MA 544 Numerical Linear Algebra of Big Data

Classification of Movie Reviews with Latent Semantic Analysis

Kristin Kim

Taekkeun Nam

Houming Kuang

"I pledge my honor that I have abided by the Stevens Honor System"

Kristin Kim, Taekkeun Nam, Houming Kuang

A. Motivation

Natural Language Processing (NLP) is one of the popular and fast-growing machine learning and deep learning domains. On the subject of the project, our team agreed to take advantage of the knowledge that we've learned in this course, to use NLP techniques and to do something interesting, which is related to our daily life. From many text data, such as Twitter messages, comments on Amazon, or news headlines, we decided to work on movie reviews as watching movies was a common interest all of our team members shaed. The reason that our team specifically selected Latent Semantic Analysis (LSA) for the NLP method to analyze movie reviews is that the technique heavily relies on Single Value Decomposition (SVD), one of the important topics that this course covered.

B. Introduction

Natural Language Processing (NLP) is a broad set of techniques to extract meaningful insights by deriving hidden meanings or patterns from words in a dataset. In this paper, we will cover one of the widely known NLP techniques, LSA technique. The method is a part of unsupervised learning techniques in which models are not supervised using the training datasets, but instead, find the hidden patterns and insights from the given data. The main benefit of this technique is that the process reduces the dimensionality of the original text-based datasets. The core methodology of LSA involves reducing dimensions with SVD, specifically, the truncated SVD, for the method of finding topics. We will also implement another famous classifying machine learning algorithm, Support Vector Machine (SVM) to see which machine learning techniques show better performance.

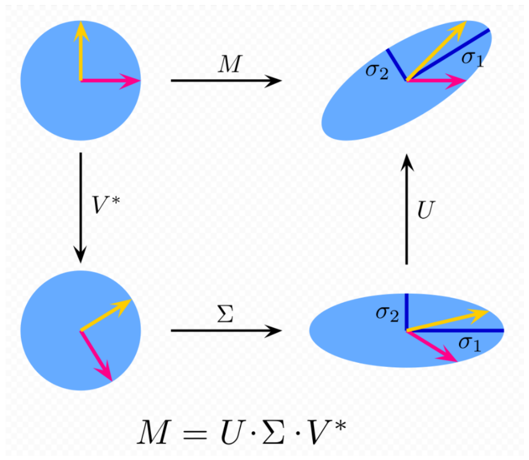
C. Theoretical Background

1. Single Value Decomposition (SVD) and Truncated SVD

a. Mathematics of SVD

$M = U\sum V^*$, U and V are orthogonal matrices, the geometric meaning is rotation, Rotate,stretch,then

rotate again.



(Figure C.1.1)

b. Mathematics of Truncated SVD

Truncated is to remove some singular values in order to simplify the matrix. There is a general method to take the truncated point. But for convenience,our processing in the project is to keep only two singular values which are topic 1 and topic 2.As for topic3, topic4, topic5... truncate directly.

c. Truncated SVD Utilizations in Python

TruncatedSVD from sklearn package is utilized for the implementation of dimensionality reduction for the data set. This specific class takes a parameter, “n_components” which is the desired dimensionality of output data to instantiate a class, and this must be less than the number of features of the data set. We set this as a 2, and this will be covered in the Implementation section with more details. Another parameter to mention is its SVD solver algorithm. By default, it will use a “randomized” SVD solver, however, ARPACK, a numerical software library for solving large-scale eigenvalue problems could be selected for custom use.

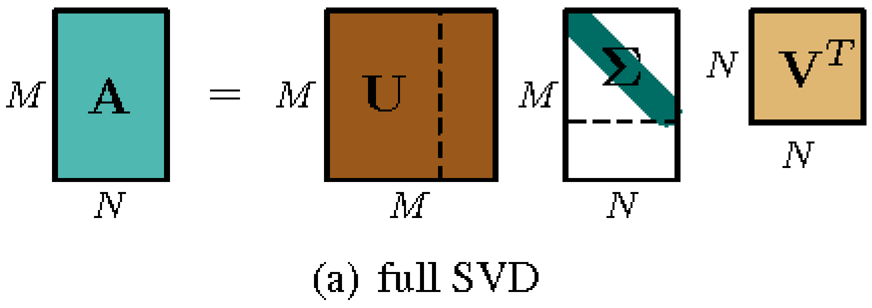
d. Different types of SVD

Suppose the shape of A is $m \times n$,written as $\underset{m\times n}{A}$,with rank r.

In full SVD:

- U is composed of r orthonormal columns that span the column space of A and $m - r$ orthonormal columns that span the left null space of A.
- $\sum$ is diagonal and composed of the square root of eigenvalues of $A^T A$(or AA^T) padded with zero rows and columns to be of shape $m \times n$. The diagonal elements are also called the singular value of A.
- V is composed of r orthonormal columns that span the row space of A and n-r orthonormal columns that span the null space of A.

$$\underset{m\times n}{A} = \underset{m\times m,}{U}\underset{m\times n,}{\Sigma}\underset{n\times n}{V^T}$$



(Figure C.1.2)

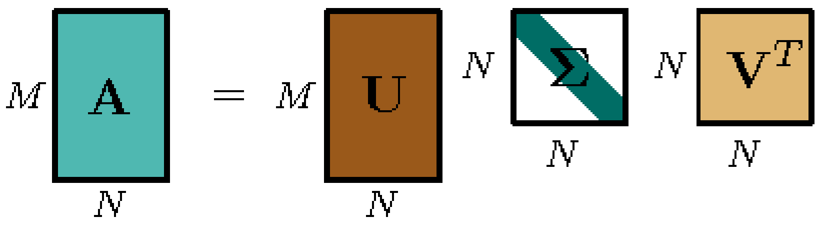
In reduced SVD:

- The $m - r$ columns that span the left null space are removed from U.
- The padded rows and columns of zeros are removed from $\sum$.
- The $m - r$ columns that span the null space are removed from V.

So it becomes

$$\underset{m\times n}{A} = \underset{m\times r, r\times r,\ r\times n}{U_r\, \Sigma_r\, V_r^T}$$

Note, both reduced SVD and full SVD results in the original A with no information loss.



(b) reduced SVD

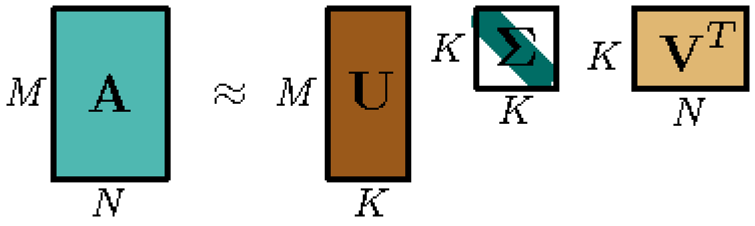
(Figure C.1.3)

In truncated SVD:

- We take k largest singular values ($0 < k < r$,thus truncated) and their corresponding left and right singular vectors,

$$\underset{m\times n}{A} \approx \underset{m\times k, k\times k,\ k\times n}{U_t\, \Sigma_t\, V_t^T}$$

- A constructed via truncated SCD is an approximation to original A.



(c) truncated SVD

(Figure C.1.4)

2. Latent Semantic Analysis (LSA)

LSA is also called Dimensionality reduction using truncated SVD.

- Advantages of LSA

- It is efficient and easy to implement.
- It also gives decent results that are much better compared to the plain vector space model.
- It is faster compared to other available topic modeling algorithms, as it involves document term matrix decomposition only.

- Disadvantages of LSA

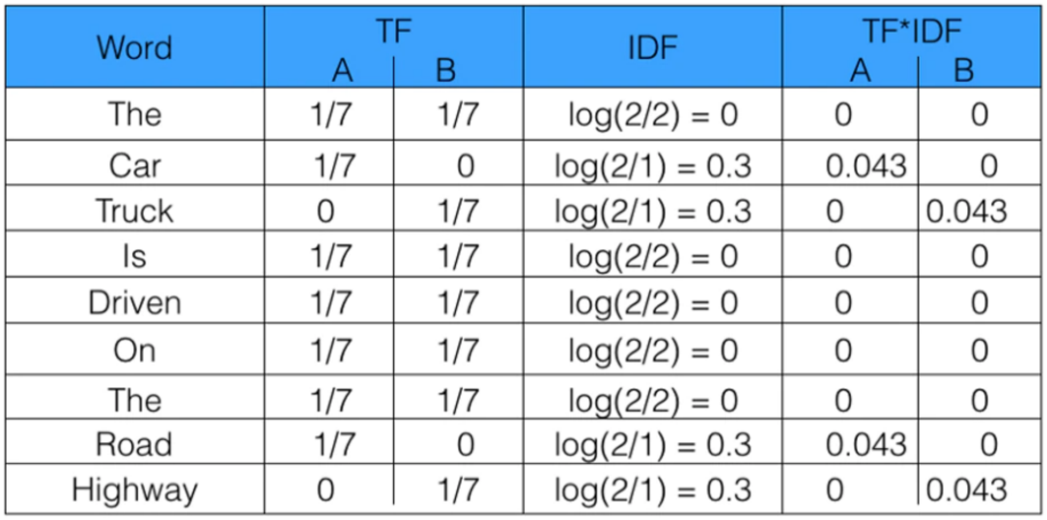
- Since it is a linear model, it might not do well on datasets with non-linear dependencies.
- LSA assumes a Gaussian distribution of the terms in the documents, which may not be true for all problems.
- LSA involves SVD, which is computationally intensive and hard to update as new data comes up.
- Lack of interpretable embeddings (we don't know what the topics are, and the components may be arbitrarily positive/negative)

3. TF-IDF

a. Mathematics of TF-IDF

When processing data in the LSA method, if the frequency is used, we will have a higher possibility to get noise. In order to reduce noise, we use TF-IDF to replace the frequency

$$tf\,idf(t, d, D) = tf(t, d) \, . \, idf(t, D)$$



Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	log(2/2) = 0	0	0
Car	1/7	0	log(2/1) = 0.3	0.043	0
Truck	0	1/7	log(2/1) = 0.3	0	0.043
Is	1/7	1/7	log(2/2) = 0	0	0
Driven	1/7	1/7	log(2/2) = 0	0	0
On	1/7	1/7	log(2/2) = 0	0	0
The	1/7	1/7	log(2/2) = 0	0	0
Road	1/7	0	log(2/1) = 0.3	0.043	0
Highway	0	1/7	log(2/1) = 0.3	0	0.043

A = "The car is driven on the road"; B = "The truck is driven on the highway" Image from freeCodeCamp - How to

(Figure C.3.1)

For example,the word "THE" appears in both sentences,if we use frequency to analyze .Because this word appears too many times, it is likely to become a keyword. But we all know that it is meaningless to analyze 'the'. TF-IDF can solve this problem very well. If all sentences have this word, Then IDF=0, the word will not become a keyword.

b. Term Frequency and Inverse Document Frequency (TF-IDF)

- Motivation:

  How to identify important words (or phrases, named entities) in a text in a collection or corpus? When searching for documents, we'd like to have these important words matched.

