Правительство Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего профессионального образования
Национальный исследовательский университет
«Высшая школа экономики»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»
Департамент анализа данных и искусственного интеллекта

КУРСОВАЯ РАБОТА

на тему

Исследование возможности встраивания контекстной информации в алгоритмы коллаборативной фильтрации на основе матричных разложений

Выполнил студент группы 203
Стеценко Макар Александрович

Научный руководитель:
Доцент, кандидат технических наук,
Игнатов Дмитрий Игоревич

Москва 2015

Оглавление

Аннотация

"Исследование возможности встраивания контекстной информации в алгоритмы коллаборативной фильтрации на основе матричных разложений"

Стеценко Макар Александрович

Факультет компьютерных наук. Второй курс. 203 группа. 2015 год.

Игнатов Дмитрий Игоревич

Факультет компьютерных наук. Департамент анализа данных и искусственного интеллекта.

Научный руководитель. dignatov@hse.ru

Наличие контекстной информации является одним из важнейших факторов для построения личных рекомендаций. Однако, классические алгоритмы коллаборативной фильтрации, основанные на матричных разложениях, таких как SVD разложение, используют только информацию о пользователях и предметах и не предоставляют явных методов включения дополнительных факторов. В данной работе будет показан один из методов встраивания контекстной информации в алгоритм, использующий SVD разложение. Для тестирования рассматриваемого метода будет использоваться открытый банк данных MovieLens. База данных содержит пользователей портала MovieLens, каждый из которых оценил не менее 20 фильмов, а так же информацию о каждом фильме.

Abstract

"Integrating Contextual Information into Collaborative Filtering Algorithms based on Matrix Decomposition"

Stetsenko Makar
Faculty of Computer Science. 2nd course. 203 group. 2015 year.
Ignatov Dmitry
Faculty of Computer Science. School of Data Analysis and Artificial Intelligence.
Scientific adviser. dignatov@hse.ru

Context has always been an important factor in personalized Recommender systems. However, standard collaborative filtering algorithms based on matrix factorization rely mainly on user and subject information and don't provide any methods for incapsulating extra data. This work demonstrates such a method based on SVD decomposition. To test results an open data base taken from MovieLens is used. The database provides information about users and movies.

Введение

Рекомендательные системы стали неотъемлемой частью жизни людей и бизнеса, особенно после популяризации BigData и открытых данных. Задача таких систем угадывать предпочтения пользователей основываясь на их предыдущих действиях. Более формально, задача о рекомендациях состоит в угадывании значения пары (пользователь, предмет). В простейшем варианте, ответом на данный вопрос будет либо 1 - предмет интересен пользователю, либо 0 - предмет не стоит рекомендовать.

Коллаборативная фильтрация один из способов построения рекомендаций. Данный метод использует информацию о других пользователях системы, чтобы понять, какой предмет скорее всего понравится рассматриваемому пользователю. Несмотря на то, что упрощенная модель *(пользователь, предмет)* подходит для моделирования многих ситуаций, очень часто приходится сталкиваться с дополнительными параметрами, которые играют важную роль при решении задачи о рекомендациях. Таким параметром может быть время, тогда уже имеется 3-х мерный вектор *(пользователь, предмет, время)*. Множество переменных, которые влияют на отношение пользователя к предмету и следовательно на рекомендации для этого пользователя, называется *контекстом*.

Целью данной работы будет решение задачи о рекомендациях с использованием контекстной информации.

Глава 1

1.1 Введение

Рекомендательные системы стали независимой областью для исследования примерно в середине 90-ых. Сегодня можно дать такое определение: Рекомендательные системы - это программы, которые помогают пользователю принять решение, стараясь найти из общего множества товаров, набор товаров наиболее схожий с уже понравившимся ему. Что считать понравившимся товаром, зависит от области, в которой будет применяться рекомендательная система, это может быть высокая оценка фильму или переход по ссылке на страницу продукта. Поиск таких товаров имеет первостепенную роль для онлайн бизнеса. Интернет магазины и сервисы по предоставлению контента имеют в своем распоряжении тысячи наименований, но лишь малая доля будет интересна конечному пользователю, поэтому построение правильных рекомендаций гарантированно увеличивает прибыль таких сервисов. В качестве примера можно привести интернет гиганта в сфере E-Commerce - Amazon, а так же Netflix, компанию, специализирующуюся на показе фильмов и сериалов.

