Санкт-Петербургский государственный университет

Прикладная математика и информатика

Выпускная квалификационная работа

# Сравнение некоторых методов машинного обучения на примере построения рекомендательных систем

Выполнила:
Ситчихина М.С., группа 421
Научный руководитель:
к.ф-м.н Ананьевский М.С
Кафедра теоретической кибернетики

Санкт-Петербург

2022

Saint Petersburg State University

Applied Mathematics and Informatics

Graduation Project

# Comparison of some machine learning methods based on the example of building recommender systems

Perfomed:
Sitchikhina M.S., группа 421
Scientific Supervisor:
Candidate of Physical and Mathematical Sciences
Ananevskii M.S
Control and Processing of
Information in Cybernetic and Robotic Systems

Saint Petersburg

2022

# Оглавление

# 1. Введение

Задача рекоммендательних систем предсказать желания пользователя и предложить товар, который будет интересен ему.

Свой прогноз система строит на активности клиента: оценки на купленные им продукты, история покупок, схожее поведение с другими пользователями, история поиска в системе и так далее. На основе собранных данных формируется список рекомендаций для клиента, которые будут релевантны для него и смотивируют на покупку или совершение целевого действия, выгодного для бизнеса.

Чем лучше система угадывает желания клиента, тем больше промотируется нужный продукт, выручка значительно увеличивается.
Следовательно, результатом хорошей рекоммендательной системы является:

- повышение лояльности и удовлетворенности клиента;
  Персонализированная рекомендация позволяет расположить клиента к компании, потому что он чувствует, что его интересы и желания учитываются.

- увеличение выручки;
  Засчет того, что клиенту могут предлагаться товары, которые он сам бы никогда не стал искать.

Применяя рекомендательную систему, бизнес не только увеличивает свой заработок, но и улучшает опыт пользователя от использования продукта. Отсюда следует вывод, что рекомендательные системы нужны в любой сфере от киносервисов до продаж бытовой технки. Именно поэтому задача разработки такой системы с наилучшей точностью рекомендации является важной.
Для ее решения будут исследованы несколько методов машинного обучения, построенных на разных математических моделях, и будет проведено сравнение эффективности исполнения на открытых данных с сайта (1). Все алгоритмы реализованы на языке Python с помощью открытой библиотеки scikit-learn и ее модулей (2), фреймворка PyTorch (3), коды методов представлены в приложениях.

## 2. Постановка задачи

Рассмотрим киносервис, который агрегирует оценки, выставляемые пользователями, и будем преследовать цель — подобрать такие фильмы, которые потенциально могут быть интересны для пользователя.
Для того, чтобы составить рекомендацию на отдельного пользователя, можно решать задачи прогнозирования оценки пользователя или оценки вероятности того, что пользователь выбрал бы некоторый фильм для просмотра.
Оба варианта задач коррелируют между собой: чем выше прогнозируемая оценка пользователя, тем вероятность просмотра будет выше; чем ниже вероятность просмотра, тем ниже должна быть оценка.

## 3. Описание данных

Movielens (1) — это платформа, в которой люди делятся своим мнением о фильмах для того, чтобы получить персонализированную подборку фильмов, которые будут им по вкусу. Начиная с 1988 года разработичики системы из лаборатории Grouplens проводили множество экспериментов по улучшению рекомендательных систем (4), что также позволило создать наборы реальных данных, пригодных для использования в машинном обучении.

Датасеты с сайта Movielens часто используются в статьях о коллаборативной фильтрации данных (5), задачах top-N (6) и других подзадач рекомендательных систем.

Исполнение алгоритмов будет оцениваться на версии датасета "movielens 100k dataset"(7), состоящей из 100 тысяч оценок, полученных от 1000 пользователей по 1600 фильмам. Пользователь может оценить фильм от 1 до 5, нулем обозначается отсутствие оценки, то есть фильм не был просмотрен или не оценен.

| | user_id | item_id | ratings |
|---|---|---|---|
| 0 | 196 | 242 | 3 |
| 1 | 186 | 302 | 3 |
| 2 | 22 | 377 | 1 |
| 3 | 244 | 51 | 2 |
| 4 | 166 | 346 | 1 |

Таблица 1. Пример набора данных

Важно учитывать, что при работе с данными часто могут возникать следующие проблемы (8):

- объем;

  Когда данные занимают много места по памяти, то это ведет не только к снижению скорости выполнения алгоритма, но также требует большей вычислительной мощности компьютера.

- разреженность;

  Если большая часть данных имеет нулевое или пустое значене, то работа с таким датасетом усложняется. Большой перевес в сторону одного класса грозит тем, что практически любой простой алгоритм будет прогнозировать значение этого класса, чтобы повысить эффективность работы.

  Его больше, следовательно, он вероятнее совпадает с тестовой выборкой и функция потерь будет меньше.

- достоверность.

  Невозможно отследить, что собранные данные полностью отражают реальность. Рекомендации, построенные на мнении пользователей, напрямую от них зависят, поэтому если в системе оценивания возникают мошенники, которые намерено занижают или завышают рейтинг, то практическая польза от рекомендации теряется.

### 3.1. Обработка датасета

Прежде всего, чтобы сконцетрировать внимание на самих алгоритмах, а не на методах их оптимизации, было принято решение выбрать датасет на 100 тысяч оценок, который является относительно маленьким в мире больших данных.

На рис.1 представлено распределение оценок фильмов от 1 до 5. Оно похоже на нормальное, поэтому будем считать, что все данные корректны, то есть все пользователи были добропорядочными и оценивали фильмы, опираясь только на свое мнение.

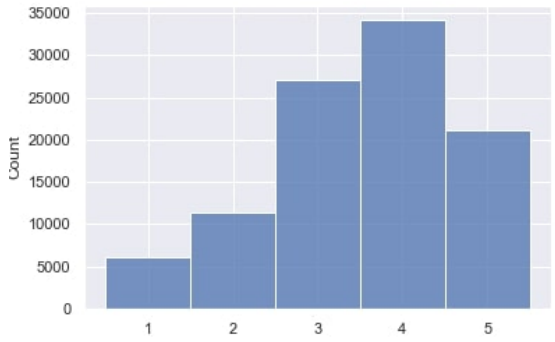



Рис. 1. Распределение оценок пользователей

При анализе данных было выяснено, что данные довольно разреженные — более 90 процентов оценок не проставлены. Так как идеальный датасет в рассматриваемой задаче — это массив данных, где соотношение классов примерно равное, то такой вывод усложняет решение.

Преобразуем исходный датасет в матрицу, в которой $i$-столбец будет являться оценками фильмов, просмотренных $i$-м пользователем.

Справиться с разреженными способами можно либо с помощью усложнения алгоритма, либо препроцессингом данных.
Для начала оставим только тех пользователей, которые дали свое мнение на не менее чем 20 фильмов. Число 20 взято, потому что в таком случае число фильмов

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | 4 | 0 | ... | 5 | 0 | 0 |
| 3 | 0 | 0 | ... | 0 | 0 | 5 |
| 4 | 0 | 0 | ... | 0 | 0 | 0 |
| | | | ... | | | |
| 0 | 0 | 0 | ... | 0 | 0 | 0 |
| 0 | 0 | 0 | ... | 0 | 0 | 0 |
| 0 | 0 | 0 | ... | 0 | 0 | 0 |

Таблица 2. Перобразование исходных данных

и пользователей примерно равно — матрица становится размера 939 на 943.
И будем считать, что если оценка выше трех, то условно фильм был просмотрен не зря и подобные фильмы пользователю нравятся, и будем относить его к классу 1, если ниже — то его лучше не рекомедовать, и такому фильму будем сопоставлять 0. Дальнейшие модификации датасета рассмотрим в каждом методе по отдельности.

В итоге получена матрица размера 939 на 943, где на $(i, j)$ позиции стоит оценка за $i$-ый фильм $j$-го пользователя.

# 4. Рекомендация методом логистической регрессии

## 4.1. Мотивация

Метод логистической регрессии применяется для классификации объектов с помощью моделирования вероятностной оценки принадлежности тому или иному классу (9).