- Intuition:
  - In a document, if a word/term/phrase is repeated many times, it is likely important.

- However, if it appears in most of the documents in the corpus, then it has little discriminating power in determining relevance.
- For instance, a collection of documents on the auto industry is likely to have the term auto in almost every document. Search by "auto" you may get all the documents.

- TF-IDF: is composed by two terms:
    - TF (Term Frequency): which measures how frequently a term, say w, occurs in a document.
    - IDF (Inverse Document Frequency): measures how important a term is within the corpus.
- TF-IDF provides another way to remove stop words

c. Term Frequency (TF)

- Measures how frequently a term, say w, occurs in a document, say *d*. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones.
- Thus, the frequency of w in *d*, denoted as $freq(w, d)$ is often divided by the document length (a.k.a. the total number of terms in the document, denoted as |*d*| as a way of normalization:

$$tf(w, d) = \frac{freq(w,d)}{|d|}$$

d. Inverse Document Frequency (IDF)

- Measures how important a term is within the corpus.
- However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance.
- Thus we need to weigh down the frequent terms while scale up the rare ones.

- Let $|D|$ denote the number of documents, $df(w,D)$ denotes the number of documents with term w in them. Then,

$$idf(w) = ln(\frac{|D|}{df(w,D)}) + 1$$

or a smoothed version:

$$idf(w) = ln(\frac{|D|+1}{df(w,D)+1}) + 1$$

e. Normalize the TF-IDF

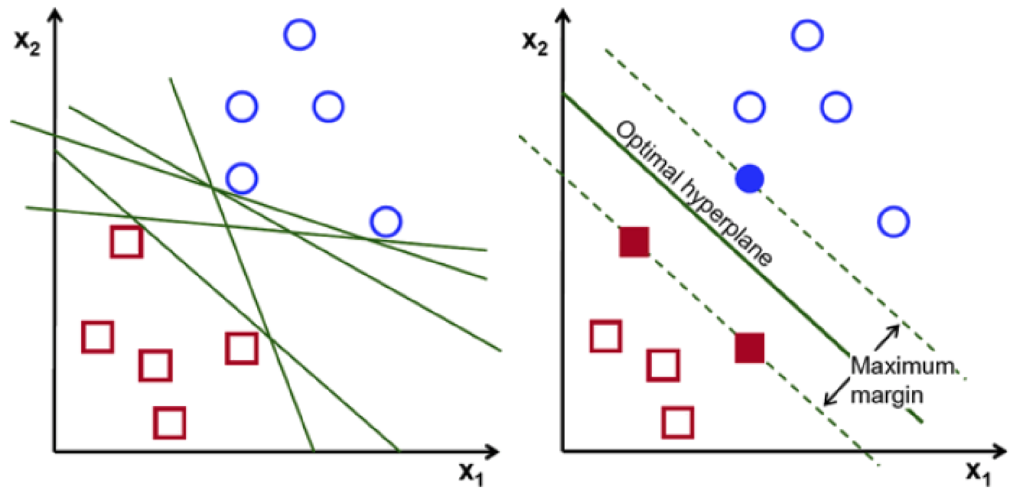
- Let $s(w,d)=tf(w,d)*idf(w)$, normalize the TF-IDF score of each word in a document normalized by the Euclidean norm, then

$$tfidf(w,d) = \frac{s(w,d)}{\sqrt{\sum w \in d \ s(w,d)2}}$$

- $*$ <- element wise multiplication

4. SVM

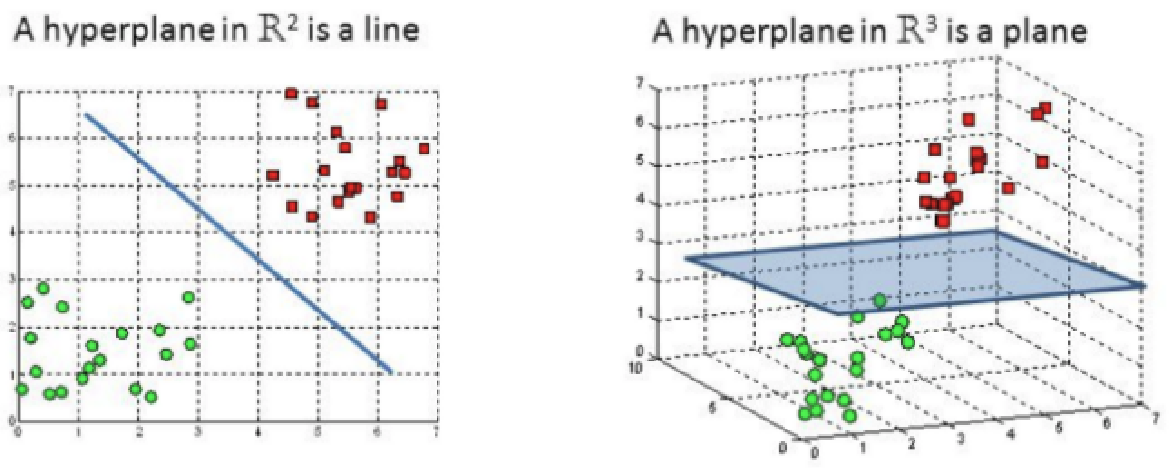
Support vector machine is highly preferred by many as it produces significant accuracy with less computation power. Support Vector Machine, abbreviated as SVM can be used for both regression and classification tasks. But, it is widely used in classification objectives. The objective of the support vector machine algorithm is to find an optimal hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.



Possible hyperplanes

(Figure C.4.1)

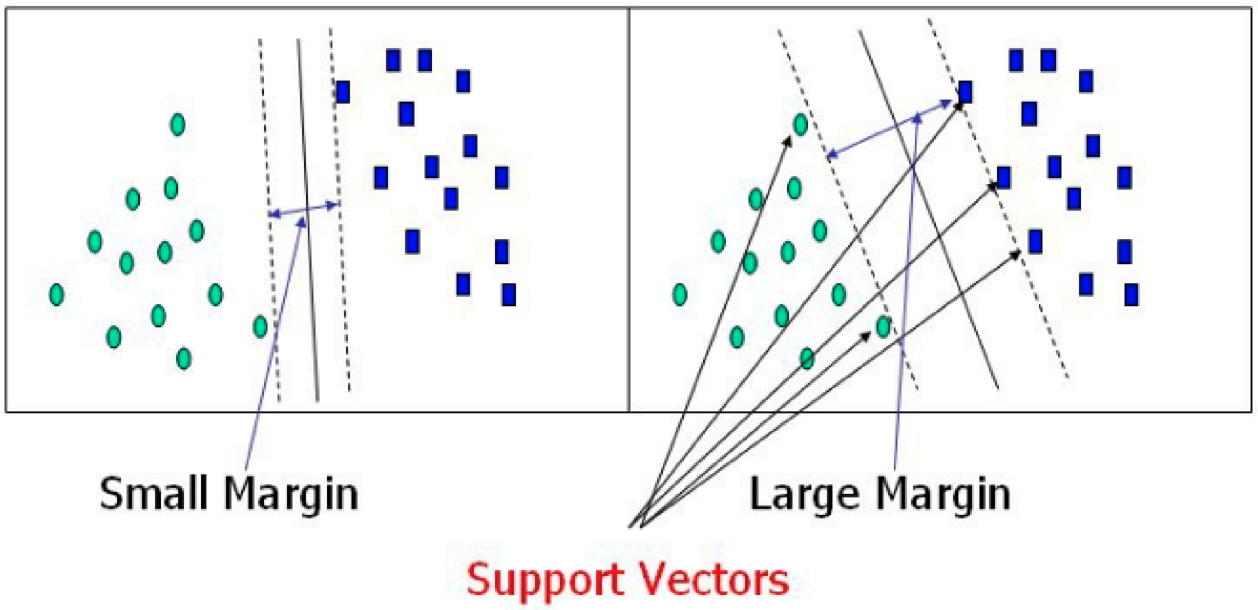
To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.



Hyperplanes in 2D and 3D feature space

(Figure C.4.2)

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.