1.2 Типы рекомендательных систем

Существует два основных подхода для нахождения рекомендаций: *Content-based Filtering* и *Collaborative Filtering (CF)*.

1.2.1 Content-based Filtering

Данный метод использует свойства и содержание рекомендуемых объектов. В системе основанной на данном подходе выделяют 3 основных компонента:

- **Парсер**. Этот компонент нужен для выделения нужной информации из объекта и ее структурирования. В качестве объекта может выступать веб-страница, набор действий пользователя, текстовый документ и так далее. Используются различные методы *feature extraction* для выделения ключевых свойств объекта, например, текстовый документ можно представить вектором ключевых слов.

- **Конструктор пользовательского профиля**. Этот модуль получает уже структурированные данные и пытается на их основе построить профиль пользователя.

- **Фильтр**. Данный компонент отвечает за построение конкретных рекомендаций. Имея готовый профиль пользователя и некоторую метрику, позволяющую измерить сходство между двумя объектами, строится ранжированный список наиболее похожих предметов, который и является результатом работы всей системы.

1.2.2 Collaborative Filtering

Данный метод основывается на большом количестве собранных от пользователей отзывов, а не свойствах рекомендуемого объекта. Главной целью является поиск схожих пользователей, используя только оценки, которые они поставили предмету. Найдя такие группы, довольно легко построить рекомендации.

В данной работе будет исследоваться коллаборативная фильтрация, поэтому формализуем решаемую ей задачу, она так же называется *задачей о рекомендациях*.

Прежде чем ввести формальное определение задачи, обозначим множество пользователей за U, множество предметов за I и множество всех оценок за R. Так же оговорим, что никакой пользователь не оценивал один и тот же предмет дважды, тогда оценка пользователя u предмету i запишем, как r_{ui}. Можно составить матрицу, где по строкам будут пользователи, по столбцам предметы, а элемент матрицы будет из множества оценок R. В среднем каждый пользователь оценивает не больше 10-20 предметов, поэтому в матрице будет очень много элементов для которых оценка r_{ui} неизвестна. Задача рекомендательной системы предсказать значение r_{ui}.

1.3 Способы реализации CF

Существует два основных метода решения задачи о рекомендациях, используя CF.

- Метод соседей (Neighborhood based)

- Матричное разложение

1.3.1 Метод соседей

Данный метод очень простой в реализации и основывается на поиске множества пользователей, похожих на пользователя u. Так как каждый пользователь представляет собой вектор оценок, то схожесть двух пользователей можно

оценить, посчитав косинус угла между двумя векторами. После того, как были найдены n схожих пользователей, легко предсказать непроставленные оценки пользователя u, используя имеющиеся оценки найденных пользователей.

1.3.2 Матричное разложение

Поскольку матрица оценок пользователей может быть очень большой, а заполненных ячеек в ней очень мало, можно уменьшить размер пространства путем поиска некоторого набора факторов f_i которые будут общими, как для пользователей, так и для предметов. Смысл матричной факторизации заключается в приближении исходной большой матрицы произведением нескольких, но меньших по размеру.

В данной работе будет рассмотрено наиболее популярное и применяемое разложение - SVD (Singular Value Decomposition).

