Санкт-Петербургский государственный университет

СУББОТИНА Екатерина Андреевна

Выпускная квалификационная работа

Оптимизация затрат на рекламу в социальных сетях с использованием моделей влияния

Уровень образования: бакалавриат

Направление 01.03.02

«Прикладная математика и информатика»

Основная образовательная программа: СВ.5005.2020 «Прикладная математика, фундаментальная информатика и программирование»

Научный руководитель:
Кандидат физ.-мат. наук,
доцент кафедры математической
теории экономических решений
Лежнина Елена Александровна

Рецензент:
Доктор физ.-мат. наук,
профессор кафедры
высшей математики
Калинина Елизавета Александровна

Санкт-Петербург

2024

Содержание

Введение

В настоящее время социальные сети стали неотъемлемой частью повседневной жизни миллионов людей по всему миру. Учитывая то, какое невероятное влияние стали оказывать социальные сети на их пользователей, появление в них рекламного контента оставалось лишь делом времени. Это естественным образом поставило перед рекламодателями задачу о наиболее эффективном способе распространения информации о своих товарах и услугах в конкурентной среде.

Пользователи или агенты взаимодействуют в социальных сетях и влияют друг на друга, а иногда даже побуждают друг друга к действиям. Анализ этих влияний, а также выявление самых влиятельных агентов может помочь в проведении наиболее эффективных рекламных кампаний при минимальных затратах. Для такого анализа применяются модели влияния в социальных сетях. В данной работе будет рассмотрена литература, связанная с исследованием таких моделей, проанализированы основные из таких моделей. Также на примере социальной сети «Вконтакте» будет проделана работа по выявлению наиболее влиятельных из группы агентов и определена доля тех, кому следует выдать бесплатные образцы продукции, чтобы добиться желаемого результата рекламной кампании.

Таким образом, результаты данной работы могут быть полезны как для компаний, занимающихся рекламой в социальных сетях, так и для исследователей в области маркетинга и аналитики данных, стремящихся улучшить эффективность своих рекламных стратегий и оптимизировать затраты на рекламу.

Постановка задачи

Компания, занимающаяся организацией юмористических концертов, стоит перед задачей разрекламировать свои услуги в некоторой социальной сети. Ее целью является максимизация эффективности рекламной кампании с учетом ограниченного бюджета. В рамках данной кампании предполагается раздача бесплатных билетов на концерт нескольким из заданных блогеров, с целью мотивировать их рассказать о впечатлениях от шоу, тем самым оказывая влияние на целевую аудиторию. Требуется определить группу пользователей, которым следует раздать бесплатные образцы, чтобы треть агентов социальной сети в результате приобрела товар.

Обзор литературы

В настоящее время публикуется большое количество работ, связанных с исследованием взаимодействий в социальных сетях, в том числе и посвященных разработке различных моделей влияния. В некоторых из них больше внимания уделено взаимодействиям между агентами социальной сети, а в других – самой сети в целом. Проанализируем статьи и публикации, чтобы представить краткий обзор основных моделей влияния.

Обратимся к классификации моделей влияния, представленной в [1] и [2]:

Будем рассматривать стохастические модели распространения. В этом виде моделей социальная сеть моделируется как направленный граф $G(V, E)$, где V – конечное множество узлов, а $E \subseteq V \times V$ – множество направленных ребер, соединяющих эти узлы. Каждый узел представляет собой пользователя социальной сети, а ребра – взаимосвязь между ними. Такие отношения – направленные, тогда $G(V, E)$ является социальным графом.

Распространение информации происходит с дискретными временными интервалами $t = 0, 1, 2, \ldots$. Каждый узел $v \in V$ имеет два возможных

состояния: активное и неактивное, которые достигаются, когда узел принимает и не принимает новую идею соответственно. Пусть $S_t \subseteq V$ – множество активных узлов в момент времени t (активный набор). S_0 будем называть начальным набором, а узлы в этом наборе – точками распространения влияния.

Стохастическая модель распространения (с дискретными временными шагами) для социального графа $G(V, E)$ определяет процесс случайной генерации активных наборов $S_t \; \forall \, t \geq 1$ с учетом набора S_0. Перейдем к классификации таких моделей.

1. Модель независимых каскадов. Модель была впервые представлена в текущем виде Кемпе [3] на основе моделей систем взаимодействующих частиц [4] и маркетинговых исследований [5]. Ключевой особенностью данной модели является то, что события распространения в каждом ребре социального графа являются взаимно-независимыми.

В модели независимых каскадов каждое ребро $(u, v) \in E$ имеет свою вероятность воздействия $p(u, v) \in [0, 1]$, которая отражает степень влияния узла u на узел v. После активации узла u в момент времени $t-1$, на ближайшем следующем шаге t у него есть единственный шанс активировать каждого из своих неактивных соседей v с вероятностью $p(u, v)$. Эта активация не зависит от других активаций.

2. Линейная пороговая модель. Предложена социологом [6] для моделирования более сложных случаев распространения, в которых активация может быть вызвана несколькими независимыми источниками. Когда сумма всех положительных сигналов, полученных объектом, превышает определенный порог, объект активируется, т.е. $\sum_i w_{ij} \geq \varphi_j$, где $\varphi_i \in [0, 1]$ – пороговое значение активации агента j под влиянием его соседей i.

В упомянутой статье значение порога фиксируется одинаковым для

всех агентов, но оно также может выбираться, например, согласно некоторому вероятностному распределению [7]. Пороговое значение зависит от многих факторов, таких как опыт агента, его убежденность, личностные черты, воздействие средств информации и др.

Также можем рассмотреть и модели отличающейся структуры:

3. Модель эпидемий. Эта модель влияния изначально применялась для изучения передачи болезней среди биологических популяций, но также может быть использована для анализа распространения информации. Наиболее детально модель разбирается в [8].

В классических эпидемических моделях каждый человек может контактировать с любым другим, что может способствовать передаче болезни. Эти модели обычно рассматриваются как непрерывные, использующие дифференциальные уравнения для анализа динамики эпидемии.

В таких моделях каждый узел переходит из одного состояния в другое, обычно S (уязвимый), I (инфицированный) и R (выздоровевший или умерший). Уязвимый человек подвержен заражению при контакте с инфицированным с вероятностью передачи заболевания β.

4. Модель голосования. В данной модели на каждом шаге каждый агент может изменить свое мнение, случайно выбрав одного из соседей и приняв его мнение. Модель голосования чем-то похожа на пороговую, поскольку агент с большей вероятностью изменит свое мнение на поддерживаемое большинством соседей. В статье [9] рассматривается применение данной модели для решения задачи максимизации влияния.

Социальная сеть здесь представляется неориентированным графом с петлями $G(N, E)$. Каждый узел v имеет множество соседей $N(v)$ и произвольно инициализируется (значением 1 или 0). В каждый момент времени каждый узел случайно выбирает одного из соседей и принимает его мнение с некоторой вероятностью.

5. Модель Изинга. Данная модель описывает возникновение намагничивания материала. Модель хорошо описана, например в [10]. Модель Изинга учитывает только взаимодействие ближайших атомов-соседей в кристаллической решетке. Аналогично может использоваться и при анализе социальных сетей.

6. Модель на основе клеточных автоматов. Клеточный автомат [11] состоит из набора объектов, обычно образующих регулярную решетку. Состояние каждого отдельно взятого объекта или агента в каждый дискретный момент времени здесь характеризуется некоторой переменной. Данные состояния синхронно изменяются через дискретные интервалы времени. Модель на основе клеточных автоматов используется для анализа масштабных, сложных систем, взаимодействие между агентами которых приводит к коллективному поведению.