(Figure C.4.3)

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

a. Large Margin Intuition

In logistic regression, we take the output of the linear function and squash the value within the range of [0,1] using the sigmoid function. If the squashed value is greater than a threshold value(0.5) we assign it a label 1, else we assign it a label 0. In SVM, we take the output of the linear function and if that output is greater than 1, we identify it with one class and if the output is -1, we identify is with another class. Since the threshold values are changed to 1 and -1 in SVM, we obtain this reinforcement range of values([-1,1]) which acts as margin.

b. Cost Function and Gradient Updates

In the SVM algorithm, we are looking to maximize the margin between the data points and the hyperplane. The loss function that helps maximize the margin is hinge loss.

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases} \qquad c(x, y, f(x)) = (1 - y * f(x))_+$$

Hinge loss function (function on left can be represented as a function on the right)

The cost is 0 if the predicted value and the actual value are of the same sign. If they are not, we then calculate the loss value. We also add a regularization parameter the cost function. The objective of the regularization parameter is to balance the margin maximization and loss. After adding the regularization parameter, the cost functions looks as below.

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

Loss function for SVM

Now that we have the loss function, we take partial derivatives with respect to the weights to find the gradients. Using the gradients, we can update our weights.

$$\frac{\delta}{\delta w_k}\lambda \parallel w \parallel^2 \quad = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k}\left(1 - y_i\langle x_i, w\rangle\right)_+ \quad = \begin{cases} 0, & \text{if } y_i\langle x_i, w\rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

Gradients

When there is no misclassification, i.e our model correctly predicts the class of our data point, we only have to update the gradient from the regularization parameter.

$$w = w - \alpha \cdot (2\lambda w)$$

Gradient Update — No misclassification

When there is a misclassification, i.e our model makes a mistake on the prediction of the class of our data point, we include the loss along with the regularization parameter to perform gradient update.

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$

Gradient Update — Misclassification

c. Pros and Cons associated with SVM

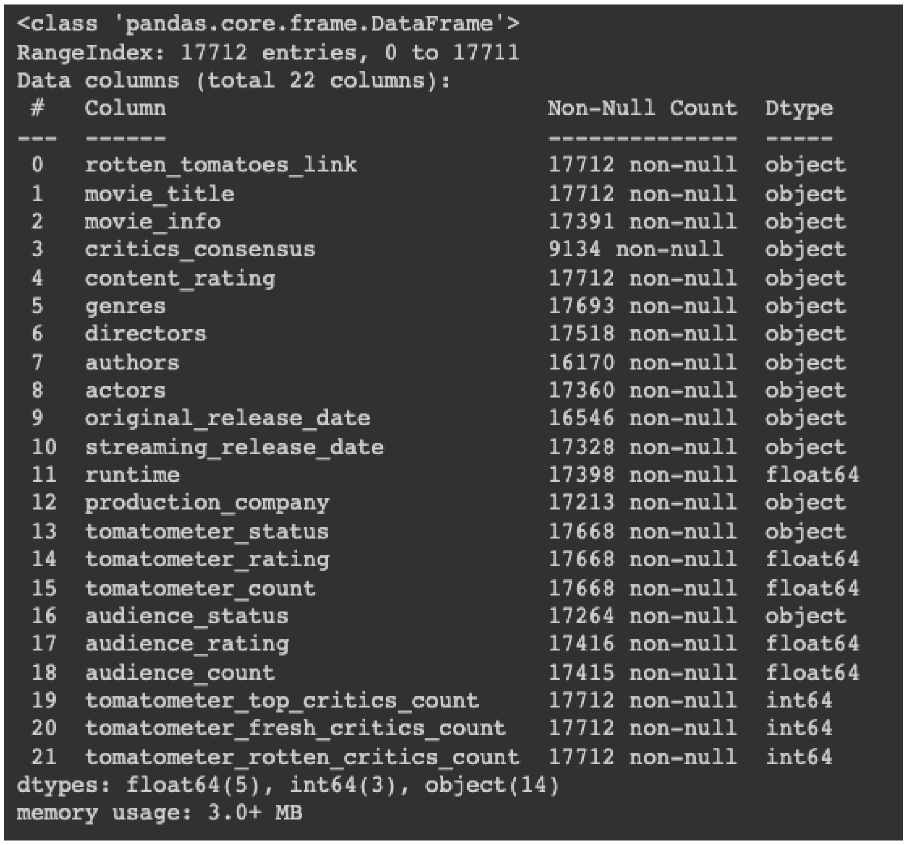
- Pros:
    - It works well with a clear margin of separation between classes.
    - It is effective in high dimensional spaces.
    - It is effective in cases where the number of dimensions is greater than the number of samples.
    - It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Cons:

- It does not perform well when we have large data sets because the required training time is higher.
- It also does not perform very well, when the data set has more noise i.e. target classes are overlapping.
- In cases where the number of features for each data point exceeds the number of training data samples, SVM will underperform.
- As the support vector classifier works by putting data points, above and below the classifying hyperplane, there is no probabilistic explanation for the classification. It does not directly provide probability estimates.

D. Model Implementation

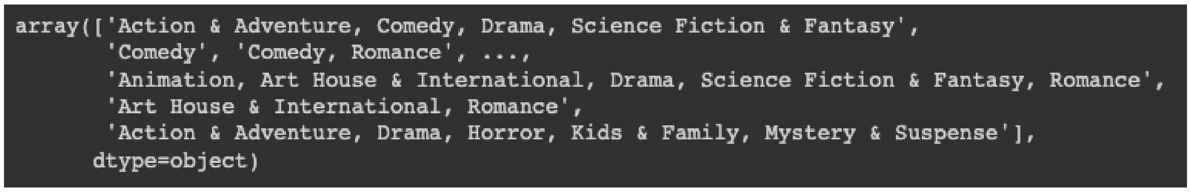
1. Data

Our data is based on the dataset “Rotten Tomatoes Movies critics and reviews” from Kaggle and it consists of 21 columns and 17,712 entries. As you may guess easily from the number of columns, this dataset has exhaustive information on the movie list on the website “Rotten Tomatoes”. Our main objective in the project is to derive the patterns from movie reviews and classify them. Therefore, we removed all the irrelevant columns except for “critics_consensus” and “genres”. The “Critics_consensus” column is the reviews, the subject data in which we analyzed and extracted the patterns. “Genres” column contains the genres of the movie, which allows us to label the text data to “Romantic Comedy” or “Action”.


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17712 entries, 0 to 17711
Data columns (total 22 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   rotten_tomatoes_link              17712 non-null  object
 1   movie_title                       17712 non-null  object
 2   movie_info                        17391 non-null  object
 3   critics_consensus                 9134 non-null   object
 4   content_rating                    17712 non-null  object
 5   genres                            17693 non-null  object
 6   directors                         17518 non-null  object
 7   authors                           16170 non-null  object
 8   actors                            17360 non-null  object
 9   original_release_date             16546 non-null  object
 10  streaming_release_date            17328 non-null  object
 11  runtime                           17398 non-null  float64
 12  production_company                17213 non-null  object
 13  tomatometer_status                17668 non-null  object
 14  tomatometer_rating                17668 non-null  float64
 15  tomatometer_count                 17668 non-null  float64
 16  audience_status                   17264 non-null  object
 17  audience_rating                   17416 non-null  float64
 18  audience_count                    17415 non-null  float64
 19  tomatometer_top_critics_count     17712 non-null  int64
 20  tomatometer_fresh_critics_count   17712 non-null  int64
 21  tomatometer_rotten_critics_count  17712 non-null  int64
dtypes: float64(5), int64(3), object(14)
memory usage: 3.0+ MB
```

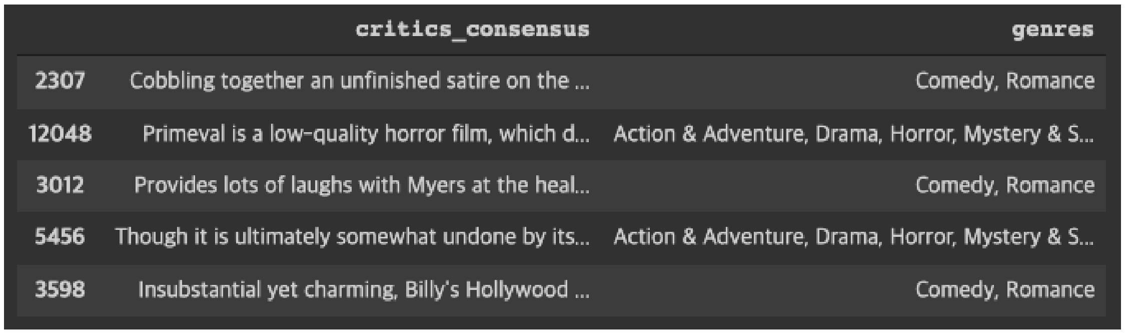
(Figure D.1.1)

The dataset was significantly desirable for our project in that it provided movie genres meticulously and detaily separated.


