§4 Заключительный этап: командная часть

Постановка задачи. Участникам командной части заключительного этапа было необходимо решить серию задач по анализу графа пользователей социальных сетей: предсказать возраст пользователей, не указавшего его в своем профиле; предсказать регион проживания пользователя; предположить, кто из других пользователей социальной сети является знакомым пользователя.

Участники должны были писать программы на языке Python. Продолжительность командной части заключительного этапа — 3 дня (всего 18 астрономических часов). Участники имели доступ к сети Интернет и могли пользоваться своими телефонами и ноутбуками.

Всего командам предлагалось 3 задачи — по одной на каждый день. Условие задачи становилось известно участникам утром соответствующего дня. Каждая задача оценивалась в баллах (подробнее см. далее).

Для каждой задачи было подготовлено два подграфа реальной социальной сети «Одноклассники»:

1. участникам представлялся специально подготовленный, очищенный и анонимизированный подграф;
2. проверка качества решения осуществлялась автоматически на полном графе, в котором присутствовали данные, вычищенные из первого графа.

Для каждой задачи участникам предоставлялось работающее базовое решение с низкой эффективностью, и участники стояли перед выбором: программировать с нуля свое собственное решение, которое сможет решить поставленную задачу качественнее, или дорабатывать предложенное решение. При этом можно было использовать базовое решение частично, например только модель данных или только распознаватель входных данных.

4.1. Описание исходных данных

Во всех задачах участникам предоставлялся граф пользователей (связи между пользователями) и файл с демографией (анонимизированные данные по каждому пользователю).

4.1.1. Граф пользователей

Граф сохранен в формате разреженной матрицы, где по каждой связи есть информация о её типе (родственник, друг и т.д.) в виде битовой маски. Каждая строка

матрицы соответствует друзьям одного пользователя и имеет формат:

```
ID_пользователя1 {(ID_друга1,маска1), (ID_друга2,маска2),…}
```

Матрица партиционирована по ID пользователя на 16 файлов, каждый из которых сжат стандартным протоколом сжатия GZip.

Пары в списке связей отсортированы по ID друга (по возрастанию). Пример записей из графа:

```
102416
{(5362439,0),(7321627,0),(7345280,0),(9939258,0),(9976393,0),(11260492,0),
(11924364,0),(16498676,0),(16513827,0),(21716731,0),(21826340,0),(23746537,0),
(23751503,0),(24412936,0),(24423533,0),(30287856,0),(32321147,0),(34243036,0),
(37592142,0),(39485706,0),(41505243,0),(42791620,0),(52012206,0),(52671472,0),
(54652307,0),(57293803,0),(59242794,0),(59252048,0),(62535397,0),(62563866,0),
(62567154,0),(64588902,0)}

102608
{(4167808,32784),(6019974,32),(6152844,16),(9570536,64),(10699806,33),
(13290514,0),(15064491,128),(16432948,512),(24473204,0),(24655822,0),
(25833075,256),(28000951,64),(30834507,2048),(34567533,16),(35766667,0),
(37385121,0),(40123805,512),(43134386,1024),(45439608,0),(45484652,0),
(47562525,0),(52378153,256),(52403136,512),(52493894,1024),(53483990,0),
(54048767,0),(54286279,2048),(57401158,0),(57956631,0),(58183281,0),
(61117236,32),(61898065,0),(61936634,0),(64512205,512),(65014849,0),
(65112662,0),(65259449,0)}
```

В маске связи могут быть установлены следующие биты:

1. Любовь
2. Супруг или Супруга
3. Родитель
4. Ребенок
5. Брат или сестра
6. Дядя или тетя
7. Родственники
8. Близкие друзья
9. Коллеги
10. Одноклассники
11. Племянник
12. Дед или бабушка
13. Внук или внучка
14. Однокурсник
15. Дружба в армии
16. Приемный родитель

17. Приемный ребенок
18. Крестный отец
19. Крестный сын
20. Совместная игра в спортивные игры

Помимо перечисленных битов в маске отношений может быть установлен, а может и не быть установлен нулевой бит. Этот бит играет чисто техническую роль и не имеет физического смысла. В итоге, например, отношение типа Ребенок может кодироваться числами 16 или 17.

