République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École nationale Supérieure d'Informatique (ESI-SBA)

Mini Project Report

Topic : Movie Recommendation System

Prof : Dif

Students :

- **SAADI Ahmed**
- **TOUMI Adem**

Table des matières

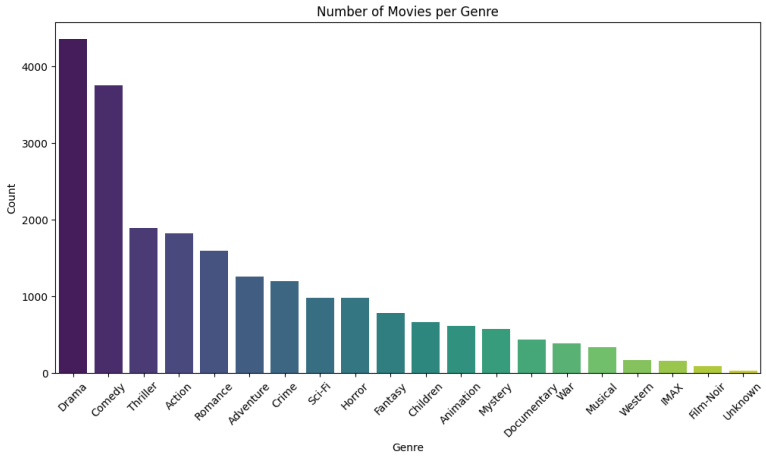
1.What's Recommendation System ?

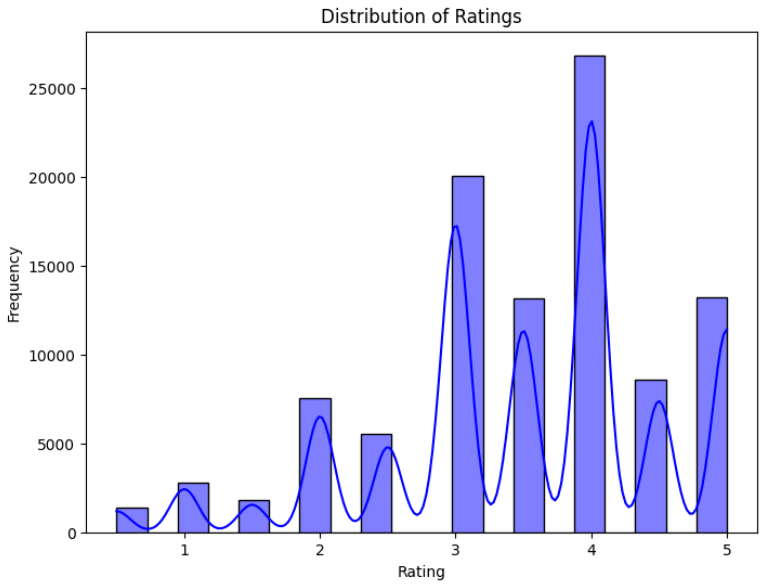
A recommendation system, powered by artificial intelligence and machine learning, leverages Big Data to deliver customized suggestions to users. These systems analyze various data points, such as browsing behavior, purchase history, and user demographics, to generate personalized recommendations. By understanding user preferences and behavior patterns through interactions like clicks, views, likes, and purchases, recommendation systems enhance user satisfaction by helping them discover relevant products or services. Their predictive algorithms are instrumental for businesses, as they effectively connect users with items that match their interests, ranging from movies and books to fitness programs and fashion.

2.Dataset Visualization :

In this section, we will visualize our datasets using the `Matplotlib` library. Matplotlib is a popular Python library for creating static, animated, and interactive visualizations. By leveraging its functionality, we can gain insights into the structure and patterns of the data.