$$M_{m,n} = U_{m,f} K_{f,f} I_{n,f}$$

Здесь матрица слева - это исходная матрица оценок (m = $|\mathcal{U}|$, n = $|\mathcal{I}|$). Матрицы справа и есть искомое разложение, рассмотрим их подробнее. Элемент матрицы U содержит веса каждого из факторов для конкретного пользователя, другими словами, элемент $u_{i,j}$ говорит насколько важен пользователю U_i фактор f_j. Тоже самое, но для предметов, содержит матрица I. Матрица K диагональная, на ее диагонали в убывающем порядке, находятся сингулярные числа матрицы M, они показывают, насколько фактор f_i важен, в контексте рекомендательных систем, он показывает какую долю оценки занимает фактор f_i, например, жанр фильма может иметь большое значение для пользователя, в то время как год выпуска - нет. Сами факторы f не имеют никакой интерпретации, и SVD разложение не говорит, что это за факторы. Найдя такое разложение, не только уменьшается объем памяти, затрачиваемой на работу рекомендательной системы, но и легко решается задача о рекомендациях. Перемножив эти три матрицы, получится исходная матрица оценок, но она уже будет полной, а не разреженной.

В данной работе, будет использоваться именно SVD разложение.

Глава 2

2.1 Контекстная информация

Несмотря на удобство и простоту стандартных методов коллаборативной фильтрации, данная модель не содержит в себе огромное количество другой доступной информации, будем называть такую информацию *контекстной*. Контекстной информацией может быть все что угодно, дата покупки (выходной или будний день), жанр фильма, возраст пользователя и так далее. Наличие такой информации должно повысить точность рекомендаций и их полезность. Так, например, на выходных можно рекомендовать фильмы для семейного просмотра, а по будням непродолжительные сериалы.

2.2 Встраивания контекстной информации

Существует несколько общих методов встраивания контекстной информации в рекомендательную систему.

- **Contextual Pre-Filtering**. Суть данного метода заключается в фильтрации не подходящих под текущий контекст оценок, дальше задача решается стандартными алгоритмами.
- **Contextual Post-Filtering**. После нахождения списка рекомендаций стандартными методами, предметы, не подходящие под текущий контекст, удаляются.
- **Contextual Modeling**. Контекстная информация встраивается на уровне модели. Стандартные методы решения задачи о рекомендациях уже не работают, или же требуется их адаптация.

В данной работе будет рассмотрен последний из методов, а именно, метод встраивания контекстной информации в матрицу отзывов.

2.3 Контекстная информация в CF

В работе будет использоваться набор данных MovieLens, он содержит отзывы на фильмы, а так же ряд дополнительной информации: возраст, пол и род дея-

тельности пользователя, жанр и год выпуска фильма (один фильм может иметь несколько жанров).

Обозначим контекстную информацию, относящуюся к пользователю за $\mathcal{C}_{\mathcal{U}}$, а к предметам за $\mathcal{C}_{\mathcal{I}}$.

$$\mathcal{C}_{\mathcal{U}} = M, F, Job_1, ..., Job_{19}$$
$$\mathcal{C}_{\mathcal{I}} = genre_1, ..., genre_{19}$$

К исходной матрице отзывов по строкам допишем информацию о предметах, а по столбцам информацию о пользователях, таким образом, матрица отзывов размера m на n примет размер $m + |C_I|$ на $n + |C_U|$. Если предмет или пользователь содержит контекстную информацию, то в соответствующей ячейке матрицы ставится оценка 5, иначе 0.