```
array(['Action & Adventure, Comedy, Drama, Science Fiction & Fantasy',
       'Comedy', 'Comedy, Romance', ...,
       'Animation, Art House & International, Drama, Science Fiction & Fantasy, Romance',
       'Art House & International, Romance',
       'Action & Adventure, Drama, Horror, Kids & Family, Mystery & Suspense'],
      dtype=object)
```

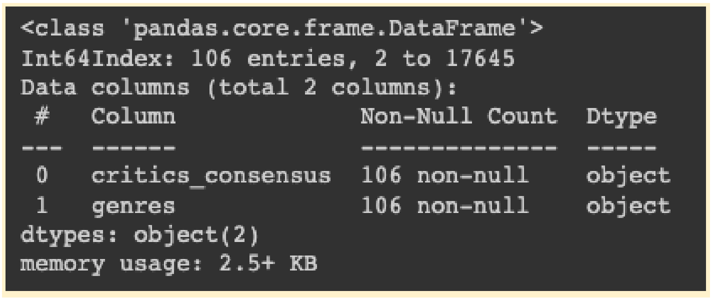
(Figure D.1.2)

As our main objective is to classify two classes : Action and Romantic comedy, we cleaned movie lists. For the purpose of the analysis, we selected movies with labels “Romance” and “Comedy” for one group: “Romcom (or Romantic Comedy)”, and “Action & Adventure”, “Horror” and “Mystery & Suspense” for the other group: “Action”. As a result, the dataset has narrowed down to 106 rows of movies and 2 columns of features: “Critics_consenssus” for actual reviews and “Genres” for labels.



	critics_consensus	genres
2307	Cobbling together an unfinished satire on the ...	Comedy, Romance
12048	Primeval is a low-quality horror film, which d...	Action & Adventure, Drama, Horror, Mystery & S...
3012	Provides lots of laughs with Myers at the heal...	Comedy, Romance
5456	Though it is ultimately somewhat undone by its...	Action & Adventure, Drama, Horror, Mystery & S...
3598	Insubstantial yet charming, Billy's Hollywood ...	Comedy, Romance

(Figure D.1.3)



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 106 entries, 2 to 17645
Data columns (total 2 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   critics_consensus  106 non-null    object
 1   genres             106 non-null    object
dtypes: object(2)
memory usage: 2.5+ KB
```

(Figure D.1.4)

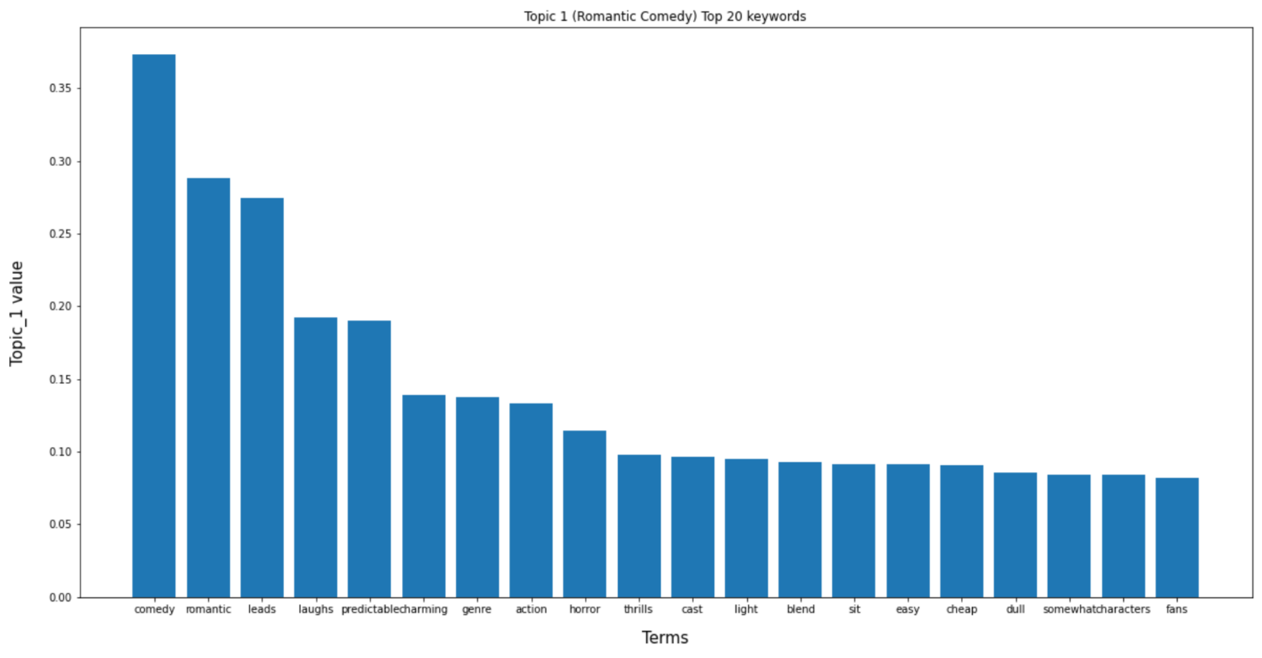
2. LSA by Countvectorizer

a. CountVectorizer()

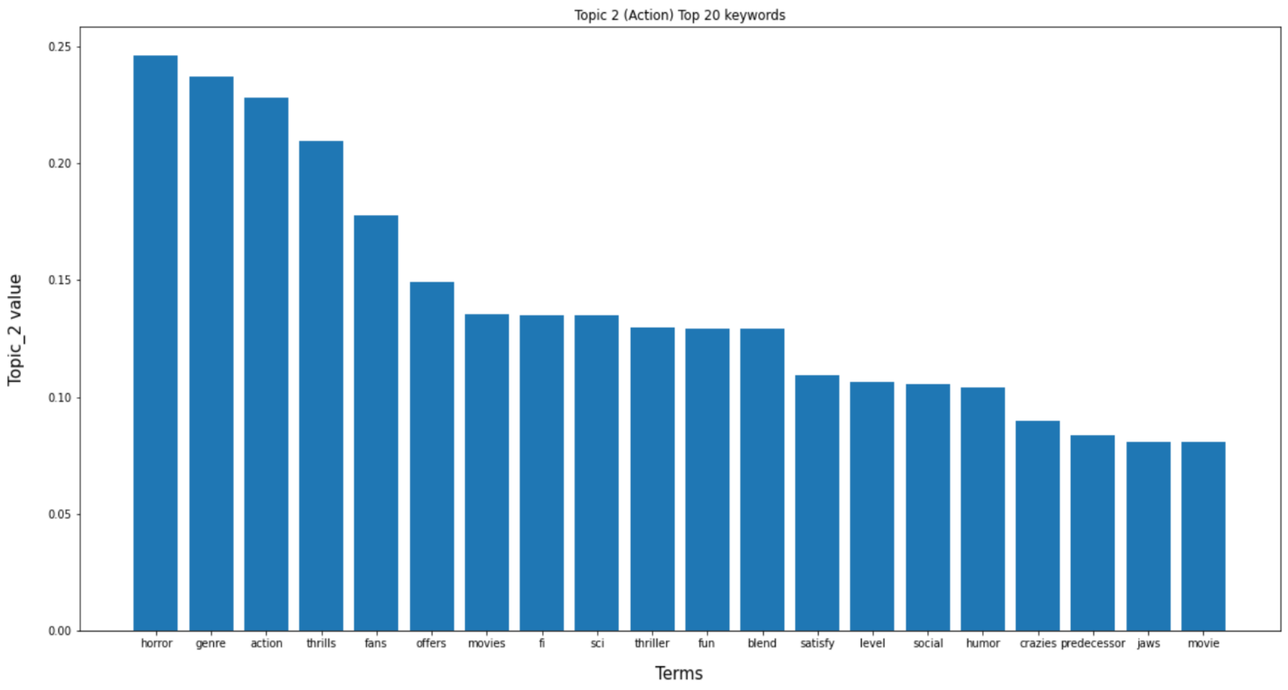
```
count_vec = CountVectorizer(min_df=1, stop_words='english')
bag_of_words_count = count_vec.fit_transform(action_romcom.critics_consensus)
bag_of_words_count.todense()
```

(Figure D.2.1)

The CountVectorizer() feature transforms the words of reviews into a matrix of rows of reviews by a word appearing in the entire data set of reviews. The main goal is to convert texts to values that are comprehensible by a computer. This generates useful byproducts such as the dictionary, a set of all words shown in at least one review, and the combination of the dictionary and the encoding matrix for a greater understanding of what each topic represents. For example, when sorted by the absolute values of "topic_1" and "topic_2", the words that are likely to define each genre are located at the top and are considered relevant for the reviews we analyze. Figure 5.2 shows that some top keywords for "topic_1" are "comedy", "romantic", "leads", "predictable", "laughs", "charming", "easy", and "shallow", while the Figure 5.3 shows "genre", "horror", "action", "thrills", "fans", and "crazies".



(Figure D.2.2)



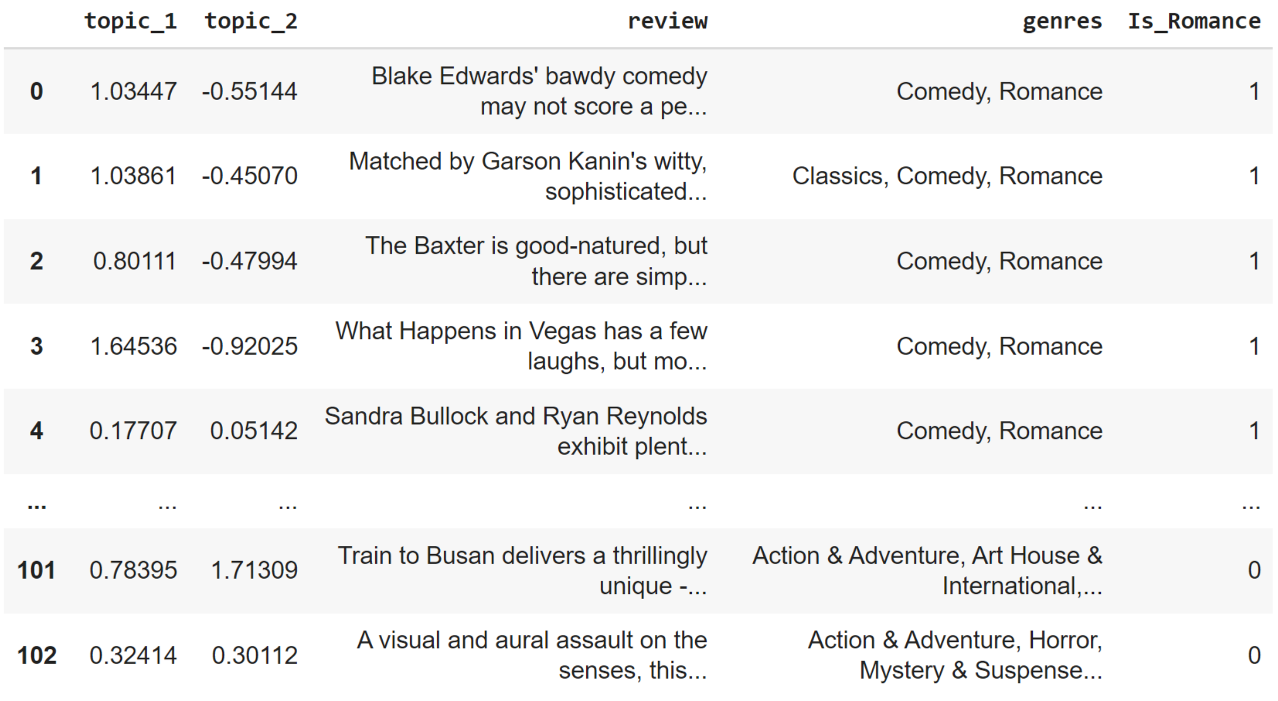
(Figure D.2.3)

b. TruncatedSVD()

```
svd = TruncatedSVD(n_components=2,n_iter=100)
lsa = svd.fit_transform(bag_of_words_count)
```