1. First, we import the necessary libraries, including `matplotlib.pyplot` and `pandas` for data handling.
2. Next, we load the dataset into a DataFrame using `pandas` and begin by examining the structure of the data, such as checking for missing values and understanding the distribution of numerical features.
3. We can then create various types of plots, including histograms, scatter plots, and line graphs, depending on the type of data and the patterns we wish to explore. For example :
   — A histogram can be used to observe the distribution of a specific feature.
   — A scatter plot can reveal the relationship between two numerical variables.
   — A line graph can be used to visualize trends over time or other continuous variables.
4. Finally, we customize the visualizations by adjusting parameters such as labels, title, and axis limits to enhance the clarity and presentation of the plots.





3.Introduction to Association Rules :

Association rules analysis related algorithms, such as Apriori and FP-growth tree algorithms, can be defined as developed techniques that are used for transactional data types. These algorithms compute the strength of associations between data values and their relative combinations. For example, if a customer is 70% likely to buy milk, the algorithm can determine how likely the customer is to a lso buy bread. In this context, we focus on two main types of algorithms : the Apriori and the FP-growth. Data are structured as shown in Table **??**.

TID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

- Table 1. *Transactional data example, adapted from Tan and etl (2013) Association Analysis: basic concepts and algorithms in Introduction to data mining*

3.1.THEORETICAL PART :

3.1.1.Major Computation in Association Rules :

Association rules in general have the listed below major computations in the dataset :

Support : The percentage of the transactions that contains the items from the data set, which means how many times did the items occur in the dataset [**?**].

$$\text{support}(A \to C) = \text{support}(A \cup C) \quad (\text{range}[0,1])$$

Confidence : The probability that items on the left-hand side and right-hand side of ruleset are occurring together, with a higher confidence reflecting a higher likelihood of the items being purchased together based on the given rule [**?**].

$$\text{confidence}(A \to C) = \frac{\text{support}(A \cup C)}{\text{support}(A)} \quad (\text{range } [0,1])$$

Lift : Probability of all items set all occurring together in the rule. Lift value more than 1 reflects that the presence of the item will increase with the presence of the other items and its occurrence in the same transaction, so the lift summarizes the strength of the association rule as a link between the items of both sides [**?**].

$$\text{lift}(A \to C) = \text{confidence}(A \to C) \cdot \frac{1}{\text{support}(C)} \quad (\text{range}[0, \infty])$$

— Note that left-hand side and right-hand side of the rules are identified also as "antecedents" and "consequents".

3.1.2. STEPS TO COMPUTE ASSOCIATION RULES :

Generate Frequent Itemsets :

Two Algorithms (Apriori, and FPGrowth) are going to be discussed to generate frequent itemsets, which are going to be used later on to create association rules.

Apriori Algorithm :

A pseudocode in Figure **??** implements the frequent itemsets generation of the Apriori algorithm. In this, C_k adopts a set of candidate k-itemsets, and the frequent set of itemsets is represented by F_k. Apriori follows the below steps (Tan et al., 2013) :

1. The algorithm initially makes a single pass over the dataset to determine the support of each item. Upon completion of this step, the set of all frequent 1-itemsets K is obtained.
2. The algorithm generates new candidate k-itemsets and prunes unnecessary candidates that are guaranteed to be infrequent, given the frequent $(k-1)$ itemsets found in the previous iteration.

3. The algorithm makes an additional pass to count the support of the generated candidates. The subset function is used to determine all the candidate itemsets in C_k that are contained in each transaction t.
4. After counting support, the algorithm eliminates all candidate itemsets whose support counts are less than the minimum support, N, which is chosen by the user.
5. The algorithm terminates when there are no new frequent itemsets generated, i.e., $F_k = \emptyset$.

Algorithm 5.1 Frequent itemset generation of the *Apriori* algorithm.

1: $k = 1$.
2: $F_k = \{ i \mid i \in I \wedge \sigma(\{i\}) \geq N \times minsup\}$. {Find all frequent 1-itemsets}
3: **repeat**
4: $k = k + 1$.
5: C_k = candidate-gen(F_{k-1}). {Generate candidate itemsets.}
6: C_k = candidate-prune(C_k, F_{k-1}). {Prune candidate itemsets.}
7: **for** each transaction $t \in T$ **do**
8: C_t = subset(C_k, t). {Identify all candidates that belong to t.}
9: **for** each candidate itemset $c \in C_t$ **do**
10: $\sigma(c) = \sigma(c) + 1$. {Increment support count.}
11: **end for**
12: **end for**
13: $F_k = \{ c \mid c \in C_k \wedge \sigma(c) \geq N \times minsup\}$. {Extract the frequent k-itemsets.}
14: **until** $F_k = \emptyset$
15: Result = $\bigcup F_k$.