Данные были подготовлены с использованием инструмента для хранения больших данных Apache Pig и содержат два соответствующих файла с заголовками, позволяющие участникам использовать этот инструмент и для предварительной обработки/фильтрации данных.

4.1.2. Демография пользователей

Данные о демографии предоставлены для того же миллиона пользователей, что и информация о социальных связях в формате списка атрибутов:

`userId create_date birth_date gender ID_country ID_Location loginRegion`

где:

- `userId` – идентификатор пользователя
- `create_date` – дата создания пользовательского аккаунта (количество миллисекунд от 01.01.1970)
- `birth_date` – дата рождения пользователя (количество дней от 01.01.1970, может быть отрицательным!)
- `gender` – пол пользователя (1 – мужчины, 2 – женщины)
- `ID_country` – идентификатор страны, указанной в профиле
- `ID_Location` – идентификатор региона/города, указанный в профиле.
- `loginRegion` – идентификатор региона, откуда чаще всего авторизуются в данной социальную сети пользователь (может отсутствовать!)

Пример данных:

```
44053078 1166032023073 3067 1 10414533690 2423601 99
12495764 1177932393270 1138
2 10405172143 188081
25646929 1165304175170 3756 2 10414533690 3953941 22
25646999 1160728984480 3884 2 10414533690 241372 120
12495833 1176909723643 3363 2 10414533690 2724941 11
```

Демография партиционирована по той же схеме, что и граф, но не сжата (передается в виде открытых текстов). Так же может быть обработана с помощью стандартного инструмента хранения больших данных Apache Pig или любого другого инструмента, поддерживающего CSV.

4.2 Формулировки задач

Задача 4.2.1 «Дата рождения»

Представленный для анализа фрагмент социального графа включает информацию о связях **100 тысяч пользователей**, попавших в двухшаговую окрестность сотни случайно выбранных пользователей. Участникам предоставляются файлы графа социальной сети со всеми связями и файл демографии, в котором указан данные по пользователям, включая возраст, однако возраст указан не для всех пользователей.

По пользователям которые присутствуют в графе, но не присутствуют в демографии необходимо установить значение их атрибута `birth_date` (дату рождения).

Данные записываются в файл в формате:

(<id_ползователя>\t(знак табуляции)<birth_date>)

Посчитанные результаты участников принимаются в файле формата txt и сравнивается с полными данными специально написанной программой, которая считает расхождение между данными участников и настоящими данными.

Чем меньше расхождение, тем выше оценивается результат команды согласно схеме, представленной далее в разделе «Методика оценки».

Предоставляемое участникам базовое решение (дает 1 балл):

```
 1. import random
 2. import math
 3.
 4. from GraphParser import graphParser
 5. def printall(res):
 6.     for i in range(0,len(res)):
 7.         print(res[i])
 8.         print("---------")
 9. cols = list()
10. cols.append(2)
11. (demog,fd) = graphParser.parseFolder("Task1\\Task1\\trainDemography",0,"",0,
cols)
12. c=1
13. for key, value in demog.items():
14.     print(str(key) + " val "+ str(value))
15.     c+=1
16.     if c==3:
17.         break
18.
```

```
#print(demog)
graph =  graphParser.parseFolderBySchema("Task1\\Task1\\graph",30,"")
#print(graph[0])
minBD = 9999999999999999
maxBD = 0
keyVal = 0 #'birth_date'
for key, value in demog.items():
    #print(key)
    bd = int(value[keyVal])
   # print( people)
    if bd>maxBD:
        maxBD=bd
    if bd<minBD:
        minBD=bd
print(minBD)
print(maxBD)
randCount=15
diffSum1 = 0
diffSum = [0]*randCount
for people in graph[0]:
    pId =  people['from']
    if pId not in demog.keys():
        print("err for "+str(pId))
        continue
    dateSum = 0
    totalLen=0
    maxBDp = 0
    minBDp = 9999999999999999
    #print(people['links'])
    for links in people['links']:

        pIdr = links['to']
        print(links['to'])


        if pIdr not in demog.keys():
            print("err for "+str(pIdr))
            continue
        totalLen+=1
        bd = int(demog[pIdr][keyVal])
        if bd>maxBDp:
            maxBDp=bd
        if bd<minBDp:
            minBDp=bd
        dateSum+=int(bd)

    if  (totalLen == 0):
        continue
    #    avg=propBirthDate
    #else:
    avg=(dateSum)/(totalLen)

    if (totalLen>=4):
        print("TOTAL Len big!"+str(totalLen))
        avg=(dateSum-maxBDp-minBDp)/(totalLen-2)
    else:
        print("total len small!"+str(totalLen))
        avg=(dateSum)/(totalLen)

    #avg=propBirthDate

