PES University, Bangalore

(Established under Karnataka Act No. 16 of 2013)

MAY 2020: IN SEMESTER ASSESSMENT (ISA) B.TECH. IV SEMESTER

UE18MA251- LINEAR ALGEBRA

MINI PROJECT REPORT

ON

LATENT SEMANTIC ANALYSIS

Submitted by

1. Name:	Sooraj J S	SRN:	PES1201801892
2. Name:	Krishna P Hegde	SRN:	PES1201801896
3. Name:	Prathvik Nayak	SRN:	PES1201802006

Branch & Section : CSE - E section

PROJECT EVALUATION

(For Official Use Only)

Sl.No.	Parameter	Max Marks	Marks Awarded
1	Background & Framing of the problem	4	
2	Approach and Solution	4	
3	References	4	
4	Clarity of the concepts & Creativity	4	
5	Choice of examples and understanding of the topic	4	
6	Presentation of the work	5	
	Total	25	

Name of the Course Instructor : Mr. Ramesh Bhat H

Signature of the Course Instructor :

TABLE OF CONTENTS

1.INTRODUCTION:

Have you ever worked on a dataset with more than a thousand features? How about over 50,000 features? Having a high number of variables is both a boon and a curse. It's great that we have loads of data for analysis, but it is challenging due to size.

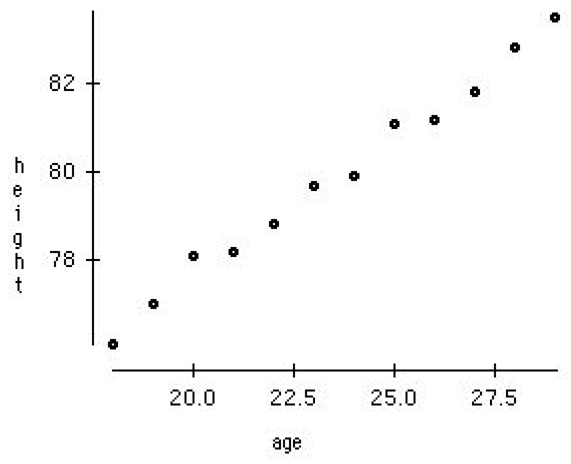
It's not feasible to analyze each and every variable at a microscopic level. It might take us days or months to perform any meaningful analysis and we'll lose a ton of time and money for our business! Not to mention the amount of computational power this will take. We need a better way to deal with high dimensional data so that we can quickly extract patterns and insights from it. So how do we approach such a dataset?

Using dimensionality reduction techniques, of course. You can use this concept to reduce the number of features in your dataset without having to lose much information and keep (or improve) the model's performance. It's a really powerful way to deal with huge datasets, as you'll see in this article.

We are generating a tremendous amount of data daily. In fact, 90% of the data in the world has been generated in the last 3-4 years! The numbers are truly mind boggling. Below are just some of the examples of the kind of data being collected:

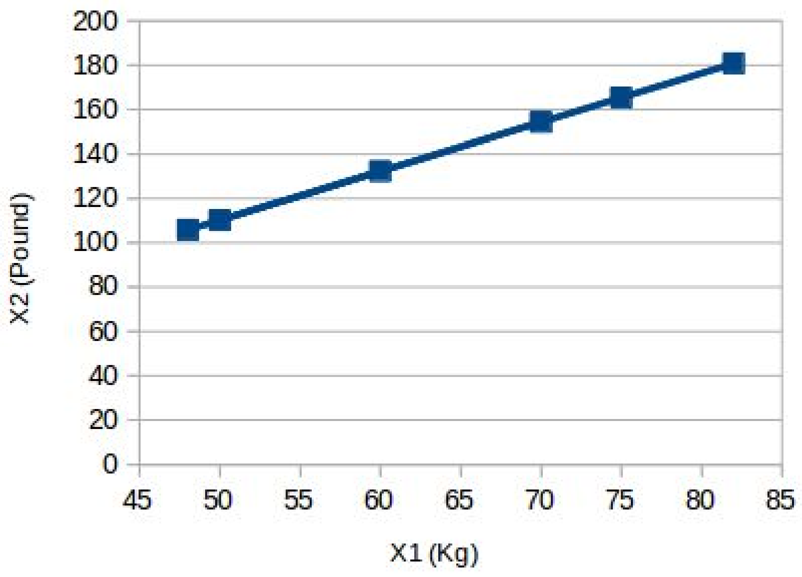
- Facebook collects data of what you like, share, post, places you visit, restaurants you like, etc.
- Your smart phone apps collect a lot of personal information about you.
- Amazon collects data of what you buy, view, click, etc. on their site.
- Casinos keep a track of every move each customer makes.

As data generation and collection keeps increasing, visualizing it and drawing inferences becomes more and more challenging. One of the most common ways of doing visualization is through charts. Suppose we have 2 variables, Age and Height. We can use a scatter or line plot between Age and Height and visualize their relationship easily:



Now consider a case in which we have, say 100 variables (p=100). In this case, we can have 100(100-1)/2 = 5000 different plots. It does not make much sense to visualize each of them separately, right? In such cases where we have a large number of variables, it is better to select a subset of these variables (p<<100) which captures as much information as the original set of variables.

Let us understand this with a simple example. Consider the below image:



Here we have weights of similar objects in Kg (X1) and Pound (X2). If we use both of these variables, they will convey similar information. So, it would make sense to use only one variable. We can convert the data from 2D (X1 and X2) to 1D (Y1) as shown below:

Similarly, we can reduce p dimensions of the data into a subset of k dimensions (k<<n). This is called dimensionality reduction.