7. Модель на основе цепей Маркова. В работе [12] демонстрируется модель цепей Маркова, влияние внутри которой рассматривается с точки зрения команды агентов. Она обладает двухуровневой структурой, поскольку на индивидуальном уровне моделируются действия каждого отдельно взятого агента, а на групповом – всей группы в целом.

Глава 1. Математическая модель

Будем использовать математическую модель, описанную в [13].

Пусть в нашей социальной сети n агентов. Степени их влияния друг на друга определим матрицей $A = \left[a_{ij}\right]_{n\times n}$ с элементами $a_{ij} > 0$, где a_{ij} будет обозначать силу влияния i – го агента на j – го.

Для $n > 1$ матрица A – приводима, если для некоторой матрицы перестановок P:

$$\tilde{A} = PAP^T = \begin{bmatrix} B & C \\ \mathbb{0} & D \end{bmatrix},$$

где B и D – квадратные матрицы. В противном случае матрица A –

неприводима.

Величина a_{ij} зависит от многих факторов, например, такими факторами могут быть общие интересы, дружеские отношения и т.д. При составлении модели мы будем использовать некоторые из таких факторов, с помощью которых в дальнейшем сможем определить влияние агентов. Введем эти факторы: $x_{ij}^{l} \in [0;1],\ l = 1, \dots, 5$ (условие достигается с помощью нормировки).

Здесь за x_{ij}^{1} примем количество общих подписчиков у i – го и j – го агентов. На главной диагонали матрицы (т.е. при $i = j$) запишем количество подписчиков i – го агента. Важный фактор, обозначающий пересекающуюся аудиторию агентов, способную влиять и распространять информацию.

x_{ij}^{2} – количество общих подписок (групп и пользователей) у i – го и j – го агентов. На главной диагонали – количество подписок i – го агента. Большое количество общих подписок свидетельствует о пересекающихся интересах пользователей, а значит и об их способности повлиять друг на друга.

x_{ij}^{3} – количество общих друзей у i – го и j – го агентов. На главной диагонали – количество друзей i – го агента. Фактор, схожий с количеством общих подписчиков, однако, свидетельствующий о возможном распространении влияния между пользователями вне социальной сети.

x_{ij}^{4} – отношения у i – го и j – го агентов. (1 – если i – ый агент есть в друзьях у j – го, в противном случае – 0). Данный фактор может свидетельствовать о доверительных взаимоотношениях между агентами и, как следствие, возможном распространении ими влияния друг на друга.

x_{ij}^{5} – количество репостов у i – го агента со страницы j – го. На главной диагонали – общее количество репостов на стене i – го агента. Если на странице одного агента найдется много репостов со страницы влияния другого, очевидно, что первый часто читает страницу второго, а значит,

подвержен распространяемому им влиянию.

Далее по формуле

$$a_{ij} = \sum_{l=1}^{5} \alpha_l \, x_{ij}^{l}, \tag{1.1}$$

где $\alpha_l \in [0; 1],\ l = 1, \dots, 5$ – некоторый весовой вектор, найдем элементы матрицы влияния A.

В зависимости от того, какой продукт рекламируется в ходе рекламной кампании, значения весового вектора будут меняться. В данной работе выбраны довольно общие факторы влияния, которые можно применять для исследования любого продукта. Однако, если бы в исследовании присутствовал фактор, например, общих аудиозаписей пользователей, очевидно, что значение весового вектора было бы наибольшим, если бы рекламным продуктом был, скажем, билет на концерт или музыкальный альбом, и, соответственно, наименьшим, если бы рекламируемый товар не относился к музыке.

Будем считать, что построенная матрица A – матрица смежности некоторого ориентированного графа $G(A)$, тогда рассматриваемая социальная сеть представима в виде этого графа. Если влияние i – го агента на j – го слишком мало, будем считать что пути из i в j нет.

Глава 2. Нахождение наиболее влиятельных агентов

В предыдущей главе была представлена математическая модель, с помощью которой будем описывать влияние в социальной сети. Перейдем к анализу второй части задачи, где необходимо будет среди всех агентов сети найти самых влиятельных для раздачи бесплатных образцов.

2.1. Иерархическая кластеризация агентов влияния

Тогда при указанных условиях построим матрицу T транзитивного

замыкания графа $G(A)$ по алгоритму Флойда-Уоршелла. Используем формулу:

$$t_{ij} = \ t_{ij} \vee \left(t_{ik} \wedge t_{kj}\right) \forall\, i,j,k = \overline{1,n}. \tag{2.1}$$

Далее воспользуемся алгоритмом нахождения канонической формы полученной матрицы T [14]. Здесь возможны четыре случая:

1. T – матрица, состоящая из единиц, тогда граф $G(A)$ – сильно связный, а матрица A – неприводима [15]. Направленный граф является сильно связным, если любые два его различных узла соединены направленным ребром.

2. T – блочная верхнетреугольная матрица, не содержащая нулей, симметричных относительно главной диагонали. Тогда матрица A – слабо приводимая (ее можно привести к блочно-диагональному виду путем удаления некоторых строк и столбцов), а граф $G(A)$ – однонаправлено связный, т.е. для любых его узлов u и v существует либо направленный путь из u в v, либо из v в u.

3. T – блочная верхнетреугольная матрица, содержащая нули, симметричные относительно главной диагонали. Тогда матрица A – слабо приводимая, а граф $G(A)$ – слабо связный, имеющий как минимум одну пару узлов, не соединенную друг с другом.

4. Если ни одно из вышеперечисленных условий не выполнено, то $G(A)$ – несвязный.

В случаях 2-4 составляем S – вектор, состоящий из сумм строк матрицы T. Далее строится матрица перестановок P такая, что $PS = \tilde{S}$, где $\tilde{S}$ – вектор, содержащий значения вектора S, отсортированные в порядке убывания.

Вектор $\tilde{S}$ состоит из k групп равных значений, в каждой из которых n_j $(j = \overline{1,\ k})$ таких значений, то есть

$$\sum_{j=1}^{k} n_j = n = \dim(A).$$

Далее, составим матрицу $\mathcal{T} = PTP^T$, которая будет иметь блочную структуру:

$$\mathcal{T} = PTP^T = \begin{pmatrix} T_1 & * & \dots & * \\ 0 & T_2 & \dots & * \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & T_j \end{pmatrix}, \tag{2.2}$$

где T_j – квадратная матрица размера n_j.

В полученной матрице найдем нули, симметричные относительно ее главной диагонали. Если таких нулей нет, то матрица $\mathcal{T}$ ассоциирована с односторонне связным графом $G(A)$. В противном случае граф $G(A)$ – слабосвязный, а симметричные нули обозначают не связанные между собой пары узлов.

Последним шагом, используя полученные результаты, найдем итоговую матрицу влияния

$$\tilde{A} = PAP^T. \tag{2.3}$$

В результате получим m кластеров $C_1, C_2, \dots, C_m$ с n_i $(i = \overline{1,\ k})$ элементами в каждом. Здесь агенты, находящиеся в первом кластере влияют на агентов второго кластера, которые в свою очередь влияют на элементы третьего и т.д., но не влияют на предыдущие.

2.2. Определение необходимой доли агентов

Следующий шаг – определить долю агентов, которой нужно раздать бесплатные материалы, чтобы заданное количество пользователей в итоге приобрело рекламируемый продукт. Для этого используем пороговую модель q-влияния, описанную в [16].