- Figure 2 *Apriori algorithm Pseudocode, adapted from Tan and etl (2013) Association Analysis: basic concepts and algorithms in Introduction to data mining*

FP-Growth Algorithm :

The FP-Growth algorithm does not create candidate sets in the same fashion as Apriori in terms of multiple passes over the entire dataset. It requires only two scans over the dataset to extract frequent itemsets, making it faster and less computationally expensive than Apriori.

1. The algorithm scans the whole dataset once to calculate each item's support count, ignores the infrequent items, and sorts the generated item set in descending order.
2. The algorithm constructs the FP-tree by making a second pass over the dataset. After reading the first transaction, it creates a node for each item in the transaction and generates a path to encode their relation. Every node along the path has a frequency count of 1.
3. This process continues until all the generated transactions are mapped onto their relative paths constructed by the FP-Tree.

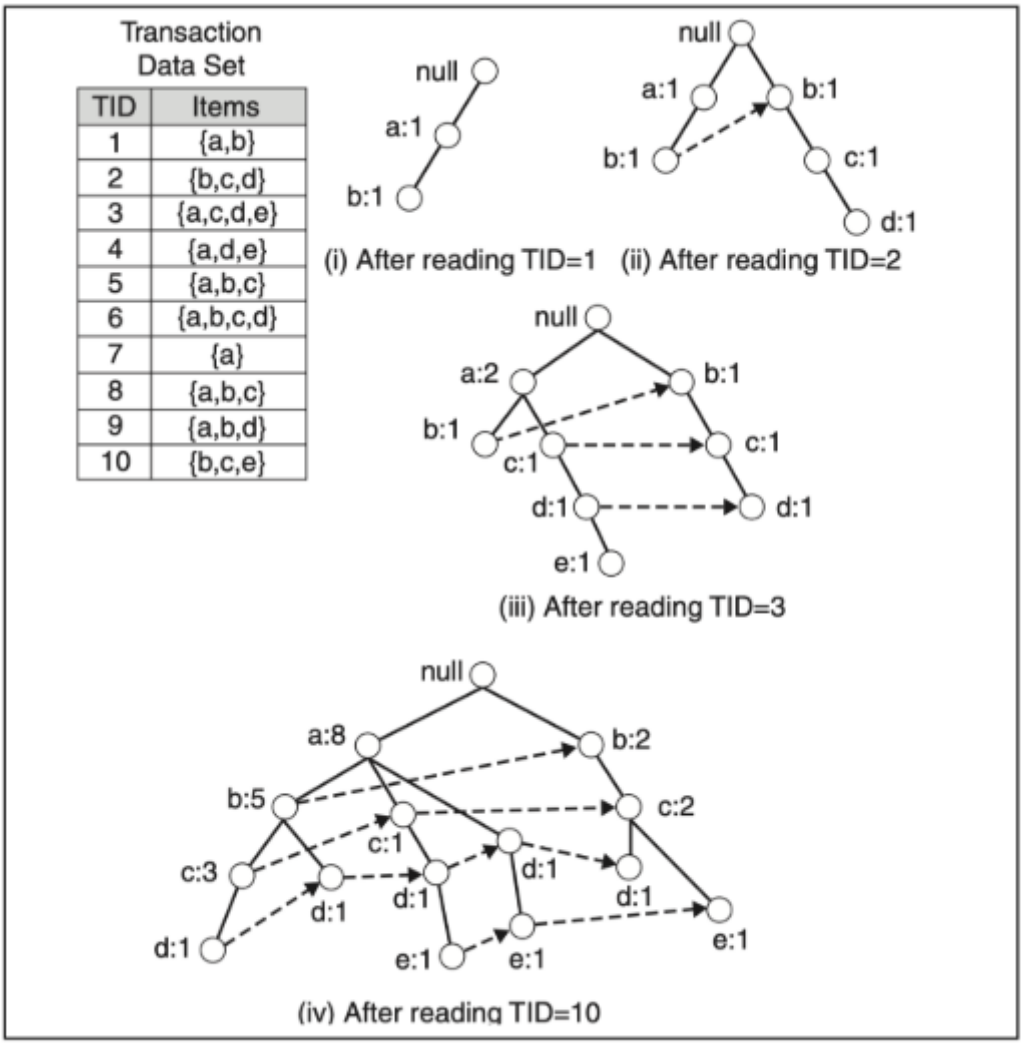
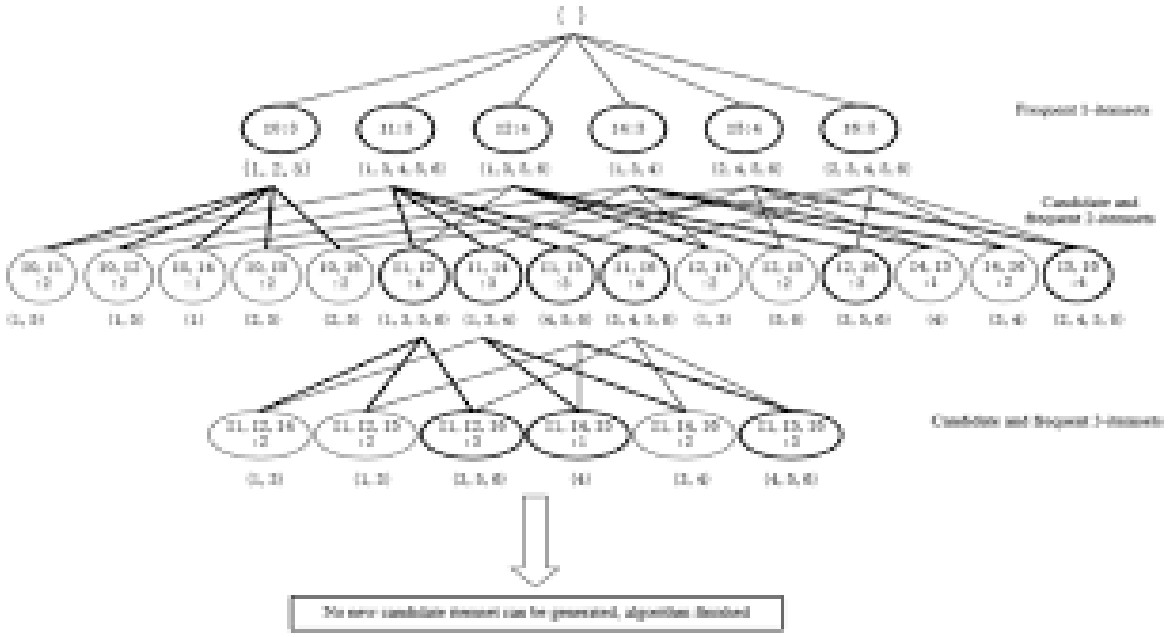


Figure 3. Fp-Growth algorithm generation, adapted from Tan and etl (2013) Association Analysis: basic concepts and algorithms in Introduction to data mining.

Eclat Algorithm :

The ECLAT algorithm stands for Equivalence Class Clustering and bottom-up Lattice Traversal. It is one of the popular methods of Association Rule mining. It is a more efficient and scalable version of the Apriori algorithm. While the Apriori algorithm works in a horizontal sense, imitating the Breadth-First Search of a graph, the ECLAT algorithm works in a vertical manner, just like the Depth-First Search of a graph. This vertical approach of the ECLAT algorithm makes it a faster algorithm than the Apriori algorithm.

