Project 4: Cluster Analysis, ANN, and Text Mining Report

CSC 177-01: Data Warehousing and Data Mining

Professor: Jagan Chidella

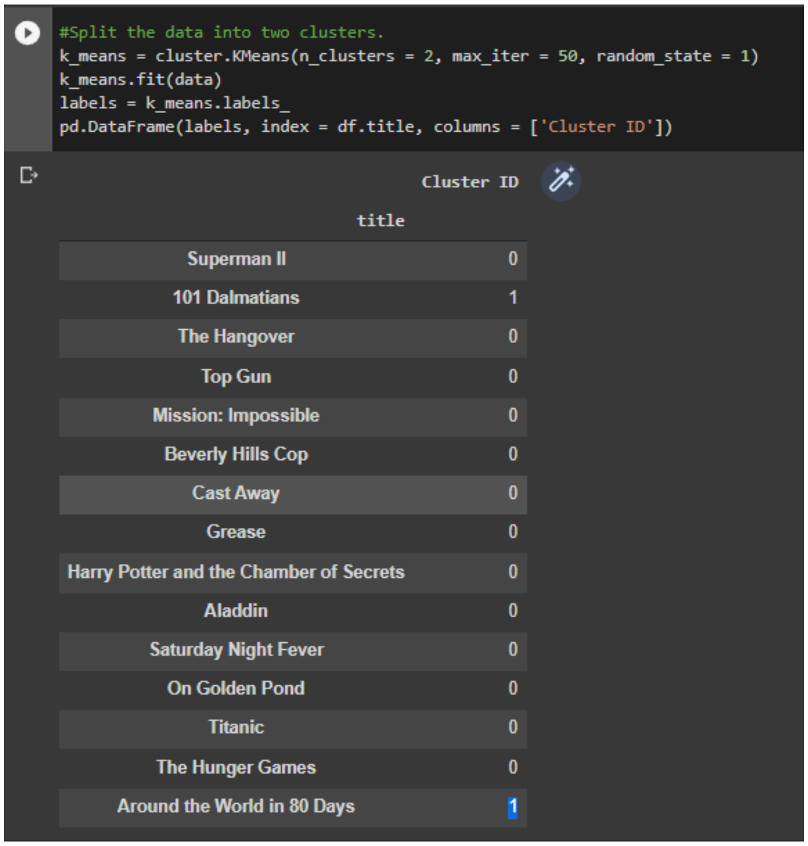
Group: Jason Phillips, Mohammad Ameri, Ryon Faroughi, Youser Alalusi,

Yusran Sadman

Cluster Analysis:

When performing clustering on large datasets, the goal is to make sense of large and complex datasets and partition them into a grouping that makes sense and is easy to understand. The goal for the particular dataset that we were tasked with analyzing was to group movies together that shared some common characteristics. The dataset contained many attributes so this process could have been done numerous ways (and it was). The attributes that we chose to group the movies together was High rated movies vs low rated movies. Since the dataset was so large, we took an additional step and used the top 200 category to select our movies from. Only movies that were included in the column would be selected to cluster together.

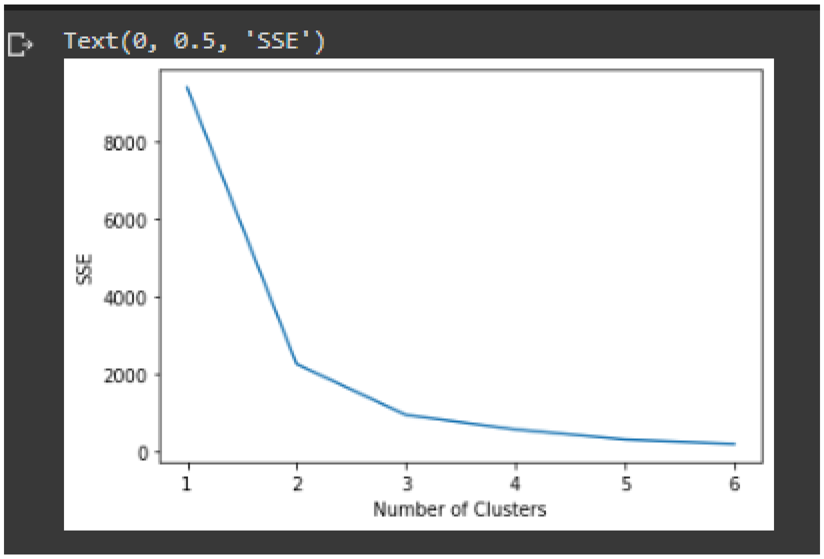
After processing the data and removing any rows/columns that were not going to be useful in clustering, the first action performed was the KMeans analysis.



```
#Split the data into two clusters.
k_means = cluster.KMeans(n_clusters = 2, max_iter = 50, random_state = 1)
k_means.fit(data)
labels = k_means.labels_
pd.DataFrame(labels, index = df.title, columns = ['Cluster ID'])
```