В решении можно так же использовать линейную регрессию, но она имеет ряд недостатков:

- Восприимчива к выбросам;
- В силу линейности не может улавливать более сложные взаимосвязи между классами;
- На выходе может выдавать отрицательное или больше 1 число, что невозможно в случае прогноза вероятности.

Для того, чтобы вывести логистическую регрессию из линейной, вместо $y$ в выражении линейной регрессии рассмотрим вероятность $P$ принадлежности к некоторому классу. Для простоты записи возьмем однофакторную регрессию:

$$y = \alpha_0 + \alpha_1 x. \tag{1}$$

Так как необходимо, чтобы прогнозируемая вероятность принимала не любое значение, а строго от 0 до 1, рассмотрим преобразование:

$$P = \alpha_0 + \alpha_1 x,$$
$$\frac{P}{1-P} = \alpha_0 + \alpha_1 x.$$

Если оставить все таким образом, то область допустимых значений для множества признаков уменьшится в силу ограничений на $P$, что усложнит моделирование. Необходимо, чтобы переменные могли варьироваться в любом диапозоне. Для этого рассмотрим логарифм от выражения слева:

$$\ln(\frac{P}{1-P}) = \alpha_0 + \alpha_1 x. \tag{2}$$

Теперь область значений — все множество действительных чисел, и можно приступить к вычислению функции вероятности:

$$e^{\ln(\frac{P}{1-P})} = e^{\alpha_0+\alpha_1 x}$$
$$\frac{P}{1-P} = e^{\alpha_0+\alpha_1 x}$$
$$P = e^{\alpha_0+\alpha_1 x} - e^{\alpha_0+\alpha_1 x}P$$
$$P(1 + e^{\alpha_0+\alpha_1 x}) = e^{\alpha_0+\alpha_1 x}$$
$$P = \frac{e^{\alpha_0+\alpha_1 x}}{1 + e^{\alpha_0+\alpha_1 x}}$$
$$P = \frac{1}{1 + e^{-(\alpha_0+\alpha_1 x)}}.$$

Теперь перепишем полученный результат в контексте задачи рекомендации. Признаками является история просмотров пользователя, функция должна прогнозировать вероятность того, что пользователю понравится фильм, обозначим это событием A. Тогда получим, что вероятность наступления события $A$ при условии вектора признаков $x$:

$$P(A = 1|x) = \frac{1}{1 + e^{x\theta^T}}, \tag{3}$$

где $\theta$ — параметры регрессии, а $x$ – вектор признаков.

### 4.2. Постановка задачи

Пусть $V$— множество пользователей(viewers), $M$— множество фильмов(movies).
Задана матрица размера $M \times V(|M| = 939, |V| = 943)$, где $x_{ij}$ — флаг удовлетворенности $j$-го пользователя от просмотра $i$-го фильма.
Необходимо оценить вероятность пользователей посмотреть $i-$ый фильм.

В соответствии с моделью регрессии вероятность того, что $k-$му пользователю понравится $i-$ый фильм равна:

$$\forall k \in V \; P_k = \frac{1}{1 + e^{x\theta^T}}, x\theta^T = \theta_0 + \sum_{\substack{j=1 \\ j\neq i}}^{M} \theta_j \cdot x_{jk}. \tag{4}$$

Требуется найти такое значение вектора $\theta = [\theta_0, \theta_1, \ldots, \theta_M]$, при котором достигается

минимум функции потерь:

$$f(x, P, \theta) \to \min_{\theta} \tag{5}$$

### 4.3. Теоретическая часть

Параметры в логистической регрессии оцениваются с помощью логистической функции ошибки(log loss):

$$-\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\{x_i = 1\} \ln P(x_i) + \mathbb{1}\{x_i = 0\} \ln(1 - P(x_i)) \tag{6}$$

Выведем функцию ошибки — через метод максимального правдоподобия (10). Его суть состоит в том, чтобы максимизировать функционал правдоподобия. Функцией правдободобия называется совместное распределение выборки, математически функция равна произведению условных вероятностей наблюдать каждое событие при заданных параметрах. В задаче рассматривается бинарная классификация, то есть событие принимает значения 0 или 1, это позволяет интерпретировать датасет как распредление Бернулли. Тогда функция правдоподобия примет следующий вид:

$$L(\theta) = \prod_{i=1}^{N} \sigma(\theta^T x^i)^P (1 - \sigma(\theta^T x^i))^{1-P}, \tag{7}$$

за $\sigma$ — обозначена сигмоидная функция, $P$ — вероятность наступления успеха (равенства 1). Чтобы облегчить вычисления, прологорифмируем:

$$\ln L(\theta) = \sum_{i=1}^{N} P \ln(\sigma(\theta^T x^i)) + (1 - P) \ln(1 - \sigma(\theta^T x^i)) \tag{8}$$

В машинном обучении более привычно минимизировать функцию, поэтому верно равенство:

$$\max \ln L(\theta) = \min(-\ln L(\theta)). \tag{9}$$

Итого в постановке задачи логистической регрессии (5) равносильно минимизации функционала максимального правдоподобия:

$$-\sum_{i=1}^{N} P \ln(\sigma(\theta^T x^i)) + (1 - P) \ln(1 - \sigma(\theta^T x^i)) \to \min_{\theta} \tag{10}$$

Если бы использовали среднюю квадратичную ошибку, как в линейной регрессии, то в силу нелинейности функции (3), функция потерь имела бы несколько локальных минимумов. При обычном градиентном спуске наличие локальных минимумов сильно усложняет задачу, нельзя гарантировать, что алгоритм нашел именно глобальный минимум, поэтому стоит использовать log loss, чтобы минимизировать ошибку.
Выведем шаг градиентного спуска. Для этого продифференцируем функцию потерь(log loss) по ее параметру:

$$\begin{aligned} LL(\theta) &= p\ln(\sigma(\theta^T x)) + (1-p)\ln(1-\sigma(\theta^T x)), \\ -\frac{\partial LL(\theta)}{\partial \theta_i} &= -\frac{\partial LL(\theta)}{\partial \sigma(\theta^T x)}\frac{\partial \sigma(\theta^T x)}{\partial \theta_i} = \\ &= -\frac{\partial LL(\theta)}{\partial \sigma(\theta^T x)}\frac{\partial \sigma(\theta^T x)}{\partial \theta^T x}\frac{\partial \theta^T x}{\partial \theta_i}. \end{aligned}$$

Распишем каждый из множителей:

$$\begin{aligned} \frac{\partial LL(\theta)}{\partial \sigma(\theta^T x)} &= \frac{\partial(p\ln(z)+(1-p)\ln(1-z))}{\partial z} = \\ &= \frac{p}{z} - \frac{1-p}{1-z} = \frac{p}{\sigma(\theta^T x)} - \frac{1-p}{1-\sigma(\theta^T x)}, \\ \frac{\partial \sigma(\theta^T x)}{\partial \theta^T x} &= \frac{\partial \sigma(z)}{\partial z} = \frac{d}{dz}(1+\exp^{-z})^{-1} = \\ &= -(1+\exp^{-z})^{-2}\frac{d}{dz}(1+\exp^{-z}) = \exp^{-z}(1+\exp^{-z})^{-2} = \\ &= \frac{\exp^{-z}+1-1}{(1+\exp^{-z})^{-2}} = (1-\frac{1}{(1+\exp^{-z})})\frac{1}{(1+\exp^{-z})} = \\ &= \sigma(\theta^T x)(1-\sigma(\theta^T x)), \\ \frac{\partial \theta^T x}{\partial \theta_i} &= x_i. \end{aligned}$$

И подставим все вычисления в начальную производную:

$$\begin{aligned} -\frac{\partial LL(\theta)}{\partial \theta_i} &= -(\frac{p}{\sigma(\theta^T x)} - \frac{1-p}{1-\sigma(\theta^T x)})\sigma(\theta^T x)(1-\sigma(\theta^T x))x_i = \\ &= -(p(1-\sigma(\theta^T x)) - (1-p)\sigma(\theta^T x)) = (\sigma(\theta^T x) - p)x_i. \end{aligned}$$

В итоге получаем, что на каждом шаге спуска параметр пересчитывается по формуле:

$$\theta_{k+1} = \theta_k - \alpha(\sigma(\theta^T x) - p)x_i, \tag{11}$$

где $\alpha$ — это шаг, с которым изменяется параметр.

