Содержание

1. Введение

С каждым годом рынок товаров и услуг растет, и становится все труднее и труднее найти в этом спектре разнообразий именно то, что подходит конкретному потребителю. Именно поэтому разработке рекомендательных систем уделяется столько времени и внимания: они влияют на то, какие новости мы читаем, какую музыку слушаем, что покупаем. Алгоритмы рекомендаций могут применяться не только для составления развлекательного контента или подсказок при онлайн-покупках, но и, например, для подбора акций в инвестиционный портфель или дисциплин по выбору в новом семестре.

Цель работы – построить рекомендательную систему, используя для этого векторное представление графа на основе методов случайного блуждания. Для этого были рассмотрены различные виды рекомендательных систем и проанализированы особенности реализации на графах, после чего был составлен и оценен алгоритм рекомендаций.

В работе реализован алгоритм случайного блуждания, используемый для векторизации графа. Были учтены недостатки рекомендательных систем, такие как проблема «холодного старта», и предложен вариант решения. Работа алгоритма протестирована на смоделированных и реальных данных, взятых из открытых источников.

Реализация алгоритма представлена на языке программирования Python с преимущественным использованием библиотеки Networkx.

2. Анализ реального графа

Для проверки работы алгоритма был взят граф internet-industry-partnerships [1]. Каждый узел представляет компанию, которая конкурировала в интернет-индустрии в период с 1998 по 2001 год. Две компании связаны ребром, если они объявили о создании совместного предприятия, стратегического альянса или другого партнерства [2].

Рассмотрим основные характеристики графа.

Таблица 1. *Основные характеристики графа*

Количество узлов	219.00000
Количество ребер	630.00000
Минимальный узел	1.00000
Максимальный узел	219.00000
Средняя степень узла	3.00000
Плотность	0.02639
Диаметр	6.00000

Как можно заметить, наименьший индекс узла в данном графе – 1, а не 0, как обычно. Это требуется учесть для дальнейшей работы, чтобы избежать ошибок.

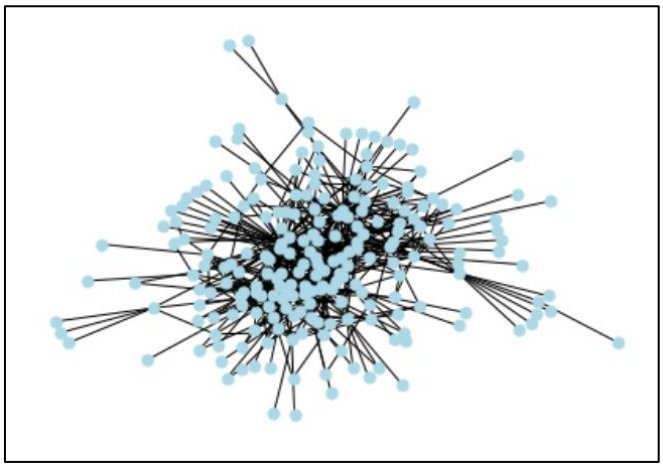

Рис 1. *Визуализация графа*

Так как в графе относительно много узлов и ребер, то визуализация не очень информативна. Посмотрим на распределение степеней узлов.

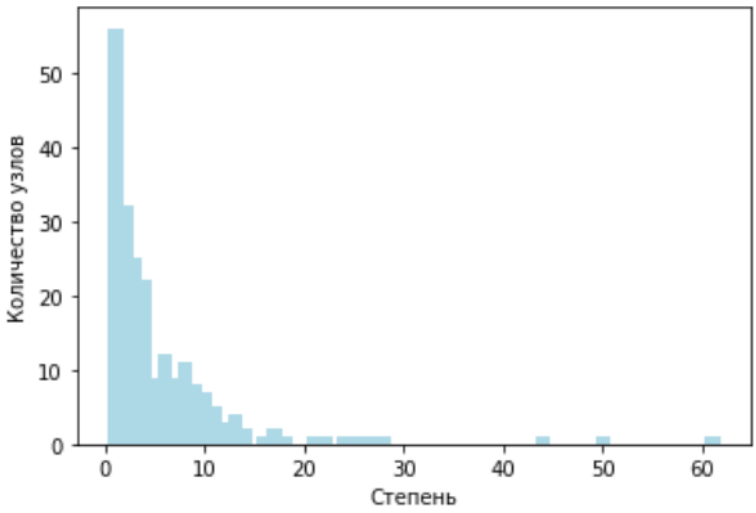

Рис 2. *Распределение степеней узлов в графе*

Большинство узлов имеют небольшую степень, но при этом есть и некоторое количество узлов, у которых количество связей с другими участниками огромно. Это согласуется со степенным законом распределения.

3. Теоретическая справка

3. 1. Рекомендательные системы

Рекомендательная система – это комбинация алгоритмов обучения, статистических инструментов и алгоритмов распознания. Как правило, выделяют три стратегии создания рекомендательных систем: на основе содержания (content-based или item-based), на основе коллаборативной функции (collaborative или user-based) и смешанная.

Рекомендательная система на основе содержания в качестве точки опоры использует online-историю потребителя. Это может быть история заказов, поставленные лайки, часто используемые тэги, ключевые слова поиска в браузере и даже переход по ссылкам.

Коллаборативная функция опирается на поиск пользователей с похожими паттернами и предлагает пользователю те же объекты рекомендации, что были у них.