(Figure D.2.4)

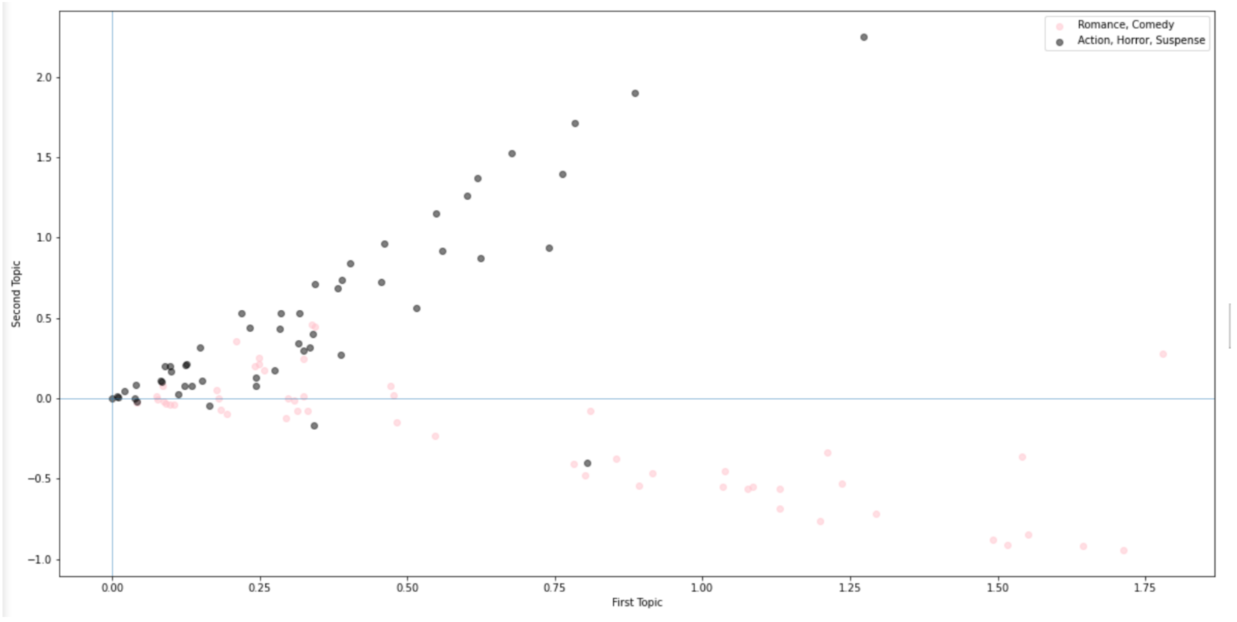
This specific type of SVD returns the numerical values of two columns assigned to be “topic_1” and “topic_2” in our data set shown in Figure 5.6. By finding hidden patterns, SLA outputs the two columns of distinct numbers that well represent the “genre” column. The categorical values of the “genre” column are converted to binary values and inserted into the “Is_Romance” column, and this additional step of conversion allows a smoother evaluation of the LSA.



	topic_1	topic_2	review	genres	Is_Romance
0	1.03447	-0.55144	Blake Edwards' bawdy comedy may not score a pe...	Comedy, Romance	1
1	1.03861	-0.45070	Matched by Garson Kanin's witty, sophisticated...	Classics, Comedy, Romance	1
2	0.80111	-0.47994	The Baxter is good-natured, but there are simp...	Comedy, Romance	1
3	1.64536	-0.92025	What Happens in Vegas has a few laughs, but mo...	Comedy, Romance	1
4	0.17707	0.05142	Sandra Bullock and Ryan Reynolds exhibit plent...	Comedy, Romance	1
...	...	...	...	...	...
101	0.78395	1.71309	Train to Busan delivers a thrillingly unique -...	Action & Adventure, Art House & International,...	0
102	0.32414	0.30112	A visual and aural assault on the senses, this...	Action & Adventure, Horror, Mystery & Suspense...	0

(Figure D.2.5)

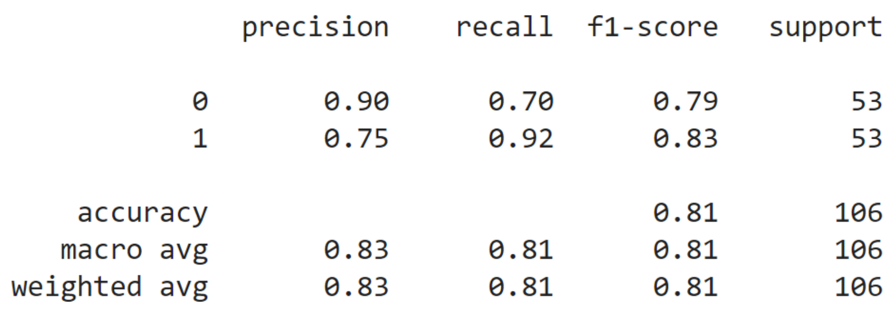
c. Visualization



(Figure D.2.6)

This scattered plot displays a clear distinction between two different genres. Values scored high in the first topic but low in the second topic belong to Romance/Comedy and opposite values belong to Action/Horror. Despite the clear trending of each review depending on its genre, misclassification of data points of Romance/Comedy is clustered around the lower range of values, especially those that have a value of “second_topic” in a range of 0 to 0.5.

d. Evaluation



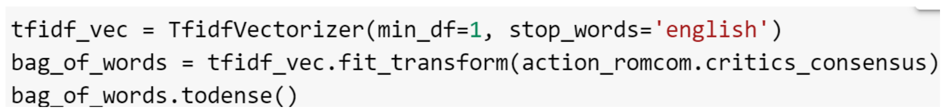
```
              precision    recall  f1-score   support

           0       0.90      0.70      0.79        53
           1       0.75      0.92      0.83        53

    accuracy                           0.81       106
   macro avg       0.83      0.81      0.81       106
weighted avg       0.83      0.81      0.81       106
```

(Figure D.2.7)

Based on the scatter plot of LSA findings, “topic_1” refers to a distinct group of Romance/Comedy reviews, and “topic_2” refers to a distinct group of Action/Horror. So the “predicted” column was added and if a row has a higher “topic_1” value, a value “1” was inserted and if not, “0”, an implication of Action/Hoor, was inserted. Then, the “predicted” column and the “Is_Romance” column were taken out for the classification report to evaluate precision and recall. As shown in Figure D.2.7, F1-score for CounteVectorizer recorded 83%

3. LSA by TF-IDF

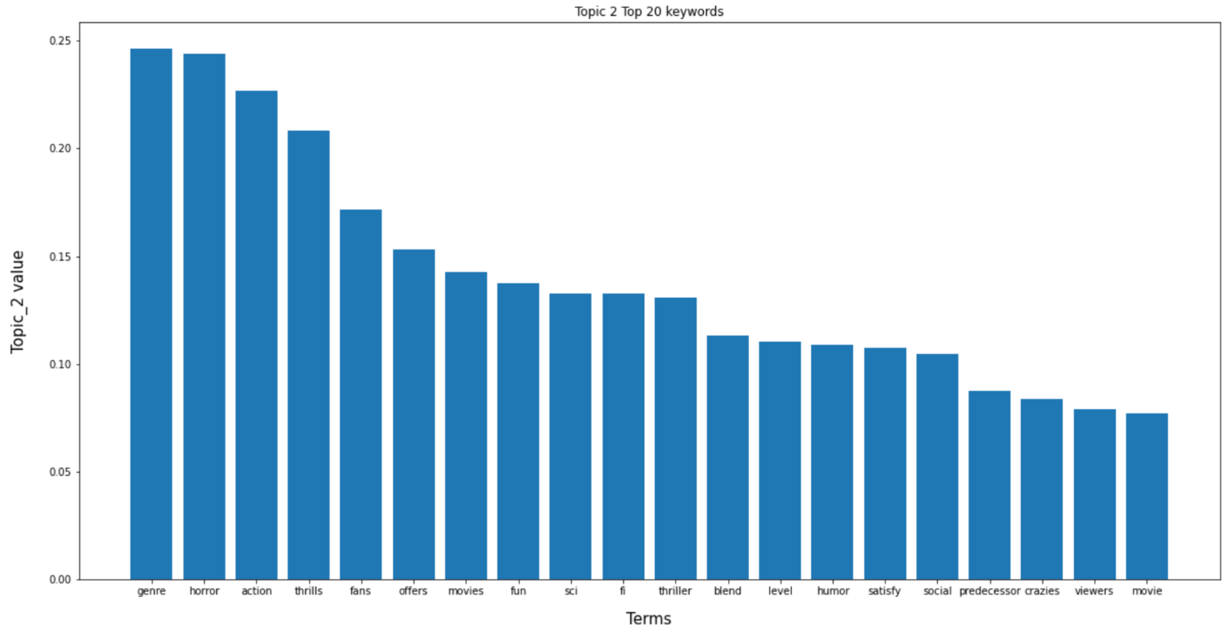
a. TfidfVectorizer()