Для того, чтобы была возможность сравнить все методы между собой, будем оценивать точность (accuracy), которая показывает отношение числа правильных ответов ко всем.

$$accuracy = \frac{|prediction == sample|}{|prediction|}. \tag{12}$$

### 4.4. Реализация

Будет использовано готовое решение LogisticRegression() из библитотеки sklearn.

Прежде всего необходимо преобразовать матрицу оценок так, чтобы осталось только два класса — 1 и 0 в соответствии с последним параграфом из раздела 3.1 Обработка датасета.
Выберем для прогноза первый фильм. Разобьем матрицу оценок на обучающую и тренировочную выборки с помощью функции traintestsplit() таким образом, чтобы тест составлял 20 процентов от общей выборки, тогда в тестовой матрице будет 188 пользователей и их оценки на первый фильм, а в выборке для обучения — 751.

Обучим модель на тренировочном датасете и сделаем прогноз удовлетворенности пользователей фильмом. Точность на тесте окажется равной 90 процентам. Можно предположить, что модель переобучилась, так как значение метрики высокое. Но точность на тренировочном датасете не уступает тесту — 84%, то есть модель действительно хорошо справляется с предсказанием.
Осталось сделать прогноз оценок для других фильмов и рассчитать среднюю точность на всех данных.

В результате рекомендательная система методом логистической регрессии с точностью 85 процента предсказывает понравится пользователю фильм или нет.

# 5. Рекомендация методом кластеризации

## 5.1. Мотивация

Это метод машинного обучения, который представляет собой разделение объектов на группы таким образом, чтобы объекты одного класса были похожи между собой. В настоящем контексте это означает, что в одном кластере должны быть люди с одинаковыми интересами о фильмах.

Конкретному пользователю можно будет спрогнозировать оценку на непросмотренный им фильм, как среднее арифметическое от всех оценок, поставленных пользователями из его кластера. Это похоже на то, что было проделано в регрессии, но здесь среднее будет считаться по людям из одного кластера, а значит, что все пользователи разделяют вкусы друг друга, поэтому среднее будет выражать более точное мнение кластера о фильме.

Существует множество способов кластеризации, будем использовать алгоритм K-means, он относится к категории машинного обучения без учителя, то есть не требует разметки данных. Формально, как и алгоритм Ллойда (11), K-means стремится разбить исходное множество на заданное количество групп, при этом минимизировать среднеквадратичное отклонение на точках каждого кластера.

## 5.2. Постановка задачи

Пусть $V$ — множество пользователей, $K$ — множество меток классов. На множестве $V$ задана метрика $\rho(v_i, v_j)$, вычисляющая расстояние между двумя объектами множества. Необходимо каждому объекту $v_i \in V$ сопоставить метку $k_i \in K$ так, чтобы среднее расстояние между объектами внутри одного кластера было минимальным или, наоборот, среднее расстояние между объектами разных кластеров было макси-

мальным:

$$\sum_{i=1}^{|K|} \frac{1}{|K_i|} \sum_{\substack{v_m, v_n \in K_i, \\ m \neq n}} \rho(v_m, v_n) \to min,$$

$$\sum_{i,j \in \{1,|K|\}} \sum_{\substack{v_m \in K_i, \\ v_n \in K_j}} \rho(v_m, v_n) \to max$$

### 5.3. Теоретическая часть

Прежде всего необходимо указать $k$ — количество искомых кластеров, затем случайным образом выбираются $k$ точек — центроид, рассчитывается расстояние от каждой точки до центроид, и кластер формируется из ближайших к нему точек. На следующей итерации вычисляется новая центроида для каждой группы, как среднее точек кластера. Так продолжается до тех пор, пока центроиды не перестанут изменяться.

Важной задачей является выбрать нужное значение $k$, чтобы достичь наилучшей рекомендации. Существует несколько способов для оптимизации поиска. Рассчитаем для каждой точки метрику, называемую silhouette score:

$$s(v_i) = \frac{b(v_i) - a(v_i)}{max(b(v_i), a(v_i))},$$

$$b(v_i) = \min_{I \neq J} \frac{1}{K_J} \sum_{v_j \in K_J} d(v_i, v_j),$$

$$a(v_i) = \frac{1}{|K_I| - 1} \sum_{\substack{v_j \in K_I, \\ i \neq j}} d(v_i, v_j),$$

где $a(i)$ — среднее расстояние от точки $v_i$ до других точек в кластере $I$, $b(v_i)$ — наименьшее среднее расстояние от точки $v_i$ кластера $I$ до точек других кластеров, $d(v_i, v_j)$ — расстояние между точками. Если кластер состоит из одной точки, то по определнию $a(v_i) = 0$.

Фактически $a(i)$ показывает, насколько точка схожа со своим кластером, чем меньше значение, тем лучше она подходит. В свою очередь $b(v_i)$ отвечает за похожесть с другими кластерами, если его значение велико, то $v_i$ не подходит в другие группы.

Для того, чтобы осознать, как вычисляется $d(v_i, v_j)$, предположим, что пользователи давали оценки только на два фильма. Тогда каждого пользователя можно представить на $R^2$, задав парой $(x, y)$ — оценки на первый и второй фильм, очевидно, что в таком случае $d(v_i, v_j)$ — это корень из суммы квадратов разностей координат. В общем случае, получается что пользоваетель $v_i$ задается вектором $(x_1, x_2, \ldots, x_{|M|})$ и $d(v_i, v_j)$ становится евклидовым расстоянием в $|M|-$мерном пространстве:

$$d(v_i, v_j) = \sqrt{\sum_{k=1}^{|M|}(x_i^2 - y_j^2)}\ , \tag{13}$$

где $v_j = (y_1, y_2, \ldots, y_{|M|})$.

Заметим, что $s(v_i)$ лежит в промежутке от -1 до 1. Таким образом, если $s(v_i)$ около -1, отсюда следует, что $v_i$ не подходит к этому кластеру, если $s(v_i)$ около 1 — точка была кластеризована верно, $s(v_i) = 0$ — точка лежит на границе двух кластеров.

Определив, что значит эта метрика для каждой точки по отдельности, становится ясно, что среднее значение $s(v_i)$ показывает, насколько эффективна была кластеризация. Единственный минус такой оценки — это переобучаемость, поэтому важно также смотреть на количество элементов в каждом классе.

Как у любого алгоритма, у K-means есть свои плюсы и минусы. К достоинствам алгоритма можно отнести:

- интуитивно понятный результат;
- прост в вычислении, то есть выполяется относительно быстро;
- динамичность классов;

  Изначальные классы могут сильно отличаться от коненчных, так как на каждой итерации центроиды пересчитываются.
- есть сходимость.

И можно выделить следующие недостатки (12):

- необходимо дополнительно оценивать количество кластеров;
- ограничение на форму кластеров;

  В силу того, что обучение основывается на вычислении расстояния, алгоритм будет эффективнее работать на тех датасетах, где данные могут быть разделены на сферические кластер;
- не чувствителен к выбросам.

### 5.4. Реализация

K-means возьмем из модуля sklearn.cluster, а оценивать качество будем функцией silhouette score из sklearn.metrics.
Как и в прошлом методе, сразу выберем пользователей, которые оценили хотя бы 20 фильмов и будем применять алгоритм на матрице, в которой проставлены рейтинги фильмов, без бинарного преобразования.

Запустив алгоритм на $k$ от 2 до 12, были построены прогнозы для каждого $k$. Необходимо соблюсти баланс между оптимальной ошибкой и количеством классов, так как чем больше групп, тем меньше там человек.