Here are some of the benefits of applying dimensionality reduction to a dataset:

- Space required to store the data is reduced as the number of dimensions comes down.
- Less dimensions lead to less computation/training time/
- Some algorithms do not perform well when we have large dimensions. So reducing these dimensions needs to happen for the algorithm to be useful.
- It takes care of multicollinearity by removing redundant features. For example, you have two variables – 'time spent on the treadmill in minutes' and 'calories burnt'. These variables are highly correlated as the more time you spend running on a treadmill, the more calories you will burn. Hence, there is no point in storing both as just one of them does what you require.
- It helps in visualizing data. As discussed earlier, it is very difficult to visualize data in higher dimensions so reducing our space to 2D or 3D may allow us to plot and observe patterns more clearly.

One such dimensionality reduction technique is SVD (Single value decomposition). SVD decomposes the original variables into three constituent matrices. It is essentially used to remove redundant features from the dataset. It uses the concept of Eigenvalues and Eigenvectors to determine those three matrices.

Have you ever been inside a well-maintained library? We have always been incredibly impressed with the way the librarians keep everything organized, by name, content, and other topics. But if you gave these librarians thousands of books and asked them to arrange each book on the basis of their genre, they will struggle to accomplish this task in a day. However, this won't happen if these books came in a digital format. All the arrangement seems to happen in a matter of seconds, without requiring any manual effort, thanks to Natural Language Processing (NLP).

All languages have their own intricacies and nuances which are quite difficult for a machine to capture (sometimes they're even misunderstood by us humans!). This can include different words that mean the same thing, and also the words which have the same spelling but different meanings. For example, consider the following two sentences:

1. I liked his last **novel** quite a lot.
2. We would like to go for a **novel** marketing campaign.

In the first sentence, the word 'novel' refers to a book, and in the second sentence it means new or fresh. We can easily distinguish between these words because we are able to understand the context behind these words. However, a machine would not be able to capture this concept as it cannot understand the context in which the words have been used. This is where Latent Semantic Analysis (LSA) comes into play as it attempts to leverage the context around the words to capture the hidden concepts, also known as topics.

So, simply mapping words to documents won't really help. What we really need is to figure out the hidden concepts or topics behind the words. LSA is one such technique that can find these hidden topics. Let's now deep dive into the inner workings of LSA.

2.REVIEW OF LITERATURE:

2.1 An Article by Peter W. Foltz on LatentSemanticAnalysisForText

Latent semantic analysis (LSA) is a statistical model of word usage that permits comparisons of semantic similarity between pieces of textual information.This Paper Summarizes three experiments that illustrate how LSA may be used in text-based research. Two experiments describe methods for analyzing a subject's essay for determining from what text a subject learned the information and for grading the quality of information cited in the essay. The third experiment describes using LSA to measure the coherence and comprehensibility of texts.

2.2 A paper released by Josef Steinberger and Karel Ježek on Using Latent Semantic Analysis in Text Summarization and Summary Evaluation

This paper deals with using latent semantic analysis in text summarization. They describe a generic text summarization method which uses the latent semantic analysis technique to identify semantically important sentences. Then they propose two new evaluation methods based on LSA, which measure content similarity between an original document and its summary. In the evaluation part they compare seven summarizers by a classical content-based evaluator and by the two new LSA evaluators. Influence of summary length on its quality from the angle of the three mentioned evaluation methods is also noted.

3.REPORT ON PRESENT INVESTIGATION:

3.1 Latent Semantic Analysis (LSA)

Latent semantic analysis (LSA) is a technique in natural language processing, in particular distributional semantics, of analyzing relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms. LSA assumes that words that are close in meaning will occur in similar pieces of text (the distributional hypothesis). A matrix containing word counts per document (rows represent unique words and columns represent each document) is constructed from a large piece of text and a mathematical technique called singular value decomposition (SVD) is used to reduce the number of rows while preserving the similarity structure among columns.

3.2 Singular Value Decomposition (SVD)

In linear algebra, the singular value decomposition (SVD) is a factorization of a real or complex matrix that generalizes the eigendecomposition of a square normal matrix to any $m \times n$ matrix via an extension of the polar decomposition.

Specifically, the singular value decomposition of an $m \times n$ rectangular diagonal matrix with mon-negative real numbers on the diagonal, and V is an $n \times m$ real or complex unitary matrix. If M is real, U and $V^T = V^*$ are real orthonormal matrices.

The diagonal entries $\sigma_i = \Sigma_{ii}$ of Σ are known as the singular values of M. The number of non-zero singular values is equal to the rank of M. The columns of U and the columns of V are called the left-singular vectors and right-singular vectors of M, respectively.

The SVD is not unique. It is always possible to choose the decomposition so that the singular values Σ_{ii} are in descending order. In this case, Σ (but not always U and V) is uniquely determined by M.

The term sometimes refers to the compact SVD, a similar decomposition $M = U\Sigma V^*$ in which Σ is square diagonal of size $r \times r$, where $r \leq min\{m, n\}$ is the rank of M, and has only the non-zero singular values. In this variant, U is an $m \times r$ matrix and V is an $n \times r$ matrix, such that $U^*U = V^*V = I_{r \times r}$.

Mathematical applications of the SVD include computing the pseudo inverse, matrix approximation, and determining the rank, range, and null space of a matrix. The SVD is also extremely useful in all areas of science, engineering, and statistics, such as signal processing, least square fitting of data, and process control. In LSA, the SVD is used as a rank reduction algorithm, which decomposes the term- document matrix into a term-concept matrix U,a singular value matrix S and a concept-document matrix V.

