

שם הקורס: ניהול מידע מבוזר (096224)

מגישים: נימרוד סולומון (ת.ז 206574733) ומתן שילוני (ת.ז 208634469)

מספר תרגיל הבית: פרויקט חלק א', חלק 3

תאריך הגשה: 09.06.2022

חלק ג' – Design

במסמך זה נציע תיכון למסד הנתונים בהתבסס על התובנות אליהן הגענו בחלק הקודם של הפרויקט. כפי שצינו בחלק הקודם, התובנות שלנו היו מבוססות ומונחות בעיקר על הערים, מתוך תקווה למצוא תובנות מעניינות שיהיו רלוונטיות לתכנון החלוקה האופקית והאנכית של הטבלאות במסד. בחלק הקודם מצאנו לא מעט תובנות מעניינות על המסד, שברובן נשתמש לתיכון.

בשורות הבאות נציג את התיכון למסד, טבלה-טבלה. קצרה היריעה מלהרחיב על כל עמודה ושורה בכל טבלה, לכן נתייחס לחלוקה בקווים מעט יותר כלליים, בדגשים, וככל שלא נציין פרט מסוים, ניתן יהיה להניח שהוא אינו קריטי לתיכון בהתבסס על התובנות שמצאנו בחלק הקודם.

הטבלה movies :

חלוקה אופקית - דגשים

1. ז'אנרים

א. **תל אביב:** נשים לב כי מהתובנות שמצאנו עולה כי הז'אנרים

Action, Drama, Science Fiction מהווים חלק הארי מבין כלל השאילתות בהן

מופיעה התייחסות לתל אביב. כלומר, נוכל להשמיט ז'אנרים שאינם נמנים על הקבוצה

לעיל לטובת הפרגמנטציה באתר *Tel Aviv*.

ב. **ירושלים:** ניתן לראות מהתובנות כי בעיר זו הז'אנרים מאוזנים יחסית זה לזה ואין רוב

מובהק לאף קבוצה מצומצמת (בגודל שכדאי לעשות עבורו פרגמנטציה).

ג. **חיפה:** מהתובנות שמצאנו בחלק הקודם עולה כי הז'אנר *Drama* מחזיק ברוב מובהק

בהופעות בשאילתות מעיר זו. על כן, באתר *Haifa* נוכל להשמיט סרטים שאינם מז'אנר

זה כלל (כלומר מספיק שאחד הז'אנרים של סרט הוא *Drama* על מנת שהוא כן יישמר

באתר זה).

ד. **אילת:** ניתן לראות מהתובנות כי בעיר זו הז'אנרים מאוזנים יחסית זה לזה ואין רוב מובהק

לאף קבוצה מצומצמת (בגודל שכדאי לעשות עבורו פרגמנטציה).

ה. **טבריה:** נשים לב כי מהתובנות שמצאנו עולה כי הז'אנרים *Drama,*

Documentary, Family מהווים חלק הארי מבין כלל השאילתות בהן מופיעה

התייחסות לטבריה. כלומר, נוכל להשמיט ז'אנרים שאינם נמנים על הקבוצה לעיל לטובת

הפרגמנטציה באתר *Kibutz Gesher*.

ו. את הרשומות המכילות סרטים מז'אנרים שלא צוינו עד כה נשמור באתר אקראי, לצורך

העניין - *Tel Aviv* (בהמשך נדאג לשמור גם באתרים אחרים לצורך איזון), ובלבד שהן גם

לא עונות על אף תנאי שיצוין בהמשך (אחרת נשמור אותו במקום אחר בהתאם לתנאי).

2. שפה

- א. ניתן להסיק מהתובנות של ניתוח הנתונים כי בערים תל אביב וירושלים יש עניין רב בסרטים באנגלית ובעברית, בעוד שאר השפות אינן זוכות להתעניינות משמעותית בערים אלו/בערים האחרות.
- ב. מהתבוננות קצרה בדאטה קל להתרשם שרוב הסרטים הם, בין היתר, גם באנגלית.
- ג. לכן, את כל הסרטים שאינם בעברית לא נפריד ע"פ מאפיין השפה.
- ד. הסרטים בעברית ימוקמו בתל אביב וירושלים בצורה משוכפלת, לאור ההתעניינות שצוינה לעיל.

3. מדינה

- א. מכיוון שהסרטים הישראליים פופולריים מאוד, ואך ורק בירושלים ובתל אביב, נמקם את הרשומות בהן מופיעים סרטים ישראליים בצורה משוכפלת בתל אביב ובירושלים.
- ב. בערים אחרות אין התעניינות במאפיין זה.
- ג. לכן, רשומות שאינן עוסקות בסרטים ישראליים, נשמור באתר אקראי, לצורך העניין – *Haifa*, ובלבד שהן גם לא עונות על אף תנאי שצוין לעיל או יצוין בהמשך (אחרת נשמור אותו במקום אחר בהתאם לתנאי).