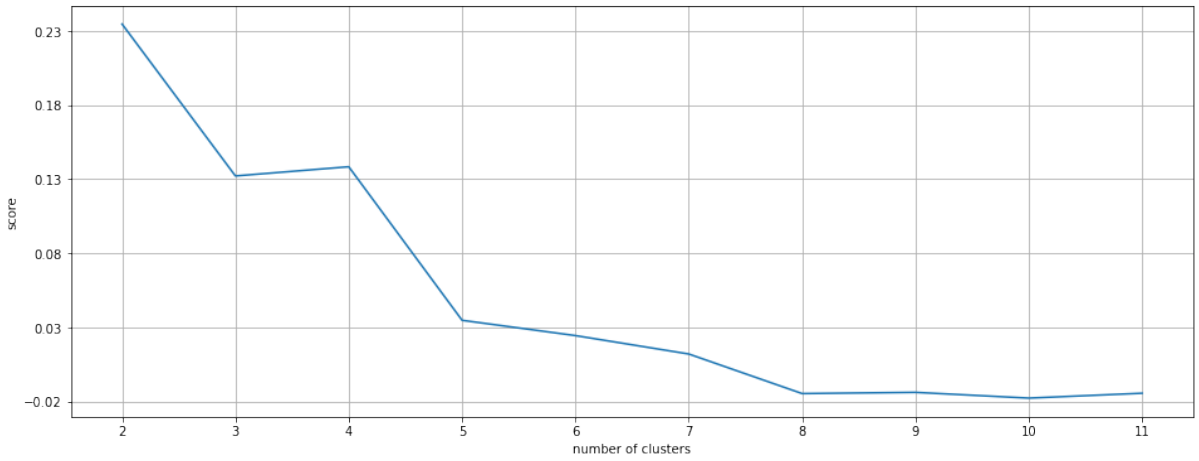



Рис. 2. Значение silhouette score

При $k = 2$ размер одного класса сильно превышает размер второго, при $k = 4$ образуются два маленьких класса размером около 100 объектов и два больших, но при $k = 3$ распределение классов удовлетворительное.

Запустим K-means с параметром $k = 3$, в результате каждому пользователю будет соответствовать номер класса от 0 до 2.

По аналогии с тестированием в разделе о логистической регрессии будем оценивать точность по каждому фильму отдельно, а затем найдем среднее значение. Всем пользователям из тестовой выборки сопоставим среднее за фильм по оценкам его кластера и применим бинарное преобразование — 1, если оценка за фильм больше 3, в противном случае — 0.

Точность [1] на каждом кластере представлена в таблице 3.

| **Номер кластера** | **Количество пользователей в кластер** | **Точность** |
|---|---|---|
| 0 | 133 | 0.948 |
| 1 | 268 | 0.866 |
| 2 | 542 | 0.916 |

Таблица 3. Среднее значение метрики accuracy на каждом кластере

Рекомендательная система, построенная с помощью метода K-means, может с 90-процентной точностью предсказать будет удовлетворен пользователь просмотром фильма или нет.

[1] Доля правильных ответов на тестовой выборке.

# 6. Рекомендация с помощью нейронной сети

## 6.1. Мотивация

Нейронные сети хороши тем, что на базе вложенных в них алгоритмов формируют более точное решение. В отличие от обычных алгоритмов машинного обучения, которые дают ответ, основываясь на данных для обучения. Данное отличе может быть полезным, когда нет очевидных связей между параметрами модели и ее выходом, как получилось в рассматриваемом в этой работе датасете. Неизвестен жанр фильма, его актеры или сценаристы, то есть нет признаков, за которые можно зацепиться и увидеть корреляцию человеческим глазом. Будем пытаться предсказать оценки пользователей на последний фильм.

## 6.2. Постановка задачи

Пусть дана матрица размера $V \times M$ ($|M|= 939$, $|V|= 943$), элементы которой равны оценке, выставленой пользоватем на фильм, $k$ — номер фильма, оценки пользователей которого будут прогнозироваться. Необходимо $\forall v \in V$ сопоставить 1, если пользователь $v$ был удовлетворен фильмом, и 0 в обратном случае. При этом значение функция потерь $f(r_k, \theta)$, где $r_k$ — вектор, состоящий из реакций пользователей на $k-$ый фильм, а $\theta$ — вектор параметров нейронной сети, должно стремиться к минимуму.

## 6.3. Структура нейронной сети

Нейронные сети состоят из входного слоя, затем может быть один или несколько скрытых слоев и выходной слой, отвечающий за результат. Каждый слой состоит из нейронов.

На вход нейрон обычно принимает числа в диапозоне от 0 до 1 или от -1 до 1, аналогично, с выходом, поэтому перед тем, как приступить к работе с нейронной сетью, данные должны быть нормализованы. Начальный слой можно рассматривать как линейную регрессию:

$$\sum_{1}^{n} \theta_i x_i, \tag{14}$$

$x_i$ — это данные. поданные на вход нейронной сети, $\theta_i$ — это вес, присвоенный нейронам во входном слое.

В процессе обучения веса меняются в зависимости от влияния значения $x_i$ на результат. Чем важнее является нейрон, тем больше у него будет вес.

Определим структуру нейронной сети, выполняющей задачу классификации — она будет состоять из 4 линейных слоев, пусть на вход подается бинарный вектор размера $|M|-1$, где будет стоять 1 — если фильм понравился, его оценили 4 и 5, и 0 — если оценка была ниже или фильм не был просмотрен. На выходе будем получать 1 нейрон, который будет означать вероятность принадлежности к классу 1 или 0.

Стандартный выбор для нейронных сетей-классификаторов является гиперболический тангенс, возвращающий значения от -1 до 1, поэтому применим к последнему нейрону сигмоидную функцию, чтобы значение было от 0 до 1.

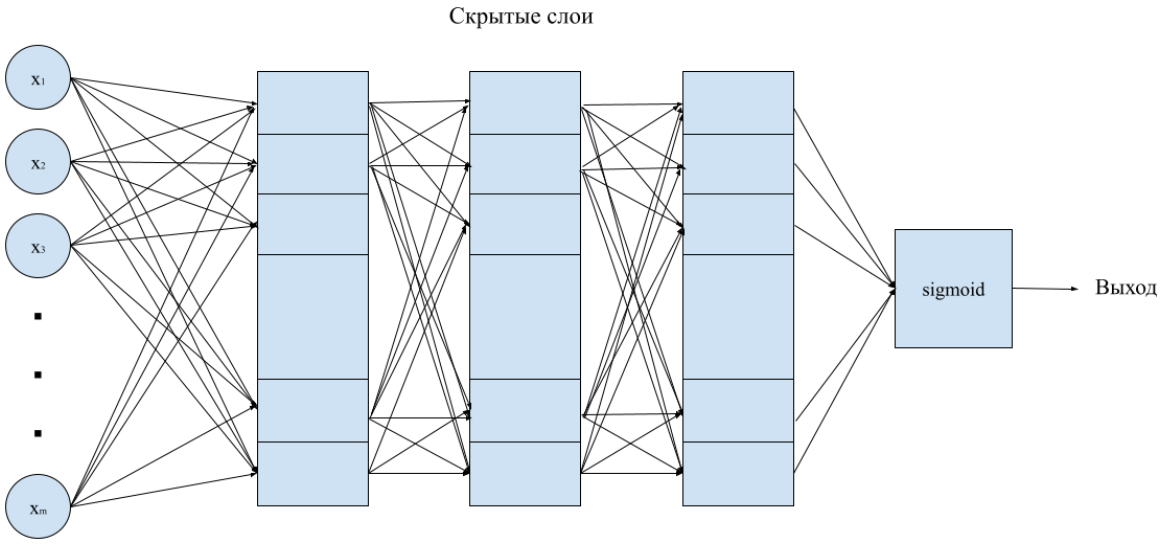



Рис. 3. Структура нейронной сети

**Функция активации**

Каждый нейрон скрытого слоя помимо входных и выходных значений характеризует функция активации.

Чаще всего используют сигмоидную функцию, которая упоминалась раннее: $f(x) = \frac{1}{1+e^{-x}}$, и гиперболический тангенс: $f(x) = \frac{e^{2x}-1}{e^{2x}+1}$.

Они отличаются областью значений, в рассматриваемой задаче лучше использовать сигмоидную функцию, так как ее область значений равна области значений вероятности, которую необходимо предсказать. В сетях-классификаторах принято использовать вторую функцию активации, поэтому к выходному слою будет в конце применена сигмоидная функция, чтобы было возможно интерпретировать результат нейронной сети, а для остальных слоев функцией активации будет оставлен гиперболический тангенс в качестве функции активации.