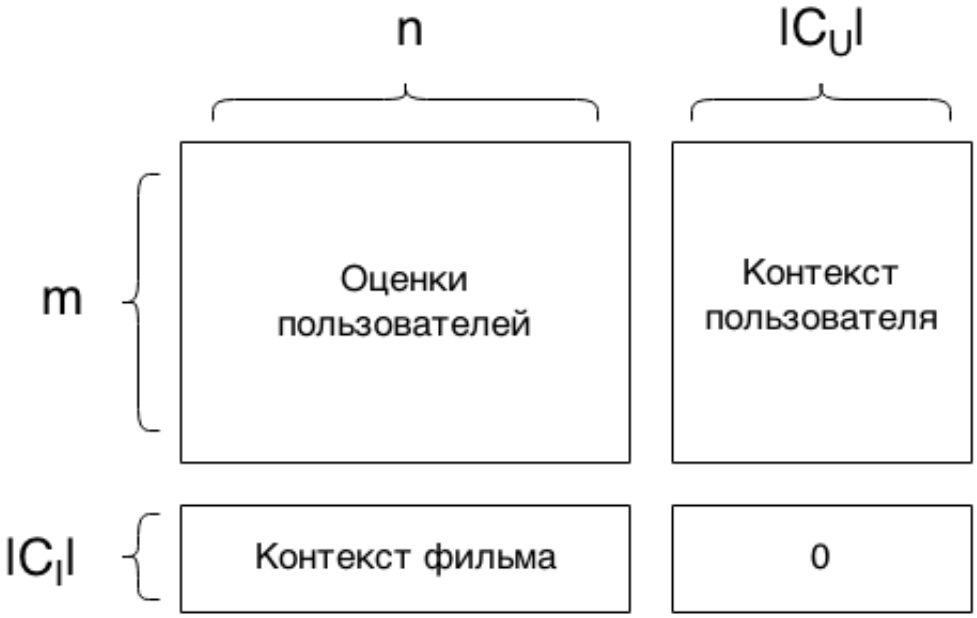

Рис. 2.1: Схема расширенной матрицы отзывов с контекстной информацией

Таким образом мы получили *расширенную матрицу отзывов*, которая содержит в себе всю необходимую информацию. Отличие от стандартной матрицы заключается в том, что добавленные строки, как бы являются пользователями, которые очень любят фильмы определенного жанра. Алгоритм CF учтет таких пользователей и найдет наиболее схожих с ними. Аналогично для столбцов. К новой матрице можно применить уже описанный ранее метод разложения SVD.

Глава 3

3.1 Реализация на Python

Для проверки предложенного метода встраивания контекстной информации был написан прототип рекомендательной системы на языке Python.

3.1.1 Реализация SVD

Обычные алгоритмы поиска SVD разложения не подходят для матрицы отзывов, потому что в такой матрице значения многих элементов неизвестны, их предсказание и является задачей системы. Однако, все классические алгоритмы поиска SVD ожидают полную матрицу. Существует два способа решения данной проблемы:

- Попытаться заполнить пустые ячейки определенными значениями, например, средней оценкой для всех фильмов или нулями. Такой метод вносит неточность в результат разложения и значительно увеличивает объем памяти для хранения матрицы.
- Применить метод, предложенный Симоном Фанком [1], который основан на градиентном спуске.

3.1.2 FunkSVD

Суть предложенного Фанком алгоритма заключается в использовании метода стохастического градиентного спуска только на известных значениях матрицы отзывов. Результатом работы алгоритма будут две матрицы, а не три, как в классическом разложении.

Для нахождения разложения, используется стохастический градиентный спуск. Каждый признак обучается по отдельности при помощи следующих правил:

$$U_{f_i} = U_{f_i} + L(2eI_{f_i} - GU_{f_i})$$
$$I_{f_i} = I_{f_i} + L(2eU_{f_i} - GI_{f_i})$$

Здесь L - это константа, отвечающая за скорость обучения, G - нормализующая константа для борьбы с переобучением, $e = r_{kj} - (U_k, I_j)$ разница между предсказанным значением оценки и настоящим. Поскольку обучение происходит на известных оценках, то и r_{kj} всегда известны.

3.2 Оценка эффективности

Существует несколько методов оценивания работы рекомендательной системы. После того, как были найдены матрицы признаков U и I, выбирается тестовый набор данных, из него берутся оценки пользователей, которые на момент обучения были неизвестны, и система предсказывает эти оценки $\widehat{r_i} = (U_i, I_i)$. Далее высчитываются следующие метрики:

$$\textit{Root Mean Square Error (RMSE)} = \sqrt{\frac{1}{n}\sum_{i}^{n}(r_i - \widehat{r_i})^2}$$

$$\textit{Mean Absolute Error (MAE)} = \sum_{i}^{n}|r_i - \widehat{r_i}|$$

Разница между MAE и RMSE заключается в том, что RMSE намного сильнее реагирует на большие отклонения.