Предположим, что компания предоставляет бесплатные образцы в

качестве рекламы $\rho < 1$ доле агентов социальной сети. Пусть пользователи, получившие бесплатный образец, купят продукт по собственному желанию с вероятностью p^+, в то время как пользователи, не получившие бесплатный образец, также могут купить продукт с вероятностью p^-. И пусть $p^+ > p^-$. Пользователи, которые покупают продукт, могут повлиять на своих знакомых (в социальной сети), чтобы они тоже купили продукт с вероятностью q.

Пусть φ_i – случайная величина, имеющая распределение Бернулли. Она принимает значение 1, если агент i решил купить продукт по собственному желанию (не под влиянием других пользователей), т.е. $P(\varphi_i = 1) = p^+$, и 0, если пользователь приобрел товар под влиянием рекламы, т.е. $P(\varphi_i = 0) = p^-$, тогда φ_i имеет параметр μ, такой что:

$$\mu = \rho p^+ + (1 - \rho)p^-. \tag{2.4}$$

Таким образом:

$$1 - E[Y] = (1 - \mu)\sum_{k=0}^{\infty} P_1(k+1)(1 - qE[Y])^k, \tag{2.5}$$

$$1 - E[X] = (1 - \mu)\sum_{k=1}^{\infty} P_0(k)(1 - qE[Y])^k. \tag{2.6}$$

Здесь X – случайная величина, показывающая купил ли продукт некоторый узел r, Y – случайная величина, которая демонстрирует приобретет ли товар некорневой узел i. Данные величины распределены в соответствии с распределением Бернулли.

Также $P_1(k)$ – это вероятность того, что дочерний узел имеет степень k, а $P_0(k)$ – вероятность того, что корневой узел имеет степень k. Данные величины вычисляются по формулам:

$$P_0(k) = \frac{k^{-\gamma}}{\zeta(\gamma)},$$

$$P_1(k) = \frac{k^{1-\gamma}}{\varsigma(\gamma - 1)}$$

для $k = 1, 2, \ldots$. Здесь γ – положительная константа ($2 < \gamma < 3$),

$$\varsigma(x) = \sum_{k=1}^{\infty} k^{-x} - \text{дзета функция Римана.}$$

Таким образом, находим математическое ожидание $N = E[X]$. Оно будет обозначать долю пользователей, которые в результате под действием влияния или по собственной инициативе приобретут рекламируемый товар.

Пример

Рассмотрим поставленную задачу на примере реальных данных социальной сети «Вконтакте». В нашей задаче нужно прорекламировать некоторое юмористическое шоу, а раздаваемым в рамках рекламной кампании бесплатным образцом является билет на данное шоу.

Для начала выберем блогеров, среди которых будем искать наиболее влиятельных. В поставленных условиях целесообразно будет выбрать агентов социальной сети, состоящих в популярной группе «Вконтакте», специализирующейся на создании юмористического контента. Пусть такой группой будет, например, группа «Импроком».

Далее, чтобы получить необходимые данные, обратимся к API «Вконтакте». Для доступа к API создадим приложение «Вконтакте», и через него получим токен авторизации, который позволит формировать запросы для извлечения нужной информации в удобном формате JSON.

Затем, на языке Python программно реализуем алгоритм нахождения необходимых для исследования агентов. Зададим выбранную группу по ID «Вконтакте» и запросим информацию о ее участниках. Среди них выберем, например, 20 страниц пользователей, подходящих под условия:

1. Профиль открыт и не заблокирован администрацией «Вконтакте». Условие необходимо, чтобы в дальнейшем была возможность

получить доступ к информации на странице.

2. Страница «живая», т.е. пользователь был в сети в последние полгода. Исключаем из исследования страницы без активности.

3. Количество подписчиков страницы находится в промежутке $[10000; 20000]$. Количество подписчиков является основным фактором, влияющим на быстродействие программы. Условие необходимо, чтобы исследовать примерно равные по активности страницы, а также для удобства проверки результатов.

После выполнения программы получим 20 ID пользователей, которые далее подадим на вход основной программы.

Переходим к основной программе. Здесь, получив ID агентов для исследования, используя вышеописанные запросы к API для получения данных о них, составим матрицы факторов влияния x_{ij}^{l}, $l = 1, \dots, 5$. Пример такой матрицы представлен на рисунке 1.

```
[[16013     9    45     8    39   217    21    22    31     0     4     7     3     8     2     7     1     8    24     1]
 [    9  7630    26     2     9    13     3     6     6     4     0     9     8     1    10     1     0     4     4     0]
 [   45    26  6946    26    23    37    45    16     9     1     2     1    11     4     2    14     0    13    11     3]
 [    8     2    26  1386     2     5     5     1     0     0     0     0     0     0     0     1     0     1     2     0]
 [   39     9    23     2 10301     7    49     0     1     0     0     0     3     2     0     2     2     2     4     1]
 [  217    13    37     5     7 13596     4    25    42     3     0     8     5    17     3    17     1    14     8     0]
 [   21     3    45     5    49     4  4303     2     1     0     1     0     2     1     0     1     2     0     3     2]
 [   22     6    16     1     0    25     2  3350     8     0     0     0     4     3     0    12     0     4     6     0]
 [   31     6     9     0     1    42     1     8  4380     2     0    25     1     7     3    38     0     9    21     1]
 [    0     4     1     0     0     3     0     0     2  1900     0     1     0     2     0     1     0     3     0    12]
 [    4     0     2     0     0     0     1     0     0     0  4231     0     0     0     0     0     0     0     2     2]
 [    7     9     1     0     0     8     0     0    25     1     0  5372     1    15     0     0     1     3     8     0]
 [    3     8    11     0     3     5     2     4     1     0     0     1  3164     2     6     1     0     0     0     0]
 [    8     1     4     0     2    17     1     3     7     2     0    15     2  2432     1     2     0    13     5     1]
 [    2    10     2     0     0     3     0     0     3     0     0     0     6     1  5796     1     0     0     1     0]
 [    7     1    14     1     2    17     1    12    38     1     0     0     1     2     1  1004     0     5     7     0]
 [    1     0     0     0     2     1     2     0     0     0     0     1     0     0     0     0  3644     0     0     0]
 [    8     4    13     1     2    14     0     4     9     3     0     3     0    13     0     5     0  5716     2     0]
 [   24     4    11     2     4     8     3     6    21     0     2     8     0     5     1     7     0     2  3462     1]
 [    1     0     3     0     1     0     2     0     1    12     2     0     0     1     0     0     0     0     1   511]]
```

Рис. 1 – Матрица общих подписчиков x_{ij}^{1}.

Очевидно, что данные матриц факторов будут меняться вместе с обновлениями на страницах выбранных пользователей, т.е. очень часто. Поскольку считывание данных посредством запросов является довольно медленным процессом, будем учитывать обновления лишь раз в день. Также видим, что итоговое количество подписчиков, расположенное на главной диагонали не всегда принадлежит указанному в условии 3 промежутку. Так происходит потому, что в ходе получения данных о подписчиках также

исключаем из исследования страницы неактивных пользователей.

Для выполнения условия $x_{ij}^{l} \in [0;1],\ l = 1, \dots, 5$ произведем нормировку полученных матриц так, чтобы сумма элементов в строке каждой из них была равна 1.