**Оптимизатор**

Он помогает ускорить обучение подбором параметров. В решении будет использована оптимизация с помощью Adam (13), он гарантирует быструю сходимость.
Adaptive Moment Estimation(сокр. Adam) смотрит на математическое ожидание и дисперсию, то есть на первый и второй моменты. В отличие от градиентного спуска Adam движется с меньшей скоростью, что позволяет минимизировать риск проскочить глобальный минимум.

Обучение нейронной сети немного отличается от обучения предыдущих алгоритмов. Из тренировочного сета формируются массивы небольшого размера, называемые батчами. Далее происходит обучение на эпохах, где одна эпоха — это тренировка сети по одному разу на каждом батче. Каждую эпоху формируются разные батчи из одного тренировочного массива. Чем больше эпох, тем лучше нейронная сеть обучается и минимизирует ошибку.

На каждой итерации Adam будет рассматривать один батч. Для выполнения Adam нужно передать несколько гиперпараметров:

- $\alpha$ — скорость обучения, обычно берут 0.001
- $\beta_1$ и $\beta_2$ — коэффициенты, регулирующие затухание первых и вторых моментов, соответственно. По умолчанию значения — 0.9 и 0.999
- $\epsilon$ — небольшое значение, чтобы при реализации недопустить деление на ноль.

Задача Adam состоит в том, чтобы найти параметр $\theta$, при котором математическре ожидание функции потерь будет минимальным. Для этого рассмотрим скользящие средние:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{15}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{16}$$

где $g_t$ – градиент, вычисленный на текущием батче, а $m_0$ и $v_0$ задаются равными нулю.

В силу начальных значений $m_0$ и $v_0$ оценки моментов будут смещены относительно нуля, поэтому следующие равенства будет неверны:

$$E[m_t] = E[g_t] \tag{17}$$

$$E[v_t] = E[g_t^2]. \tag{18}$$

Распишем первые несколько шагов для $m_t$:

$$\begin{aligned}
m_0 &= 0, \\
m_1 &= \beta_1 m_0 + (1 - \beta_1) g_1 = (1 - \beta_1) g_1, \\
m_2 &= \beta_1 m_1 + (1 - \beta_1) g_2 = \beta_1 (1 - \beta_1) g_1 + (1 - \beta_1) g_2, \\
m_3 &= \beta_1 m_2 + (1 - \beta_1) g_3 = \beta_1 \cdot (\beta_1 (1 - \beta_1) g_1 + (1 - \beta_1) g_2) + (1 - \beta_1) g_3 \\
&= \beta_1^2 (1 - \beta_1) g_1 + \beta_1 (1 - \beta_1) g_2 + (1 - \beta_1) g_3.
\end{aligned}$$

Отсюда можно вывести общую формулу для вычисления $m_t$:

$$m_t = (1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-i} g_i. \tag{19}$$

И подставим в формулу (11), чтобы доказать смещенность:

$$E[m_t] = E[(1-\beta_1)\sum_{i=0}^{t-1}\beta_1^t g_i = E[g_t](1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}g_i + \zeta = E[g_t](1-\beta_1^t) + \zeta. \quad (20)$$

Аппроксимируем $g_i$ с помощью $g_t$, тогда можно будет вынести $g_t$ из суммы, из-за приближения возникнет $\zeta$, осталось воспользоваться суммой геометрической прогрессии. Таким образом, получаем смещенную оценку первого момента, аналогично, можно доказать и для второго. Подкорректируем оба момента и получим:

$$\bar{m}_t = \frac{m_t}{1-b_1^t}, \quad (21)$$

$$\bar{v}_t = \frac{v_t}{1-b_2^t}. \quad (22)$$

$$(23)$$

Итого, формула для пересчета параметров будет выглядеть следующим образом:

$$\theta_t = \theta_{t-1} - \alpha\frac{\bar{m}_t}{\sqrt{\bar{v}_t}+\epsilon} \quad (24)$$

**Функция потерь**

В реализации будем использовать BCELoss — метрика, вычисляющая бинарную кросс-энтропию. Бинарная кросс-энтропия показывает, насколько отличается изначальное распределение классов от распределения, предсказанного нейронной сетью. Данная метрика является векторным продолжением функции ошибки log loss, рассмотренной раннее.

$$L = (l_1, l_2, \ldots, l_V), \quad (25)$$

$$l_i = -\theta_i\Big(\mathbb{1}\{y_i = 1\}\ln P(x_i) + \mathbb{1}\{y_i = 0\}\ln\Big(1 - P(x_i)\Big)\Big), \quad (26)$$

где $(x_i, y_i)$ — это пара значения из тестовой выборки и выхода нейронной сетки, $\theta_i$ — её параметры. И затем вычисляется среднее значение вектора $L$ — сумма всех координат на размерность вектора.

### 6.4. Реализация

Для построения нейронной сети будем использовать библиотеку torch.

Чтобы была возможность сравнить результаты алгоритмов, будем считать помимо

BCELoss, точность прогноза на каждой итерации обучения.

Заменим оценки в исходной матрице на 1 и 0, как описывалось в разделе 3.1 Обработка датасета, и рассмотрим бинарную матрицу удовлетворенности пользователей фильмами. Обучим на 50 эпохах. На Рис. 4 видно — точность с увеличением эпохи снижается с 70 процентов до 66, что вызывает с первого взгляда недоумение. Обратим внимание на значение не только точности, но и кросс-энтропии.

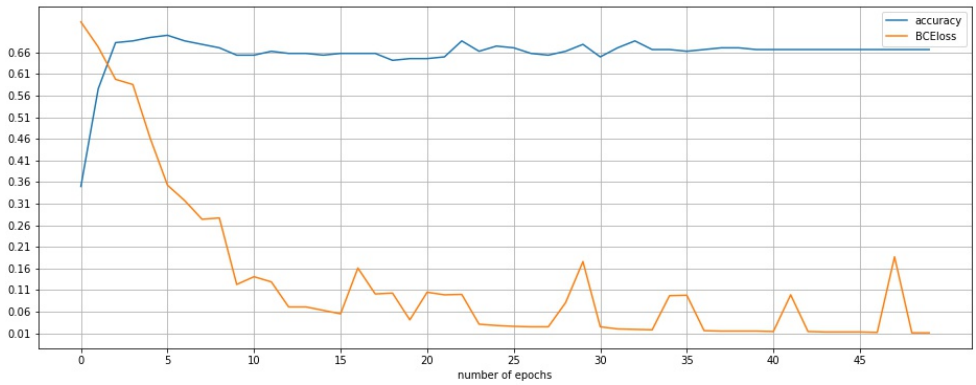



Рис. 4. BCELoss и accuracy

Метрика уменьшается с каждой эпохой, а точность не изменяется, отсюда следует вывод, что сеть переобучается, она все больше подстраивается под тренировочный датасет.

Таким образом, из-за того, что набор данных небольшой, нужно уменьшить количество итераций обучения, чтобы предотвратить переобучение. Примерно после десятой эпохи начинается переобучение, поэтому стоит остановить обучение на этом этапе.

Итоговая точность рекомендательной системы на базе нейронной сети с предсказанием будет оценен фильм пользователем положительно или нет равна примерно 67 процентам.

# 7. Заключение

В работе исследовалась задача построения рекомендательной системы на основе истории действий пользователя.

Изучено три метода машинного обучения: логистическая регрессиия, кластеризация алгоритмом K-means, линейная нейронная сеть — с целью выявить, какой из них покажет наилучшую эффективность рекомендации.

Помимо технической реализации алгоритмов было приведено их теоретическое описание. Были рассмотрены частые пробелемы, возникающие при работе с большими данными, и способы борьбы с ними.