У обеих стратегий есть недостатки, например, проблема «холодного старта»: как рекомендовать то, чем никто еще не пользовался и не приобретал, и что рекомендовать пользователю, который только появился в системе и не имеет истории? Решить эту проблему помогает использование смешанной стратегии и рекомендация самых популярных объектов (Top-N). [11]

Для построения рекомендательных систем активно используют глубокое обучение, однако исследования, приведенные в статье [6], показали, что классические подходы для решения данной задачи подходят лучше. Авторы утверждают, что графовый алгоритм, основанный на методах случайного блуждания, значительно превосходит все существующие подходы. В 2020 году на Хабре была опубликована статья [11], в которой авторы предлагают свою версию улучшенного алгоритма, а в статье [9] автор рассказывает о построении рекомендательной системы на основе графов в сфере логистики. Помимо областей применения представленные в них системы рекомендации значительно отличаются своей реализацией: так, например, авторы [11] и [9] выбрали разные метрики оценивания, что позволяет ознакомиться с

преимуществами обеих. Подробнее об оценке схожести объектов говорится в [5], где рассказывается о коэффициенте сходства Джаккарда, косинусном расстоянии и центрированном косинусном расстоянии.

После анализа литературы было решено остановиться на рекомендательной системе на основе содержания. Во-первых, при работе с такой системой информацию о пользователях и товарах можно хранить отдельно, что снижает время работы алгоритма. Во-вторых, информация о пользователях (интересы, геолокация, возраст и т.д, на основе которых делается вывод о схожести между пользователями) может отсутствовать, в то время как информация о товарах есть всегда.

3. 2. Теория графов

Граф – это модель, состоящая из множества вершин (узлов) и множества соединяющих их ребер. Так как мы рассматриваем item-based систему, то в качестве узлов выступают объекты рекомендаций (книги, продукты, акции, online-курсы и т.д.), а ребро между узлами означает, что данные товары/услуги хотя бы раз приобретались вместе.

Важную роль имеют такие два показателя вершины, как ее степень и вес. Степень – это количество ребер, исходящих из вершины. Если степень вершины равна 1, то ее называют висячей, а если нулю – то изолированной. Вес – число, которое ставят в соответствие данной вершине. Это может быть как ее степень, так и любой другой показатель важности.

Различают ориентированные и неориентированные графы. Если граф ориентированный, значит, его ребрам присвоено направление, которое нужно учитывать при обходе графа. Поскольку в графе из товаров/услуг обычно не имеет значение, в каком порядке пользователь добавил их корзину, а важно, что он приобрел их вместе, то при построении системы будем иметь дело только с неориентированными графами (которые всегда можно получить из ориентированных путем игнорирования заданного направления).

Путь (иногда называют маршрутом) – это последовательность смежных ребер. Задается перечислением вершин, по которым он пролегает. Таким образом, длина пути – это количество ребер в пути.

3. 3. Случайное блуждание и векторизация

В случае рекомендательных систем удобнее всего работать не с самими графами, а с их векторными представлениями, в которых можно зафиксировать расположение узлов в графе и их расположение по отношению к другим вершинам. Наиболее распространенным является подход к векторизации (эмбеддингу) узлов на основе случайных блужданий, так как она учитывает предположение, что схожие узлы имеют тенденцию существовать при коротких случайных обходах по графу.

Случайное блуждание по графу – это случайный процесс перехода между вершинами, определяемый матрицей перехода P, где p_{ij} означает вероятность перехода из узла $node_i$ в узел $node_j$ [8]. Если мы для каждой вершины совершим k случайных блужданий длины l, то вершины в получившимся массиве блужданий можно будет интерпретировать как слова в некотором тексте и использовать любой инструмент, преобразующий текст в массив векторов [10].

Как правило, для получения матрицы перехода используют формулу $P = D^{-1}A$, где D – это матрица, на диагонали которой располагается вектор сумм степеней вершин соответствующей строки, а A – это матрица смежности. Однако, чем больше вершин и ребер в графе, тем больше времени занимает нахождение данных матриц. Вторая проблема в этой задаче – это память. В библиотеке Networkx реализована функция adjacency_matrix, которая возвращает матрицу смежности для графа. К сожалению, для записи результата уходит довольно много оперативной памяти, и для графа с 80 000 узлами уже невозможно найти матрицу смежности. После анализа возможных путей решения была выбрана стратегия не использовать всю матрицу переходов для построения случайного блуждания, а лишь отдельную ее часть.

Для дальнейших вычислений была использована библиотека sympy (код см. Приложение 3).