Throughout the implementation, we use a variation of SVD called the TruncatedSVD which calculates the SVD and reduces the number of columns to the number mentioned by us. For example, if the matrix is a $n \times n$ matrix, the SVD results in a n column matrix. The truncated SVD returns a matrix with m columns where m is mentioned by the user.

3.3 DataSet and Text Preprocessing

As we often deal with text data for LSA, before we dive into the applications of LSA it would be helpful to understand the dataset, text preprocessing and text representation for better understanding of the implementation.

3.3.1 Dataset

For this project, we are using the 20 newsgroups dataset present in sklearn.datasets. This dataset is a collection of 11314 articles belonging to 20 different newsgroups. These 20 newsgroups may or may not belong to another general category. The headers, footers and quotes were removed from the dataset before loading it to memory. Each newsgroup has been assigned a number from 0 to 19 and stored separately in another list.

3.3.2 Text Preprocessing

The text in the dataset consists of upper, lower case alphabets, digits, special characters and words that might not contribute to grasping a brief understanding of the document. We first start by removing all characters that are not alphabets or spaces. Then we convert all letters in the text to lowercase. This makes our data more consistent. Next we remove words like ‘and’, ‘or’, ‘but’, ‘not’, ‘if’ from the text. These words are called stopwords. Stopwords are words that do not contribute to the meaning of the document and so removing them would make our code more efficient as it saves memory and time of execution. To remove stop words, we start by tokenizing the text (splitting the text), identify stop words, remove them from the list and detokenize them to gain the new string. A list of stop words is readily available to us in python’s nltk package.

3.3.3 Document Term Matrix

As stated earlier, it is inconvenient and not very helpful to represent the text data as strings for our applications. We therefore introduce the Document-Term Matrix. This matrix is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. The rows signifies the documents in the collection and the columns correspond to. While it is logical to have all the words in the document as a column in the matrix, due to the limitations of storage and computational power, it is better to choose N number of words based on the words importance/frequency. We use the TF-IDFVectorizer to construct the DTM.

3.3.4 TF-IDFVectorizer

The TF-IDFVectorizer uses the term frequency and inverse document frequency to assign weights to words in the corpus that would help in mathematical and statistical calculations. The tf-Idf weights is the product of Tf and Idf. Term frequency helps to gauge the importance of a word in a given document. The less the frequency, the more important the word is. Words such as ‘the’ are frequent but don’t necessarily help in defining the document. But this is not always true. Words can be unique to that document and even though its frequency is high, it might be essential in defining the document. To account for this, we introduce the Inverse Document Frequency. This term, assigns lesser score to words that appear in high frequency in all the documents present in the document. In this way, if a word is unique to a particular document and has high frequency, it would still end up with a higher score thereby solving the problem.

TFIDF

For a term i in document j:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

3.4 Applications of LSA

3.4.1 Text Classification

Text Classification deals with classifying a given piece of text or document based on the contents of the text. This can be achieved using SVD as a rank reduction algorithm. In this case SVD acts as a dimensionality reduction algorithm that helps in better computational speed and results. We start by Text preprocessing as discussed in 3.3.a and 3.3.b. Then we split the dataset to 80:20 to train and test data. We normalize the tf-idf vectors using L1 normalization. Then we train ML models like SVM, in our case, using the training data. We use cross validation to tune the hyper parameters so as to avoid overfitting. This is followed by testing the built model to check accuracy of our model. The reason to use LSA is that it takes the data and forms better and lesser features from them. The new features that we get are a linear combination of the original features, which helps reduce and avoid highly correlated features in the dataset. Therefore, we can see how SVD naturally fits in and gets the job done.

For Classification, we use rbf-SVM. Support vector machines is a supervised learning algorithm that classifies data by identifying support vectors. The decision boundaries are identified by maximizing the margin distance i.e the distance between the decision boundary and support vectors. The Radial basis function is a kernel trick used to map the data to infinite dimensions to make the data linearly separable.