Результативность прогноза оценивалась на одном датасете, состоящим из рейтингов фильмов, проставленных пользователями. С помощью машинного обучения прогнозировались фильмы, которые могут понравиться пользователю, а затем вычислялся процент правильных ответов на тестовой выборке — точность рекомендации.

| **Метод** | **Точность рекомендации** |
|---|---|
| Логистическая регрессия | 85 % |
| Кластеризация | 90 % |
| Нейронная сеть | 67 % |

Таблица 4. Результаты работы алгоритмов

Регрессия и кластеризация показывают очень хороший результат, а нейронная сеть из-за переобучения может выдавать только два правильных ответа из трех. Такая сеть с двумя скрытыми слоями очень простая, для улучшения результата есть смысл рассматривать более сложные структуры, требующие больше вычислительной мощности, времени обучения и так далее.

Несмотря на то, что у кластеризации точноть ответов выше, отсюда не следует, что этот метод наилучший вариант для рекомендательной системы. В силу особенностей алгоритма K-means добавление новых пользователей приводит либо к полному перезапуску алгоритма и перераспределению кластеров, либо расчету расстояний от нового пользователя ко всем кластерам, чтобы определить, куда его отнести. Оба случая вычислительно затратные.

Таким образом, рекомендательная система методом кластеризации показала наилучший результат, но с точки зрения практического применения разумнее пожертвовать незначительно точностью ответов и остановить выбор на регресси в рассматриваемой задаче.

# Список литературы


1. MovieLens. — Режим доступа: https://movielens.org (дата обращения: 2022-03-20).

2. Библиотека scikit-learn. — Режим доступа: https://scikit-learn.org/stable/ (дата обращения: 2022-03-20).

3. Модуль PyTorch. — Режим доступа: https://pytorch.org (дата обращения: 2022-03-20).

4. Harper F. Maxwell, A.Konstan. Joseph. The MovieLens datasets: History and context. // ACM Trans. Interact. Intell. Syst. 5. — 2015. — Vol. 4, no. 5.

5. Joint Neural Collaborative Filtering for Recommender Systems / Wanyu Chen, Fei Cai, Honghui Chen, Maarten de Rijke // AACM Trans. Intell. Syst. Technol. — 2019. — Vol. 37, no. 4.

6. Oliveira Samuel E. L., Victor Diniz Anisio Lacerda Luiz Merschmanm, Pappa Gisele L. Is Rank Aggregation Effective in Recommender Systems? An Experimental Analysis // AACM Trans. Intell. Syst. Technol. — 2020. — Vol. 11, no. 2.

7. MovieLens 100K Dataset. — Режим доступа: https://grouplens.org/datasets/movielens/100k/ (дата обращения: 2022-03-20).

8. Machine Learning With Big Data: Challenges and Approaches / Alexandra L'Heureux, Katarina Grolinger, Hany F. Elyamany, Miriam A. M. Capretz // IEEE ACCESS. — 2017. — Vol. 5. — P. 7776–7797.

9. Cramer J.S. The Origins of Logistic Regression // Tinbergen Institute Working Paper. — 2002.

10. Ratnaparkhi Adwait. A Linear Observed Time Statistical Parser Based on Maximum Entropy Models // Second Conference on Empirical Methods in Natural Language Processing. — 1997.

11. Lloyd S.P. Least-Squares Quantization In PCM // IEEE Transactions On Information Theory. — 1982. — Vol. 28, no. 2. — P. 129–137.

12. Research issues on K-means Algorithm: An Experimental Trial Using Matlab / Joaquín Ortega, Ma Rocío, Boone Rojas, María García // CEUR Workshop Proceedings. — 2009. — 01. — Vol. 534.

13. Kingma Diederik, Ba Jimmy. Adam: A Method for Stochastic Optimization // International Conference on Learning Representations. — 2014. — 12.

# Приложение 1

### Рекомендательная система методом логистической регрессии

Импортирование нужных библиотек.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

Выгружаем датасет и формируем его в нужную матрицу. По горизонтали матрицы – фильмы, по вертикали – пользователи.

[2]:
```
movies=1682
viewers=943
table = pd.read_table('u.data', sep='\t', names=["user_id", "item_id",␣
 ↪"rating", "timestamp"])
table = table.sort_values(['user_id'], ascending=True)
ratings = np.empty((movies, viewers))
for record in zip(table["item_id"], table["user_id"], table["rating"]):
    ratings[record[0] - 1][record[1] - 1] = record[2]
```

Удаление пропущенных значений: оставим только те строки – фильмы, где стоит хотя бы 20 оценок.

[3]:
```
drop_indexes = []
s = 0
for movie_id in range(movies):
    if len(ratings[movie_id][ratings[movie_id]>0]) < 20:
        drop_indexes.append(movie_id)
    s += len(ratings[movie_id][ratings[movie_id]>0])
#print(s/movies)
ratings = np.delete(ratings, drop_indexes, axis=0)
movies = ratings.shape[0]
viewers = ratings.shape[1]
print(ratings.shape)
```

```
(939, 943)
```

Замена оценок на флаги удовлетворенности просмотром фильма.

[4]:
```
ratings[ratings==1] = 0
ratings[ratings==2] = 0
ratings[ratings==3] = 0
ratings[ratings==4] = 1
ratings[ratings==5] = 1
```

Разделяем выборку для теста и обучения в отношении 2 к 8. Прогнозировать будем оценки первого пользователя на фильмы.

[5]:
```
y = ratings
yi = y[:,0]
X = list(range(movies))
(X_train, X_test,
 y_train, y_test) = train_test_split(X, yi,
                                     test_size=0.2,
                                     random_state=0)
X_train= np.reshape(X_train, (-1, 1))
X_test= np.reshape(X_test, (-1, 1))
```

Обучаем логистическую регрессию, формируем предсказание для первого пользователя.

[6]:
```
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
gen_acc = logreg.score(X_test, y_test)
```

[7]:
```
print('Точность прогноза на тесте:', round(sum(y_test==y_pred)/
 ↪len(y_test), 3))
```

```
Точность прогноза на тесте: 0.904
```

Проверим значение точности на обучении.

[8]:
```
y_pred = logreg.predict(X_train)


print('Точность прогноза на обучении:', round(sum(y_train==y_pred)/
 ↪len(y_train), 3))
```

```
Точность прогноза на обучении: 0.846
```

Осталось вычислить среднюю точность. Для этого измерим повторим те же действия с другими пользователями.

[10]:
```
accurace_list = [gen_acc]
for viewer_id in range(1,viewers):
    yi = y[:,viewer_id]
    if sum(yi) != 0:
        (X_train, X_test, y_train, y_test) = train_test_split(X, yi,
                                            test_size=0.2,
                                            random_state=0)
        X_train= np.reshape(X_train, (-1, 1))
        X_test= np.reshape(X_test, (-1, 1))
        y_pred = logreg.predict(X_test)
        accurace_list.append(logreg.score(X_test, y_test))

print('Точность на всех данных:',round(sum(accurace_list)/
 ↪len(accurace_list),3))
```

```
Точность на всех данных: 0.856
```

# Приложение 2

### Рекомендательная система методом кластеризации

Импортирование нужных библиотек.

[1]:
```
import pandas as pd
import numpy as np
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.pyplot as plt
```

Выгружаем датасет и формируем его в нужную матрицу. По горизонтали матрицы – фильмы, по вертикали – пользователи.

[2]:
```
movies=1682
viewers=943
table = pd.read_table('u.data', sep='\t', names=["user_id", "item_id",␣
 ↪"rating", "timestamp"])
table = table.sort_values(['user_id'], ascending=True)
ratings = np.empty((movies, viewers))
for record in zip(table["item_id"], table["user_id"], table["rating"]):
    ratings[record[0] - 1][record[1] - 1] = record[2]
```

Удаление пропущенных значений: оставим только те строки – фильмы, где стоит хотя бы 20 оценок.

[3]:
```
drop_indexes = []
s = 0
for movie_id in range(movies):
    if len(ratings[movie_id][ratings[movie_id]>0]) < 20:
        drop_indexes.append(movie_id)
    s += len(ratings[movie_id][ratings[movie_id]>0])
#print(s/movies)
ratings = np.delete(ratings, drop_indexes, axis=0)
movies = ratings.shape[0]
viewers = ratings.shape[1]
print(ratings.shape)
```

```
(939, 943)
```

Транспонируем матрицу для корректной подачи в алгоритм. По горизонтали матрицы – пользователи, по вертикали – фильмы.