title	Cluster ID
Superman II	0
101 Dalmatians	1
The Hangover	0
Top Gun	0
Mission: Impossible	0
Beverly Hills Cop	0
Cast Away	0
Grease	0
Harry Potter and the Chamber of Secrets	0
Aladdin	0
Saturday Night Fever	0
On Golden Pond	0
Titanic	0
The Hunger Games	0
Around the World in 80 Days	1

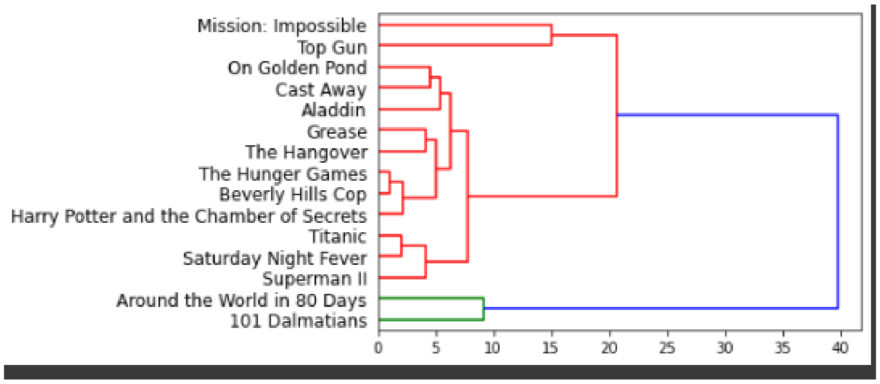
After creating a subset of the data that considers audience and critics scores/ratings, the code above separates the movies into 2 clusters. Cluster 0 is a cluster where the movie is either loved by either the audience or critics. Cluster 1 is a cluster where the movie is disliked by both critics and audiences.



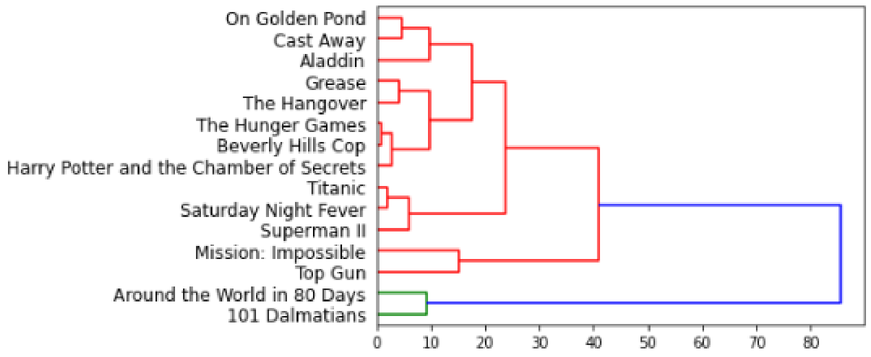
```
Text(0, 0.5, 'SSE')
```

After analyzing the graph above, a rough estimate of an ideal cluster would be around 2-3.

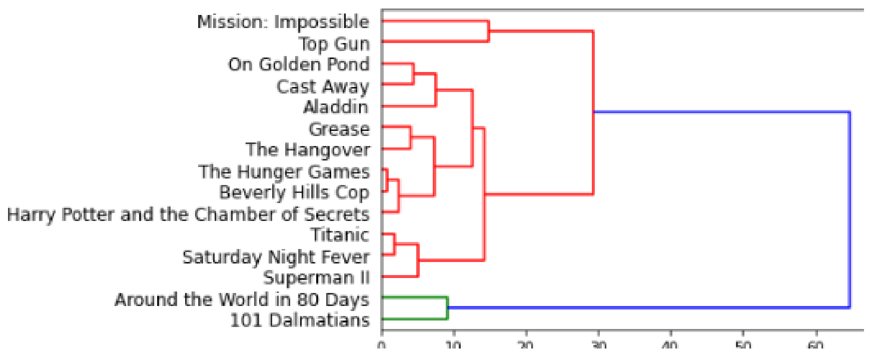
Below are the figures produced when applying multiple versions of hierarchical analysis on the dataset.