$$K(x_i, x_j) = e^{-\gamma(x_i - x_j)^2}$$

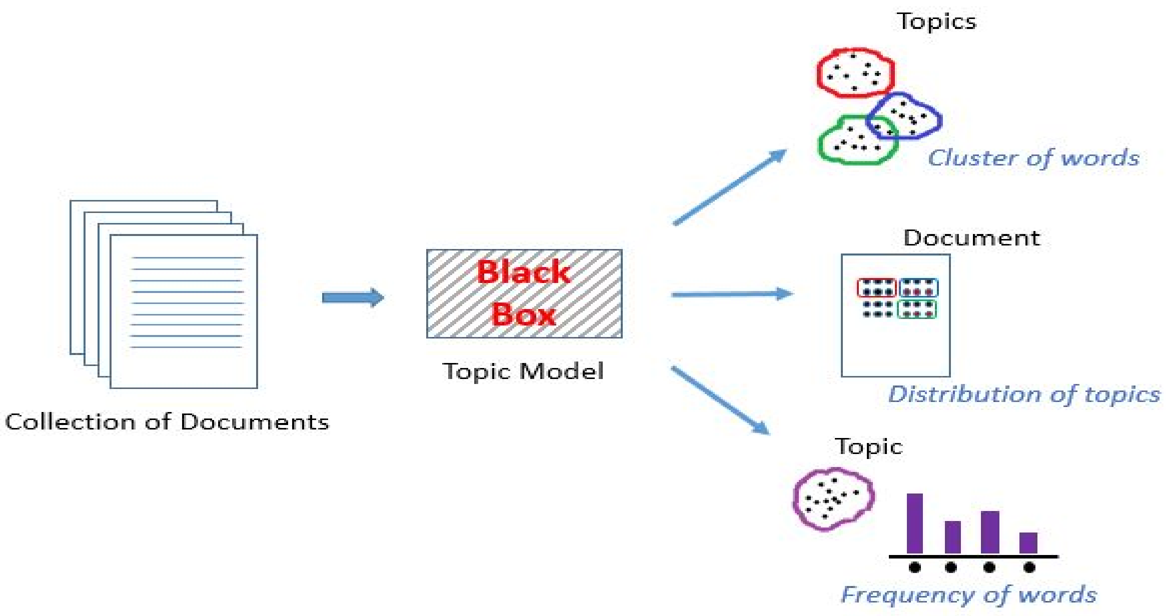
Radial Basis Function

In our project, we try to classify science newsgroups to its sub groups i.e cryptography, electronics, medicine and space.

3.4.2 Topic Modeling

In this project we will look into a text mining approach called Topic Modeling. It is an extremely useful technique for extracting topics, and one you will work with a lot when faced with NLP challenges.

A Topic Model can be defined as an unsupervised technique to discover topics across various text documents. These topics are abstract in nature, i.e.words which are related to each other form a topic. Similarly, there can be multiple topics in an individual document. For the time being, let’s understand a topic model as a black box, as illustrated in the below figure:



This black box (topic model) forms clusters of similar and related words which are called topics. These topics have a certain distribution in a document, and every topic is defined by the proportion of different words it contains.

The use of LSA here is as an unsupervised learning algorithm that helps to find topics/clusters in the text data.The principal components are formed using svd,where each new feature is a linear combination of different features. As the principal components are independent, a new component is formed when that feature varies significantly from others. Therefore each principal component can be treated as a topic of the corpus.So by defining the number of components needed, we define the number of topics we are going to find.

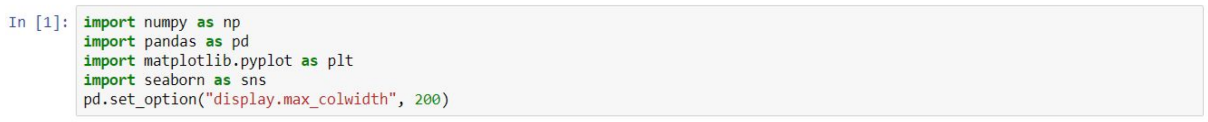
Topic modeling helps in exploring large amounts of text data, finding clusters of words, similarity between documents, and discovering abstract topics. As if these reasons weren't compelling enough, topic modeling is also used in search engines wherein the search string is matched with the results.

4.IMPLEMENTATION IN PYTHON:

4.1 Topic Modeling

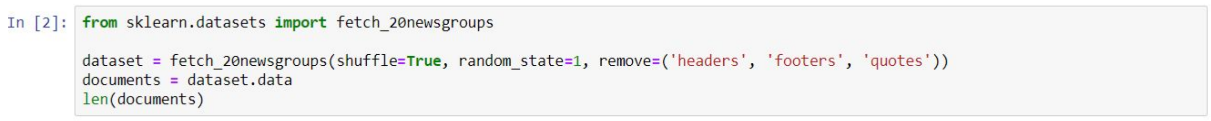
At first we used LSA in a Topic Modeling problem. The steps have been mentioned below:

Lets load the required libraries first



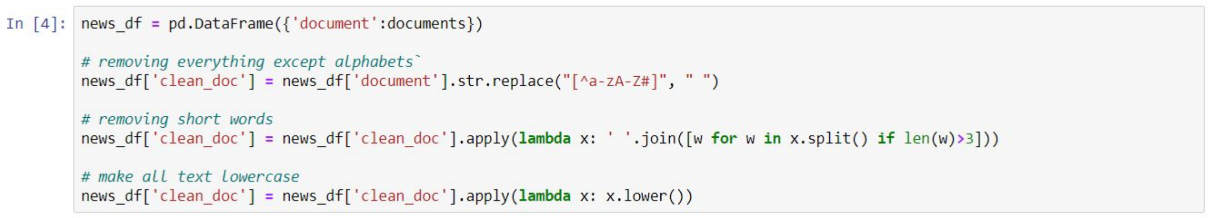
```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        pd.set_option("display.max_colwidth", 200)
```

We have used the 20 Newsgroup dataset from sklearn



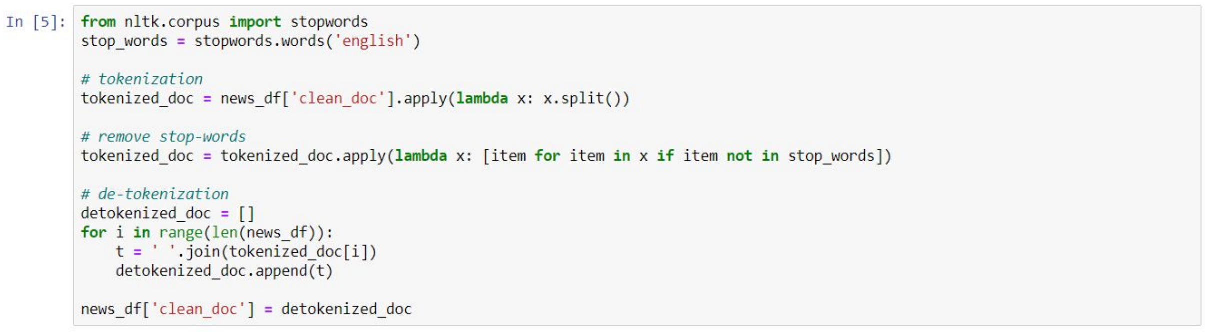
```
In [2]: from sklearn.datasets import fetch_20newsgroups

        dataset = fetch_20newsgroups(shuffle=True, random_state=1, remove=('headers', 'footers', 'quotes'))
        documents = dataset.data
        len(documents)
```