Зададим некоторый весовой вектор α. Пусть $\alpha = [0.75, 0.8, 0.65, 0.5, 0.5]$. По формуле (1.1), исключая слишком малые влияния составим матрицу влияния A (Рис. 2).

```
[[0.   0.   0.   0.   0.   0.04 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.56 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.03 0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.54 0.   0.05 0.   0.03 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.06 0.   0.08 0.   0.05 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.   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.   0.   0.   0.  ]
 [0.06 0.   0.   0.   0.03 0.05 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.65 0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.05 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.57 0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.1 ]
 [0.06 0.   0.   0.   0.07 0.   0.   0.   0.   0.   0.   0.   0.   0.06 0.   0.   0.   0.   0.06 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.   0.04 0.   0.   0.   0.   0.  ]
 [0.08 0.04 0.03 0.   0.07 0.07 0.   0.05 0.   0.   0.1  0.04 0.05 0.   0.03 0.   0.   0.54 0.09 0.  ]
 [0.04 0.03 0.04 0.   0.03 0.04 0.   0.06 0.   0.   0.03 0.   0.09 0.04 0.   0.51 0.   0.   0.04 0.  ]
 [0.   0.   0.   0.   0.   0.07 0.   0.39 0.34 0.   0.   0.   0.   0.   0.26 0.   0.   0.   0.   0.26]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.03 0.03 0.03 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.53 0.   0.03 0.   0.   0.   0.03]
 [0.05 0.   0.   0.   0.04 0.   0.   0.   0.   0.   0.04 0.   0.   0.04 0.   0.   0.   0.   0.   0.  ]
 [0.07 0.   0.   0.   0.   0.   0.   0.   0.   0.1  0.   0.   0.   0.   0.   0.52 0.   0.   0.   0.  ]]
```

Рис. 2 – Матрица влияния A.

Так как данная матрица представляет собой матрицу смежности графа $G(A)$, то можем изобразить этот граф (Рис. 3).

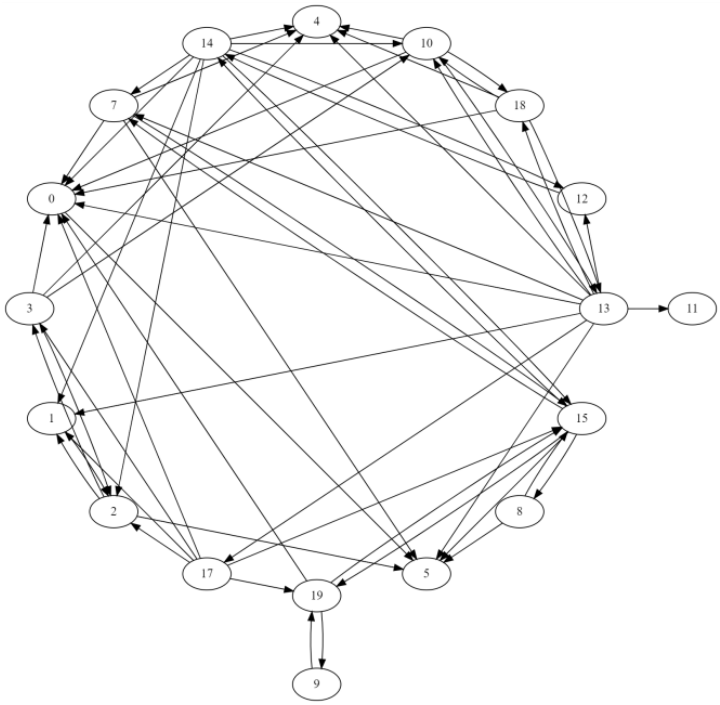

Рис. 3 – Граф $G(A)$.

Следуя алгоритму, описанному в п. 2.1., в частности используя формулы (2.1), (2.2). Получим иерархическую кластеризацию агентов влияния (Рис. 4).

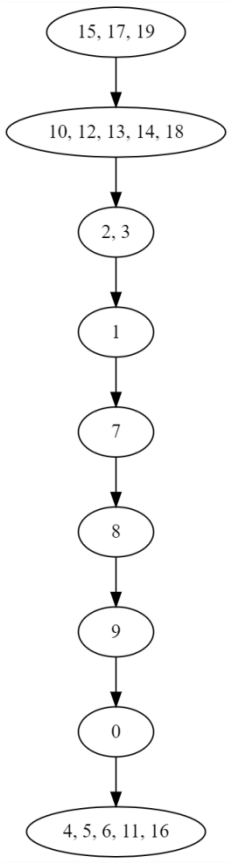

Рис. 4 – Иерархическая кластеризация агентов влияния.

Таким образом, наиболее влиятельные среди нашей группы агентов это пользователи, ID которых находятся под номерами 15, 17 и 19. Чтобы определить долю агентов, которым нужно раздать бесплатные образцы обратимся к алгоритму, описанному в п. 2.2.

Параметр q в формулах (2.5), (2.6) представляет собой вероятность того, что на некоторого пользователя социальной сети будет оказано влияние. Мы же, по полученным результатам, можем подсчитать среднее значение влияния на каждого из агентов сети и использовать его в качестве данного параметра:

$$\bar{q} = \frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{n} a_{ij} = \frac{1}{20} \times 8.24 \approx 0.41$$

Для различных $\mu = [0.02, 0.04, 0.06, 0.08, 0.1]$ и для полученного

значения $q = 0.41$ подсчитаем согласно (2.5) и (2.6) математическое ожидание $E[X]$. График зависимости $E[X]$ от параметра μ представлен на рис. 5.

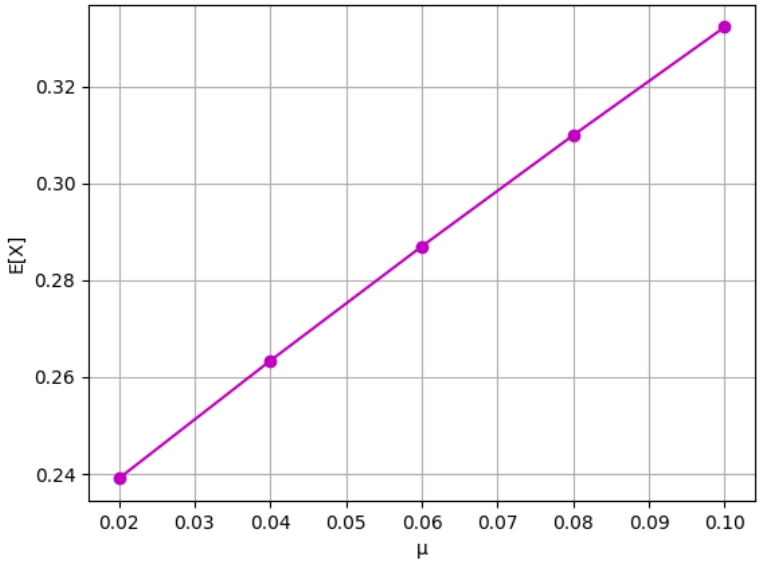

Рис. 5 – График зависимости $E[X]$ от μ.

На графике видно, что даже не самая затратная рекламная кампания может в заданных условиях принести неплохую прибыль.