Single Link



Max Link



Average Link

There were many other categories that we tested, but struggled to find a correlation between attributes that would properly distinguish a defined class. We tried categorizing the movies by genre, but there isn't much other data besides mpaa rating that would help categorize a movie as a horror movie as opposed to a comedy movie. So after countless trials, we stuck with critics and audience ratings/scores to have a decent cluster of movies like or disliked by people.

Text Mining:

Vectorization is nothing but converting text into numeric form. As we know, regardless of how machines are capable and powerful, still they do not understand words and sentences in the same manner as humans do so, to make that possible for machines to understand texts first we need to convert them to a numerical representation.

TF-IDF can be broken down into two parts *TF (term frequency)* and *IDF (inverse document frequency).*

What is TF (term frequency)?

Term frequency works by looking at the frequency of a *particular term* you are concerned with relative to the document. There are multiple measures, or ways, of defining frequency:

· Number of times the word appears in a document (raw count).

· Term frequency adjusted for the length of the document (raw count of occurrences divided by the number of words in the document).

· Logarithmically scaled frequency (e.g., log (1 + raw count)).

· Boolean frequency (e.g., 1 if the term occurs, or 0 if the term does not occur, in the document).

What is IDF (inverse document frequency)?

Inverse document frequency looks at how common (or uncommon) a word is amongst the corpus. IDF is calculated as follows where ***t*** is the term (word) we are looking to measure the commonness of and ***N*** is the number of documents (d) in the corpus (D). The denominator is simply the number of documents in which the term, ***t***, appears in.

Scikit-Learn

- $\mathrm{IDF}(t) = \log \frac{1+n}{1+\mathrm{df}(t)} + 1$

Standard notation

- $\mathrm{IDF}(t) = \log \frac{n}{\mathrm{df}(t)}$

The reason we need IDF is to help correct for words like “of”, “as”, “the”, etc. since they appear frequently in an English corpus. Thus, by taking inverse document frequency, we can minimize the weighting of frequent terms while making infrequent terms have a higher impact.

Finally, IDFs can also be pulled from either a background corpus, which corrects for sampling bias, or the dataset being used in the experiment at hand.

Putting it together: TF-IDF

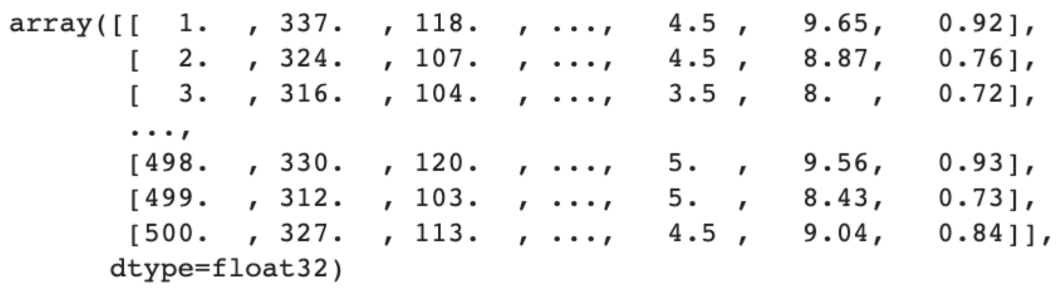
To summarize the key intuition motivating TF-IDF is the importance of a term is inversely related to its frequency across documents.TF gives us information on how often a term appears in a document and IDF gives us information about the relative rarity of a term in the collection of documents. By multiplying these values together, we can get our final TF-IDF value.