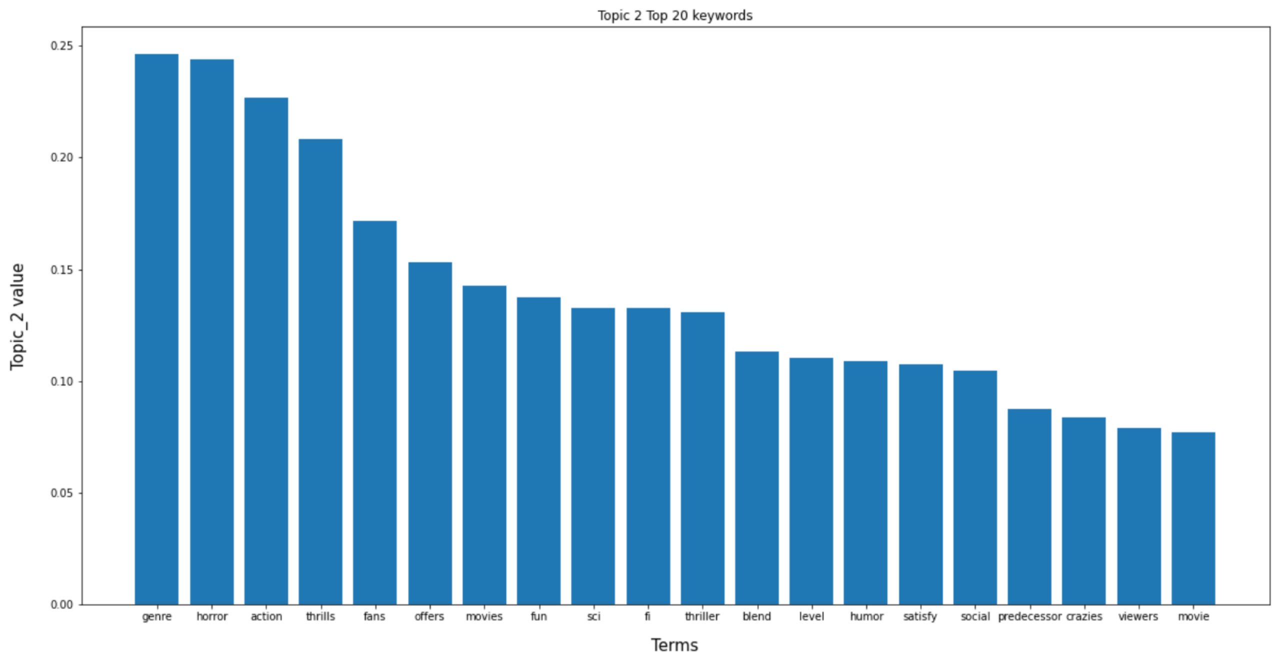
```
tfidf_vec = TfidfVectorizer(min_df=1, stop_words='english')
bag_of_words = tfidf_vec.fit_transform(action_romcom.critics_consensus)
bag_of_words.todense()
```

(Figure D.3.1)

TfidfVectorizer() feature transforms the words of reviews into a matrix of rows of reviews by words appearing in the entire data set of reviews. However, unlike the CountVectorizer(), this considers the frequency of a word appearing in the entire data set, hence expect better accuracy in the separation of reviews into two categories. Very similar to words that appeared for CountVectorizer LSA, Figure 5.10 shows that some top keywords for "topic_1" are "comedy", "romantic", "leads", "predictable", "laughs", "charming", "easy", and "shallow", and Figure 5.11 shows "genre", "horror", "action", "thrills", "fans", and "crazies".



(Figure D.3.2)



(Figure D.3.3)

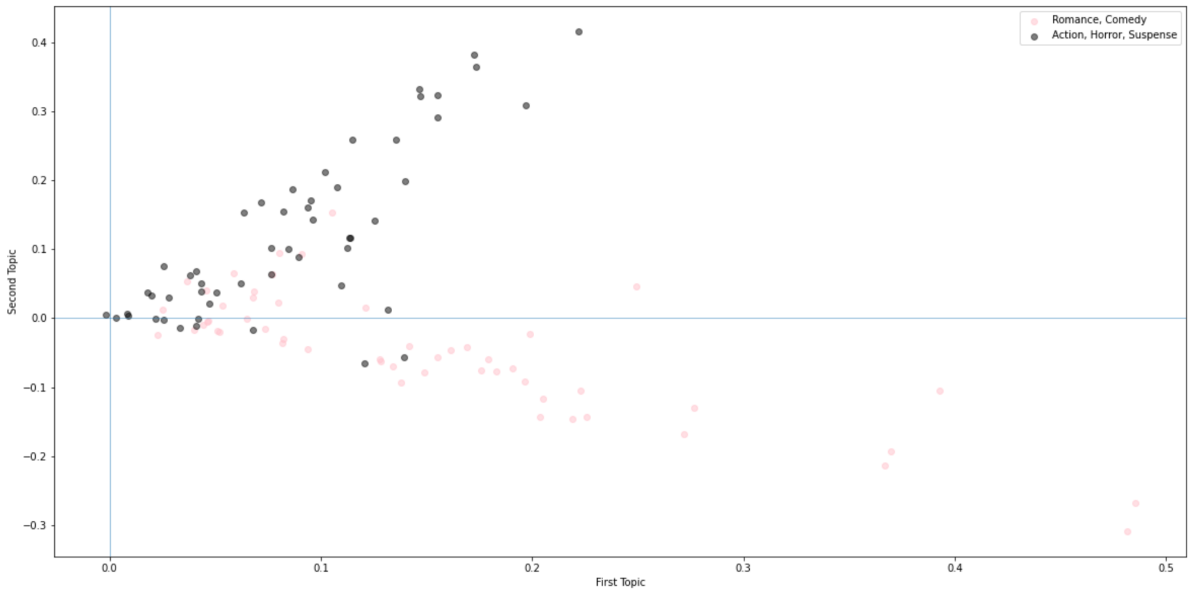
b. TruncatedSVD()

```
svd = TruncatedSVD(n_components=2,n_iter=100)
lsa = svd.fit_transform(bag_of_words_count)
```

(Figure D.3.4)

Same procedures for CountVectorizer LSA were taken here, so refer to the TruncatedSVD() section in LSA by Count Vectoizer section to see more details.

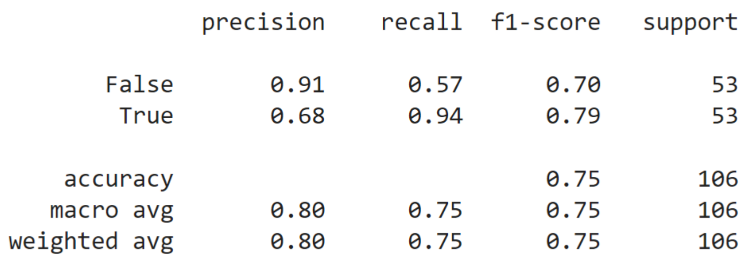
c. Visualization



(Figure D.3.5)

This scattered plot also displays a clear distinction like how the Countectorizer did. However, the data points for TF-IDF are much lower than that of Countvectorizer since the log is applied to the quotient of a total number of reviews and the number of reviews containing that certain word.

d. Evaluation



```
              precision    recall  f1-score   support

       False       0.91      0.57      0.70        53
        True       0.68      0.94      0.79        53

    accuracy                           0.75       106
   macro avg       0.80      0.75      0.75       106
weighted avg       0.80      0.75      0.75       106
```

(Figure D.3.6)

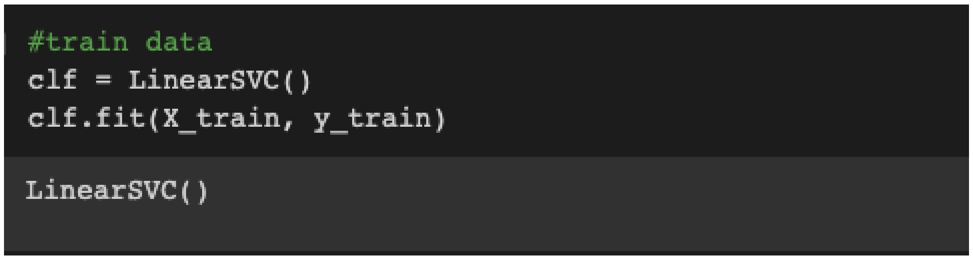
Utilized the same logic used in LSA by Count Vectorizer, F1-score for came out to be 79%

4. Support Vector Machine (SVM)

For the purpose of comparison to LSA, we implemented SVM to the same dataset. SVM is one of widely used classification machine learning techniques for its relatively simple yet powerful performance to classify data points. Our team wondered what difference this technique would make from LSA. Our work on SVM consists of three parts: First, to train the model without parameters. Second, to implement the GridsearchCV method to find the best parameters. Third, to train the model again with parameters obtained from GridsearchCV.

a. SVM without any hyperparameters

First, we trained the SVM model without any hyperparameters (so called “A vanilla model”) to observe how the factors give impact on the performances.