Так как требуется, чтобы доля пользователей, которые в результате приобретут товар, была равна $N = E[X] \approx 0.33$, то согласно графику (рис. 5), для этого параметр μ должен быть равен ~0.1. Из формулы (2.4) выразим искомое значение ρ:

$$\rho = \frac{\mu - p^-}{p^+ - p^-}, \text{где } \mu = 0.1 \qquad (3.1)$$

Тогда итоговая величина будет зависеть от параметров p^+ и p^-. Допустим, пользователь, получивший бесплатный билет, захочет пойти на следующий концерт с вероятностью $p^+ = 0.3$, а агент, не получивший его с вероятностью $p^- = 0.05$, тогда по формуле (3.1) $\rho = 0.2$. Следовательно, в рамках рекламной кампании необходимо раздать бесплатные билеты

$\rho \times n = 0.2 \times 20 = 4$ наиболее влиятельным пользователям.

Согласно предыдущим результатам (Рис. 4), таковыми являются узлы

под номерами 15, 17 и 19. Однако для достижения желаемых результатов нам необходимо выбрать четырех пользователей. Обратимся к кластеру C_2. Здесь наиболее влиятельными являются агенты под номерами 10, 12, 13, 14, 18, среди которых нужно выбрать одного самого влиятельного.

Повторно применим вышеописанный алгоритм, но теперь будем рассматривать только пользователей из кластера C_2. Получим следующую кластеризацию (Рис.6):

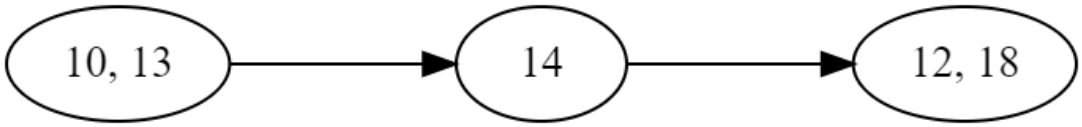

Рис. 6 – Иерархическая кластеризация агентов влияния внутри кластера C_2.

Из агентов 10, 13 можно выбрать любого, так как их влияния приблизительно равны. Возьмем, например, пользователя под номером 13, так как внутри группы он влияет на большее количество агентов, чем узел 10.

Таким образом, для раздачи бесплатных образцов выберем агентов под номерами 13, 15, 17, 19. Они будут способны оказать влияние на других пользователей так, чтобы треть из них приобрела билет на следующий юмористический концерт.

Выводы

В ходе работы была решена поставленная задача, а именно, был получен набор пользователей, которым следует в рамках рекламной кампании раздать бесплатные образцы продукции, чтобы треть агентов социальной сети приобрела товар.

Для этого была реализована программа на языке программирования Python, которая по реальным данным смоделировала взаимоотношения взятых за исследуемую группу агентов, и выдала необходимый результат по представленным алгоритмам.

Заключение

В современном мире реклама является очень важным аспектом продвижения товара, а социальные сети на сегодняшний день представляют собой, наверное, самую масштабную и эффективную площадку для ее размещения

Можно отметить, что решение задачи об определении оптимального количества бесплатных образцов товара для распространения в социальной сети представляет собой важную задачу для маркетологов и бизнес-аналитиков.

Подход к решению задачи, представленный в работе, может быть применим и полезен как для крупных корпораций, так и для небольших предприятий, стремящихся оптимизировать свои расходы и маркетинговые стратегии. Данный подход может быть адаптирован под конкретные условия и потребности компании, что позволит повысить эффективность распределения бесплатных образцов и, как следствие, увеличить прибыльность бизнеса.

Поскольку полученные результаты уже основаны на реальных данных социальной сети, можно говорить о возможности использования выводов в конкретных бизнес-сценариях для достижения конкурентных преимуществ и увеличения прибыли при наличии некоторых дополнительных статистических данных для более точного анализа.

Список литературы

1. CHEN W., LAKSHMANAN L., CASTILLO C. Information and Influence Propagation in Social Networks. Synthesis //Lectures on Data Management. – 2013.– P. 1-177.
2. ГУБАНОВ Д. А., НОВИКОВ Д. А., ЧХАРТИШВИЛИ А. Г. Модели влияния в социальных сетях // УБС. – 2009. №27.
3. KEMPE D., KLEINBERG J., TARDOS É. Maximizing the Spread of Influence through a Social Network // Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. – 2003. – P. 137-146.
4. DURRETT R. Lecture Notes on Particle Systems and Percolation. – 1988. – P. 98-201.
5. GOLDENBERG J., LIBAI B., MULLER E. Talk of the Network: A Complex Systems Look at the Underlying Process of Word-of-Mouth// Marketing Letters. – 2001.– Vol. 12. – P. 211-223.
6. GRANOVETTER M. Threshold Models of Collective Behavior // American Journal of Sociology. – 1978. – Vol. 83., No. 6. – P. 1420-1443.
7. MORRIS S. Contagion // The Review of Economic Studies. – 2000. – Vol. 67, №1. – P. 57-78.
8. NEWMAN M. Networks: An Introduction / Oxford University Press, Inc., USA. – 2010.
9. EVEN-DAR E., SHAPIRA A. A Note on Maximizing the Spread of Influence in Social Networks / Internet and Network Economics. – 2007. – P. 281-286.
10. CHANDLER, D. Introduction to Modern Statistical Mechanics/ New York, Oxford University Press. – 1987. – P. 119-124.
11. SCHIFF J. L. Cellular Automata: A Discrete View of the World/ NY: Wiley. – 2007.
12. HANG D., GATICA-PEREZ D., BENGIO S., ROY D. Learning

Influence among Interacting Markov Chains // Neural Information Processing Systems (NIPS). – 2005. – P. 132-141.

13. LEZHNINA E. A., KALININA E. A. Algorithm of Hierarchical Matrix Clusterization and Its Applications // Contributions to Game Theory and Management. – 2022.

14. САВИЦКАЯ Д. В. Нормальная форма (0,1)-матриц и алгоритмы ее построения // Вестник Санкт-Петербургского университета. Сер. 10. Прикладная математика. Информатика. Процессы управления. – 2008. – Вып. 4. – С. 85–97.

15. GUO H., LI M. I., SHUAI Zh. Global Stability of the Endemic Equilibrium of Multigroup SIR Epidemic Models// Can. Appl. Math. Q. 2006. – Vol. 14. – P. 259–284.

16. ZHAO B., LI Y. K., LUI J. C. S., CHIU D.-M. Mathematical Modeling of Advertisement and Influence Spread in Social Networks // In NetEcon: Workshop on the Economics of Networks, Systems and Computation. – 2009.

