Final Project - An Exploration of Linear Algebra for Machine Learning

Name: Nicolas Jorquera

Overview

In this project I utilize a variety of Linear Algebra techniques to discover patterns in the data. I also explore how certain algorithms are better depending on the data given.

In order to do this I utilized the GoodReads Data Set provided by UCSD Book Graph. There website can be found here : https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home.

For the purpose of this project, I utilized the comic book subsection which included only the books labeled as comic books. (The full dataset had over 13 million entries).

- ***Also for the purpose of cleansiness, I have refrained from displaying datasets. An example of these datasets can be found in the PowerPoint presentation given; were I felt they had better usage.***

Loading the Data

In [1]:

```
# Importing Necessary Packages:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Database 1 - Reviews of Comic Books by Individuals

In [2]:

```
# Reading the csv file:
df = pd.read_json('goodreads_reviews_comics_graphic.json.gz', lines=True)
```

In [3]:

```
# Sample of what the database includes:
# df.head(1)
```

In [4]:

```
# Number of Unique Users:
df.user_id.nunique()
```

Out[4]:

```
59347
```

In [5]:

```
# Number of Unique Books:
df.book_id.nunique()
```

Out[5]:

```
89311
```

In [6]:

```
# Getting a list of the Columns:
```

```
col = df.columns
```

- **This database gives us the interaction between users and books; and provides the rating given, a written wreview, and how much attention their feedback received.**

Dataframe 2 - Information on Books in Dataframe 1

In [7]:

```
df2 = pd.read_json('goodreads_books_comics_graphic.json.gz', lines=True)
```

In [8]:

```
# Sample of what the database includes:
# df2.head(2)
```

In [9]:

```
# Number of Unique Books:
df2.book_id.nunique()
```

Out[9]:

```
89411
```

- **Here we can use the book_id as the foreign key; which connects one dataframe to another. In this dataframe we can utilize the average rating, title, publication year, author and description.**

Dimensionality Reduction Techniques

Throughout this semester we were introduced to a variety of Dimensionality Reduction techniques. In this section the benefits of 2 techniques are shown by utilizing the datasets shown above.

1. ***Singular Value Decomposition*** **will be used to create a Collaborative Filtering Recomendation System. This becomes useful when there is alot of sparce data; and is a more robust matrix factorization technique. We will also explore how to utilize SVD from the sci-kit learn library.**
2. ***Principal Component Analysis*** **will be used to reduce the data into 2 components, and try to find any patterns that emerge from the 2-dimensional representation of the data.**

1. Singular Value Decomposition : Collaborative FIltering

In [10]:

```
ratings = df[['user_id','book_id','rating']]
```

In [11]:

```
print('Number of Reviews is -',len(ratings.index))
```

```
Number of Reviews is - 542338
```

Setting restrictions to amount of reviews left - Allows us to limit matrix to include only books/users with more than 10 reviews, to try to limit the sparcity of the matrix and make the algorithm more accurate

In [12]:

```
# Trying to restrict to users with more than ___ reviews. - In this case 10
user_count = ratings[['user_id','book_id']].groupby('user_id').count()
user_count = user_count[user_count['book_id'] >= 10]
```

In [13]:

```
# Trying to restrict to books with more than ___ reviews - In this case 10
book_count = ratings[['user_id','book_id']].groupby('book_id').count()
book_count = book_count[book_count['user_id'] >= 10]
```

In [14]:

```
# Reducing the overall ratings matrix to include books/users with more than 10 reviews.
ratings = ratings[ratings["user_id"].isin(user_count.index) & ratings["book_id"].isin(bo
ok_count.index)]
```

Creating a Sparce Matrix - The rows will the individual users, the columns will be indiviudal books and the values will be the raings given; with a 0 if no rating has been given.

In [15]:

```
reviewmatrix = ratings.pivot(index="user_id", columns="book_id", values="rating").fillna
(0)
```

In [16]:

```
reviewmatrix.head(2)
```

Out[16]:

book_id	866	868	869	870	871	873	2879	2880	2881	2882	...	35541317	35657815	35669801
user_id														
0008931c0cde961e9c802c5a58196d23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
000efb30c5236d7437c3cdf4bf3e4dc7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0

2 rows × 9307 columns

Taken from Notes in Class:

Theorem: Singular Value Decomposition

Any arbitrary matrix $A \in \mathcal{C}^{m \times n}$ can be factored as

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

where U is an $m \times m$ unitary matrix, Σ is an $m \times n$ diagonal matrix, and V is an $n \times n$ unitary matrix.

As mentioned above, SVD is a type of matrix factorization that decomposes a matrix A into the components seen above. In the case of our matrix, the vh values are the right singular vectors of A that pertain the books, while the u values pertain to the people reviewing the books. As discussed in class, SVD utilizes the properties of the singular values (values on the diagonal vector) and eigenvectors (Eigenvectors of A dot A.T) reveal alot about the structure of the matrix.

In [17]:

```
# Singular value decomposition - Linear Algebra Method
from numpy.linalg import svd
matrix = reviewmatrix.values
u, s, vh = svd(matrix, full_matrices=False)
```

In [18]:

```
# Vh - sample dataframe of SVD scores given to books. The closer the score
pd.DataFrame(vh).head(2)
```

Out[18]:

0	1	2	3	4	5	6	7	8	9	9297	9298

	0	1	2	3	4	5	6	7	8	9	...	9297	9298
0	0.003274	0.00573	0.004088	0.013791	0.004718	0.007947	0.000883	0.005844	0.000838	0.000682	...	0.001419	0.000803
1	-0.002586	-0.00255	-0.002800	-0.005325	-0.002117	-0.003141	0.000152	-0.002632	-0.000466	0.000308	...	0.001374	-0.000318

2 rows × 9307 columns

Cosine Similarity

To determine which books are closest to one another we need to Cosine Similarity, which is a metric that measures how similar two vectors are in a multi-dimensional space. In this case, how close books are to one another.

In [19]:

```
# Defition of Cosine Similarity
def cosine_similarity(v,u):
    return (v @ u)/ (np.linalg.norm(v) * np.linalg.norm(u))

highest_similarity = -np.inf
highest_sim_col = -1
```

Below; cosine similarity is utilized for the vh matrix, which correlates to the book matrix. Books that are similar to one another will have a value closer to one! One way to quickly check that no data has been lost is to make sure that the dimensionality of matrix vh (nxn) is equal to the number of books: n. This is the case as proven below:

In [20]:

```
print('Both have the same dimensionality -', len(vh) == len(reviewmatrix))
```

```
Both have the same dimensionality - True
```

In [21]:

```
# Finding the highest similarity - Movie to Movie
for col in range(1,vh.shape[1]):
    similarity = cosine_similarity(vh[:,0], vh[:,col])
    if similarity > highest_similarity:
        highest_similarity = similarity
        highest_sim_col = col
```

In [22]:

```
# Finding the book that is most similar to book n
n = 0
print("Column %d (book id %s) is most similar to column 0 (book id %s)" %
        (highest_sim_col, reviewmatrix.columns[col], reviewmatrix.columns[n])
)
```

```
Column 7548 (book id 36232397) is most similar to column 0 (book id 866)
```

2. Singular Value Decomposition - Utilizing Sci-Kit Learn

Here we utilize sci-kit learn, a powerful Machine Learning algorithm with alot of the tools we have been learnign throughout the semester built in. In order to make the SVD algorithm more accurate, we will utilize it in another way below; and make use of the tools we used for ML Models.

In [23]:

```
# Importing Necessary Packages

from sklearn.decomposition import TruncatedSVD
```

```
from sklearn.model_selection import train_test_split
```

In [24]:

```
text = df[['review_text','rating']].sample(n=10000)
```

In [25]:

```
# text.head()
```

Text Vectorization

In this example we will use a text vectorizer to represent the reviews of each individual as a sparce matrix, where the rows are each individual review, while the columns represent frequency of word appearance. A 0 appears if the word has not appeared.

In [26]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
```

In [27]:

```
bag = vectorizer.fit_transform(text.review_text)
```

Singular Value Decomposition

Here we can utilize another purpose of SVD, specifically to reduce dimensionaliuty when working with ML problems. Forunately in this case the dat is not significant, but if I were to utilize the full dataset of goodreads it would most certainly come in handy. To utilize this application I will use a Truncated version of SVD, a factorized data matrix where the number of columns is equal to the truncation. In this way it is similar to PCA, but the estimator "does not center the data".

