Отчёт по лабораторной работе №9 "Рекомендательные системы"

In [1]:
```
import numpy as np
from scipy.io import loadmat
from scipy.sparse.linalg import svds
```

In [2]:
```
def load_file(filename, keys=None):
    if keys is None:
        keys = ['X', 'y']
    mat = loadmat(filename)
    ret = tuple([mat[k].reshape(mat[k].shape[0]) if k.startswith('y') else mat[k] for k in keys])
    return ret
```

1. Загрузите данные ex9_movies.mat из файла.

In [3]:
```
Y, R = load_file('ex9_movies.mat', keys=['Y', 'R'])
print(f'Y shape: {Y.shape}')
print(f'R shape: {R.shape}')
```

```
Y shape: (1682, 943)
R shape: (1682, 943)
```

2. Выберите число признаков фильмов (n) для реализации алгоритма коллаборативной фильтрации.

Пусть число признаков фильмов будет равно 20.

In [4]:
```
N_FACTORS = 20
```

Все функции, которые необходимо реализовать в последующих заданиях, будут реализованы в классе `Recommender`.

In [5]:
```
class Recommender:
    def __init__(self, n_factors=N_FACTORS, learning_rate=0.5, reg_L=0.1, max_steps=1e+3):
        self.n_factors = n_factors
        self.learning_rate = learning_rate
        self.reg_L = reg_L
        self.max_steps = int(max_steps)
        self.cost_history = []

    def fit(self, Y, R):
        self.n_m, self.n_u = Y.shape
        self.X = np.random.rand(self.n_m, self.n_factors)
        self.Theta = np.random.rand(self.n_factors, self.n_u)

        for cur_step in range(self.max_steps):
            self.gradient_descent(Y, R)
            cost = self.cost_func(Y, R)
            self.cost_history.append(cost)

    def cost_func(self, Y, R):
        hypotesis = np.dot(self.X, self.Theta)
        mean_error = R * (hypotesis - Y)
        mean_squared_error = mean_error ** 2
        cost = mean_squared_error.sum() / 2
        regularized_cost = cost + (self.reg_L / 2) * ((self.X ** 2).sum() + (self.Theta ** 2).sum())
        return regularized_cost

    def gradient_descent(self, Y, R):
        hypotesis = np.dot(self.X, self.Theta)
        mean_error = R * (hypotesis - Y)
        dX = np.dot(mean_error, self.Theta.T)
        dTheta = np.dot(self.X.T, mean_error)
        regularized_dX = dX + self.reg_L * self.X
        regularized_dTheta = dTheta + self.reg_L * self.Theta
        self.X -= self.learning_rate * regularized_dX
        self.Theta -= self.learning_rate * regularized_dTheta

    def predict(self, user_id, R, top=5):
        predictions = np.dot(self.X, self.Theta)
        user_ratings = (R[:, user_id] != 1) * predictions[:, user_id]
        return user_ratings.argsort()[-top:][::-1]
```

3. Реализуйте функцию стоимости для алгоритма.

Функция стоимости реализована в классе `Recommender` в методе `cost_func`.

```
    def cost_func(self, Y, R):
        hypotesis = np.dot(self.X, self.Theta)
        mean_error = R * (hypotesis - Y)
        mean_squared_error = mean_error ** 2
        cost = mean_squared_error.sum() / 2
        regularized_cost = cost + (self.reg_L / 2) * ((self.X ** 2).sum() + (self.Theta ** 2).sum
    ())
        return regularized_cost
```

4. Реализуйте функцию вычисления градиентов.

Функция вычисления градиентов реализована в классе `Recommender` в методе `gradient_descent`.

```
    def gradient_descent(self, Y, R):
        hypotesis = np.dot(self.X, self.Theta)
        mean_error = R * (hypotesis - Y)
        dX = np.dot(mean_error, self.Theta.T)
        dTheta = np.dot(self.X.T, mean_error)
        regularized_dX = dX + self.reg_L * self.X
        regularized_dTheta = dTheta + self.reg_L * self.Theta
        self.X -= self.learning_rate * regularized_dX
        self.Theta -= self.learning_rate * regularized_dTheta
```

5. При реализации используйте векторизацию для ускорения процесса обучения.