To start with, we will try to clean our text data as much as possible. The idea is to remove the punctuations, numbers, and special characters all in one step.Then we will remove shorter words because they usually don’t contain useful information. Finally, we will make all the text lowercase to nullify case sensitivity.



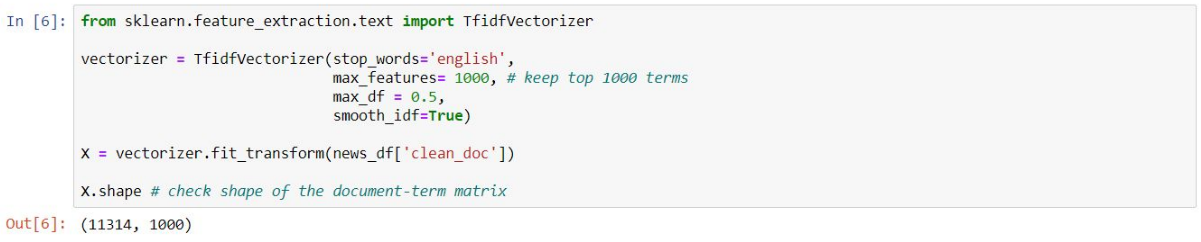
```
In [4]: news_df = pd.DataFrame({'document':documents})

        # removing everything except alphabets`
        news_df['clean_doc'] = news_df['document'].str.replace("[^a-zA-Z#]", " ")

        # removing short words
        news_df['clean_doc'] = news_df['clean_doc'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>3]))

        # make all text lowercase
        news_df['clean_doc'] = news_df['clean_doc'].apply(lambda x: x.lower())
```

Now let's remove the stop-words from the text data as they are mostly clutter and hardly carry any information. Stop-words are terms like ‘it’, ‘they’, ‘am’, ‘been’, ‘about’, ‘because’, ‘while’, etc. To remove stop-words from the documents, we will have to tokenize the text, i.e., split the string of text into individual tokens or words. We will stitch the tokens back together once we have removed the stop-words.



```
In [5]: from nltk.corpus import stopwords
        stop_words = stopwords.words('english')

        # tokenization
        tokenized_doc = news_df['clean_doc'].apply(lambda x: x.split())

        # remove stop-words
        tokenized_doc = tokenized_doc.apply(lambda x: [item for item in x if item not in stop_words])

        # de-tokenization
        detokenized_doc = []
        for i in range(len(news_df)):
            t = ' '.join(tokenized_doc[i])
            detokenized_doc.append(t)

        news_df['clean_doc'] = detokenized_doc
```

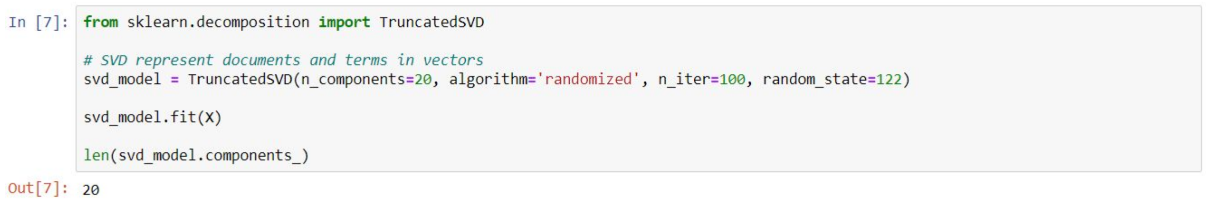
We will use sklearn's tf-idfVectorizer to create a document-term matrix with 1000 terms



```
In [6]: from sklearn.feature_extraction.text import TfidfVectorizer

        vectorizer = TfidfVectorizer(stop_words='english',
                                     max_features= 1000, # keep top 1000 terms
                                     max_df = 0.5,
                                     smooth_idf=True)

        X = vectorizer.fit_transform(news_df['clean_doc'])

        X.shape # check shape of the document-term matrix
Out[6]: (11314, 1000)
```

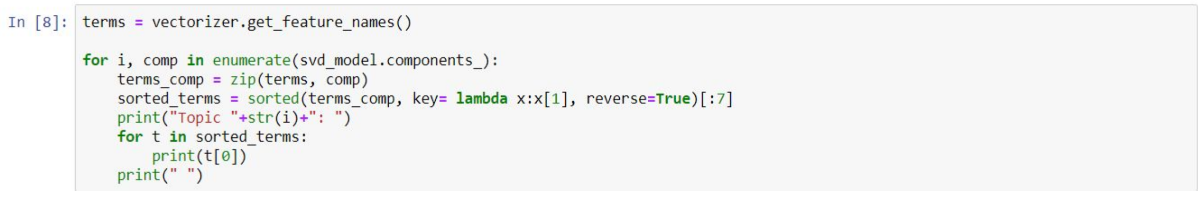
The next step is to represent each and every term and document as a vector. We will use the document-term matrix and decompose it into multiple matrices. We will use sklearn’s TruncatedSVD to perform the task of matrix decomposition.