In [28]:

```
# Sample Sparce Matrix
bag.A
```

Out[28]:

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

In [29]:

```
X = bag.A
y = text.rating
```

In [30]:

```
# Singular Value Decomposition
svd = TruncatedSVD(n_components=5, n_iter=7, random_state=42)
svd.fit(X)
```

Out[30]:

```
TruncatedSVD(n_components=5, n_iter=7, random_state=42)
```

In [31]:

```
# Splitting the data into testing / training data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=4
2)
```

Linear Regression - Utilizing my own Linear Regression Code from Programming Assignment 9

Machine learning model that can predict the score based on the review of the user. I will not delve too much into the optimizing this ML model as other students in the class are pursuing this. Instead I chose to focus on how SVD can be utilized in this scenario.

In [32]:

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

In [33]:

```
# Utilizing Linear Regression Code from Programming Assigment 9

lr = LinearRegression()
lr.fit(X_train, y_train)
print ("R-squared for the training data: ",lr.score(X_train, y_train))
print ("R-squared for the test data: ",lr.score(X_test, y_test))
```

```
R-squared for the training data:  0.9794473380392279
R-squared for the test data:  -7.084773366941225e+16
```

Observation

- **Here multiple improvements could be made to the model to make it more accurate, however I chose to focus on the Linear Algebra applications. In the future, a Neural Network is usually used for NLP problems. Furthermore, k-crossfold validation and various feature engineering techniques could also be utilized to optimize the cost function.**

3. Principal Component Analysis - Data Analysis

Here we hope to use Dimensionality Reduction for another purpose: Data Analysis. In order to do this we will utilize Dataset 2.

In [36]:

```
# Importing Necessary Packages
from sklearn.decomposition import PCA
```

In [37]:

```
# Utilizing Numerical Info for PCA and Removing all book_id that were not included in SVD
- too much sparcity / few reviews
books = df2[['title','book_id','text_reviews_count','average_rating','ratings_count']]
books = books[books["book_id"].isin(book_count.index)]
```

In [38]:

```
# Dropping the title to run PCA
books2 = books.drop(columns='title')
```

Classification Function:

At this point we have reduced the dimensionality of the dataset into 2 components, PCA 1 and PCA 2. However, to have a good graphical sense if there is any correlations here we want to have three different classifications for this data: good, average and poor. Because not enough information was provided; we will seperate these books into different 'grades'. In this case these grades will be added as new columns to compare to PCA clustering. The cutoff point for the classification was determined using the statistical information of each column. (Comparison of the mean and standard deviation)

In [39]:

In [39]:

```
def classification_rating (row):
    if row['average_rating'] >= 4:
        return 'Good'
    if row['average_rating'] >= 2.5:
        return 'Average'
    else:
        return 'Poor'
```

In [40]:

```
def classification_popularity (row):
    if row['average_rating'] >= 3000:
        return 'Good'
    if row['average_rating'] >= 1750:
        return 'Average'
    else:
        return 'Poor'
```

In [41]:

```
books['class_rating'] = books.apply (lambda row: classification_rating(row), axis=1)
books['class_popularity'] = books.apply (lambda row: classification_popularity(row), axis
=1)
books.reset_index(drop=True, inplace=True)
```

In [42]:

```
books.head(2)
```

Out[42]:

	title	book_id	text_reviews_count	average_rating	ratings_count	class_rating	class_popularity
0	A Game of Thrones: The Graphic Novel, Vol. 4	22889758	71	4.09	637	Good	Poor
1	Gambit, Volume 2: Tombstone Blues	17277799	34	3.75	370	Average	Poor

Principal Component Analysis Algorithm

In [43]:

```
# Principal Component Analysis - 2 components were chosen; to depict them graphically.
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(books2)
```

In [44]:

```
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
```

In [45]:

```
finalDf = pd.concat([principalDf, books.class_rating, books.class_popularity], axis = 1)
finalDf.head(2)
```