Пусть мы имеем граф с пятью вершинами и некоторым количеством ребер. Тогда в матрице смежности A_{5x5} на позиции (ij) будет стоять 1, если между вершинами i и j есть ребро, и 0 в противном случае. Обозначим через Sum_i сумму элементов i-й строчки матрицы A. Матрица переходов P будет иметь вид:

$$\begin{bmatrix} \frac{a_{11}}{Sum_1} & \frac{a_{12}}{Sum_1} & \frac{a_{13}}{Sum_1} & \frac{a_{14}}{Sum_1} & \frac{a_{15}}{Sum_1} \\ \frac{a_{21}}{Sum_2} & \frac{a_{22}}{Sum_2} & \frac{a_{23}}{Sum_2} & \frac{a_{24}}{Sum_2} & \frac{a_{25}}{Sum_2} \\ \frac{a_{31}}{Sum_3} & \frac{a_{32}}{Sum_3} & \frac{a_{33}}{Sum_3} & \frac{a_{34}}{Sum_3} & \frac{a_{35}}{Sum_3} \\ \frac{a_{41}}{Sum_4} & \frac{a_{42}}{Sum_4} & \frac{a_{43}}{Sum_4} & \frac{a_{44}}{Sum_4} & \frac{a_{45}}{Sum_4} \\ \frac{a_{51}}{Sum_5} & \frac{a_{52}}{Sum_5} & \frac{a_{53}}{Sum_5} & \frac{a_{54}}{Sum_5} & \frac{a_{55}}{Sum_5} \end{bmatrix}$$

Рис 2. *Матрица переходов для графа с пятью узлами*

Вектор начальных состояний имеет вид: $p^0 = [p_1, p_2, p_3, p_4, p_5]^T$. Тогда матрица перехода в момент времени t=1 можно найти так: $P^1 = P^T \cdot p^0 =$

$$\begin{bmatrix} \frac{a_{51}p_5}{Sum_5} + \frac{a_{41}p_4}{Sum_4} + \frac{a_{31}p_3}{Sum_3} + \frac{a_{21}p_2}{Sum_2} + \frac{a_{11}p_1}{Sum_1} \\ \frac{a_{52}p_5}{Sum_5} + \frac{a_{42}p_4}{Sum_4} + \frac{a_{32}p_3}{Sum_3} + \frac{a_{22}p_2}{Sum_2} + \frac{a_{12}p_1}{Sum_1} \\ \frac{a_{53}p_5}{Sum_5} + \frac{a_{43}p_4}{Sum_4} + \frac{a_{33}p_3}{Sum_3} + \frac{a_{23}p_2}{Sum_2} + \frac{a_{13}p_1}{Sum_1} \\ \frac{a_{54}p_5}{Sum_5} + \frac{a_{44}p_4}{Sum_4} + \frac{a_{34}p_3}{Sum_3} + \frac{a_{24}p_2}{Sum_2} + \frac{a_{14}p_1}{Sum_1} \\ \frac{a_{55}p_5}{Sum_5} + \frac{a_{45}p_4}{Sum_4} + \frac{a_{35}p_3}{Sum_3} + \frac{a_{25}p_2}{Sum_2} + \frac{a_{15}p_1}{Sum_1} \end{bmatrix}$$

Рис. 3. *Матрица перехода на шаге t=1*

Так как вектор начальных состояний содержит только одну единицу (соответствует вершине, в который мф находимся в момент времени t=0) и четыре нуля, то очевидно, что нет необходимости вычислять всю матрицу смежности для того, чтобы найти распределение вероятностей перехода на шаге t=1. Если $p_i^0 = 1$, то $p^1 = \left[\frac{a_{i1}}{\sum_{j=1}^{n} a_{ij}}; \ldots; \frac{a_{in}}{\sum_{j=1}^{n} a_{ij}}\right]^T$, i = 1,…n; n = 5.

Легко заметить, что формула будет справедлива для любого n, а значит, вместо всей матрицы смежности достаточно сделать рассчеты для i-ого узла.

Таким образом, мы упростили задачу построения случайного блуждания для некоторого начального узла. Поскольку для векторизации граф будет представлен в виде некоторого текста, где каждое случайное блуждание – это отдельное предложение, то в качестве длины каждого блуждания возьмем 15 узлов (как средняя длина предложения в тексте).

В качестве метода векторизации используется Word2Vec, широко распространенный в рекомендательных механизмах. Это алгоритм машинного обучения, основанный на контекстной близости: мы считаем, что слова, входящие в одно предложение, имеют схожий смысл, следовательно, их векторные представления будут похожи. Здесь в качестве метрики близости используется косинусное расстояние: $cos(\theta) = \frac{A \cdot B}{||A|| \cdot ||B||}$. [4].

3.4. Некоторые проблемы и их решение

При построении рекомендательных систем с помощью векторного представления графа на основе случайных блужданий можно встретить несколько проблем. В этом разделе описаны сами проблемы и решения, которые будут использованы в дальнейшем.

Как уже упоминалось, для item-based рекомендательной системы характерны проблемы «холодного старта» для пользователя и объекта. Пусть новому пользователю, у которого еще нет «истории», на основе которой мы можем строить рекомендацию, будет предложено top-5, то есть пять самых популярных объектов. В нашем случае «популярные» вершины определим как вершины с наибольшей степенью. Что касается проблемы «холодного старта» для нового объекта, то предлагается «подкидывать» пользователю в список рекомендаций один новый или наименее популярный объект: то есть вершину с нулевой или наименьшей степенью. Такой способ может негативно повлиять на качество рекомендаций, однако он

активно используется в онлайн-магазинах, которым важнее продать как можно больше.

Следующая проблема, это выбор момента векторизации графа. Если представлять граф в численном виде каждый раз при создании новой рекомендации, то может возникнуть ситуация, что для одного и того же набора исходных вершин алгоритм выдает совершенно другие рекомендации. Это связано с использованием случайного блуждания, из-за чего каждый раз граф будет представлен по-другому. Поэтому процесс эмбеддинга вынесем перед непосредственным построением рекомендаций и будем считать, что он меняется только при добавлении новых связей в граф.