1. The basic idea is to use Transaction ID Sets (tidsets) intersections to compute the support value of a candidate and to avoid the generation of subsets that do not exist in the prefix tree.
2. In the first call of the function, all single items are used along with their tidsets.
3. Then, the function is called recursively, and in each recursive call, each item-tidset pair is verified and combined with other item-tidset pairs.
4. This process continues until no candidate item-tidset pairs can be combined.



Drawbacks in association rules :

Due to the iterative process of rules generation in association rules mining where every item in the dataset is compared against each item and so on, the increase in items and transaction can make the computation very expensive, (Tan and etl 2013) listed some limitations that can be solved by decrease the dimensionality in datasets.and reduce the number of item sets in terms of columns and rows that need to be computed, that can be reduced by setting threshold to compute items sets that pases the threshold only.

2.PRACTICAL PART :

2.1. Dataset :

This dataset (**ml-latest**) describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service. It contains **27,753,444 ratings** and **1,108,997 tag applications** across **58,098 movies**. These data were created by **283,228 users** between January 09, 1995, and September 26, 2018. This dataset was generated on September 26, 2018.

More information about the data and description can be found in the following link : `http://files.grouplens.org/datasets/movielens/ml-latest-README.html`.

2.2. Selected Data :

From the directory of the uncompressed file, we observe that there are a total of **5 datasets**. We will select and examine the dataset that contains the necessary data required for the implementation of association rules.

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

The selected dataset should resemble the structure shown in **Table 1**.

```python
data_movies = pd.read_csv('ml-latest-small/movies.csv')
data_tages = pd.read_csv('ml-latest-small/tags.csv')
data_ratings = pd.read_csv('ml-latest-small/ratings.csv')
```

Movies Data : This is the only dataset that contains the titile of the movies and each movie unique key.

```python
data_movies.head()
data_movies.movieId.value_counts(),print('Number of duplicated unique ids are: ',data_movies.movieId.duplicated().sum())
```

Movies Tags Data : This dataset contains the added tags to the movie plus the user ID, which is valuable to be considered as an index to each transaction. On the other hand, the dataset has fewer movie IDs than the rating dataset, which made us select it instead.

```
data_tages
✓ 0.0s
```

	userId	movieId	tag	timestamp
0	2	60756	funny	1445714994
1	2	60756	Highly quotable	1445714996
2	2	60756	will ferrell	1445714992
3	2	89774	Boxing story	1445715207
4	2	89774	MMA	1445715200
...	...	...	...	...
3678	606	7382	for katie	1171234019
3679	606	7936	austere	1173392334
3680	610	3265	gun fu	1493843984
3681	610	3265	heroic bloodshed	1493843978
3682	610	168248	Heroic Bloodshed	1493844270

3683 rows × 4 columns

Movies Rating Data : Movies rating data contain more that 9k movies ids which can be usefull to be merged with the movies title dataset.

```
data_ratings
```
✓ 0.0s

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
...	...	...	...	...
100831	610	166534	4.0	1493848402
100832	610	168248	5.0	1493850091
100833	610	168250	5.0	1494273047
100834	610	168252	5.0	1493846352
100835	610	170875	3.0	1493846415

100836 rows × 4 columns

```
data_ratings.movieId.value_counts()
```
✓ 0.0s

```
movieId
356       329
318       317
296       307
593       279
2571      278
         ...
138966      1
140162      1
140737      1
140816      1
121169      1
Name: count, Length: 9724, dtype: int64
```

2.3. Merging Datasets :

In this case, *inner join* is selected as it ensures that only the data where `movieId` labels exist in both datasets is included.

More details about merging can be found in the Pandas documentation.

```python
# --- Step 2: Data Preprocessing ---
# Merge the datasets movie with tags using movieId
merge = data_movies.merge(data_tages,on = 'movieId',how = 'inner')
# Drop unnecessary columns
merge.drop(columns=['tag','timestamp','genres'],inplace=True)
# Group titles by userId
merge_list = merge.groupby(by = ["userId"])["title"].apply(list).reset_index()
```

2.4. Data Transformation :

Here, data is transformed into a binary input format to be accepted by the algorithm. Several methods are available in various packages, such as :

2.5. Generating Frequent Itemsets :

This section implements the use of the itemsets generation according to the selected parameters, `min_support=0.01, max_len=2`, as a threshold to optimize the algorithm by limiting memory usage. A support value of 1% and a maximum length of two items in each row are selected.