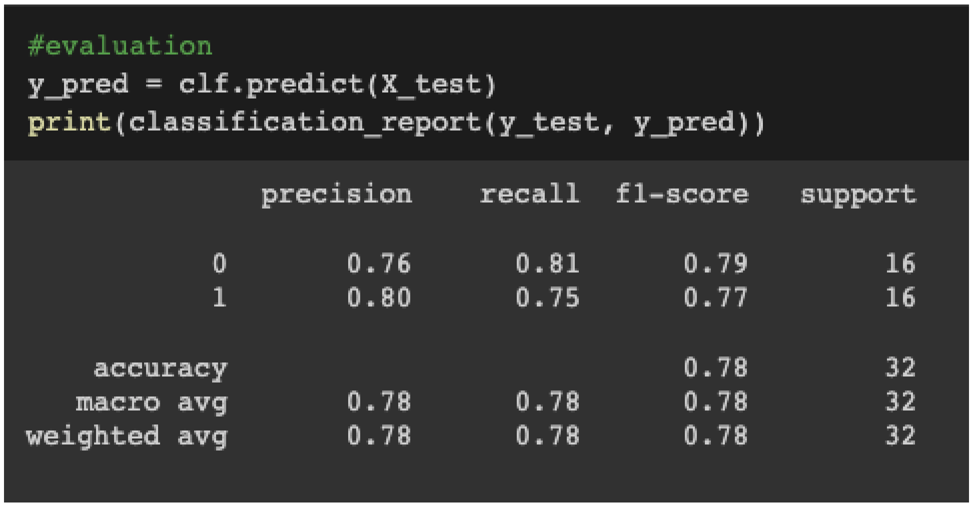
```
#train data
clf = LinearSVC()
clf.fit(X_train, y_train)
```

```
LinearSVC()
```

(Figure D.4.1)

b. Evaluation

It showed a f1-score of 0.77 and accuracy score of 0.78, which were not perfect and a little less than those of LSA. However, we already expected such a result before conducting the model. Our interest was more about how much difference the hyperparameters would make to the performances.



```
#evaluation
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.81      0.79        16
           1       0.80      0.75      0.77        16

    accuracy                           0.78        32
   macro avg       0.78      0.78      0.78        32
weighted avg       0.78      0.78      0.78        32
```

(Figure D.4.2)

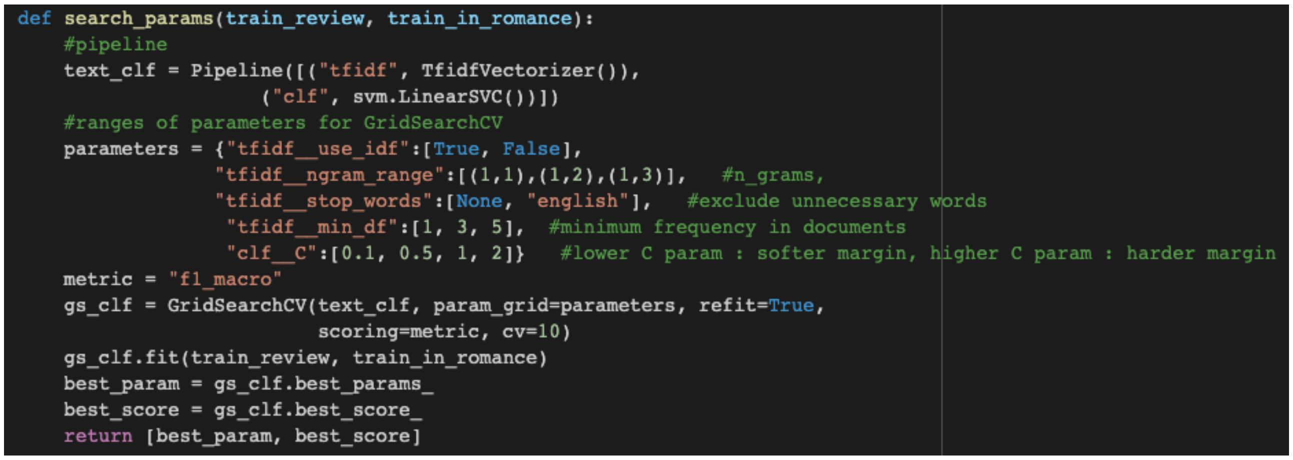
c. GridsearchCV

To fine tune the hyperparameters, GridsearchCV has been utilized. GridSearchCV is a library function that is a member of sklearn's model_selection package. It helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, you can select the best parameters from the listed hyperparameters. One of the best features of this technique, aside from tuning parameters manually, is it allows you to tune hyperparameters for multiple models at the same time. As you can see the following picture shows that we implemented tuning parameters for TfidfVEctorizer (for vectorizing words) and LinearSVC (for SVM model). The variable "text_clf" contains the function allowing us to conduct parameter tuning for both models and the variable "parameters" has a range of subject hyperparameters to be tested in combination with each parameter.

The subject hyperparameters are as follows:

- Tfidf__use_idf [True, False] : To use idf or not.
- Tfidf__ngram_range [(1,1), (1,2), (1,3)] : How many ngram to be allowed.
- Tfidf__stop_words [None, "english"] : Not to use stop words or to use "english:.

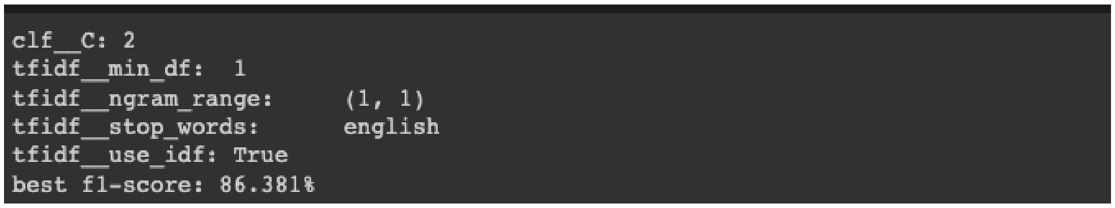
- Tfidf__min_df [1, 3, 5] : To assign minimum appearance of words in documents.
- clf__C [0.1, 0.5, 1, 2] : How hard or soft margins are allowed when setting a hyperplane.



```python
def search_params(train_review, train_in_romance):
    #pipeline
    text_clf = Pipeline([("tfidf", TfidfVectorizer()),
                         ("clf", svm.LinearSVC())])
    #ranges of parameters for GridSearchCV
    parameters = {"tfidf__use_idf":[True, False],
                  "tfidf__ngram_range":[(1,1),(1,2),(1,3)],   #n_grams,
                  "tfidf__stop_words":[None, "english"],   #exclude unnecessary words
                   "tfidf__min_df":[1, 3, 5],  #minimum frequency in documents
                   "clf__C":[0.1, 0.5, 1, 2]}   #lower C param : softer margin, higher C param : harder margin
    metric = "f1_macro"
    gs_clf = GridSearchCV(text_clf, param_grid=parameters, refit=True,
                          scoring=metric, cv=10)
    gs_clf.fit(train_review, train_in_romance)
    best_param = gs_clf.best_params_
    best_score = gs_clf.best_score_
    return [best_param, best_score]
```

(Figure D.4.3)

With the different hyperparameters, GridsearchCV conducted all the possible combinations of them, and it gave us the information that if you plug in the suggested parameters, you would obtain the highest prediction scores.

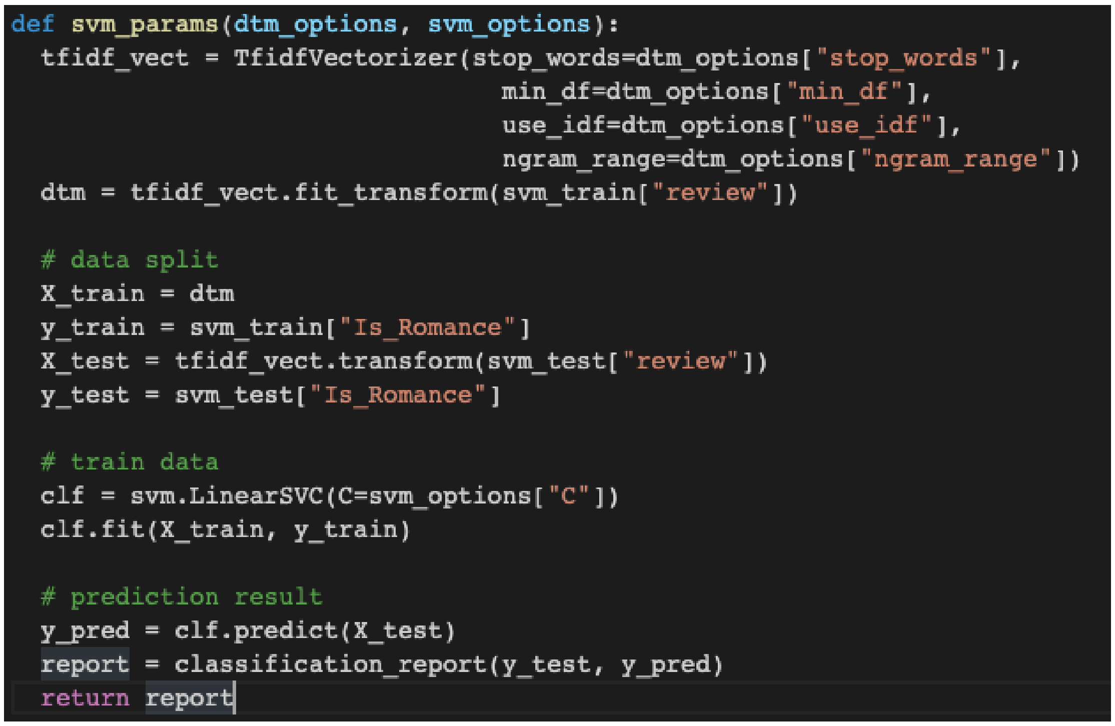


```
clf__C: 2
tfidf__min_df:  1
tfidf__ngram_range:     (1, 1)
tfidf__stop_words:      english
tfidf__use_idf: True
best f1-score: 86.381%
```

(Figure D.4.4)

d. SVM with the hyperparameters acquired from GridsearchCV

Then the SVM model was trained to the same dataset again with the tuned hyperparameters assisted by GridsearchCV. The parameter tuning method told us that we would have an f1-score of 86.381%.