Число рекомендаций на одну вершину должно иметь обратную зависимость от числа исходных вершин. Пусть у нас имеется пользователь, который в прошлом сделал только две покупки. Сколько рекомендаций мы должны ему дать? С одной стороны, чем больше у пользователя покупок, тем проще нам будет найти похожие товары и порекомендовать их. С другой стороны, нам нужно простимулировать пользователя сделать много новых покупок, и чем больше мы ему покажет, тем больше он может купить. Возьмем фиксированное число рекомендаций на одну вершину, например, пять. Тогда пользователь с двумя покупками в прошлом получит десять рекомендаций, а пользователь с двадцатью покупками – сто рекомендаций, но просмотрит он значительно меньше. Число рекомендаций на одну вершину можно найти по следующей формуле:

$$k = [\frac{5}{\mathrm{n}} + 1],$$

где n – количество вершин в «истории», а [ ] – означает целую часть числа. Тогда при n = 2 получим k = 3, следовательно, шесть рекомендаций, а при n = 20, k = 1, что дает 20 рекомендаций. Указанная формула была получена произвольно и может быть заменена любой другой.

3. 5. Алгоритм

Пусть имеется *пользователь* (например, покупатель в online-магазине или компания, которая хочет создать стратегический альянс) и список объектов (*base*), с которыми этот пользователь связан (его предыдущие покупки или компании, с которыми раньше сотрудничали).

3.5.1. Алгоритм векторизации графа

1. Для всех узлов графа G строится случайное блуждание:

1.1. Рассчитывается вектор переходных состояний для i-ого узла по формуле:

$$P = A/D,$$

где вектор A – i-ая строчка матрицы смежности, на позиции (ij) которой стоит 1, если между вершинами i и j есть ребро, и 0 в противном случае;

$D = \sum_{i=0}^{n} a_i$.

1.2. Если вершина изолирована, то случайное блуждание представляет собой путь длины 15, состоящий только из начальной вершины.

1.3. Если вершина не изолирована, то с учетом вектора переходных вероятностей из списка соседей случайно выбирается следующая вершина, для которой повторяются пункты 1.1-1.3, пока не будет получен путь длины 15.

2. ассив со случайными блужданиями векторизуется с помощью Word2Vec.

3.5.2. Алгоритм построения рекомендаций

1. На вход рекомендательной системе подается список base. Если это новый пользователь, то на вход подается список из одного элемента ‘None’.

2. Если base ==‘None’, то возвращается top-5 и алгоритм заканчивает свою работу. Иначе переход к пункту 3.

3. Рассчитывается $k = [\frac{5}{\mathrm{n}} + 1]$, где n – количество вершин в base, [ ] – целая часть числа.

4. Вершины проверяются на изолированность. Изолированные вершины убираются из списка base. Если все вершины из списка base – изолированные, то возвращается top-5 и алгоритм заканчивает свою работу.

5. Для каждой вершины b из списка base и каждой вершины графа (за исключением тех, что входили в base) рассчитывается косинусное расстояние.

6. Выбирается k вершин, чье косинусное расстояние наибольшее.

7. В окончательную подборку добавляется одна вершина с наименьшей степенью во всем графе.

4. Проверка алгоритма на модельных данных

Проверка качества алгоритма рекомендательной системы имеет некоторые трудности. Во-первых, в отличие от задач регрессии и классификации, где возвращается одно значение, мы имеем дело с целым списком объектов без однозначного отображения один-ко-одному. Во-вторых, для одного и того же набора исходных вершин могут (и скорее всего будут) возвращаться разные рекомендации; кроме того, возвращаемый набор содержит одну «подкинутую» вершину, которая не связана с «историей» пользователя.

Тем не менее, чтобы протестировать алгоритм и получить некоторое представление о его работе, был реализован следующий метод проверки:

1. Создается случайный граф *G_test* с *n* узлами и с *e* ребрами. Считаем, что это граф связи объектов в момент времени t = 2.

2. Случайным образом выбирается 20% от всех ребер. Создаётся список *x_nodes* из первых вершин в каждой пара ребер, причем первый элемент списка – это список из одной вершины, второй элемент – список из двух вершин, и так далее. Последний элемент – это список из всех вершин. Аналогичным образом создается список *y_nodes* из вторых вершин в паре ребер.

3. Создается граф G: копия графа G_test, из которой удаляются использованные ребра. Считаем, что это граф связей в момент времени t = 1, а удаленные ребра – это будущие покупки пользователя.

4. Граф G векторизуется.

5. Для каждого списка вершин из x_nodes создаются рекомендации *rec_4_user*.

6. Рассчитываются precision и recall по следующим формулам [7]:

$$\text{precision} = \frac{\#\,(\text{rec\_4\_user} \cap \text{y\_nodes[i]})}{\#\,\text{rec\_4\_user}}$$

$$\text{recall} = \frac{\#\,(\text{rec\_4\_user} \cap \text{y\_nodes[i]})}{\#\,\text{y\_nodes[i]}}$$

где # - количество уникальных элементов, ∩ - пересечение множеств.

7. Подсчитывается среднее значение каждой метрики.

С помощью метода dense_gnm_random_graph из библиотеки Networkx создадим небольшой граф *G_test* с n = 150 и e = 4*n.