```

```
80.     trueVal = int(demog[pId][keyVal])
81.     diffSum1+= abs(trueVal-avg)
82.     for ind in range(0,len(diffSum)):
83.         avg =  random.randrange(minBD,maxBD)
84.         diffSum[ind]+= abs(trueVal-avg)
85.
86. print(diffSum1)
87. for ind in range(0,randCount):
88.     print(diffSum[ind])
```

СПОСОБ ОЦЕНКИ РЕЗУЛЬТАТА:

Для получения количественной оценки правильности результата использовалась следующая программа-компаратор на языке Python:

```
import pandas as pd
import numpy as np
from random import randint
import math
dir_name = 'testDemography'
files = ['part-v004-o000-r-00000', 'part-v004-o000-r-00001', 'part-v004-o000-r-
00002', 'part-v004-o000-r-00003', 'part-v004-o000-r-00004', 'part-v004-o000-r-
00005', 'part-v004-o000-r-00006', 'part-v004-o000-r-00007', 'part-v004-o000-r-
00008', 'part-v004-o000-r-00009', 'part-v004-o000-r-00010', 'part-v004-o000-r-
00011', 'part-v004-o000-r-00012', 'part-v004-o000-r-00013', 'part-v004-o000-r-
00014', 'part-v004-o000-r-00015']
files = [dir_name + '/' + i for i in files]
df = pd.DataFrame()
frames = []
for file_name in files:
    d = pd.read_csv(file_name, sep='\t', names=['id', 'date', 'num', 'bla1',
'bla2', 'bla3', 'bla4', 'bla5'])
    del d['date']
    del d['bla1']
    del d['bla2']
    del d['bla3']
    del d['bla4']
    del d['bla5']
    frames.append(d)
test_data = pd.concat(frames, ignore_index=True)
answers = pd.read_csv('results.txt', sep='\t', names=['id', 'num'])

def compare(res, test):
    to_sum = []
    for i, row in res.iterrows():
        vals = test[test['id'] == row['id']]['num'].values
        if(len(vals)):
            stds = []
            for v in vals:
                if ((not math.isnan(v)) and not math.isnan(row['num'])):
                    stds.append(math.pow(v - row['num'], 2))
            if(len(stds)):
                print('stds', stds)
                to_sum.append(min(stds))
    return sum(to_sum)

s = compare(answers, test_data)
print(s)
```

РЕШЕНИЕ:

В первую очередь надо определить что поставленная задача является регрессионой задачей, после чего можно провести исследование особенностей данной задачи, найти корреляции между свойствами пользователей и провести исследование на поиск лучшей модели для работа с этими данными.

Также важные результаты могут показать гипотезы о возрасте друзей пользователя.

Задача 4.2.2 «Регион»

Представленный для анализа фрагмент социального графа включает информацию о связях **100 тысяч пользователей**, попавших в двухшаговую окрестность сотни случайно выбранных пользователей. Участникам предоставляются файлы графа социальной сети со всеми связями и файл демографии, в котором указан данные по пользователям, включая регион, однако регион указан не для всех пользователей.

По пользователям которые присутствуют в графе, но не присутствуют в демографии необходимо установить их аттрибут `ID_Location` (регион).

Ответ записывается в текстовый файл в формате:

`(<id_ползователя>\t(знак табуляции)<ID_Location>)`

Посчитанные результаты участников принимаются в файле формата txt и сравнивается с полными данными специально написанной программой, которая считает расхождение между данными участников и настоящими данными.