4. חברות הפקה

- א. נשים את כל הסרטים של חברות ההפקה: Pixar Animation Studios, Walt Disney Pictures, Warner Bros באילת, שכן חברות ההפקה מאוד פופולריות באילת ופופולריות כמו כל חברת הפקה אחרת בשאילתות הנוגעות לערים אחרות.
- ד. את הרשומות הנותרות נשמור באתר אקראי, לצורך העניין – *Kibutz Gesher* ובלבד שהן גם לא עונות על אף תנאי שצוין לעיל או יצוין בהמשך (אחרת נשמור אותו במקום אחר בהתאם לתנאי).

5. שנה

- א. מהתבוננות קצרה על הדאטה ניתן לראות כי ישנם סרטים שהופקו לפני שנת 1990, ואולם כפי שראינו בחלק הקודם, השאילתות מתייחסות רק לסרטים משנת 1990 ואילך. לכן, נמקם רשומות של סרטים שיצאו לפני 1990 באחד האתרים באקראי שכן אף שאילתה לא ניגשת אליהם. לצורך העניין – *Eilat*, ובלבד שהן גם לא עונות על אף תנאי שצוין לעיל או יצוין בהמשך (אחרת נשמור אותו במקום אחר בהתאם לתנאי).

חלוקה אנכית - דגשים:

1. את העמודות: title, tagline, revenue, overview, נשים באחד האתרים באקראי שכן אף שאילתה לא ניגשת אליהן. את עמודת מפתח הטבלה movie_id נשים בכל אחד מהאתרים.
2. מימשנו את האלגוריתם¹ לביצוע פרגמנטציה אנכית שנלמד בכיתה והרצנו אותו על רלציה זו, בתקווה למצוא קורלציות חזקות בין עמודותיה השונות.

¹ ראה נספח מצורף בסוף המסמך

```
vertical fragmentation: [[3], [5, 4, 0, 2, 1]]  
max res = -0.300117708900000006
```

- המספר 3 מייצג את העמודה *country*, והיא מהווה לצורך העניין פרגמנט נפרד משאר העמודות - *genres, cities, production company, release date, language*.
4. לאחר הרצת האלגוריתם, הסקנו כי אין קורלציה משמעותית בין העמודות, וכי אין בידינו נתונים שעל בסיסם ניתן לחלק אנכית.
5. לכן, לא נפריד את הרלציה לפי העמודות אליהן יש התייחסות בשאילתות המשתמשים.
6. כלומר, בכל אתר תהיינה העמודות ששאילתות ניגשות אליהן, ואילו את העמודות הנותרות נשמור באחד האתרים באקראי, לצורך העניין *Tel Aviv*.

הטבלה credits :

חלוקה אופקית - דגשים :

1. שחקנים

- א. ניתן להסיק מהתובנות של ניתוח הנתונים כי בערים תל אביב וירושלים יש עניין בזהות השחקן המשחק בסרט, וכמעט אך ורק בשחקנים *Tom Hanks, Johnny Depp, Brad Pitt*.
- ב. בערים אחרות יש שאילתות מעטות מאוד בהן השדה *actors* אינו ריק. על כן, הסקנו כי למשתמשים בערים אלו זהות השחקן פחות חשובה.
- ג. לכן, בערים תל אביב וירושלים נשמור רשומות בהן מופיעים אחד מהשחקנים לעיל.
- ד. את הרשומות הנותרות נשמור באתר אקראי, לצורך העניין - *Jerusalem*.

חלוקה אנכית - דגשים :

1. ברלציה זו קיימות מעט מדי עמודות כדי לבצע חלוקה אנכית שתייעל את מסד הנתונים (שתי עמודות למעט המפתח).

הטבלאות tickets ו-users :

הטבלאות *tickets* ו-*users* שונות מהטבלאות *movies* ו-*credits* : הן קטנות יותר מבחינת כמות המידע (ניתן אף לחזות בכך אם נתבונן בגודל הקבצים של הטבלאות שסופקו) ולמעט הערים השונות שמופיעות בכל רשומה (ב-*tickets* תחת השדה *city* וב-*users* תחת השדה *user_location*) לא ניתן למצוא פרדיקטים שאילתות מפרידות רשומות בטבלאות הללו על פיהם. לכן נחלק את הטבלה חלוקה אופקית בלבד, ונשים בכל אתר את הרשומות שבהן מופיעה העיר המתאימה לאתר.

