Лабораторная работа №9 “Рекомендательные системы”

In [52]:

```
import numpy as np
from scipy.sparse.linalg import svds
from scipy.io import loadmat
```

1. Загрузите данные ex9_movies.mat из файла.

In [53]:

```
mat = loadmat('data/ex9_movies.mat')
R = mat['R']
Y = mat['Y']
```

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

In [54]:

```
NUM_FEATURES = 15
```

3. Реализуйте функцию стоимости для алгоритма.

CollaborativeFiltering.cost_func

Функция стоимости:

$J(x^{(i)},...,x^{(n_m)},\Theta^{(I)},...,\Theta^{(n_u)}) = \frac{1}{2} \sum ((\Theta^{(j)})^Tx^{(i)} - y^{(i,j)})^2 + (\frac{\lambda}{2} \sum^{n_u}_{j=1}\sum^n_{k=1} (\Theta^{(j)}_k)^2) + (\frac{\lambda}{2} \sum^{n_m}_{i=1}\sum^n_{k=1} (x^{(i)}_k)^2)$

Вычисление градиента:

$\frac{\partial J}{\partial x^{(i)}_k} = \sum ((\Theta^{(j)})^Tx^{(i)} - y^{(i,j)})\Theta_k^{(j)} +\lambda x^{(i)}_k$

$\frac{\partial J}{\partial \Theta^{(j)}_k} = \sum ((\Theta^{(j)})^Tx^{(i)} - y^{(i,j)})x_k^{(i)} +\lambda \Theta^{(j)}_k$

In [55]:

```
class CollaborativeFiltering:
    def __init__(self, num_features=NUM_FEATURES, gradient_step=0.5, reg_lambda=0.1, max_iters=5000
):
        self.num_features = num_features
        self.gradient_step = gradient_step
        self.reg_lambda = reg_lambda
        self.max_iters = max_iters

    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_lambda / 2) * ((self.X ** 2).sum() + (self.Theta ** 2).s
um())
        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_lambda * self.X
        regularized_dTheta = dTheta + self.reg_lambda * self.Theta
```

```
        self.X -= self.gradient_step * regularized_dX
        self.Theta -= self.gradient_step * regularized_dTheta

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

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

    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]
```

4. Реализуйте функцию вычисления градиентов.

CollaborativeFiltering.gradient_descent

5. При реализации используйте векторизацию для ускорения процесса обучения.

Все функции, реализованные в классе CollaborativeFiltering используют векторизацию

6. Добавьте L2-регуляризацию в модель.

Функция стоимости(cost_func) реализована с L2-регуляризацией.

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

In [56]:

```
rec = CollaborativeFiltering(gradient_step=0.001, reg_lambda=10)
rec.fit(Y, R)
```

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

In [57]:

```
my_ratings, presence = np.zeros(Y.shape[0], dtype=int), np.zeros(R.shape[0], dtype=int)
my_ratings[54], presence[54] = 5, 1 # Professional, The (1994)
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[540], presence[540] = 5, 1 # Mortal Kombat (1995)
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?

In [58]:

```
rec = CollaborativeFiltering(gradient_step=0.001, reg_lambda=10)
```

```
rec = CollaborativeFiltering(gradient_step=0.001, reg_lambda=10)
rec.fit(my_Y, my_R)
```

In [59]:

```
top_movies = rec.predict(user_id, my_R, top=5)
```

In [60]:

```
with open('data/movie_ids.txt', errors='ignore') as f:
    movie_names = f.read().split('\n')[:-1]

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

```
168 Monty Python and the Holy Grail (1974)
173 Princess Bride, The (1987)
89 Blade Runner (1982)
250 Fifth Element, The (1997)
172 Empire Strikes Back, The (1980)
```

Все рекомендации совпадают с предпочтениями, которые мы обозначили хорошо оценивая боевики.

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

In [61]:

```
class CollaborativeFilteringSVD(CollaborativeFiltering):
    def fit(self, Y, R):
        self.X, _, self.Theta = svds(Y.astype('float64'), k=NUM_FEATURES)
```

In [62]:

```
svd_rec = CollaborativeFilteringSVD()
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)
```

```
79 Fugitive, The (1993)
22 Braveheart (1995)
12 Usual Suspects, The (1995)
568 Speed (1994)
684 In the Line of Fire (1993)
```

Все рекомендации также совпадают с предпочтениями, которые мы обозначили хорошо оценивая боевики.