Since the data comes from 20 different newsgroups, let's try to have 20 topics for our text data. The number of topics can be specified by using the *n_components* parameter.

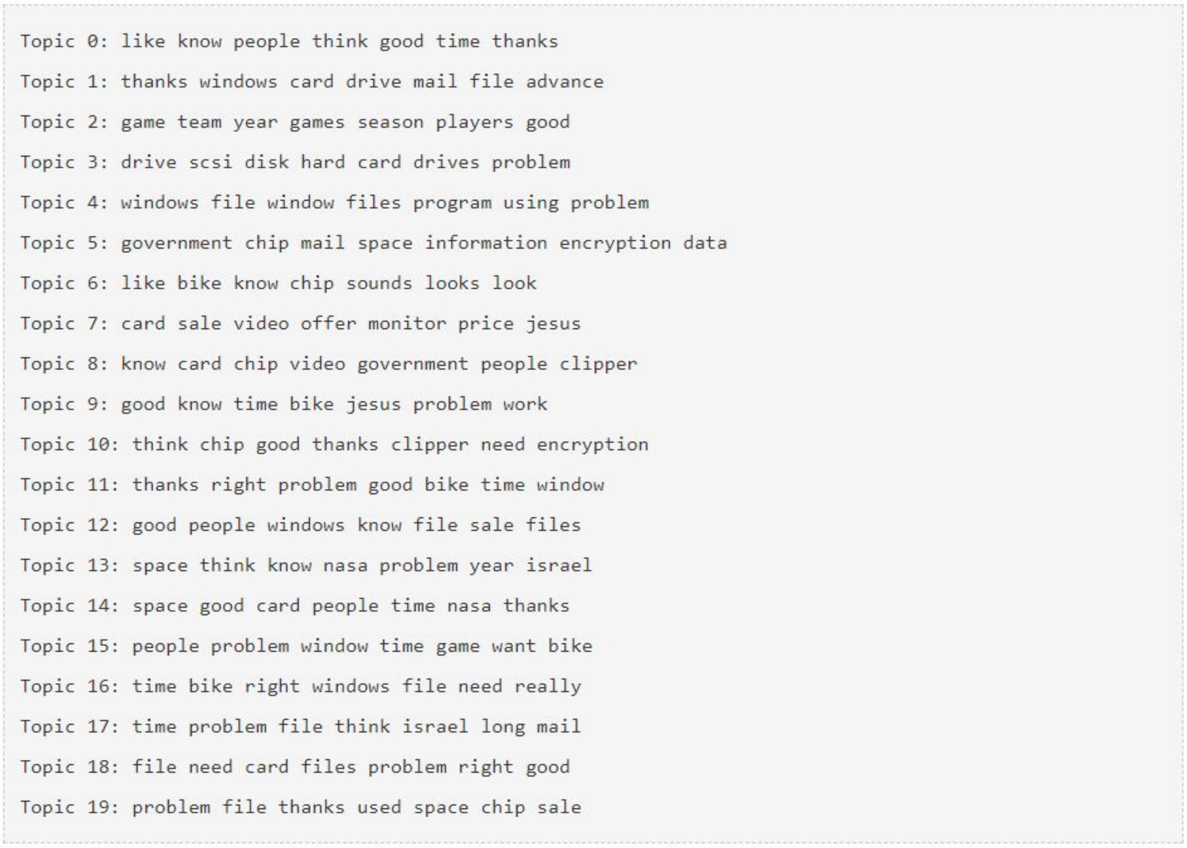


```
In [7]: from sklearn.decomposition import TruncatedSVD

        # SVD represent documents and terms in vectors
        svd_model = TruncatedSVD(n_components=20, algorithm='randomized', n_iter=100, random_state=122)

        svd_model.fit(X)

        len(svd_model.components_)

Out[7]: 20
```

The components of *svd_model* are our topics, and we can access them using *svd_model.components_*. Finally, let's print a few most important words in each of the 20 topics and see how our model has done.



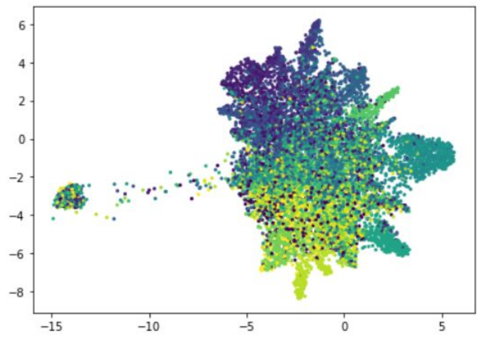
```
In [8]: terms = vectorizer.get_feature_names()

        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])
            print(" ")
```



```
Topic 0: like know people think good time thanks
Topic 1: thanks windows card drive mail file advance
Topic 2: game team year games season players good
Topic 3: drive scsi disk hard card drives problem
Topic 4: windows file window files program using problem
Topic 5: government chip mail space information encryption data
Topic 6: like bike know chip sounds looks look
Topic 7: card sale video offer monitor price jesus
Topic 8: know card chip video government people clipper
Topic 9: good know time bike jesus problem work
Topic 10: think chip good thanks clipper need encryption
Topic 11: thanks right problem good bike time window
Topic 12: good people windows know file sale files
Topic 13: space think know nasa problem year israel
Topic 14: space good card people time nasa thanks
Topic 15: people problem window time game want bike
Topic 16: time bike right windows file need really
Topic 17: time problem file think israel long mail
Topic 18: file need card files problem right good
Topic 19: problem file thanks used space chip sale
```