הטבלה queries :

מההבחנה הפשוטה כי אף שאילתה לא ניגשת לרלציה זו, נוכל למקם את כולה או חלקים ממנה בכל אתר שנרצה.

```
# Section 3 - Design
# here we will work on vertical fragmentations only.
# for horizontal fragmentation look section's pdf.

import sys
import numpy as np
from collections import Counter

# statistics about the nature of queries
# from here and on, we will consider these as generic queries with the calculated query-access
def row_counter(my_array):
    # count identical rows in matrix
    list_of_tups = [tuple(ele) for ele in my_array]
    return Counter(list_of_tups)

# build use matrix for queries and shrink it
shape = (queries_df.count(), len(queries_df.columns))
queries_use = np.zeros(shape)
queries_itr = queries_df.rdd.toLocalIterator()
for i, row in enumerate(queries_itr):
    for j, col in enumerate(queries_df.columns):
        if type(row[col]) != list:
            queries_use[i, j] = 1
        else:
            queries_use[i, j] = 0 if row[col][0] == "" else 1
```

```
access = row_counter(queries_use)
```

```
for key, value in access.items():
    print(f"{key} : {value}")
```

```
# user_id | genres | lang | actors | director | cities | country | from_realese_date | production_company
```

```
(1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0) : 10046
(1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0) : 12416
(1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0) : 31305
(1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0) : 965
(1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0) : 21205
(1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0) : 12620
(1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0) : 9983
(1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0) : 1308
(1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0) : 64
(1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0) : 3
(1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0) : 3
(1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0) : 70
(1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0) : 5
(1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0) : 4
(1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0) : 1
(1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0) : 2
```

```

[25] n_queries = 8
     n_attributes = 2
     #attribute usage matrix on movies

     # actors | director
     aum = [ [0.0, 1.0],
             [0.0, 0.0],
             [1.0, 0.0],
             [0.0, 1.0],
             [0.0, 0.0],
             [1.0, 0.0],
             [1.0, 1.0],
             [1.0, 1.0]]

     #number of sites
     n_sites = 5

     #access matrix
     acc = [[10046/200000, 10046/200000,10046/200000,10046/200000,10046/200000, 10046/200000],
            [12412/600000, 12416/200000,12416/200000,12416/200000,12416/200000, 12416/200000],
            [31305/200000, 31305/200000,31305/200000,31305/200000,31305/200000, 31305/200000],
            [965/200000, 965/200000,965/200000,965/200000,965/200000, 965/200000],
            [21205/200000, 21205/200000,21205/200000,21205/200000,21205/200000, 21205/200000],
            [12620/200000, 12620/200000,12620/200000,12620/200000,12620/200000, 12620/200000],
            [9983/200000, 9983/200000,9983/200000,9983/200000,9983/200000, 9983/200000],
            [1308/200000, 1308/200000,1308/200000,1308/200000,1308/200000, 1308/200000],
            ]

```

```

✓ [26] #prefix sum for each query
     pre = [0 for i in range(n_queries)]
     for i in range(n_queries):
         for j in range(n_sites):
             pre[i] = pre[i]+acc[i][j]

     #attribute affinity matrix
     aam = [[0 for i in range(n_attributes)] for j in range(n_attributes)]

     #calculation of the aam
     for i in range(n_attributes):
         for j in range(n_attributes):
             for q in range(n_queries):
                 if aum[q][i]==1 and aum[q][j]==1:
                     aam[i][j] = aam[i][j]+pre[q]

     print("Attribute affinity matrix")
     for i in range(n_attributes):
         print(aam[i])
     print("Access Site Sums")
     print(pre)

Attribute affinity matrix
[1.3804, 0.282275]
[0.282275, 0.55755]
Access Site Sums
[0.25115, 0.26900666666666667, 0.782625, 0.024125, 0.530125, 0.3155, 0.249575, 0.0327]

```

```

[27] def bond(Ax,Ay):
    if Ax== -1 or Ay== -1:
        return 0
    ans = 0
    for i in range(n_attributes):
        ans = ans + (aam[i][Ax]*aam[i][Ay])
    return ans

def cont(Ai,Ak,Aj):
    print("bond ",Ai, "bond", Ak, " = ", bond(Ai,Ak))
    print("bond ",Ak, "bond", Aj, " = ", bond(Ak,Aj))
    print("bond ",Ai, "bond", Aj, " = ", bond(Ai,Aj))
    return 2*bond(Ai,Ak) + 2*bond(Ak,Aj) - 2*bond(Ai,Aj)

```

```

#Bond energy algorithm
def BEA():
    ca = []
    ca.append(0)
    ca.append(1)
    index = 2
    while index < n_attributes:
        maxi = -1
        maxc = -100000
        for i in range(1,index):
            con = cont(ca[i-1],index,ca[i])
            print("Index ", i+1, " ", "cont ", ca[i],index+1,ca[i]+1, con)
            if con > maxc:
                maxi = i
                maxc = con

        #boundary left
        con = cont(-1,index,ca[0])
        print("Index ", i+1, " ", "cont ", 1,index+1,ca[0]+1, con)
        if con > maxc:
            maxi = 0
            maxc = con

        #boundary right
        con = cont(ca[index-1],index,-1)
        print("Index ", i+1, " ", "cont ", ca[index-1]+1,index+1,index+2, con)
        if con > maxc:
            maxi = index

        if maxi==index:
            ca.append(index)
        else:
            ca.append(0)
            for j in range(index,maxi,-1):