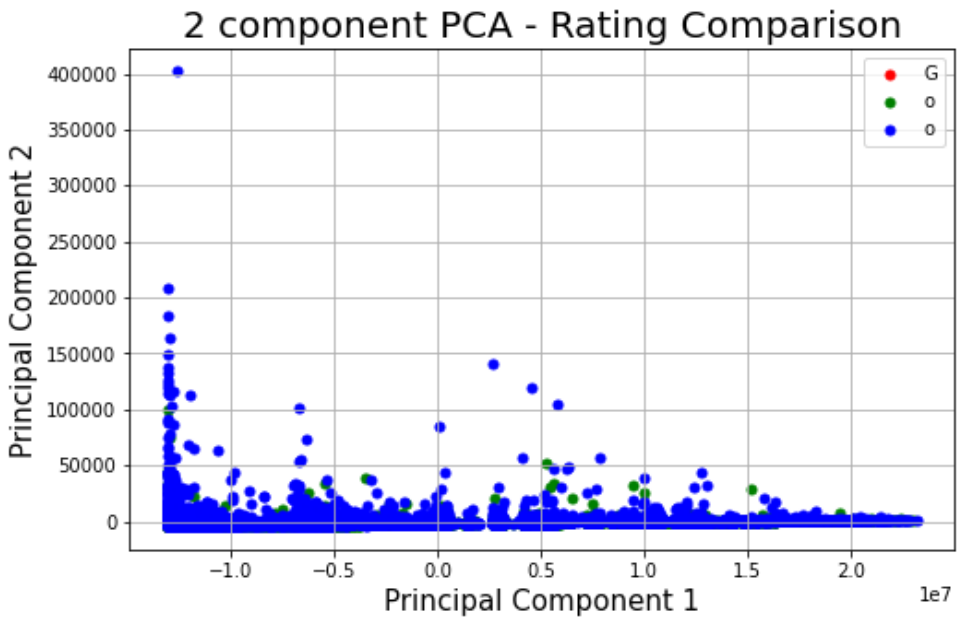
Out[45]:

	principal component 1	principal component 2	class_rating	class_popularity
0	9.813999e+06	-506.012427	Good	Poor
1	4.202040e+06	-1713.685700	Average	Poor

Graphical Representation - Data Analysis

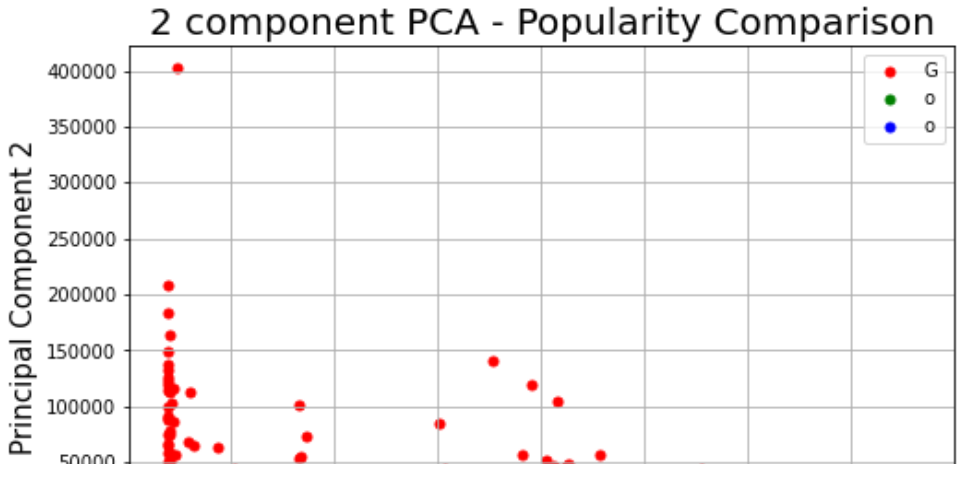
In [52]:

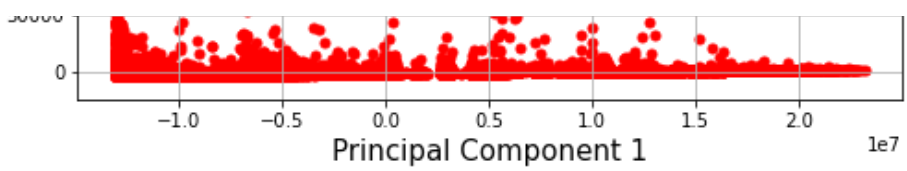
```
fig = plt.figure(figsize = (8,5))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA - Rating Comparison', fontsize = 20)
score = ['Poor', 'Average', 'Good']
colors = ['r', 'g', 'b']
for score, color in zip(score,colors):
    indicesToKeep = finalDf['class_rating'] == score
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 25)
ax.legend(score)
ax.grid()
```



In [54]:

```
fig = plt.figure(figsize = (8,5))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA - Popularity Comparison', fontsize = 20)
score = ['Poor', 'Average', 'Good']
colors = ['r', 'g', 'b']
for score, color in zip(score,colors):
    indicesToKeep = finalDf['class_popularity'] == score
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 25)
ax.legend(score)
ax.grid()
```





Explained Variance

In [48]:

```
print('The explained variance of the first Principal Component is -', pca.explained_variance_ratio_[0]*100)
```

```
The explained variance of the first Principal Component is - 99.9999179076565
```

