150, min_dist=0.5, random_state=12).fit_transform(X_topics)

    plt.figure(figsize=(7, 5))
    plt.scatter(embedding[:, 0], embedding[:, 1],
                c=newsgroups.target,
                s=10,  # size
                edgecolor='none'
                )
    plt.show()u @ np.linalg.pinv(c)
    """


"""
Clean data by removing non-alphabetic characters, making letters lowercase,
and removing stopwords, words less than 3 characters long, and words not
in the English dictionary
"""
def clean_data(raw):
    stop_words = stopwords.words('english')
    dictionary = set(words.words())
    clean_data = []
    for item in raw:
        item = re.sub("[^a-zA-Z#]", " ", item)
        item = item.lower()
        clean_data.append(
            ' '.join([w for w in item.split() if (len(w) > 3 and
                                w not in stop_words and w in dictionary)]))

    return clean_data


entrypoint()
```

7.2 Requirements (Python Libraries)

```
click==7.1.2
cur==0.0.1
cycler==0.10.0
joblib==0.14.1
kiwisolver==1.2.0
llvmlite==0.32.0
matplotlib==3.2.1
nltk==3.5
numba==0.49.0
numpy==1.18.4
pandas==1.0.3
pyparsing==2.4.7
python-dateutil==2.8.1
pytz==2020.1
regex==2020.4.4
scikit-learn==0.22.2.post1
scipy==1.4.1
six==1.14.0
tbb==2020.0.133
tqdm==4.46.0
umap-learn==0.4.2
```