[4]:
```
ratings = ratings.T
ratings.shape
```

[4]:
```
(943, 939)
```

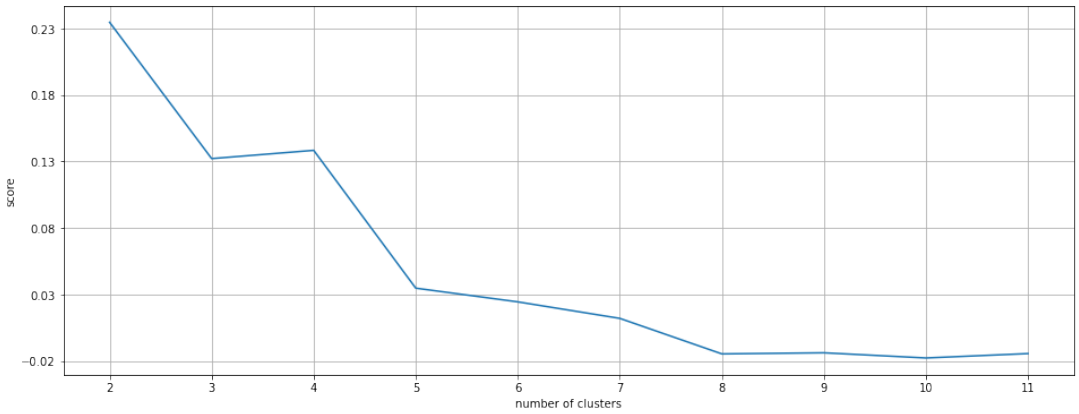
Выбор лучшего k.

[5]:
```
def clustering_errors(k, data):
    kmeans = KMeans(n_clusters=k).fit(data)
    predictions = kmeans.predict(data)
    silhouette_avg = silhouette_score(data, predictions)
    return silhouette_avg
```

[6]:
```
new_k_values = range(2, 12)
sparse_errors_k = [clustering_errors(k, ratings) for k in new_k_values]
```

[7]:
```
fig, ax = plt.subplots(figsize=(16, 6))
ax.set_xlabel('number of clusters')
ax.set_ylabel('score')
ax.plot(new_k_values, sparse_errors_k)


xticks = np.arange(min(new_k_values), max(new_k_values)+1, 5.0)
ax.set_xticks(xticks, minor=False)
ax.set_xticks(xticks, minor=True)
ax.xaxis.grid(True, which='both')
yticks = np.arange(round(min(sparse_errors_k), 2),␣
 ↪max(sparse_errors_k), .05)
ax.set_yticks(yticks, minor=False)
ax.set_yticks(yticks, minor=True)
ax.yaxis.grid(True, which='both')
plt.savefig("choicek.jpg")
```




Построение прогноза с 3 кластерами.

[8]:
```
predictions_10 = KMeans(n_clusters=3, algorithm='full').
 ↪fit_predict(ratings)
```

[9]:
```
ratings_df = pd.DataFrame(ratings)
```

Каждому пользователю сопоставлен номер кластера.

[10]:
```
predict_cluster = pd.concat([ratings_df.reset_index(), pd.
↪DataFrame({'group':predictions_10})], axis=1)
```

Размеры получившихся кластеров.

[11]:
```
for i in range(3):
    print('Номер кластера:', i, 'Количество пользователей в кластере:',␣
↪predict_cluster[predict_cluster.group == i].shape[0])
```

```
Номер кластера: 0 Количество пользователей в кластере: 133
Номер кластера: 1 Количество пользователей в кластере: 268
Номер кластера: 2 Количество пользователей в кластере: 542
```

Непросмотренным фильмам ставим среднее значение оценок по кластеру.

[12]:
```
loss = []
for cluster_id in range(3):
    cluster_i = predict_cluster[predict_cluster.group == cluster_id].
 ↪drop(['index', 'group'], axis=1)
    s=0
    for user_id in list(cluster_i.index):
        user_ratings  = cluster_i.loc[user_id, :].copy()
        control2 = cluster_i.loc[user_id, :]
        control2[control2==1] = 0
        control2[control2==2] = 0
        control2[control2==3] = 0
        control2[control2==4] = 1
        control2[control2==5] = 1
        for movie_id in range(movies):
            if user_ratings[movie_id] == 0 and␣
 ↪cluster_i[movie_id][cluster_i[movie_id]!=0].shape[0] != 0:
                user_ratings[movie_id] =␣
 ↪round(cluster_i[movie_id][cluster_i[movie_id] > 0].mean())
            elif user_ratings[movie_id] == 0 and␣
 ↪cluster_i[movie_id][cluster_i[movie_id]!=0].shape[0] == 0 :
                user_ratings[movie_id] = 0
        user_ratings[user_ratings==1] = 0
        user_ratings[user_ratings==2] = 0
        user_ratings[user_ratings==3] = 0
        user_ratings[user_ratings==4] = 1
        user_ratings[user_ratings==5] = 1
        s += sum(user_ratings == control2)/user_ratings.shape[0]
    loss.append(s/cluster_i.shape[0])
    print('Кластер:', cluster_id,'точность:', round(s/cluster_i.
 ↪shape[0],3))
print('Итоговая точность по всему датасету:', round(sum(loss)/
 ↪len(loss), 3))
```

```
Кластер: 0 точность: 0.948
Кластер: 1 точность: 0.866
Кластер: 2 точность: 0.916

Итоговая точность по всему датасету: 0.91
```

# Приложение 3

### Рекомендательная система с помощью нейронной сети

Импортирование нужных библиотек

[1]:
```
import torch
import pandas as pd
import numpy as np
import matplotlib
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import math
```

Выгружаем датасет и формируем его в нужную матрицу. По горизонтали матрицы – фильмы, по вертикали – пользователи.

[2]:
```
movies=1682
viewers=943
table = pd.read_table('u.data', sep='\t', names=["user_id", "item_id",␣
 ↪"rating", "timestamp"])
table = table.sort_values(['user_id'], ascending=True)
ratings = np.empty((movies, viewers))
for record in zip(table["item_id"], table["user_id"], table["rating"]):
    ratings[record[0] - 1][record[1] - 1] = record[2]
```

Удаление пропущенных значений: оставим только те строки – фильмы, где стоит хотя бы 20 оценок.

[3]:
```
drop_indexes = []
s = 0
for movie_id in range(movies):
    if len(ratings[movie_id][ratings[movie_id]>0]) < 20:
        drop_indexes.append(movie_id)
    s += len(ratings[movie_id][ratings[movie_id]>0])
#print(s/movies)
ratings = np.delete(ratings, drop_indexes, axis=0)
movies = ratings.shape[0]
viewers = ratings.shape[1]
print(ratings.shape)
ratings=ratings.T
```

```
(939, 943)
```

Будем пытаться предсказать оценки первого фильма (yi) для послдених 200 пользователей.