Рейтинги всех пользователей для всех фильмов предсказывается путём перемножений матриц:

`hypotesis = np.dot(self.X, self.Theta)`

Градиентный спуск также высчитывается с помощью векторизации:

```
self.Theta -= self.learning_rate * dX
self.X -= self.learning_rate * dTheta
```

где `self.X` и `self.Theta` - матрицы размера `(n_movies x n_factors)` и `(n_factors * n_users)`

6. Добавьте L2-регуляризацию в модель.

Регулязация добавляется в функцию стоимости в следующей строчке:

`regularized_cost = cost + (self.reg_L / 2) * ((self.X ** 2).sum() + (self.Theta ** 2).sum())`

В градиентный спуск:

```
regularized_dX = dX + self.reg_L * self.X
regularized_dTheta = dTheta + self.reg_L * self.Theta
```

7. Обучите модель с помощью градиентного спуска или других методов оптимизации.

Установим параметры: скорость обучения - 0.001, параметр регуляризации - 10, количество шагов - 1000.

In [6]:
```
rec = Recommender(learning_rate=0.001, reg_L=10, max_steps=1e+3)
rec.fit(Y, R)
```

8. Добавьте несколько оценок фильмов от себя. Файл movie_ids.txt содержит индексы каждого из фильмов.

Добавим нового пользователя в матрицу рейтингов. Это будут мои личные оценки для некоторых фильмов.

In [7]:
```
my_ratings, presence = np.zeros(Y.shape[0], dtype=int), np.zeros(R.shape[0], dtype=int)
my_ratings[95], presence[95] = 5, 1  # Terminator 2: Judgment Day (1991)
my_ratings[194], presence[194] = 5, 1  # Terminator, The (1984)
my_ratings[585], presence[585] = 5, 1  # Terminal Velocity (1994)
my_ratings[942], presence[942] = 5, 1  # Killing Zoe (1994)
my_ratings[1216], presence[1216] = 5, 1  # Assassins (1995)
my_ratings[312], presence[312] = 1, 1  # Titanic (1997)
my_ratings[318], presence[318] = 1, 1  # Everyone Says I Love You (1996)
my_ratings[725], presence[725] = 1, 1  # Fluke (1995)

my_Y = np.column_stack((Y, my_ratings))
my_R = np.column_stack((R, presence))
user_id = my_Y.shape[1] - 1
```

9. Сделайте рекомендации для себя. Совпали ли они с реальностью?

Обучим модель с учётом внесённых рейтингов в матрицу от себя.

In [8]:
```
rec = Recommender(learning_rate=0.001, reg_L=10, max_steps=1e+3)
rec.fit(my_Y, my_R)
```

Далее предскажем топ 5 фильмов, которые могли бы быть мне интересны.

In [9]:
```
top_movies = rec.predict(user_id, my_R, top=5)
```

In [10]:
```
with open('movie_ids.txt', encoding="ISO-8859-1") as f:
    movie_names = [' '.join(line.split(' ')[1:]).replace('\n', '') for line in f.readlines()]

for movie in np.array(movie_names)[top_movies]:
    print(movie)
```

```
Star Wars (1977)
Usual Suspects, The (1995)
Raiders of the Lost Ark (1981)
Cop Land (1997)
Fifth Element, The (1997)
```

Рекомендации совпадают с реальность. Предсказанные фильмы относятся к жанру боевики/триллеры. Фильмы этих жанров были отмечены мной высоким рейтингом.

10. Также обучите модель с помощью сингулярного разложения матриц. Отличаются ли полученные результаты?

In [11]:
```
class SVDRecommender(Recommender):
    def fit(self, Y, R):
        self.X, _, self.Theta = svds(Y.astype('float64'), k=N_FACTORS)
```

In [12]:
```
svd_rec = SVDRecommender()
svd_rec.fit(my_Y, my_R)
top_mov = svd_rec.predict(user_id, my_R, top=5)
for movie in np.array(movie_names)[top_mov]:
    print(movie)
```

```
Fugitive, The (1993)
Braveheart (1995)
Raiders of the Lost Ark (1981)
Pulp Fiction (1994)
Usual Suspects, The (1995)
```

Результаты также совпадают с реальностью. Алгоритм порекоммендовал фильмы жанра боевики/триллеры. Фильмы этих жанров были отмечены мной высоким рейтингом по сравнению с другими фильмами. Некоторые из фильмов даже совпали с предыдущими результами, где модель была обучена с помощью градиентного спуска.