Приложение

А – программа поиска агентов влияния для исследования

```
import vk_api
import time
import numpy as np
import dpath as dp

token = 'vk1.a.8xCFy0_Ge-
1ibpRljZD6TYT59quc5EvWJK4EaaIeGxffB1yXcntvnnuDMlqepOCL17GOgCjZsS
JchoABfF1O8hGVt6glOCwDtKpw6jjbz0uj05zTgBbHdtDqv1-
600_7WpZNIqpawY0ihrrz5ch9EabY0ZCB7tmBuY51ZGeeHE5ytWvOmTp4kryyMxo
6yGrorZr9eATN_hrdIn7_czsFdQ'

session = vk_api.VkApi(token=token)
vk = session.get_api()
N = 8


def get_offset_members(group_id):
    count = vk.groups.getMembers(group_id=group_id)['count']
    return count


def get_offset_fl(user_ids, count):
    response = vk.users.get(user_ids=user_ids,
fields='followers_count', count=count)
    followers = [item.get('followers_count') for item in
response]
    return followers

def get_influencers(group_id):
    sub_list = []
    offset = 0
    count = 1000
    max_offset = get_offset_members(group_id)
    while offset < max_offset:
        if abs(offset - max_offset) < 1000:
            count = max_offset - offset
        response = vk.groups.getMembers(group_id=group_id,
offset=offset, count=count, fields='last_seen')
        # print(response)
        for item in response['items']:
            try:
                if (item['last_seen']['time'] >= 1696095450) and
not(item.get('is_closed') and (item.get('can_access_closed'))):
                    sub_list.append(item['id'])
            except Exception as E:
                continue
```

```
        time.sleep(0.2)
        offset += 1000
    return sub_list


group_id = 203677279

all_influencers = get_influencers(group_id)
print(len(all_influencers))
'''i = 0

print('~~~~~~~~~~')
while i < len(all_access):
    # print(i)
    if not all_access[i]:
        all_influencers.pop(i)
        all_access.pop(i)
    i += 1'''

print(len(all_influencers))
# print(len(all_access))
print('~~~~~~~~~~')

influencers_followers_count = []

offset = 0
count = 1000
max_offset = len(all_influencers)

while offset < max_offset:
    if abs(offset - max_offset) < 1000:
        count = max_offset - offset

    temp_inf = all_influencers[offset:(offset + count)]
    temp_fl = get_offset_fl(temp_inf, count)

    if len(temp_inf) != len(temp_fl):
        del all_influencers[offset:(offset + count)]
        continue

    for item in temp_fl:
        influencers_followers_count.append(item)

    # time.sleep(0.2)
    offset += 1000

print(len(all_influencers))
# print(len(all_access))
print(len(influencers_followers_count))
#print(influencers_followers_count)

indices = [index for index, value in
```

```
enumerate(influencers_followers_count) if value == None]
#print(indices)

for i in sorted(indices, reverse=True):
    del all_influencers[i]
    del influencers_followers_count[i]


np.savez_compressed('influencers.npz', a=all_influencers,
b=influencers_followers_count)
'''loaded = np.load('influencers' + '.npz')

all_influencers = loaded['a']
influencers_followers_count = loaded['b']'''

for i in range(20):
    for j in range(len(influencers_followers_count)):
        if 10000 <= influencers_followers_count[j] <= 20000:
            print(influencers_followers_count[j])
            break

    print(all_influencers[j])

    all_influencers.pop(j)
    influencers_followers_count.pop(j)
```