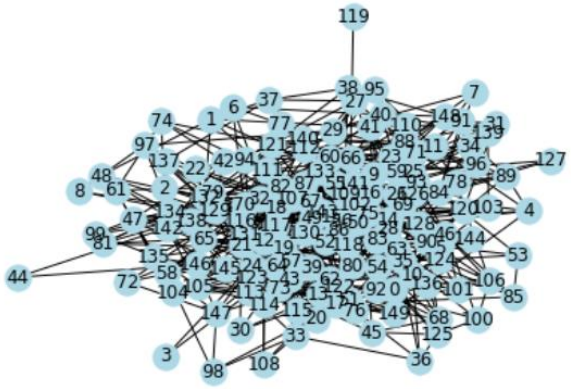

Рис. 4. *Визуализация графа G_test*

Изолированных вершин в исходном графе нет. Посмотрим на распределение степеней.

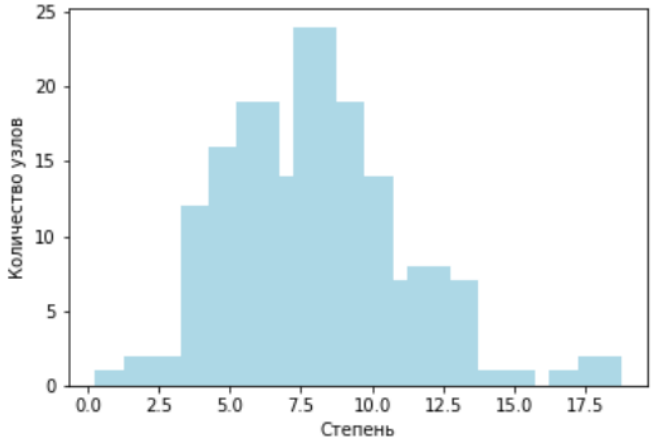

Рис 5. *Распределение степеней вершин графа G_test*

Поскольку граф создан случайным образом, он не попадает под закон распределения стеней узлов.

Проверим качество по указанному выше алгоритму.

Таблица 2. *Качество алгоритма на первых пяти элементах списка x_nodes*

	precision	recall	y_nodes	recommendation
0	0.00	0.00	[107]	{96, 4, 5, 73, 110, 54, 119}
1	0.00	0.00	[107, 36]	{96, 5, 73, 10, 50, 19, 119}
2	0.00	0.00	[107, 36, 101]	{5, 73, 10, 50, 19, 119}
3	0.12	0.25	[107, 36, 101, 19]	{5, 73, 10, 105, 108, 50, 19, 119}
4	0.11	0.20	[107, 36, 101, 19, 140]	{5, 73, 10, 105, 108, 50, 19, 116, 119}

Среднее значение Precision: 0.38

Среднее значение Recall: 0.21

Хотя числа далеки от единицы, результаты в целом неплохие. Precision отражает количество релевантных рекомендаций среди всех сделанных рекомендаций. Так как число рекомендаций должно было превышать число исходных вершин, то получить единицу по этому показателю мы бы и не смогли. Результат, близкий к 0.5, вполне хороший.

Recall – это количество релевантных рекомендаций по отношению к y_nodes, и наше значение 0.21 означает, что 79% объектов, которые пользователь «приобрел» в будущем, не были нами порекомендованы.

Посмотрим на графики. Чем больше было вершин в base, тем точнее была рекомендация.

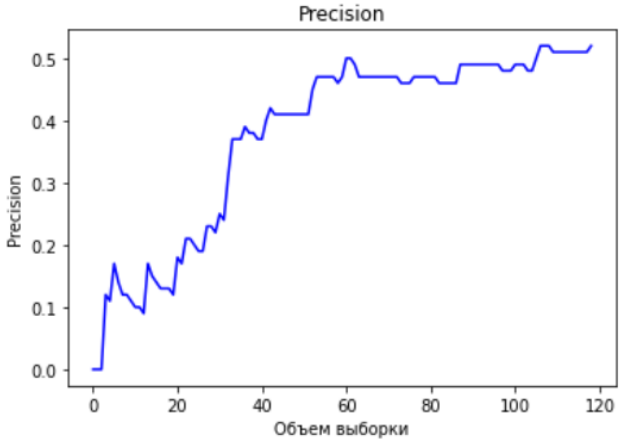

Рис. 6. *Зависимость значения precision от объема base*

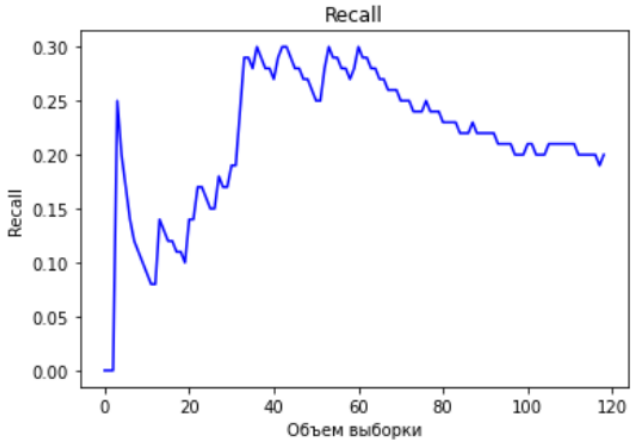

Рис. 7. *Зависимость значения recall от объема base*

Теперь проверим алгоритм на частных случаях.