In [49]:

```
print('The explained variance of the second Principal Component is -', pca.explained_variance_ratio_[1]*100)
```

```
The explained variance of the second Principal Component is - 8.204308898846555e-05
```

Reduced Sample Size

- **To ensure that there are no correlations, a random sample size of 50 was chosen to see if any correlations could be found:**

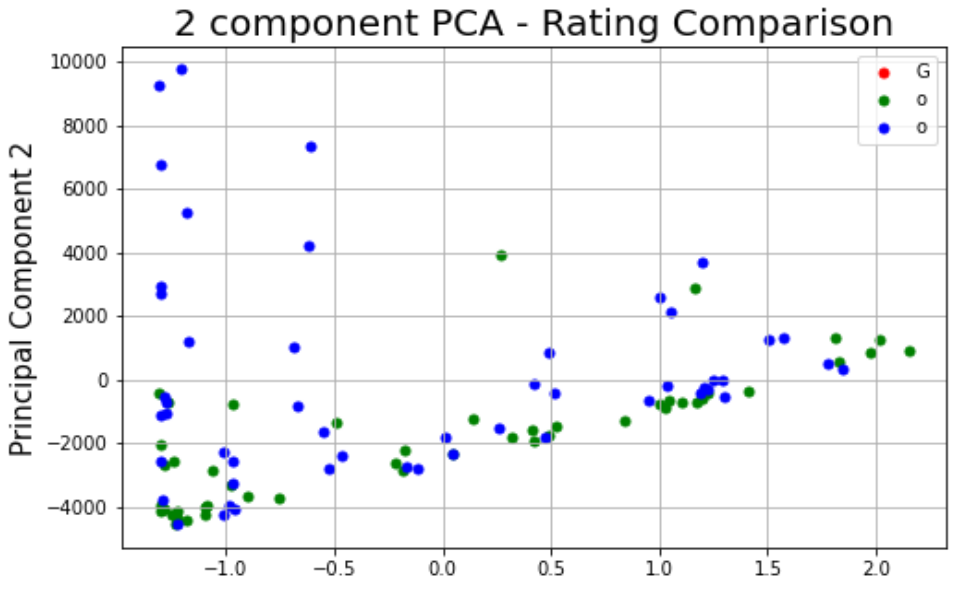
In [55]:

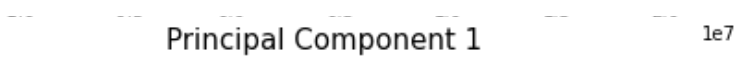
```
# Reduced Sample Size of 50

finalDf2 = finalDf.sample(n=100)

fig = plt.figure(figsize = (8,5))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA - Rating Comparison', fontsize = 20)
score = ['Poor', 'Average', 'Good']
colors = ['r', 'g', 'b']
for score, color in zip(score,colors):
    indicesToKeep = finalDf2['class_rating'] == score
    ax.scatter(finalDf2.loc[indicesToKeep, 'principal component 1']
               , finalDf2.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 25)

ax.legend(score)
ax.grid()
```





Observations:

- **Unfortunately, the plots above were not very conclusive. Although the first principal component accounts for almost 99% variance, there was not a clear distinction between data points above. Normally, PCA is a great way to visualize different classifications, however in this case this classification was self imposed by the user (me) and resulted in unsubstantial results.**
- **To improve this I would apply PCA to the overall GoodReads dataset, and I would expect the data to classify into different categories. However for the purposes of this report, I only worked on the Comic Book subsection as the overall dataset had over 13 million entries.**

Conclusion:

It's important to note, although various algorithms were used above, they were never truly optimized for real world scenarios. In each of the situations above I pointed out the limitations of the algorithm and what improvements could have been made to improve their performance. Instead, I hope to have shown how linear algebra is instrumental for the purposes of Machine Learning. However each algorithm has it's limitations, whether they be mathematical limitations or limitations imposed by the user due to the quality of the data. It is the responsibility of the data scientist to decide when to use which algorithm when

Data Set Citations:

- **Mengting Wan, Julian McAuley, "Item Recommendation on Monotonic Behavior Chains", in RecSys'18. [bibtex]**
- **Mengting Wan, Rishabh Misra, Ndapa Nakashole, Julian McAuley, "Fine-Grained Spoiler Detection from Large-Scale Review Corpora", in ACL'19. [bibtex]**