Note : The resulting frequent items are Python frozensets, which cannot be directly selected using pandas methods such as `pd.loc[]`.

Appriori :

```python
# --- Step 3: Applying Apriori Algorithm ---
from mlxtend.frequent_patterns import apriori
# Measure the time taken to run the algorithm
%time
# Find frequent itemsets
apriori_frequent_itemsets = apriori(df, min_support=0.01,use_colnames=True,max_len=2)

# Display frequent itemsets
apriori_frequent_itemsets['itemsets'].apply(lambda x: len(x)).value_counts()
```

```
CPU times: user 10 µs, sys: 0 ns, total: 10 µs
Wall time: 18.1 µs

itemsets
2    774986
1      1572
Name: count, dtype: int64
```

```python
from mlxtend.frequent_patterns import association_rules


# Step 1: Generate association rules
rules = association_rules(apriori_frequent_itemsets, metric="confidence", min_threshold=0.01)

# Function to get recommendations for a specific movie
def get_recommendations(movie_name, rules, n=10):
    # Step 2: Filter rules where the movie is in the antecedent
    movie_rules = rules[rules['antecedents'].apply(lambda x: movie_name in x)]

    # Step 3: Sort rules by confidence
    movie_rules = movie_rules.sort_values('confidence', ascending=False)

    # Step 4: Extract the consequents as recommendations
    recommendations = []
    for _, rule in movie_rules.iterrows():
        recommendations.extend(list(rule['consequents']))
        if len(recommendations) >= n:
            break

    return recommendations[:n]

# Example usage
movie_name = "Inception (2010)"  # Replace with the movie you're interested in
recommendations = get_recommendations(movie_name, rules)
print(f"Top 10 recommendations for {movie_name}:")
for i, movie in enumerate(recommendations, 1):
    print(f"{i}. {movie}")
```

Fpgrowth :

```python
from mlxtend.frequent_patterns import fpgrowth
%time
fpgrowth_frequent_itemsets = fpgrowth(df, min_support=0.01, use_colnames=True,max_len=2)
fpgrowth_frequent_itemsets.head()
```

✓ 2m 57.7s Python

```
CPU times: user 3 µs, sys: 1 µs, total: 4 µs
Wall time: 7.15 µs
```

	support	itemsets
0	0.453202	(Forrest Gump (1994))
1	0.394089	(Matrix, The (1999))
2	0.392447	(Silence of the Lambs, The (1991))
3	0.361248	(Star Wars: Episode IV - A New Hope (1977))
4	0.321839	(Fight Club (1999))

2.6. Analysis to the Results :

Let's create a new feature that can be used for further analysis :

```
fpgrowth_frequent_itemsets['length'] = fpgrowth_frequent_itemsets['itemsets'].apply(lambda x: len(x))
fpgrowth_frequent_itemsets
```

	support	itemsets	length
0	0.453202	(Forrest Gump (1994))	1
1	0.394089	(Matrix, The (1999))	1
2	0.392447	(Silence of the Lambs, The (1991))	1
3	0.361248	(Star Wars: Episode IV - A New Hope (1977))	1
4	0.321839	(Fight Club (1999))	1
...	...	...	...
215428	0.011494	(Cape Fear (1962), Manchurian Candidate, The (...	2
215429	0.011494	(Graduate, The (1967), Cape Fear (1962))	2
215430	0.011494	(Sicario (2015), Dark Knight, The (2008))	2
215431	0.011494	(Sicario (2015), Usual Suspects, The (1995))	2
215432	0.011494	(Sicario (2015), Fight Club (1999))	2