Чем меньше расхождение, тем выше оценивается результат команды согласно схеме, представленной в разделе «Методика оценки».

Предоставляемое участникам базовое решение (дает 1 балл):

```
 1. import math
 2. import sys
 3.
 4. def bl(graph, locs, fd=False):
 5.     res = list()
 6.     count = int(0)
 7.
 8.     for pId,conns  in graph.items():
 9.         count+=1
10.         if count%1000 == 0:
11.             print(count)
12.         dateSum = 0
13.         totalLen=0
14.         locIds=dict();
15.         print(pId)
16.         try:
17.             if locs[pId] != None:
18.                 continue
19.         except:
20.             pass
21.         if type(conns) == int:
22.             conns=[conns]
```

```
23.         for links in conns:
24.             totalLen+=1
25.             try:
26.                 frLoc=locs[links]
27.             except:
28.                 continue
29.             try:
30.                 locIds[frLoc]+=1 #int(demog[links])
31.             except:
32.                 locIds[frLoc]=1
33.
34.         resId=0
35.         popId=0
36.         for locId, total in locIds.items():
37.             if total>popId:
38.                 popId=total
39.                 resId=locId
40.
41.         res.append([pId,resId])
42.         if (fd):
43.             fd.write(str(pId)+'\t'+str(resId)+'\n')
44.     return res
45.
46. from GraphParser import graphParser
47.
48. pass
49. cols = list()
50. cols.append("userId")
51. cols.append("ID_Location")
52. (locs,fd) =
graphParser.parseFolderBySchema("Task2\\Task2\\trainDemography",0,"","userId",
cols, True)
53. cols = list()
54. cols.append("from")
55. cols.append("to")
56. cols.append("links")
57. (graph, fd) =
graphParser.parseFolderBySchema("Task2\\Task2\\graph",0,"","from",cols,True)
58. print("data loaded")
59. fdres=open("results.txt",'w')
60. bl(graph,locs, fdres)
```

СПОСОБ ОЦЕНКИ РЕЗУЛЬТАТА:

Для получения количественной оценки правильности результата использовалась следующая программа-компаратор на языке Python:

```
import pandas as pd
import numpy as np
import math
import ast

test_df = pd.read_csv('task2/test.tsv', sep='\t', names=['id', 'groups'])
results_df = pd.read_csv('task2/results.tsv', sep='\t', names=['id', 'groups'])

def compare(results, test):
    #Iterate over all submitters results
    score = 0
    not_found_penalty = -5
    false_found_penalty = -5
```

```
    found_reward = 10
    for i, row in test.iterrows():
        test_groups = ast.literal_eval(row['groups'])
        #No such user
        if(not (any(results.id == row['id']))):
            score = score + (len(test_groups) * not_found_penalty)
            continue
        #Get fit
        result_groups = ast.literal_eval(results[results['id'] == row['id']]
['groups'].values[0])
        for tg in test_groups:
            if (tg in result_groups):
                score = score + found_reward
                result_groups.remove(tg)
                test_groups.remove(tg)
        #Get penalty
        score = score + (len(result_groups) * not_found_penalty)#not found
        score = score + (len(test_groups) * false_found_penalty)#false found
    return score
compare(results_df, test_df)
```

РЕШЕНИЕ:

В первую очередь надо определить, что поставленная задача является задачей на классификацию, после чего можно провести исследование особенностей задачи, найти корреляции между свойствами пользователей и провести исследование на поиск лучшей модели для работа с этими данными.

Так же важные результаты могут показать гипотезы о месте атрибута `location_id` друзей пользователя, особенно тех, которые учились с ним в одной школе.

Вторая задача похожа на первую, хоть и принадлежит к другому классу задач машинного обучения, таким образом участники могли использовать свои наработки первой задачи для решения второй.

Задача 4.2.3 «Поиск связей»

Представленный для анализа фрагмент социального графа включает информацию о связях **1 миллиона пользователей**, попавших в двухшаговую окрестность сотни случайно выбранных пользователей. Участникам предоставляются файлы графа и демографии по пользователям. Часть связей в предоставленном социальном графе скрыта и задачей участников является максимально полно и точно раскрыть их.