Случай 1. Изолированная вершина в base. Согласно алгоритму рекомендательной системы, если base состоит только из изолированных вершин, то возвращается top-5 вершин. Сгенерируем случайный граф с n = 30, e = n*2. Для удобства восприятия размер и цвет вершины зависит от ее степени: чем больше степень, тем больше размер и темнее цвет.

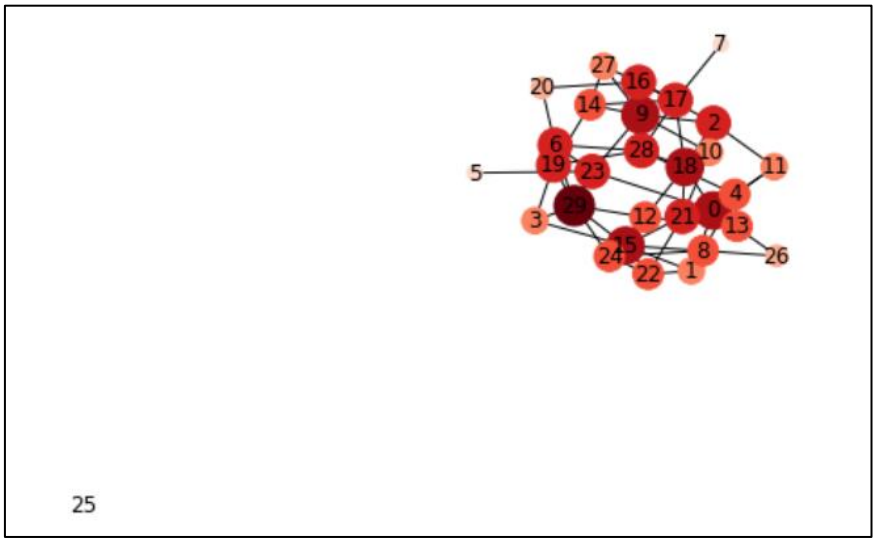

Рис. 8. *Граф с изолированной вершиной*

Запустим алгоритм с base = [25]. Результат: {0, 9, 15, 18, 29}. Это соответствует раскраске графа.

Случай 2. Новый пользователь. Оставив тот же граф, но запустим его с base = [‘None’]. Результат: {0, 9, 15, 18, 29}, что соответствует выводу для изолированной вершины. В этих местах алгоритм работает правильно.

5. Проверка алгоритма на реальных данных

Протестируем алгоритм на реальных данных. При анализе графа мы уже выяснили, что нумерация узлов в нем начинается с единицы. Перенумеруем вершины, просто вычтя из нынешнего номера единицу. После этого можно строить рекомендации. Результаты и графики представлены ниже.

Таблица 3. *Качество алгоритма на первых пяти элементах списка x_nodes*

	precision	recall	y_nodes	recommendation
0	0.0	0.0	[91]	{1, 2, 99, 5, 7, 139, 44}
1	0.0	0.0	[91, 165]	{0, 1, 5, 7, 139, 44}
2	0.0	0.0	[91, 165, 113]	{1, 5, 7, 139, 48}
3	0.0	0.0	[91, 165, 113, 67]	{1, 5, 7, 139, 48}
4	0.0	0.0	[91, 165, 113, 67, 177]	{1, 193, 5, 7, 139, 48}

Precision: 0.21

Recall: 0.05

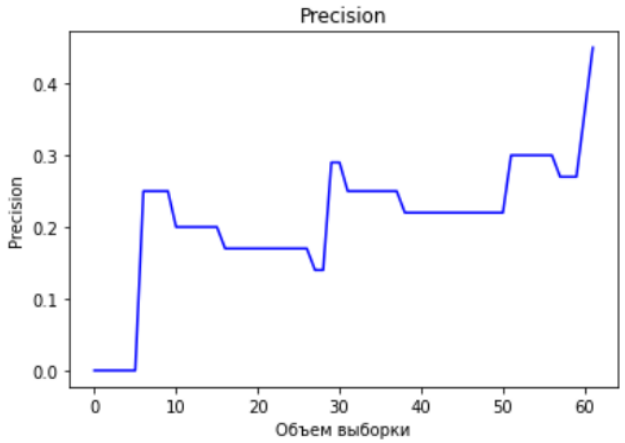

Рис. 9. *Зависимость значения precision от объема base*

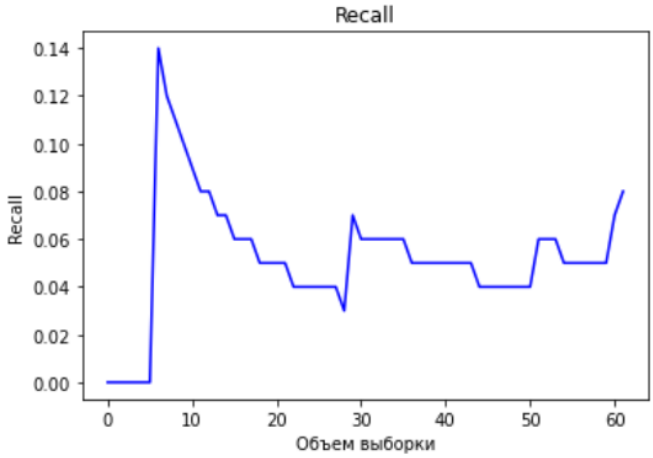

Рис. 10. *Зависимость значения recall от объема base*

К сожалению, качество на реальных данных упало еще ниже. Почему так вышло?