В части train будут оценки первых 700 пользователей на все фильмы. В тестовой – будем пытаться предсказать оценки оставшихся пользователей на первый фильм

[4]:
```
X_train = ratings[range(700),1:]
X_train = torch.FloatTensor(X_train)
X_train.shape
```

[4]:
```
torch.Size([700, 938])
```

[5]:
```
X_test = ratings[range(700, 943),1:]
X_test = torch.FloatTensor(X_test)
X_test.shape
```

[5]:
```
torch.Size([243, 938])
```

[6]:
```
y0 = ratings[:,0] # оценки пользователей на нулевой фильм
y_train = y0[0:700]
y_train = torch.FloatTensor(y_train)
y_test = y0[700:943]
y_test = torch.FloatTensor(y_test)
```

Нейронная сеть

[7]:
```
class test_net(torch.nn.Module):
    def __init__(self):
        super(test_net, self).__init__()

        self.fc1 = torch.nn.Linear(938, 32)
        self.act1 = torch.nn.Tanh()

        self.fc2 = torch.nn.Linear(32, 20)
        self.act2 = torch.nn.Tanh()

        self.fc3 = torch.nn.Linear(20, 10)
        self.act3 = torch.nn.Tanh()

        self.fc4 = torch.nn.Linear(10, 1)

    def forward(self, x):
        x = self.fc1(x)
        x = self.act1(x)
        x = self.fc2(x)
        x = self.act2(x)
        x = self.fc3(x)
        x = self.act3(x)
        x = self.fc4(x)
        return x
```

Обучение нейронной сети

[8]:
```
net = test_net()
optimizer = torch.optim.Adam(net.parameters(), lr=0.001)
epoch_num = 20
batch_size = 50

criterion = torch.nn.MSELoss()

def loss(x, y):
    return torch.sqrt(criterion(x, y))

for epoch in range(epoch_num):
    order = np.random.permutation(X_train.shape[0])
    for start_index in range(0, X_train.shape[0], batch_size):
        optimizer.zero_grad()

        batch_indexes = order[start_index : start_index + batch_size]

        x_batch = X_train[batch_indexes]
        y_batch = y_train[batch_indexes]
        preds = net.forward(x_batch)
        loss_val = loss(preds, torch.reshape(y_batch,(len(y_batch),1)))
        loss_val.backward()

        optimizer.step()
    test_preds = net.forward(X_test)
    test_preds = test_preds.reshape(-1, )
    loss_2 = loss(test_preds, y_test)
print('Mean Square Error:', round(loss_2.item(),3))
print('Первые 10 оценок:', *test_preds.detach().numpy()[:10])
```

```
Mean Square Error: 1.927
Первые 10 оценок: 0.8769811 0.23634662 4.025127 0.42982012 3.7509668 1.
↪8445486
3.714751 3.7079732 1.4442484 2.2274141
```

Погрешность вычисления довольно большая, предсказание нейронной сети не дает ответа, который можно было бы использовать на практике. Она просто выдает некоторое среднее, чтобы уменьшить ошибку. Рассмотрим нейронную сеть-классификатор.

[9]:
```
ratings[ratings==1] = 0
ratings[ratings==2] = 0
ratings[ratings==3] = 0
ratings[ratings==4] = 1
ratings[ratings==5] = 1
```

[10]:
```
X_train = ratings[range(700),1:]
X_train = torch.FloatTensor(X_train)

X_test = ratings[range(700, 943),1:]
X_test = torch.FloatTensor(X_test)

y0 = ratings[:,0] # оценки пользователей на нулевой фильм
y_train = y0[0:700]
y_train = torch.FloatTensor(y_train)

y_test = y0[700:943]
y_test = torch.FloatTensor(y_test)
```

Нейронная сеть-классфикатор

[11]:
```
class classificator_net(torch.nn.Module):
    def __init__(self):
        super(classificator_net, self).__init__()

        self.fc1 = torch.nn.Linear(938, 32)
        self.act1 = torch.nn.Tanh()

        self.fc2 = torch.nn.Linear(32, 20)
        self.act2 = torch.nn.Tanh()

        self.fc3 = torch.nn.Linear(20, 10)
        self.act3 = torch.nn.Tanh()

        self.fc4 = torch.nn.Linear(10, 1)

    def forward(self, x):
        x = self.fc1(x)
        x = self.act1(x)
        x = self.fc2(x)
        x = self.act2(x)
        x = self.fc3(x)
        x = self.act3(x)
        x = self.fc4(x)
        x =torch.sigmoid(x)
        return x
```

Обучение нейронной сети

[12]:
```
def loss(x, y):
    return sum(x==y)/len(x)

net = classificator_net()
optimizer = torch.optim.Adam(net.parameters(), lr=0.001)
epoch_num = 50
batch_size = 50

criterion = torch.nn.BCELoss()

for epoch in range(epoch_num):
    order = np.random.permutation(X_train.shape[0])
    for start_index in range(0, X_train.shape[0], batch_size):
        optimizer.zero_grad()

        batch_indexes = order[start_index : start_index + batch_size]

        x_batch = X_train[batch_indexes]
        y_batch = y_train[batch_indexes]
        #print(x_batch)
        preds = net.forward(x_batch)
        loss_val = criterion(preds, y_batch.unsqueeze(1))
        loss_val.backward()

        optimizer.step()
    test_preds = net.forward(X_test)
    test_preds = test_preds.reshape(-1, )
    accuracy = accuracy_score(np.round(test_preds.detach().numpy()),␣
↪y_test.detach().numpy())
    print('Эпоха ', epoch,': accuracy = ',round(accuracy,3), ', BCEloss␣
↪= ',round(loss_val.item(),3), sep='')
```

```
Эпоха 0: accuracy = 0.737, BCEloss = 0.66
Эпоха 1: accuracy = 0.712, BCEloss = 0.603
Эпоха 2: accuracy = 0.708, BCEloss = 0.525
Эпоха 3: accuracy = 0.695, BCEloss = 0.481
Эпоха 4: accuracy = 0.716, BCEloss = 0.339
Эпоха 5: accuracy = 0.7, BCEloss = 0.345
Эпоха 6: accuracy = 0.687, BCEloss = 0.245
Эпоха 7: accuracy = 0.679, BCEloss = 0.281
Эпоха 8: accuracy = 0.675, BCEloss = 0.135
Эпоха 9: accuracy = 0.671, BCEloss = 0.133
Эпоха 10: accuracy = 0.671, BCEloss = 0.082
Эпоха 11: accuracy = 0.675, BCEloss = 0.165
Эпоха 12: accuracy = 0.658, BCEloss = 0.123
Эпоха 13: accuracy = 0.663, BCEloss = 0.175
Эпоха 14: accuracy = 0.658, BCEloss = 0.048
Эпоха 15: accuracy = 0.667, BCEloss = 0.043
Эпоха 16: accuracy = 0.667, BCEloss = 0.039
Эпоха 17: accuracy = 0.663, BCEloss = 0.034
Эпоха 18: accuracy = 0.663, BCEloss = 0.102
Эпоха 19: accuracy = 0.667, BCEloss = 0.028
Эпоха 20: accuracy = 0.671, BCEloss = 0.026
Эпоха 21: accuracy = 0.671, BCEloss = 0.028
Эпоха 22: accuracy = 0.671, BCEloss = 0.102
Эпоха 23: accuracy = 0.671, BCEloss = 0.022
Эпоха 24: accuracy = 0.667, BCEloss = 0.021
Эпоха 25: accuracy = 0.667, BCEloss = 0.02
Эпоха 26: accuracy = 0.667, BCEloss = 0.019
Эпоха 27: accuracy = 0.663, BCEloss = 0.019
Эпоха 28: accuracy = 0.667, BCEloss = 0.017
Эпоха 29: accuracy = 0.671, BCEloss = 0.017
Эпоха 30: accuracy = 0.671, BCEloss = 0.016
```

```
Эпоха 31: accuracy = 0.667, BCEloss = 0.015
Эпоха 32: accuracy = 0.667, BCEloss = 0.014
Эпоха 33: accuracy = 0.667, BCEloss = 0.014
Эпоха 34: accuracy = 0.667, BCEloss = 0.013
Эпоха 35: accuracy = 0.667, BCEloss = 0.013
Эпоха 36: accuracy = 0.667, BCEloss = 0.012
Эпоха 37: accuracy = 0.667, BCEloss = 0.011
Эпоха 38: accuracy = 0.667, BCEloss = 0.011
Эпоха 39: accuracy = 0.667, BCEloss = 0.011
Эпоха 40: accuracy = 0.667, BCEloss = 0.011
Эпоха 41: accuracy = 0.667, BCEloss = 0.01
Эпоха 42: accuracy = 0.667, BCEloss = 0.01
Эпоха 43: accuracy = 0.663, BCEloss = 0.009
Эпоха 44: accuracy = 0.663, BCEloss = 0.01
Эпоха 45: accuracy = 0.667, BCEloss = 0.009
Эпоха 46: accuracy = 0.663, BCEloss = 0.009
Эпоха 47: accuracy = 0.663, BCEloss = 0.009
Эпоха 48: accuracy = 0.663, BCEloss = 0.009
Эпоха 49: accuracy = 0.663, BCEloss = 0.008
```