To find out how distinct our topics are, we should visualize them. Of course, we cannot visualize more than 3 dimensions, but there are techniques like PCA and t-SNE which can help us visualize high dimensional data into lower dimensions. Here we will use a relatively new technique called UMAP (Uniform Manifold Approximation and Projection).

```
In [9]: import umap

        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 = dataset.target,
                    s = 10, # size
                    edgecolor='none'
                    )
        plt.show()
```



4.2 Text Classification

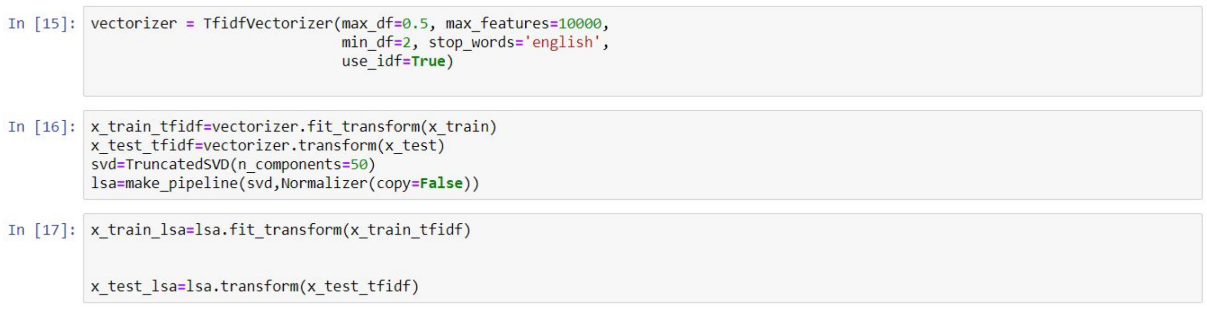
We start by loading and preprocessing the dataset as shown for topic modelling. Please refer to section 4.1 for the implementation. As we are focusing on only the 4 science groups i.e cryptography, electronics, medicine and space, we try to encode each group with numeric digits. This is done as shown in the code below.

```
In [7]: new_y=[]
        new_doc=[]
        for i in range(len(y)):
            if y[i] in [11,12,13,14]:
                new_y.append(y[i]-10)
            else:
                continue
            new_doc.append(doc[i])
        y=new_y
        doc=new_doc
```

We then continue by splitting the data into train and testing data. This is followed by applying the tf-Idf vectorization on only the training set and using this the train and test sets are transformed.

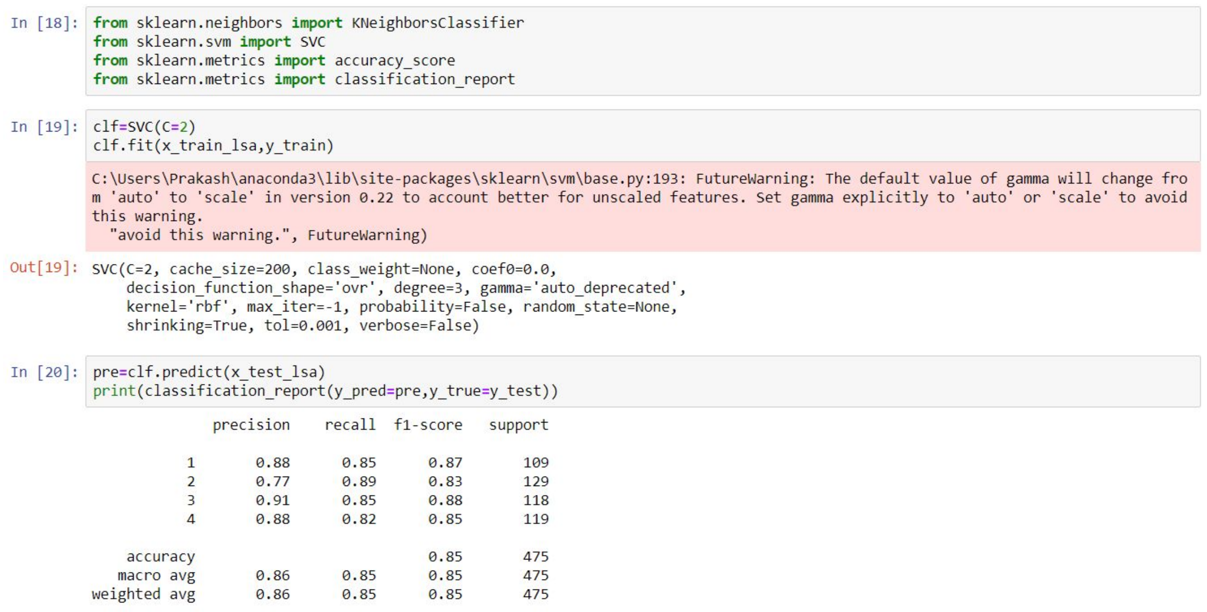
```
In [12]: from sklearn.model_selection import train_test_split

In [13]: x_train,x_test,y_train,y_test=train_test_split(df['clean'],y,test_size=0.2)

In [14]: from sklearn.feature_extraction.text import TfidfVectorizer
         from sklearn.decomposition import TruncatedSVD
         from sklearn.pipeline import make_pipeline
         from sklearn.preprocessing import Normalizer
```