Вспомним, как выглядело распределение степеней вершин для смоделированного графа и реального. В модельном графе распределение походило на нормальное, и большинство узлов имели степень 7-10. Здесь же большинство узлов имеют небольшую степень, а это имеет большое влияние на результат.

Во-первых, при отборе ребер мы могли взять узлы с единичной степенью, и после удаления соответствующих ребер в качестве base брались уже изолированные узлы, рекомендации для которых, как уже неоднократно говорилось, берутся не из правила схожести узлов.

Во-вторых, чем меньше соседей у узла, тем глубже мы можем уйти при составлении случайного блуждания: мы просто-напросто будем удаляться от начальной вершины, в то время как для графа с нормальным распределением степеней мы бы оставались вблизи вершины начала пути.

6. Заключение

Целью данной работы было построить рекомендательную систему с использованием векторного представления графа на основе методов случайного блуждания.

В ходе разработки алгоритма была выведена формула для подсчета переходных вероятностей на шаге t = 1, позволяющая хранить в памяти не массив n на n, а вектор n на 1, что значительно упростило генерацию случайного блуждания. Также было предложено решение проблемы «холодного старта» как для пользователя, так и для объекта.

Из-за сложностей оценки качества алгоритмов рекомендаций трудно дать точный ответ о релевантности рекомендаций, а метрики, предложенные для более сложных систем рекомендаций, дали невысокие результаты. Это также связано с тем, что построение данной рекомендательной системы не использовалось машинное обучение (за исключением метода Word2Vec), и делался акцент на работу с графами, которым посвящена дисциплина. В дальнейшем систему можно доработать с помощью нейронных сетей. Также имеет смысл повысить скорость работы, распараллелив вычисления, необходимые для векторизации графа.

Алгоритм реализовывался для общего случая, когда фактически нет данных, кроме информации о связях между объектами, однако при использовании алгоритма на реальных данных в конкретной сфере стоит также учитывать описание объектов, и искать похожие не только на основе наибольшего значения косинусного расстояния, но использование одинаковых слов и слов-синонимов.

7. Список литературы

[1] URL: https://networkrepository.com/internet-industry-partnerships.php

[2] URL: http://orgnet.com/netindustry.html

[3]URL:https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81_%D1%83%D0%B4%D0%BE%D0%B1%D0%BE%D1%87%D0%B8%D1%82%D0%B0%D0%B5%D0%BC%D0%BE%D1%81%D1%82%D0%B8

[4] URL: https://ru.wikipedia.org/wiki/Word2vec

[5] Bharti R., Gupta D. (2019) Recommending Top *N* Movies Using Content-Based Filtering and Collaborative Filtering with Hadoop and Hive Framework. In: Kalita J., Balas V., Borah S., Pradhan R. (eds) Recent Developments in Machine Learning and Data Analytics. Advances in Intelligent Systems and Computing, vol 740. Springer, Singapore.

[6] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. 2019. Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches. In Thirteenth ACM Conference on Recommender Systems (RecSys '19), September 16–20, 2019, Copenhagen, Denmark.ACM, New York, NY, USA, 10 pages.

[7] Neerja Doshi, Recommendation Systems — Models and Evaluation. 2018, Jun 19, Towards Data Science. URL: https://towardsdatascience.com/recommendation-systems-models-and-evaluation-84944a84fb8e

[8] Гасников, А.В. Лекции по случайным процессам: учебное пособие / А. В. Гасников, Э. А. Горбунов, С. А. Гуз и др. ; под ред. А. В. Гасникова. – Москва : МФТИ, 2019. – 285 с. ISBN 978-5-7417-0710-4

[9] Зенченко, А. Рекомендательные системы, основанные на графах. // Хабр – 2020. – 6 ноября [Электронный ресурс]. URL: https://habr.com/ru/company/epam_systems/blog/526748/

[10] Козловский, В.Е. Теоретический обзор методов кодирования графов / В.Е. Козловский, Е.Е. Лунева. URL: http://earchive.tpu.ru/handle/11683/52654

[11] Кузнецов, А. Графовые рекомендации групп в Одноклассниках. // Хабр – 2020. – 14 мая [Электронный ресурс] URL: https://habr.com/ru/company/odnoklassniki/blog/499192/