Б – основная программа

```
import vk_api
import time
import numpy as np
import os
from numpy import int64
from sklearn.preprocessing import normalize
from datetime import date
import matplotlib.pyplot as plt
import networkx as nx
from sympy import Symbol, zeta, simplify, Eq, solveset, S
from sympy.solvers import solve
from scipy.special import zeta

# token = 'vk1.a.iJ_1vVOrCs-
wlGLKu5RWobrNcHPL3WXKJYzSV3Nk1pbTTIxAku0T9wx5MhyofvKIUZCS0uS30sG
jYl3bYtrmTpkuEsZjWavDQvYIatOr33uDAYsfuYLzFqN_IjP0WBA9rKPLNJ6E3Hn
k2NHtMk2RPHL4VfV1w63zqoDeTQDwq_W3_BbDhVbKcUM4wFBQsEyyGc0bQNbEjBE
afKsJfUHMCA'
token = 'vk1.a.8xCFy0_Ge-
1ibpRljZD6TYT59quc5EvWJK4EaaIeGxffB1yXcntvnnuDMlqepOCL17GOgCjZsS
JchoABfF1O8hGVt6glOCwDtKpw6jjbz0uj05zTgBbHdtDqv1-
600_7WpZNIqpawY0ihrrz5ch9EabY0ZCB7tmBuY51ZGeeHE5ytWvOmTp4kryyMxo
6yGrorZr9eATN_hrdIn7_czsFdQ'
```

```
session = vk_api.VkApi(token=token)
vk = session.get_api()
N = 20
limit = 20

def is_profile_closed(user_ids):
    #print(type(user_ids))
    response = vk.users.get(user_ids=user_ids,
fields='personal')[0]['is_closed']
    return response

def get_offset_fl(user_id):
    count = vk.users.getFollowers(user_id=user_id)['count']
    return count


def get_offset_subs(user_id):
    count = vk.users.getSubscriptions(user_id=user_id,
extended=1)['count']
    return count


def get_offset_friends(user_id):
    count = vk.friends.get(user_id=user_id)['count']
    return count


def get_offset_wall(user_id):
    count = vk.wall.get(owner_id=user_id)['count']
    return count


def common_elements(list1, list2):
    return len(set(list1).intersection(list2))


def get_mutual(user_id1, user_id2):
    mutual_number =
len(vk.friends.getMutual(source_uid=user_id1,
target_uid=user_id2))
    return mutual_number


def get_offset_members(group_id):
    count = vk.groups.getMembers(group_id=group_id)['count']
    return count


def are_friends(user_id1, user_id2):
    if user_id2 in vk.friends.get(user_id=user_id1)['items']:
        return 1
```

```
    else:
        return 0


def show_graph_with_labels(adjacency_matrix):
    rows, cols = np.where(adjacency_matrix == 1)
    edges = zip(rows.tolist(), cols.tolist())
    gr = nx.DiGraph()
    gr.add_edges_from(edges)
    nx.draw(gr, node_size=400, with_labels=True)
    plt.show()


def bubble_sort(a, b):
    for i in range(len(a) - 1):
        for j in range(len(a) - i - 1):
            if a[j] < a[j + 1]:
                a[j], a[j + 1] = a[j + 1], a[j]
                b[j], b[j + 1] = b[j + 1], b[j]

def P1(k, gamma):
    return k ** (1 - gamma) / zeta(gamma - 1)

def P0(k, gamma):
    return k ** (- gamma) / zeta(gamma)

def EY(mu, q, Y, gamma):
    res = 0
    for k in range(limit):
        res += P1(k + 1, gamma) * (1 - q * Y) ** k

    return (1 - mu) * res

def EX(mu, q, y, gamma):
    res = 0
    for k in range(1, limit):
        res += P0(k, gamma) * (1 - q * y) ** k

    return 1 - (1 - mu) * res

def influence_groups_splitting(mylist, N):
    myset = set(mylist)
    indices = []
    splitting = []
    influence = []

    for item in myset:
        indices = [i for i in range(N) if mylist[i] == item]
        splitting.append(indices)
        influence.append(item)
    return(splitting, influence)
```

```
def sorting(S, P, N):
    S_ = [S[i] for i in range(N)]

    for i in range(N):
        temp = S_[i:N]
        maxi = max(temp)
        maxi_ind = temp.index(maxi) + i
        ind_for_P = S.index(maxi)

        if maxi in S_[0:(i)]:
            count = S_[0:(i)].count(maxi)
            indices = [index for index, value in enumerate(S) if
value == maxi]
            ind_for_P = indices[count]

        if i == N - 1:
            P[i][ind_for_P] = 1
            break

        if maxi > S_[i]:
            S_[i], S_[maxi_ind] = S_[maxi_ind], S_[i]
        P[i][ind_for_P] = 1
    return (S_, P)


def floyd_warshall(a, N):
    fw = np.zeros((N, N), dtype=int64)
    for i in range(N):
        for j in range(N):
            fw[i][j] = a[i][j]
    for i in range(N):
        for j in range(N):
            if fw[i][j] == 0:
                for k in range(N):
                    fw[i][j] = int(fw[i][j] == 1 or (fw[i][k] ==
1 and fw[k][j] == 1))
    return fw

def symm_zeros(tao):

    '''    rows = np.zeros(N, dtype=int64)
    columns = np.zeros(N, dtype=int64)'''
    indices = []
    for i in range(N - 1):
        for j in range(i + 1, N):
            if (tao[i][j] == tao[j][i]) and (tao[i][j] == 0):
                indices.append([i, j])
    return indices

def is_all_ones(t):
    for row in t:
        if any(x != 1 for x in row):
```

```
            return False
    return True


def get_influencers(group_id):
    sub_list = []
    offset = 0
    count = 1000
    max_offset = get_offset_members(group_id)
    while offset < max_offset:
        if abs(offset - max_offset) < 1000:
            count = max_offset - offset
        response = vk.groups.getMembers(group_id=group_id,
offset=offset, count=count, fields='last_seen')
        # print(response)
        for item in response['items']:
            try:
                if item['last_seen']['time'] >= 1696774497:
                    sub_list.append(item['id'])
            except Exception as E:
                continue
        time.sleep(0.2)
        offset += 1000
    return sub_list


def get_subscriptions(user_id):
    sub_list = []
    offset = 0
    count = 100
    max_offset = get_offset_subs(user_id)
    while offset < max_offset:
        if abs(offset - max_offset) < 100:
            count = max_offset - offset
        response = vk.users.getSubscriptions(user_id=user_id,
extended=1, offset=offset, count=count)
        for item in response['items']:
            sub_list.append(item['id'])
        time.sleep(0.2)
        offset += 100
    return sub_list


def get_followers(user_id):
    good_id_list = []
    offset = 0
    count = 1000
    max_offset = get_offset_fl(user_id)
    while offset < max_offset:
        if abs(offset - max_offset) < 1000:
            count = max_offset - offset
        response = vk.users.getFollowers(user_id=user_id,
```

```
offset=offset, fields='last_seen', count=count)
        time.sleep(0.2)
        offset += 1000
        for item in response['items']:
            try:
                if item['last_seen']['time'] >= 1696774497:
                    good_id_list.append(item['id'])
            except Exception as E:
                continue
    return good_id_list


def get_reposts(user_id1, user_id2):
    reposts_number = 0
    offset = 0
    count = 100
    max_offset = vk.wall.get(user_id=user_id1,
fiLlter='others')['count']
    while offset < max_offset:
        if abs(offset - max_offset) < 1000:
            count = max_offset - offset
        response = vk.wall.get(user_id=user_id1, offset=offset,
fiLlter='others', count=count)['items']
        # print(response)
        time.sleep(0.2)
        offset += 100
        for i in response:
            if i['from_id'] == user_id2:
                reposts_number += 1
    return reposts_number


'''def get_friend_relation(user_id1, user_id2):
    if are_friends(user_id1, user_id2):
        offset = 0
        count = 5000
        max_offset = get_offset_friends(user_id1)
        while offset < max_offset:
            if (abs(offset - max_offset) < 5000):
                count = max_offset - offset
            response = vk.friends.get(user_id=user_id1,
offset=offset, count=count, fields='relation')['items']
            time.sleep(0.2)
            offset += 5000
            for i in response:
                if i['id'] == user_id2 and 'relation' in i:
                    return i['relation']
                    break
        return 0
    else:
        return(-1)'''
# ID пользователей, возвращенные по результатам работы программы
```

```
influencers

influencer_id = list()
group_id = 203677279
influencer_id.append(2617)
influencer_id.append(14966)
influencer_id.append(46295)
influencer_id.append(50824)
influencer_id.append(62642)
influencer_id.append(160174)
influencer_id.append(167345)
influencer_id.append(384655)
influencer_id.append(481341)
influencer_id.append(680664)
influencer_id.append(693160)
influencer_id.append(978265)
influencer_id.append(1051399)
influencer_id.append(1313281)
influencer_id.append(1400644)
influencer_id.append(1434917)
influencer_id.append(1473948)
influencer_id.append(1705396)
influencer_id.append(2206810)
influencer_id.append(2433774)



'''all_influencers = get_influencers(group_id)
is_closed = [is_profile_closed(all_influencers[i]) for i in
range(len(all_influencers))]
# is_closed = is_profile_closed(all_influencers)


for i in range(len(all_influencers)):
    if is_closed[i]:
        all_influencers.pop(i)
        is_closed.pop(i)'''



'''for i in range(len(all_influencers) - 1):
    for j in range(len(all_influencers) - 2, i - 1, -1):
        if get_offset_fl(all_influencers[j + 1]) <
get_offset_fl(all_influencers[j]):
            all_influencers[j], all_influencers[j + 1] =
all_influencers[j + 1], all_influencers[j]'''


filename = str(date.today())

# Условие для проверки наличия файла с данными, датированного
```

```
сегодняшним днем
if os.path.exists(filename + '.npz'):
    loaded = np.load(filename + '.npz')
    x0 = loaded['a']
    x1 = loaded['b']
    x2 = loaded['c']
    x3 = loaded['d']
    x4 = loaded['e']
else:

    all_followers = [get_followers(influencer_id[i]) for i in
range(N)]
    x0 = np.zeros((N, N), dtype=int64)

    all_subscriptions = [get_subscriptions(influencer_id[i]) for
i in range(N)]
    x1 = np.zeros((N, N), dtype=int64)

    # Заполнение матриц факторов влияния

    for i in range(N):
        for j in range(N):
            if j > i:
                x0[i][j] = common_elements(all_followers[i],
all_followers[j])
                x0[j][i] = x0[i][j]
            elif i == j:
                x0[i][j] = get_offset_fl(influencer_id[i])

    for i in range(N):
        for j in range(N):
            if j > i:
                x1[i][j] = common_elements(all_subscriptions[i],
all_subscriptions[j])
                x1[j][i] = x1[i][j]
            elif i == j:
                x1[i][j] = get_offset_subs(influencer_id[i])

    x2 = np.zeros((N, N), dtype=int64)

    for i in range(N):
        for j in range(N):
            if j > i:
                x2[i][j] = get_mutual(influencer_id[i],
influencer_id[j])
                x2[j][i] = x2[i][j]
            elif i == j:
                x2[i][j] = get_offset_friends(influencer_id[i])

    x3 = np.zeros((N, N), dtype=int64)

    for i in range(N):
```

```
        for j in range(N):
            if j > i:
                x3[i][j] = are_friends(influencer_id[i],
influencer_id[j])
                x3[j][i] = x3[i][j]
            elif i == j:
                x3[i][j] = 0

    x4 = np.zeros((N, N), dtype=int64)
    for i in range(N):
        for j in range(N):
            if i != j:
                x4[i][j] = get_reposts(influencer_id[i],
influencer_id[j])
            else:
                x4[i][j] = get_offset_wall(influencer_id[i])

    np.savez_compressed(filename, a=x0, b=x1, c=x2, d=x3, e=x4)

print('~~~~~~~~~~ Матрица общих подписчиков - x1 ~~~~~~~~~~')
print(x0)
print()
print('~~~~~~~~~~ Матрица общих подписок - x2 ~~~~~~~~~~')
print(x1)
print()
print('~~~~~~~~~~ Матрица общих друзей - x3 ~~~~~~~~~~')
print(x2)
print()
print('~~~~~~~~~~ Матрица отношений - x4 ~~~~~~~~~~')
print(x3)
print()
print('~~~~~~~~~~ Матрица репостов - x5 ~~~~~~~~~~')
print(x4)
print()

n = 5
# Нормализуем матрицы x[l], чтобы сумма элементов в каждой
строке была равна 1

for l in range(n):
    globals()['x' + str(l) + '_n'] = normalize(globals()['x' +
str(l)])


alpha = [0.75, 0.8, 0.65, 0.5, 0.5]  # Весовой вектор
A = np.zeros((N, N), dtype=float)
A_ = np.zeros((N, N), dtype=int64)

for i in range(N):
    for j in range(N):
        for l in range(n):
            A[i][j] += alpha[l] * globals()['x' + str(l) +
```

```
'_n'][i][j]
        if A[i][j] < 3e-2 or i == j:
            A[i][j] = 0
            A_[i][j] = 0
        else:
            A_[i][j] = 1
A = np.round(A, 2)

# A = [[0, 1, 0, 0, 0, 0, 1, 2, 0, 2, 0, 1], [1, 0, 0, 0, 2, 0,
1, 2, 2, 0, 0, 2], [0, 1, 0, 1, 1, 0, 0, 3, 1, 2, 2, 0], [0, 0,
1, 0, 1, 0, 2, 3, 2, 0, 1, 0], [0, 0, 0, 0, 0, 2, 3, 3, 0, 0, 0,
0], [0, 0, 0, 0, 1, 0, 2, 2, 1, 2, 0, 0], [0, 0, 0, 0, 0, 2, 3,
2, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0,
0], [0, 1, 0, 0, 0, 0, 0, 1, 1, 2, 0, 1], [0, 0, 0, 0, 0, 3, 0,
0, 1, 0, 0, 0]]

'''for i in range(N):
    for j in range(N):
        if A[i][j] != 0:
            A_[i][j] = 1'''

print('~~~~~~~~~~ Матрица влияния - A ~~~~~~~~~~')
print(A)
print()
print(A_)
print()

# Ищем матрицу транзитивного замыкания - T
if np.all(A):
    T = np.ones((N, N), dtype=int64)
else:
    T = floyd_warshall(A_, N)

print('~~~~~~~~~~ Матрица транзитивного замыкания - T
~~~~~~~~~~')
print(T)
print()