Были получены следующие значения ошибок:

	RMSE	MAE
SVD	0.934347	0.758373
SVD + Context	0.919411	0.753776

Видно, что рекомендательная система, которая учитывает контекстную информацию, способна лучше предсказывать оценки пользователей.

3.3 Исходный код

Далее будет представлен исходный код. Все исходники доступны на GitHub

Листинг 3.1: Реализация SVD

```
import numpy as np
import math

class SVD():
```

```
usersCount = 0
itemsCount = 0
userRatings = {}
numberOfFeatures = -1

userFeatureMatrix = []
itemFeatureMatrix = []

def __init__(self, numberOfFeatures, numberOfUsers, numberOfItem
    self.usersCount = numberOfUsers
    self.itemsCount = numberOfItems

    self.userRatings = dict(ratings)
    self.numberOfFeatures = numberOfFeatures

    # self.userFeatureMatrix = np.random.rand(self.usersCount,
    # self.itemFeatureMatrix = np.random.rand(self.itemsCount,
    self.itemFeatureMatrix = np.full((self.itemsCount, self.num
    self.userFeatureMatrix = np.full((self.usersCount, self.num

def findDecomposition(self):
    self.__learnFeatureWithSGD()

def __learnFeatureWithSGD(self):
    learningRate = 0.005
    genRate = 0.02

    "Forces numpy to raise exceptions on warnings"
    np.seterr(all='raise')

    "Learn each feature separately"
    for f in xrange(0, self.numberOfFeatures):
        print 'Training feature #%d' % f

        "Number of iterations until convergence"
        for i in xrange(0, 10):
            print ' Iteration #%d/10' % i

            "For each known rating"
            for (userID, itemID) in self.userRatings.iterkeys():
                predicted = np.dot(self.userFeatureMatrix[userID
                error = self.userRatings[(userID, itemID)] - pre
```

```
                    "Update rules"
                    self.userFeatureMatrix[userID, f] += learningRat
                    self.itemFeatureMatrix[itemID, f] += learningRat
```

Листинг 3.2: Реализация SVD с контекстной информацией

```
import numpy as np
import math
import svd

class ContextAwareSVD():
    totalUsers = 0
    totalItems = 1682

    userContextsCount = 5
    itemContextsCount = 19

    userRatings  = {}
    userContexts = []
    itemContexts = []

    users = []
    items = []

    numberOfFeatures = -1

    userFeatureMatrix = []
    itemFeatureMatrix = []

    def __init__(self, numberOfFeatures, usersDB, itemsDB, ratings):
        self.totalUsers  = len(usersDB)

        self.userRatings = dict(ratings)
        self.users = usersDB
        self.items = itemsDB

        self.numberOfFeatures = numberOfFeatures

        self.userFeatureMatrix = np.random.rand(self.totalUsers + s
        self.itemFeatureMatrix = np.random.rand(self.totalItems + s

        self.addUserContext()
        self.addItemContext()
```

```
    def addUserContext(self):
        self.userContexts = np.empty([self.totalUsers, 3])
        for user in self.users:
            self.userRatings.update(user.getContextDict(self.totalIt

    def addItemContext(self):
        self.itemContexts = np.full([19, self.totalItems], 0, dtype=
        for item in self.items:
            self.userRatings.update(item.getContextDict(self.totalUs

    def findDecomposition(self):
        svdSolver = svd.SVD(self.numberOfFeatures,
                            self.totalUsers+self.itemContextsCount,
                            self.totalItems+self.userContextsCount,
                            self.userRatings)

        svdSolver.findDecomposition()

        self.userFeatureMatrix = svdSolver.userFeatureMatrix[:-self.
        self.itemFeatureMatrix = svdSolver.itemFeatureMatrix[:-self.
```