ANN:

ANN is (supposed to be) exactly like a human brain but simulated using software. Like a brain, a NN consist of various layers of "neurons" that work

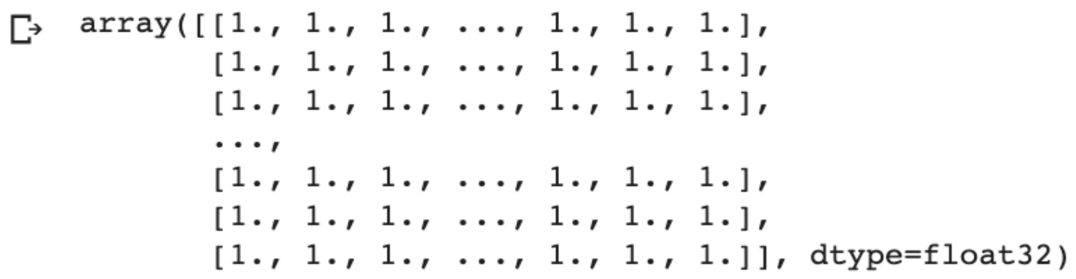
simultaneously to get the output. The task was to apply ANN on the admission dataset using code given in tutorial 6. We encoded the target attribute value to binary value.

The target attribute value to binary value after encoding



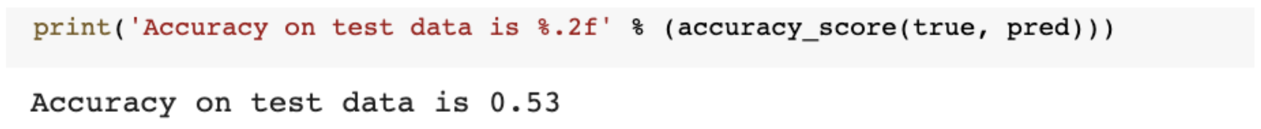
```
array([[  1.  , 337.  , 118.  , ...,   4.5 ,   9.65,   0.92],
       [  2.  , 324.  , 107.  , ...,   4.5 ,   8.87,   0.76],
       [  3.  , 316.  , 104.  , ...,   3.5 ,   8.  ,   0.72],
       ...,
       [498.  , 330.  , 120.  , ...,   5.  ,   9.56,   0.93],
       [499.  , 312.  , 103.  , ...,   5.  ,   8.43,   0.73],
       [500.  , 327.  , 113.  , ...,   4.5 ,   9.04,   0.84]],
      dtype=float32)
```

After applying ANN on admission dataset



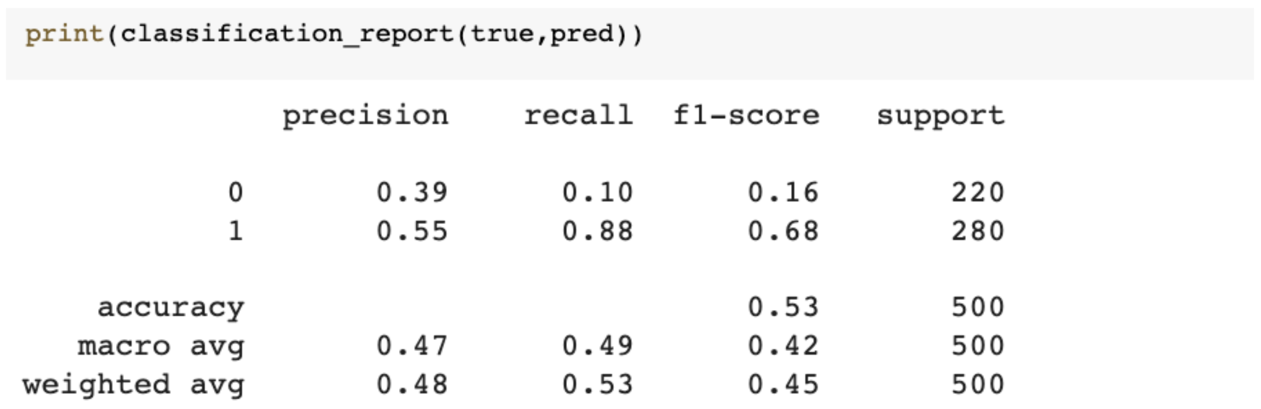
```
array([[1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       ...,
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.]], dtype=float32)
```

Yield an accuracy of 53%



```
print('Accuracy on test data is %.2f' % (accuracy_score(true, pred)))
```

```
Accuracy on test data is 0.53
```



```
print(classification_report(true,pred))
```

```
              precision    recall  f1-score   support

           0       0.39      0.10      0.16       220
           1       0.55      0.88      0.68       280

    accuracy                           0.53       500
   macro avg       0.47      0.49      0.42       500
weighted avg       0.48      0.53      0.45       500
```