We then apply a truncated SVD algorithm along with normalizing the data for better classification results.



```
In [15]: vectorizer = TfidfVectorizer(max_df=0.5, max_features=10000,
                                      min_df=2, stop_words='english',
                                      use_idf=True)

In [16]: x_train_tfidf=vectorizer.fit_transform(x_train)
         x_test_tfidf=vectorizer.transform(x_test)
         svd=TruncatedSVD(n_components=50)
         lsa=make_pipeline(svd,Normalizer(copy=False))

In [17]: x_train_lsa=lsa.fit_transform(x_train_tfidf)


         x_test_lsa=lsa.transform(x_test_tfidf)
```

This is followed by training the SVC model with the transformed training data. We can see the SVM parameters that are set in the code below. These parameters were selected using the GridSearch Algorithm and the SVM model was built based on these results.



```
In [18]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.svm import SVC
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import classification_report

In [19]: clf=SVC(C=2)
         clf.fit(x_train_lsa,y_train)

         C:\Users\Prakash\anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: The default value of gamma will change fro
         m 'auto' to 'scale' in version 0.22 to account better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid
         this warning.
           "avoid this warning.", FutureWarning)

Out[19]: SVC(C=2, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
             kernel='rbf', max_iter=-1, probability=False, random_state=None,
             shrinking=True, tol=0.001, verbose=False)

In [20]: pre=clf.predict(x_test_lsa)
         print(classification_report(y_pred=pre,y_true=y_test))

                       precision    recall  f1-score   support

                    1       0.88      0.85      0.87       109
                    2       0.77      0.89      0.83       129
                    3       0.91      0.85      0.88       118
                    4       0.88      0.82      0.85       119

             accuracy                           0.85       475
            macro avg       0.86      0.85      0.85       475
         weighted avg       0.86      0.85      0.85       475
```

<u>5.RESULTS AND DISCUSSIONS:</u>

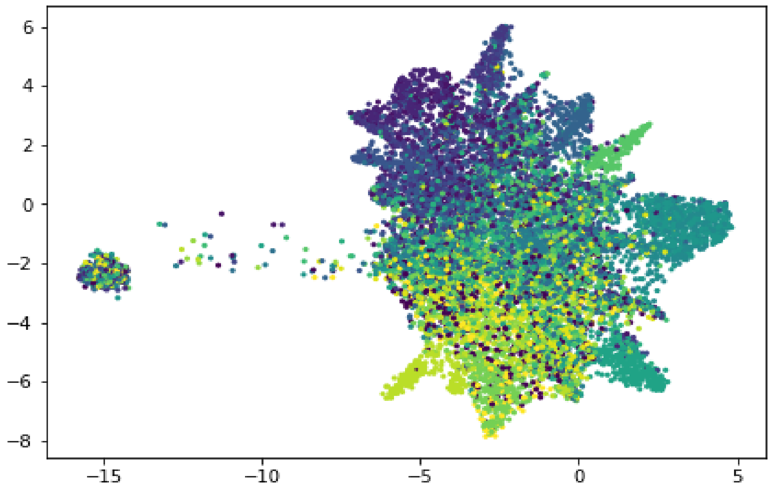
5.1 Classification Results

We used the science section of 20 newsgroups dataset to classify the dataset into the 4 science categories. We only considered 10000 words from the corpus to build the DTM and used the tf-idf score to select these words. We then used rbf-SVM to classify the text and achieved an accuracy of 0.86.

5.2 Topic Modelling Results

We tried to find 20 topics in the dataset using this approach.We represent each topic by the 7 most occurring words in them. This can be seen in the implementation part mentioned in 4.2

As we can see, from the graph above, we were able to classify the main newsgroups in the



dataset, but further subgroups become less identifiable and the subgroups are not captured well by the topics. This tells us that the subgroups share similar information and methods like lda and nfm have to be used to provide better results.

6.SUMMARY AND CONCLUSIONS:

To summarize, in this project we have looked into the concept of Latent Semantic Analysis and its applications namely Topic Modeling and Text Classification. We classify a dataset of 20 newsgroups into 4 science categories using ML models like rbf-SVM with accuracy of 0.86. We also use LSA in topic modelling, the dataset is processed as mentioned above and categorised into 20 topics.

Latent Semantic Analysis can be very useful as we saw above, but it does have its limitations. It’s important to understand both the sides of LSA so you have an idea of when to leverage it and when to try something else.

Pros:

1. LSA is fast and easy to implement.

2. It gives decent results, much better than a plain vector space model.

Cons:

1. Since it is a linear model, it might not do well on datasets with non-linear dependencies.
2. LSA assumes a Gaussian distribution of the terms in the documents, which may not be true for all problems.
3. LSA involves SVD, which is computationally intensive and hard to update as new data comes up.

7.BIBLIOGRAPHY:

7.1 Peter W. Foltz: *Latent semantic analysis for text-based research.* Chairedby Matthew S. McGlone, Lafayette College. New Mexico State University, LasCruces, New Mexico.

7.2 Josef Steinberger and Karel Ježek, *Using Latent Semantic Analysis in Text Summarization and Summary Evaluation* Department of Computer Science and Engineering, Univerzitní 22, CZ-306 14 Plzeň.

7.3 Linear Algebra And Its Applications By Gilbert Strang 4th Edition

7.4 Linear Algebra and Its Applications By David C. Lay, University of Maryland

7.5 Higher Engineering Mathematics, Sixth edition, John Bird

7.6 Higher Engineering Mathematics, 42nd edition, Dr. B.S. Grewal