Листинг 3.3: Реализация MAE

```
import numpy as np
import math

__author__ = 'makazone'

class RecSysEvaluator():
    def __init__(self, userFeatures, itemFeatures, knownRatings):
        self.userFeatures = userFeatures
        self.itemFeatures = itemFeatures
        self.knownRatings = knownRatings

    def __computeRMSE(self, ratings):
        error = 0
        for (userID, itemID) in ratings.keys():
            predicted = np.dot(self.userFeatures[userID, :], self.it
            error += math.pow(ratings[(userID, itemID)] - predicted,
        error /= len(ratings.keys())
        return error

    def __computeMAE(self, ratings):
        error = 0
```

```
        for (userID, itemID) in ratings.keys():
            predicted = np.dot(self.userFeatures[userID, :], self.it
            error += abs(ratings[(userID, itemID)] - predicted)
        error /= len(ratings.keys())
        return error

    def __extractNewRatings(self, testFile):
        newRatings = {}
        for entry in open(testFile):
            entryComponents = [int(x) for x in entry.split()]
            userID = entryComponents[0]-1
            itemID = entryComponents[1]-1
            rating = entryComponents[2]

            "Only new ratings added"
            if not (userID, itemID) in self.knownRatings:
                newRatings[(userID, itemID)] = rating

        print 'Total new ratings = %d' % len(newRatings.keys())
        return newRatings

    def newRatingsRMSE(self, testFile):
        newRatings = self.__extractNewRatings(testFile)
        error = self.__computeRMSE(newRatings)
        print 'RMSE on new ratings = %f' % error

    def newRatingsMAE(self, testFile):
        newRatings = self.__extractNewRatings(testFile)
        error = self.__computeMAE(newRatings)
        print 'MAE on new ratings = %f' % error

    def reconstructionRMSE(self):
        error = self.__computeRMSE(self.knownRatings)
        print 'reconstruction RMSE = %f' % error

    def reconstructionMAE(self):
        error = self.__computeMAE(self.knownRatings)
        print 'reconstruction MAE = %f' % error
```

Листинг 3.4: main

```
import svd
import User
import Film
```

```
import casvd
import RecSysEvaluator as rseval

userDB = {}
itemDB = {}
occupations = {}

print 'Populating DB'

id = 0
for occupation in open('data/u.occupation'):
    occupations[occupation[0:len(occupation)-1]] = id
    id += 1

for line in open('data/u.user'):
    user = User.User(line)
    userDB[user.id] = user

for line in open('data/u.item'):
    item = Film.Film(line)
    itemDB[item.id] = item

# print userDB[0]
# print itemDB[0]

print 'Reading ratings file'

dataInfo = [line.split() for line in open("data/u.info")]
dataPath = "data/u1.base"

userRatings = {}
usersUsed = {}
for entry in open(dataPath):
    entryComponents = [int(x) for x in entry.split()]
    userID = entryComponents[0]-1
    itemID = entryComponents[1]-1
    rating = entryComponents[2]

    userRatings[(userID, itemID)] = rating
    usersUsed[userID] = userID

dataSize = [len(usersUsed.keys()), int(dataInfo[1][0])] # [number o
```

```
usersList = []
for uid in usersUsed.keys():
    usersList.append(userDB[uid])

itemsList = []
for iid in itemDB.keys():
    itemsList.append(itemDB[iid])

casvdModel = casvd.ContextAwareSVD(15, usersList, itemsList, userRa
casvdModel.findDecomposition()

svdModel = svd.SVD(15, dataSize[0], dataSize[1], userRatings)
svdModel.findDecomposition()

print 'Evaluating Context Aware SVD'
casvdEvaluator = rseval.RecSysEvaluator(casvdModel.userFeatureMatrix
casvdEvaluator.reconstructionMAE()
casvdEvaluator.newRatingsMAE("data/u1.test")
casvdEvaluator.newRatingsRMSE("data/u1.test")

print '\n'

print 'Evaluating simple SVD'
simpleSVDEvaluator = rseval.RecSysEvaluator(svdModel.userFeatureMatr
simpleSVDEvaluator.reconstructionMAE()
simpleSVDEvaluator.newRatingsMAE("data/u1.test")
simpleSVDEvaluator.newRatingsRMSE("data/u1.test")
```