215433 rows × 3 columns

.Apply some basic analysis that can be used to generate some interesting insights :
.Filter by columns based on numerical conditions :

```
fpgrowth_frequent_itemsets[(fpgrowth_frequent_itemsets['length'] > 1)
                           & (fpgrowth_frequent_itemsets['support'] > 0.06)].head()
```

	support	itemsets	length
1889	0.310345	(Shawshank Redemption, The (1994), Forrest Gum...	2
1890	0.243021	(Forrest Gump (1994), Matrix, The (1999))	2
1891	0.241379	(Shawshank Redemption, The (1994), Matrix, The...	2
1892	0.234811	(Pulp Fiction (1994), Matrix, The (1999))	2
1893	0.238095	(Silence of the Lambs, The (1991), Forrest Gum...	2

2.7. Association Rules :

The `mlxtend` module computes additional measures such as :

— $leverage(A \rightarrow C) = support(A \rightarrow C) - support(A) \times support(C)$
→ range : $[-1, 1]$

— $conviction = \frac{[1-support(C)]}{[1-confidence(A \rightarrow C)]}$
→ range : $[0, \infty]$

Leverage computes the difference between the observed frequency of A and C appearing together and the frequency that would be expected if A and C were independent. A leverage value of 0 indicates independence.

A high conviction value means that the consequent is highly dependent on the antecedent. For instance, in the case of a perfect confidence score, the denominator becomes 0 (due to $1 - 1$), for which the conviction score is defined as ∞. Similar to lift, if items are independent, the conviction is 1. **.Here we are selecting the measure 'Lift' as a score evaluation method :**

```
%%time
from mlxtend.frequent_patterns import association_rules
rules = association_rules(fpgrowth_frequent_itemsets,metric="lift",min_threshold=0.01)
rules
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(Shawshank Redemption, The (1994))	(Forrest Gump (1994))	0.474548	0.453202	0.310345	0.653979	1.443019	0.095279	1.580246	0.584276
1	(Forrest Gump (1994))	(Shawshank Redemption, The (1994))	0.453202	0.474548	0.310345	0.684783	1.443019	0.095279	1.666950	0.561466
2	(Forrest Gump (1994))	(Matrix, The (1999))	0.453202	0.394089	0.243021	0.536232	1.360688	0.064420	1.306496	0.484782
3	(Matrix, The (1999))	(Forrest Gump (1994))	0.394089	0.453202	0.243021	0.616667	1.360688	0.064420	1.426430	0.437486
4	(Shawshank Redemption, The (1994))	(Matrix, The (1999))	0.474548	0.394089	0.241379	0.508651	1.290701	0.054365	1.233158	0.428635
...	...	...	...	...	...	...	...	...	...	...
427083	(Dark Knight, The (2008))	(Sicario (2015))	0.224959	0.013136	0.011494	0.051095	3.889599	0.008539	1.040003	0.958535

.A less messy way to group the top 10 related movies to the selected one. We can see here the selected movie inception has top values of lift on related movie due to the same director in them (Dunkirk) :

```
rules[rules["antecedents"].apply(lambda x: "Superman (1978)" in str(x))].groupby(
    ['antecedents', 'consequents'])[['lift']].max().sort_values(ascending=False,by='lift').head(10)
```

		lift
antecedents	consequents	
(Superman (1978))	(Mad Max Beyond Thunderdome (1985))	10.410256
	(Superman II (1980))	10.104072
	(Star Trek IV: The Voyage Home (1986))	8.408284
	(Conan the Barbarian (1982))	8.328205
	(Rocketeer, The (1991))	8.328205
	(RoboCop (1987))	7.596674
	(Star Trek III: The Search for Spock (1984))	7.435897
	(Halloween (1978))	7.287179
	(Karate Kid, The (1984))	7.157051
	(X-Files: Fight the Future, The (1998))	7.157051

```
rules[rules["antecedents"].apply(lambda x: "Superman (1978)" in str(x))].groupby(
    ['antecedents', 'consequents'])[['confidence']].max().sort_values(ascending=False,
                                                                      by='confidence').head(10).plot(kind='bar').invert_xaxis()
plt.title('Top movies that are likley to be watched with Superman (1978)');
```