Сокрытие связей коснулось только пользователей из исходного миллиона, остаток от деления аттрибут ID которых на 11 равен 7 (id % 11 == 7), сокрытию подверглось порядка 10% связей для каждого из этих пользователей. Были скрыты только ведущие в исходный миллион связи.

В прогнозе достаточно восстановить наличие связи, ее тип не важен. Результаты прогноза нужно представить в формате CSV файла вида:

```
ID_пользователя1 ID_кандидата1.1 ID_кандидата1.2 ID_кандидата1.3
ID_пользователя2 ID_кандидата2.1 ID_кандидата2.2
```

Записи в файле отсортированы по ID пользователя (по возрастанию), а затем по предсказанной релевантности кандидатов (по убыванию, саму релевантность при этом в файл писать не надо). Пример результатов:

```
5111 178542 78754
18807 982346 1346 57243
```

Результаты участников оцениваются с помощью метрики Нормализованной скидочной совокупной выгоды (Normalized Discounted Cumulative Gain, NDCG), используемой в индустрии для оценки точности работы алгоритма для этой и аналогичных ей задач. Метрика рассчитывается отдельно по каждому из пользователей, для которых есть скрытые связи, а затем усредняться. Записи в файле результата, не имеющие отношения к пользователям со скрытыми связями, при оценке результата учитываться не будут. Если по какому-то пользователю не будет предложено ни одного кандидата, то значение метрики для него будет считаться за 0.

Предоставляемое участникам базовое решение (дает 1 балл):

```
 1. # Для чтения/записи csv файлов
 2. import csv
 3. # Для работы с архивами
 4. import gzip
 5. # Для работы с файловой системой
 6. import os
 7. # Эффективные массивы простых типов
 8. import numpy
 9. # Работа с матрицами (подсчет общих друзей реализован как умножение матрицы
графа самое на себя)
10. import scipy
11. from scipy.sparse import coo_matrix, csr_matrix
12. # Пути к данным
13. dataPath = "./"
14. graphPath = os.path.join(dataPath, "trainGraph")
15. predictionPath = os.path.join(dataPath,"prediction.gz")
16.
17. # Основные параметры графа
18. numUsers = 107474
19. numLinks = int(72384968 / 2)
20. maxUserId = 9418031
21. # В этих массивах мы будем собирать данные. Инициализируем их заранее нужным
размером чтобы
22. # небыло лишнего копирования
23. form = numpy.zeros( (numLinks), dtype=numpy.int32 )
24. to = numpy.zeros( (numLinks), dtype=numpy.int32 )
25. data = numpy.ones( (numLinks), dtype=numpy.int32 )
26.
27. # Здесь храним позицию, на которую надо записать новую связь
28. current = 0
29.
30. # Итерируемся по файлам в папке
31. for file in [f for f in os.listdir(graphPath) if f.startswith("part")]:
```

```
32.     csvinput = gzip.open(os.path.join(graphPath, file), mode='rt')
33.     csv_reader = csv.reader(csvinput, delimiter='\t')
34.     # А теперь по строкам в файле
35.     for line in csv_reader:
36.         user = int(line[0])
37.         # Разбираем идшки и маски друзей
38.         for friendship in line[1].replace("{(", "").replace(")}",
"").split("),("):
39.             parts=friendship.split(",")
40.             # Записываем связь в массивы и двигаем указатель
41.             form[current] = user
42.             to[current] = int(parts[0])
43.             current += 1
44.
45.     # Не забываем закрыть файл
46.     csvinput.close()
47. # Создаем из массивов матрицу. Изначальна матрица хранится в виде списка
[i,j,v], но для эффективной
48. # дальнейшей работы нам надо преобразовать в вид [i->[j,v]]
49. fullMatrix = coo_matrix(
50.     (data, (form, to)),
51.     shape=(numLinks + 1, numLinks + 1)).tocsr()
52.
53. # Массивы больше не нужны, удаляем их из памяти
54. del form
55. del to
56. del data
57. # Считаем транспонированную матрицу (колонки и ряды поменяны местами) и её
тоже приводим в вид [i->[j,v]]
58. reversedMatrix = scipy.transpose(fullMatrix).tocsr()
59. # Поскольку прогноз нам нужно строить только для части пользователей,
остальных из
60. # исходной матрицы уберем (забьем нулями)
61. for i in range(maxUserId + 1):
62.     if i % 11 != 7:
63.         ptr = fullMatrix.indptr[i]
64.         ptr_next = fullMatrix.indptr[i+1]
65.         if ptr != ptr_next:
66.             fullMatrix.data[ptr:ptr_next].fill(0)
67.
68. # Чтобы нули не мешались при умножении, вычистим их и подуменьшим матрицу
69. fullMatrix.eliminate_zeros()
70. # Здесь и происходит основная магия - через умножение матриц получаем
счетчики общих друзей,
71. # По которым сделаем прогноз
72. commonFriends = fullMatrix.dot(reversedMatrix)
73. # Теперь осталось его записать в файл. Открываем врайтеры
74. f = open('prediction.csv', 'w')
75. writer = csv.writer(f, delimiter='\t')
76.
77. for i in range(maxUserId + 1):
78.     # Два указателя дают нам границы в которых лежат данные для этого i в
матрице
79.     ptr = commonFriends.indptr[i]
80.     ptr_next = commonFriends.indptr[i+1]
81.     # Если они не равны, значит данные есть и можно экспортировать
82.     if ptr != ptr_next:
83.         # Достаем счетчики общих друзей и создаем порядок на них от большего
к меньшему
84.         counts = commonFriends.data[ptr:ptr_next]
85.         order = numpy.argsort(-counts)
```

```
86.
87.         # Не забываем что из прогноза надо убрать себя и своих известных
друзей
88.         mineFriends =
set(fullMatrix.indices[fullMatrix.indptr[i]:fullMatrix.indptr[i+1]])
89.         mineFriends.add(i)
90.
91.         # Достаем идшки друзей, сортируем, фильтруем, обрезаем и пишем
92.         ids = commonFriends.indices[ptr:ptr_next]
93.         writer.writerow([i] + list(filter(lambda x: x not in mineFriends,
ids[order]))[:42])
94.
95. # Не забываем закрыть файл
96. f.close()
```