Листинг 3.5: Класс пользователя

```
__author__ = 'makazone'

class User():

    def __init__(self, stringData):
        userData = stringData.split('|')

        self.id = int(userData[0]) - 1

        age = int(userData[1])
        if age <= 25: self.ageGroup = 0
        elif age <= 60: self.ageGroup = 1
        else: self.ageGroup = 2
        self.exactAge = age
```

```
        self.gender = userData[2]

        self.occupation = userData[3]

    def __str__(self):
        return "{id: %d, age: %d, %c}" % (self.id, self.exactAge, s

    def getContextDict(self, contextIndexOffset):
        context = {}

        if self.ageGroup == 0:
            context[(self.id, contextIndexOffset+0)] = 5
            context[(self.id, contextIndexOffset+1)] = 0
            context[(self.id, contextIndexOffset+2)] = 0
        elif self.ageGroup == 1:
            context[(self.id, contextIndexOffset+0)] = 0
            context[(self.id, contextIndexOffset+1)] = 5
            context[(self.id, contextIndexOffset+2)] = 0
        else:
            context[(self.id, contextIndexOffset+0)] = 0
            context[(self.id, contextIndexOffset+1)] = 0
            context[(self.id, contextIndexOffset+2)] = 5

        contextIndexOffset += 3

        if self.gender == 'M':
            context[(self.id, contextIndexOffset+0)] = 5
            context[(self.id, contextIndexOffset+1)] = 0
        else:
            context[(self.id, contextIndexOffset+0)] = 0
            context[(self.id, contextIndexOffset+1)] = 5

        return context
```

Листинг 3.6: Класс фильма

```
__author__ = 'makazone'

class Film():

    def __init__(self, stringData):
        filmData = stringData.split('|')
```

```
        self.id       = int(filmData[0]) - 1
        self.title    = filmData[1]
        self.imdbURL = filmData[4]

        self.genre = [0 for x in xrange(0, 19)]
        for genreID in xrange(0, 19):
            if filmData[5+genreID] == '1':
                self.genre[genreID] = 5

    def __str__(self):
        return "{id: %d, title: %s}" % (self.id, self.title)

    def getContextDict(self, contextIndexOffset):
        context = {}

        for i in xrange(0, 19):
            if self.genre[i] == 5:
                context[(contextIndexOffset+i, self.id)] = 5
            else:
                context[(contextIndexOffset+i, self.id)] = 0

        return context
```

Заключение

В ходе работы был найден способ встраивания контекстной информации в алгоритм коллаборативной фильтрации на основе матричного разложения SVD. Предложенный подход реализован на языке Python. В ходе тестирования было выявленно, что наличие контекстной информации улучшает ключевые метрики, отвечающие за точность рекомендаций.

Литература

1. Funk Simon. Netflix Update: Try This at Home. URL: http://sifter.org/ simon/journal/20061211.html.

2. Tintarev Nava, Masthoff Judith. Recommender Systems Handbook. 2011. Т. 54. С. 479–510. URL: http://www.springerlink.com/index/10.1007/978-0-387-85820-3.

3. Ignatov Dmitry I, Mikhailova Maria, Zakirova Alexandra Yu [и др.]. Recommendation of Ideas and Antagonists for Crowdsourcing Platform Witology.

4. Xavier Amatriain, Alexandros Karatzoglou, Nuria Oliver Linas Baltrunas. Multiverse Recommendation: N-dimensional Tensor Factorization for Context-aware Collaborative Filtering // ACM Recommender Systems. 2010. URL: http://xavier.amatriain.net/pubs/karatzoglu-recsys-2010.pdf.

5. Ignatov Dmitry I, Nenova Elena, Konstantinova Natalia [и др.]. Boolean Matrix Factorisation for Collaborative Filtering : An FCA-Based Approach.

6. Pilgrim Mark. Dive into python 3 // Dive Into Python 3. 2009. С. 1–360.

7. Biclustering neighborhood-based collaborative filtering method for top-n recommender systems / Faris Alqadah, Chandan K. Reddy, Junling Hu [и др.] // Knowledge and Information Systems. 2014. . URL: http://link.springer.com/10.1007/s10115-014-0771-x.