```



```

        ca[j]=ca[j-1]
        ca[maxi] = index
        print(ca)
        index = index + 1
    print("FINAL Clustered Affinity Matrix")
    print(ca)
    return ca

```

```

✓ [29] CA = BEA()
      ca = [[0 for i in range(n_attributes)] for j in range(n_attributes)]
      for i in range(n_attributes):
          for j in range(n_attributes):
              ca[i][j] = aam[CA[i]][CA[j]]

      print(ca)

```

```

FINAL Clustered Affinity Matrix
[0, 1]
[[1.3804, 0.282275], [0.282275, 0.55755]]

```

```

✓ [30] def shift_row_aum(mat):
      row_first=[]
      for i in range(n_attributes):
          row_first.append(mat[0][i])
      for i in range(1,n_queries):
          for j in range(n_attributes):
              mat[i-1][j]=mat[i][j]
      for i in range(n_attributes):

```

```

        mat[n_queries-1][1]=row_first[1]
        # print(row_first)
        return mat

    def shift_column_aum(mat):
        col_first=[]
        for i in range(n_queries):
            col_first.append(mat[i][0])
        for i in range(n_queries):
            for j in range(1,n_attributes):
                mat[i][j-1]=mat[i][j]
        for i in range(n_queries):
            mat[i][n_attributes-1]=col_first[i]
        return mat

```


```

✓ [31] def shift_row_ca(mat):
      row_first=[]
      for i in range(n_attributes):
          row_first.append(mat[0][i])
      for i in range(1,n_attributes):
          for j in range(n_attributes):
              mat[i-1][j]=mat[i][j]
      for i in range(n_attributes):
          mat[n_attributes-1][i]=row_first[i]
      # print(row_first)
      return mat

```

```
✓ [31] def shift_column_ca(mat):  
0s  
    col_first=[]  
    for i in range(n_attributes):  
        col_first.append(mat[i][0])  
    for i in range(n_attributes):  
        for j in range(1,n_attributes):  
            mat[i][j-1]=mat[i][j]  
    for i in range(n_attributes):  
        mat[i][n_attributes-1]=col_first[i]  
    return mat
```

```
✓ [32] #Partitioning  
0s  
# this block computes access-query  
# output - list of list, in each list i we see the attributes accessed by query i  
start=n_attributes-2  
aum = [ [0.0, 1.0],  
        [0.0, 0.0],  
        [1.0, 0.0],  
        [0.0, 1.0],  
        [0.0, 0.0],  
        [1.0, 0.0],  
        [1.0, 1.0],  
        [1.0, 1.0]]  
AQ=[]  
for i in range(n_queries):  
    row=[]  
    for j in range(n_attributes):  
        if aum[i][j]==1:  
            row.append(j)  
    AQ.append(row)
```

✓  print(AQ)

```
[[1], [], [0], [1], [], [0], [0, 1], [0, 1]]
```

✓ [33] # computing TQ, BQ, OQ and costs for each permutation

```
# all permutations of attributes
shifts_list = [CA]
shifted = list(CA)
for i in range(len(CA)-1):
    shifted.append(shifted[0])
    shifted.pop(0)
    shifts_list.append(list(shifted))

# compute best vertical fragmentation
max_res = -np.inf
for shift in shifts_list:
    for seperator in range(1, len(shift)):
        TQ=[]
        BQ=[]
        OQ=[]
        until_seperator = shift[:seperator]
        from_seperator = shift[seperator:]
        # partitioned queries to TQ/BQ/OQ
        for i in range(len(AQ)):
            if set(AQ[i]).issubset(set(until_seperator)):
                TQ.append(i)
            elif set(AQ[i]).issubset(set(from_seperator)):
                BQ.append(i)
```

```
                OQ.append(i)
            elif set(AQ[i]).issubset(set(shift)):
                OQ.append(i)
        CTQ = np.array([acc[i] for i in TQ]).sum(axis=0).sum()
        CBQ = np.array([acc[i] for i in BQ]).sum(axis=0).sum()
        COQ = np.array([acc[i] for i in OQ]).sum(axis=0).sum()
        res = CTQ*CBQ - COQ**2
        if res > max_res:
            max_res = res
            res_shift = shift
            res_seperator = [until_seperator, from_seperator]

print(f"vertical fragmentation: {res_seperator}")
print(f"max res = {max_res}")
```

```
vertical fragmentation: [[1], [0]]
max res = 1.5951304620999998
```