СПОСОБ ОЦЕНКИ РЕЗУЛЬТАТА:

Для получения количественной оценки правильности результата использовалась следующая программа-компаратор на языке Python:

```
import pandas as pd
import numpy as np
dir_name = 'testDemography'
files = ['part-v004-o000-r-00000', 'part-v004-o000-r-00001', 'part-v004-o000-r-
00002', 'part-v004-o000-r-00003', 'part-v004-o000-r-00004', 'part-v004-o000-r-
00005', 'part-v004-o000-r-00006', 'part-v004-o000-r-00007', 'part-v004-o000-r-
00008', 'part-v004-o000-r-00009', 'part-v004-o000-r-00010', 'part-v004-o000-r-
00011', 'part-v004-o000-r-00012', 'part-v004-o000-r-00013', 'part-v004-o000-r-
00014', 'part-v004-o000-r-00015']
files = [dir_name + '/' + i for i in files]
df = pd.DataFrame()
frames = []
for file_name in files:
    print(file_name)
    d = pd.read_csv(file_name, sep='\t', names=['id', 'date', 'num', 'bla1',
'bla2', 'bla3', 'bla4', 'bla5'])
    del d['date']
    del d['bla1']
    del d['bla2']
    del d['bla3']
    del d['bla4']
    del d['bla5']
    frames.append(d)
result = pd.concat(frames, ignore_index=True)
result
def dcg_at_k(r, k, method=0):
    """Score is discounted cumulative gain (dcg)
    Relevance is positive real values.  Can use binary
    as the previous methods.
    Args:
        r: Relevance scores (list or numpy) in rank order
            (first element is the first item)
        k: Number of results to consider
        method: If 0 then weights are [1.0, 1.0, 0.6309, 0.5, 0.4307, ...]
                If 1 then weights are [1.0, 0.6309, 0.5, 0.4307, ...]
    Returns:
        Discounted cumulative gain
    """
    r = np.asfarray(r)[:k]
    if r.size:
        if method == 0:
            return r[0] + np.sum(r[1:] / np.log2(np.arange(2, r.size + 1)))
```

```
        elif method == 1:
            return np.sum(r / np.log2(np.arange(2, r.size + 2)))
        else:
            raise ValueError('method must be 0 or 1.')
    return 0.

def ndcg_at_k(r, k, method=0):
    """Score is normalized discounted cumulative gain (ndcg)
    Relevance is positive real values.  Can use binary
    as the previous methods.
    Args:
        r: Relevance scores (list or numpy) in rank order
            (first element is the first item)
        k: Number of results to consider
        method: If 0 then weights are [1.0, 1.0, 0.6309, 0.5, 0.4307, ...]
                If 1 then weights are [1.0, 0.6309, 0.5, 0.4307, ...]
    Returns:
        Normalized discounted cumulative gain
    """
    dcg_max = dcg_at_k(sorted(r, reverse=True), k, method)
    if not dcg_max:
        return 0.
    return dcg_at_k(r, k, method) / dcg_max

r = [3, 2, 3, 0, 0, 1, 2, 2, 3, 0]
ndcg_at_k(r, 7)
len(test[test['id'] == 15102006]['num'].values)

pd.Series([123, 0], index=['id', 'num'])
any(result.id == 115368359)
t.to_csv('results.csv',  sep='\t', index = False, header = False)
def compare(res, test):
    #Iterate over all submitters results
    to_sum = []
    for i, row in test.iterrows():
#       If there is no such id crete with zero
        if(not (any(res.id == row['id']))):
            res.append(pd.Series([row['id'], 0], index=['id', 'num']))

    for i, row in res.iterrows():
        vals = test[test['id'] == row['id']]['num'].values
        if(len(vals)):
            stds = []
            for v in vals:
                if (not math.isnan(v)):
                    stds.append(math.pow(v - row['num'], 2))
            if(len(stds)):
                to_sum.append(min(stds))

    return sum(to_sum)
t1 = result.copy()
# t1[t1['id'] == 11536835]['num'] = t1[t1['id'] == 11536835]['num'] + 1
t1.loc[1,'num'] = t1.loc[1,'num'] + 1
t1.loc[2,'num'] = t1.loc[2,'num'] + 2
t1.loc[3,'num'] = t1.loc[3,'num'] - 5
# t1[t1['id'] == 11536835]['num']
s = compare(t1, result)
s
```

РЕШЕНИЕ:

В качестве примера решения задачи, точность прогноза которого надо превзойти,

используется логистическая регрессия, натренированная на трех признаках:

1. количестве общих друзей двух пользователей,
2. разнице в возрасте и
3. факте совпадения или различия полов.

Помимо улучшения алгоритма, необходимо так же учитывать вычислительную сложность, которая ставит требование не только качественно повысить эффективность базового решения, но и успеть рассчитать результаты для всех пользователей.

Поэтому для эффективного решения необходимо выделить только те особенности для включения в модель, которые имеют достаточно высокую корреляцию с дружбой пользователей.

Это ставит перед участниками потребность в постановке гипотез о том, какие факторы имеют высокую корреляцию, а какие — низкую, и проверку гипотез на предоставленных данных.

4.3. Методика расчета баллов

По каждой из задач была написана программа-компаратор (указанная выше), которая сравнивает решение участников с полными данными, которые были не раскрыты и имеются только у жюри Олимпиады.

Чем сильнее расхождение между результатами участников и полными данными, тем ниже результат участников в баллах. Таким образом, проверка работ участников и размер начисляемых баллов были полностью автоматизированы.

Максимальный балл заключительного этапа по командной части мог составить 56 баллов. Максимальный балл по задачам распределялся следующим образом:

- Задача 1 — 10 баллов;
- Задача 2 — 16 баллов;
- Задача 3 — 30 баллов.

Для расчета количества баллов, начисляемых участникам, использовалась логистическая регрессия с округлением до целочисленного количества баллов, в которой максимальному баллу соответствовал уровень результата, в 4 раза превосходящий базовое решение, а минимальный балл (1 балл) начислялся за уровень результата, равного предложенному базовому решению.

0 баллов начислялось в случае, если результат не был показан (команда не сдала валидное решение в срок) или если точность результата была ниже базового решения.