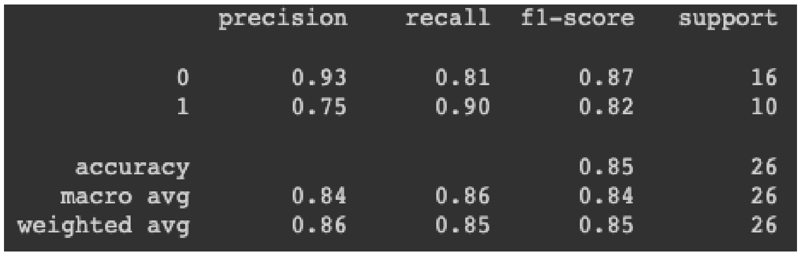


```python
def svm_params(dtm_options, svm_options):
  tfidf_vect = TfidfVectorizer(stop_words=dtm_options["stop_words"],
                               min_df=dtm_options["min_df"],
                               use_idf=dtm_options["use_idf"],
                               ngram_range=dtm_options["ngram_range"])
  dtm = tfidf_vect.fit_transform(svm_train["review"])

  # data split
  X_train = dtm
  y_train = svm_train["Is_Romance"]
  X_test = tfidf_vect.transform(svm_test["review"])
  y_test = svm_test["Is_Romance"]

  # train data
  clf = svm.LinearSVC(C=svm_options["C"])
  clf.fit(X_train, y_train)

  # prediction result
  y_pred = clf.predict(X_test)
  report = classification_report(y_test, y_pred)
  return report
```

(Figure D.4.5)

e. Evaluation

We trained the model with the assisted hyperparameters and we were able to acquire an f1-score of 82% and an accuracy score of 85%. It was down by 4% from the f1-score that GridsearchCV suggested. However, with the tuned parameters the result of our SVM model increased, from the f1-score 77% to 82% and the accuracy score 78% to 85%.



```
              precision    recall  f1-score   support

           0       0.93      0.81      0.87        16
           1       0.75      0.90      0.82        10

    accuracy                           0.85        26
   macro avg       0.84      0.86      0.84        26
weighted avg       0.86      0.85      0.85        26
```

(Figure D.4.6)

E. Conclusion

From the movie-review dataset, we conducted four different methods to extract the distinct two patterns, and classify them as “Action” and “Romantic Comedy” according to the genres. In the process, we could observe some interesting aspects while analyzing the top 20 keywords.

a. Interesting Observations from top 20 keywords

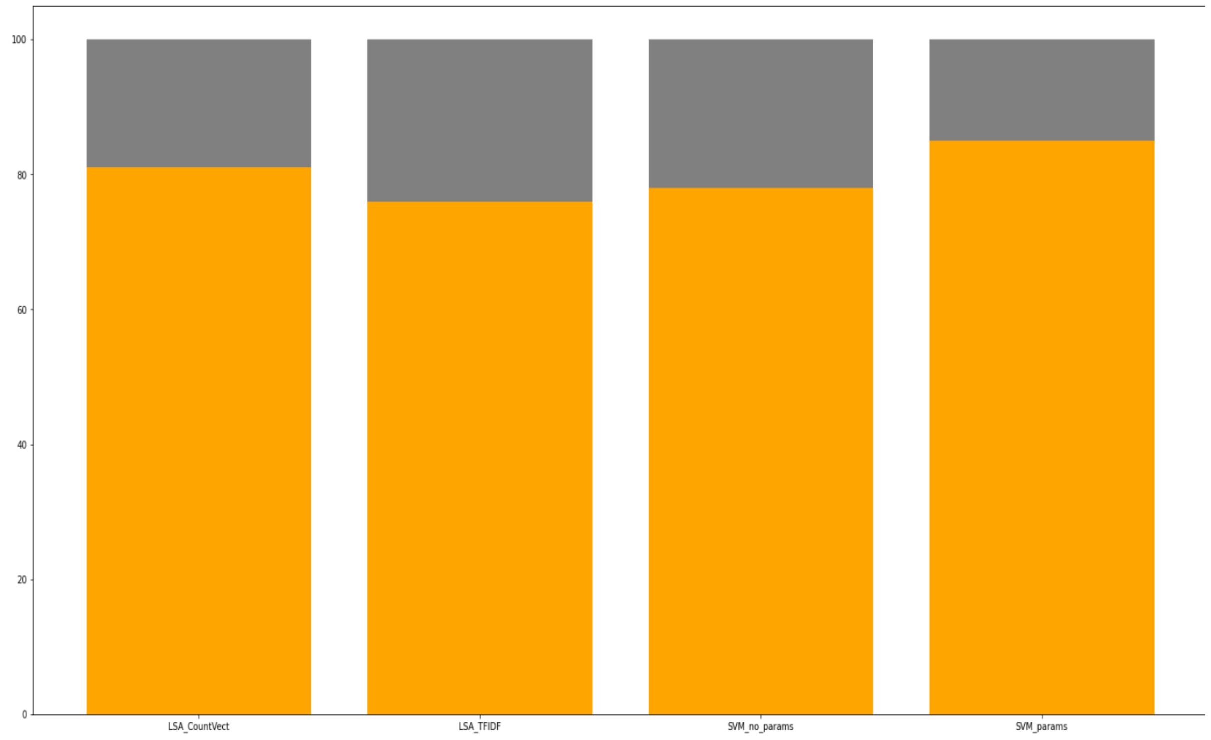
While there were some noise terms such as “sit”, “fans”, “movie” and “viewers”, the top 20 keywords obtained from both CounterVectorizer and Tf-idf clearly gave us evidence that which keywords played a role in determining the patterns and resulting classifications of each document. Further, we observed some quite interesting aspects from the top keywords revealing clues that allow us to learn some viewers’ unique sentiments toward different genres. To take some examples, in Romantic Comedy, if we remove very obvious first and second placed terms “comedy” and “romantic”, the term comes next is “leads”. It is said that male and female leads are one of the, or the most, pivotal factors in romantic comedies, and some claim that attractive leads are even more important in romantic comedies than the stories or narratives because the stories of such genre to offer are usually typical, conventional and predictable. Our work, in fact, reveals these common beliefs as the terms “predictable”, “charming”, “cast”, “easy”, “blend”, “dull”, “shallow” are on top 20 keywords. Even though these terms are fragmented that do not provide any contextualized information, we could easily assume what these words refer to.

b. Results from different models

Of the four models, the SVM with parameters showed the best result with f1-score 0.82 and accuracy score 0.85. In contrast, the SVM with no parameters showed the lowest scores. One thing unexpected was regarding the different results from methods of vectorization. Our team assumed that the Tf-idf technique would be more effective than that of CounterVectorizer in that while the latter method merely counts frequencies of terms, Tf-idf takes into account the importance of terms in a document. However, it turned out that, as opposed to our assumption, LSA with ConterVectorizer showed higher classification results than the one with Tf-idf.

	LSA		SVM	
	CounterVectorizer	TF-IDF	Without Parameters	With Parameters
f1-score	0.83	0.79	0.77	0.82
accuracy	0.81	0.75	0.78	0.85

(Figure E.1.1)



(Figure E.1.2)

F. Limitations and Future Work

Inconsistent classification measurements hindered the process of accurate evaluations for LSA. Classification results did not imply the absolute distinction between the two different categories that we looked at since they weigh more in the difference between the two values than solely in each value. Also, the F1 score for TF-IDF did not surpass the scores for CountVectorizer which is supposed to have a lower accuracy since it doesn't count the frequency of a word appearing throughout the documents. Our future work will be dedicated to revealing any hidden logic and causes behind this SLA technique so that we better understand the current result, and further enhance the quality.

Moreover, not confined to only two distinct genres, analysis of variations of genres will reveal more insightful patterns and let us find other uses of this method. Also, we can apply these methods to other types of data sets such as restaurant reviews and book reviews. Lastly, improvement of the accuracy of current work can be done by trying out various classification methods other than the ones that have been used in this project and evaluating to find one that outperforms.

G. References

[1] Rotten Tomatoes movies and critic reviews dataset

https://www.kaggle.com/datasets/stefanoleone992/rotten-tomatoes-movies-and-critic-reviews-dataset

[2] Latent Semantic Analysis & Sentiment Classification with Python

https://towardsdatascience.com/latent-semantic-analysis-sentiment-classification-with-python-5f657346f6a3

[3] Sklearn Decomposition TruncatedSVD

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html#examples-using-sklearn-decomposition-truncatedsvd

[4] Images of the full SVD and the truncated SVD

https://www.semanticscholar.org/paper/Tensor-Approximation-in-Visualization-and-Graphics-Suter/6265915841382cf7ccaa41880838490a7f91bc1b

[5] Images of the SVM

https://www.analyticsvidhya.com/blog/2021/03/beginners-guide-to-support-vector-machine-svm/

http://www.semanticscholar.org

[6] Python Implementation

https://colab.research.google.com/drive/1vIebBf_mgPH9yjfRi2Oweip_9yTsAqjf?authuser=2#scrollTo=6CGhGVBVxLbc