8. Приложения

Приложение 1

Тип процессора: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx

Объем кэш-памяти второго уровня: 512 Кб

Тактовая частота: 2.10 ГГц

В Приложении 1 представлен код, реализующий рекомендательную систему.

```
def generate_walk(G, node):
    # генерация случайного блуждания
    nodes = list(range(len(G)))
    walk_len = 15
    neighbors = list(G.neighbors(node))
    A = np.zeros(len(G))
    np.put(A, neighbors, 1)
    D = sum(A)
    if D!=0:
        walk = [node]
        cur_state = node
        for t in range(1, walk_len):
            pk = np.divide(A, D)
            cur_state = random.choices(nodes,  weights=pk)[0]
            walk.append(cur_state)
            neighbors = list(G.neighbors(cur_state))
            A = np.zeros(len(G))
            np.put(A, neighbors, 1)
            D = sum(A)
    else:
        walk = [node]*walk_len
    return walk
```

```
def cold_start(who):
    degr = {i:G.degree[i] for i in range(len(G.nodes()))}
    if who == 'user':
        # проблема "холодного старта" для пользователя
        top_k = dict(sorted(degr.items(), key=lambda x: x[1], reverse= True))
        return list(top_k.keys())[:5]
    else:
        # проблема "холодного старта" для узла
        top_k = dict(sorted(degr.items(), key=lambda x: x[1]))
        return list(top_k.keys())[0]


def recommendation(base):
    rec = []
    n = len(base)
    k = int(n**(-1)*5 + 1)
    cold = (base == ['None'])
    if cold == False:
        isolated = [b for b in base if nx.is_isolate(G, b)]
        base = [b for b in base if b not in isolated]
    if cold == False and len(isolated)!=n:
        nodes = np.array(G.nodes())
        nodes = np.delete(nodes, base)
        for b in base:
            topn = [t[0] for t in model.wv.most_similar(b, topn=k)]
            rec.extend(topn)
        cold_node = cold_start('node')
        if cold_node not in isolated:
            rec.append(cold_node)
```

```
    else:
        rec = cold_start('user')
    return set(rec)
```

Приложение 2

```
# Основные характеристики графа
description = {'Количество узлов': len(G_real),
               'Количество ребер': len(G_real.edges()),
               'Минимальный узел': min(G_real.nodes()),
               'Максимальный узел': max(G_real.nodes()),
               'Средняя степень узла': round(len(G_real.edges())/len(G_real)),
               'Плотность': round(nx.density(G_real),5),
               'Диаметр': nx.diameter(G_real)}
df_description = pd.DataFrame.from_dict(description, orient='index')
nx.draw(G_real, node_color='lightblue', node_size=50)
# распределение узлов реального графа
n = len(G_real.nodes())+1
degree = np.array([G_real.degree[i] for i in range(1,n)])
count = np.unique(degree, return_counts=True)
plt.bar(count[0], count[1], width=1.5, color='lightblue')
plt.xlabel('Степень')
plt.ylabel('Количество узлов');
# создание обучающей выборки
n = 150 # узлы
e = n*4 # ребра
G_test = nx.dense_gnm_random_graph(n, e)
G = G_test.copy()
df = pd.DataFrame(None)
df['x'] = [edge[0] for edge in list(G_test.edges())]
df['y'] = [edge[1] for edge in list(G_test.edges())]
```

```
df['egde'] = list(G_test.edges())
X_train, X_test, y_train, y_test = train_test_split(df['x'], df['y'], test_size=0.2,
shuffle=True)
for index in X_test.index:
    G.remove_edge(*df.iloc[index, 2])
x_nodes = [list(X_test[:i]) for i in range(1, len(X_test))]
y_nodes = [list(y_test[:i]) for i in range(1, len(y_test))]
# распределение степеней узлов
n = len(G_test.nodes())
degree = np.array([G_test.degree[i] for i in range(n)])
count = np.unique(degree, return_counts=True)
plt.bar(count[0], count[1], width=1.5, color='lightblue')
plt.xlabel('Степень')
plt.ylabel('Количество узлов');
# проверка качества
all_nodes = [generate_walk(G, node) for node in G.nodes()]
words = [" ".join(map(str, node)) for node in all_nodes]
model = gensim.models.Word2Vec(all_nodes, min_count=0)
# проверка качества
quality_total = pd.DataFrame(None,
                             columns=['precision', 'recall', 'y_nodes', 'recommendation'])
for i in range(len(x_nodes)):
    rec_4_user = recommendation(x_nodes[i])
    guessed = rec_4_user&set(y_nodes[i])
    precision = len(guessed) / len(rec_4_user)
    recall = len(guessed) / len(y_nodes[i])
    quality_total = quality_total.append({'precision':round(precision,2),
                                          'recall':round(recall,2),
                                          'y_nodes':y_nodes[i],
```

'recommendation':rec_4_user},

ignore_index=True)

Приложение 3

Приложение 3 содержит вывод формулы для получения вектора переходных вероятностей для шага t=1.

```
from sympy import *

a11, a12, a13, a14, a15 = symbols('a11 a12 a13 a14 a15')
a21, a22, a23, a24, a25 = symbols('a21 a22 a23 a24 a25')
a31, a32, a33, a34, a35 = symbols('a31 a32 a33 a34 a35')
a41, a42, a43, a44, a45 = symbols('a41 a42 a43 a44 a45')
a51, a52, a53, a54, a55 = symbols('a51 a52 a53 a54 a55')

A = Matrix([[a11, a12, a13, a14, a15],
            [a21, a22, a23, a24, a25],
            [a31, a32, a33, a34, a35],
            [a41, a42, a43, a44, a45],
            [a51, a52, a53, a54, a55]])
Sum1, Sum2, Sum3, Sum4, Sum5 = symbols('Sum1 Sum2 Sum3 Sum4 Sum5')
D = diag(Sum1, Sum2, Sum3, Sum4, Sum5)
P = D.inv() @ A
p1, p2, p3, p4, p5 = symbols('p1 p2 p3 p4 p5')
p0 = Matrix([[p1], [p2], [p3], [p4], [p5]])
P1 = P.T @ p0
```