if is_all_ones(T):
    print('A - неприводимая')
else:
    print('A - приводимая')


Ss = list(map(sum, T))
Ss = np.array(Ss).T.tolist()

P = np.zeros((N, N), dtype=int64)
S_, P = sorting(Ss, P, N)
Ss = np.array(Ss)
S_ = np.array(S_)
```

```
print("S = ", Ss)
print("S_ = ", S_)
print()
print('~~~~~~~~~~ Матрица перехода - P ~~~~~~~~~~')
print(P)
print("Проверка: ", P @ Ss)
print()

Tao = P @ T @ P.T
print('~~~~~~~~~~ Матрица Tao - Tao ~~~~~~~~~~')
print(Tao)
print()

print('~~~~~~~~~~ Индексы симметричных нулей Tao ~~~~~~~~~~')
ind_zeros = symm_zeros(Tao)

if len(ind_zeros) == 0:
    print('Матрица A ассоциирована с односторонне связанным
графом G(A) (симметричных нулей нет)')
else:
    print('Матрица A ассоциирована со слабо связным графом
G(A)')
    print('Симметричные нули: ')
    for item in ind_zeros:
        print(item)

print()
A__ = P @ A @ P.T
print('~~~~~~~~~~ Матрица A~ - A__ ~~~~~~~~~~')
print(A__)
print()

print('~~~~~~~~~~ Разбиение на группы по влиянию: ~~~~~~~~~~')
splitting, influence = influence_groups_splitting(Ss, N)
bubble_sort(influence, splitting)

print(splitting)
print('Отсортированное от группы наиболее влиятельных агентов к
наименее влиятельным')
print()

# show_graph_with_labels(A_)

'''for i in range(N):
    print(i, ': ', sep='')
    for j in range(N):
        if A_[i][j] == 1:
            print(j, sep='')
    print()'''
print('~~~~~~~~~~ Q-influence model ~~~~~~~~~~')
```

```
gamma = 2.5

q = 1/N * np.sum(np.sum(A, axis=1))

E_X = np.zeros(5)

'''ro = 0.1
p_plus = 0.1
p_minus = 0.01
mu = np.zeros(5)

for i in range(5):
    mu[i] = ro * p_plus + (1 - ro) * p_minus
    ro += 0.1
    p_plus += 0.02
    p_minus += 0.01'''

mu = [0.02, 0.04, 0.06, 0.08, 0.1]

for i in range(5):
    Y = Symbol('Y')

    print('mu = ', mu[i])
    E_Y = simplify(EY(mu[i], q, Y, gamma))
    # print(E_Y)

    eq1 = Eq(1 - Y, E_Y)

    sol_eq1 = solveset(eq1, Y, S.Reals)
    # print(sol_eq1)
    sol_eq1 = list(sol_eq1)
    y = sol_eq1[1]

    E_X[i] = EX(mu[i], q, y, gamma)
    print('E[X] = ', E_X[i])
    print('-----')

Mu = 0.1


plt.plot(mu, E_X, 'o-m', label='q = 0.41')
plt.xlabel('μ')
plt.ylabel('E[X]')

plt.grid()

print('~~~~~~~~~~ Наиболее влиятельные агенты внутри второго
кластера ~~~~~~~~~~')

N2 = len(splitting[1])

print('Элементы кластера: ', splitting[1])
```

```
x0_C2 = np.zeros((N2, N2), dtype=int64)
x1_C2 = np.zeros((N2, N2), dtype=int64)
x2_C2 = np.zeros((N2, N2), dtype=int64)
x3_C2 = np.zeros((N2, N2), dtype=int64)
x4_C2 = np.zeros((N2, N2), dtype=int64)

for i in range(N2):
    for j in range(N2):
        x0_C2[i][j] = x0[splitting[1][i]][splitting[1][j]]
        x1_C2[i][j] = x1[splitting[1][i]][splitting[1][j]]
        x2_C2[i][j] = x2[splitting[1][i]][splitting[1][j]]
        x3_C2[i][j] = x3[splitting[1][i]][splitting[1][j]]
        x4_C2[i][j] = x4[splitting[1][i]][splitting[1][j]]

for l in range(n):
    globals()['x' + str(l) + '_C2_n'] = normalize(globals()['x'
+ str(l) + '_C2'])

A_C2 = np.zeros((N2, N2), dtype=float)
A__C2 = np.zeros((N2, N2), dtype=int64)

for i in range(N2):
    for j in range(N2):
        for l in range(n):
            A_C2[i][j] += alpha[l] * globals()['x' + str(l) +
'_C2_n'][i][j]
        if A_C2[i][j] < 5e-2 or i == j:
            A_C2[i][j] = 0
            A__C2[i][j] = 0
        else:
            A__C2[i][j] = 1
A_C2 = np.round(A_C2, 2)

if np.all(A_C2):
    T_C2 = np.ones((N2, N2), dtype=int64)
else:
    T_C2 = floyd_warshall(A__C2, N2)

Ss_C2 = list(map(sum, T_C2))
Ss_C2 = np.array(Ss_C2).T.tolist()

P_C2 = np.zeros((N2, N2), dtype=int64)
S__C2, P = sorting(Ss_C2, P_C2, N2)
Ss_C2 = np.array(Ss_C2)
S__C2 = np.array(S__C2)

splitting, influence = influence_groups_splitting(Ss_C2, N2)
bubble_sort(influence, splitting)
print('Разбиение на группы по влиянию отсортированное от группы
наиболее влиятельных агентов к наименее влиятельным')
```

```
print(splitting)
plt.show()
```