Отчёт по лабораторной работе №9 "Рекомендательные системы"

In [1]:

```
import numpy as np
from scipy.sparse.linalg import svds
np.seterr(divide='ignore', invalid='ignore', over='ignore')

DATA_FILE_NAME_1 = 'Lab 9/ex9_movies'
DATA_FILE_NAME_2 = 'Lab 9/movie_ids'
```

Imports from 'common' file:

In [2]:

```
import os
from scipy.io import loadmat

DATA_DIRECTORY = '../Data/'


def load_data(filename, convert_type=float, separator=',', directory=DATA_DIRECTORY, encoding='utf-8',
              skip_extention=False, split_function=None):

    filepath = directory + filename
    if not skip_extention:
        filepath += '.txt'

    if not split_function:
        def split_function(line):
            return line.replace('\n', '').split(separator)

    data = []
    with open(filepath, 'r', encoding=encoding) as f:
        for line in f.readlines():
            try:
                if isinstance(convert_type, (list, tuple)):
                    data.append([
                        convert_type[j](el)
                        for row in line.replace('\n', '')
                        for j, el in enumerate(row.split(separator))
                    ])
                else:
                    data.append([convert_type(x) for x in split_function(line)])
            except TypeError:
                pass

    return np.array(data)

def load_data_from_mat_file(filename, directory=DATA_DIRECTORY):
    filepath = directory + f'{filename}.mat'
    return loadmat(filepath)
```

1. Загрузите данные ex9_movies.mat из файла.

In [3]:

```
df_data = load_data_from_mat_file(DATA_FILE_NAME_1)
Y = df_data['Y']
R = df_data['R']
```

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

In [4]:

```
features_count = 50
```

Далее пункты 3-6 будут реализованы в следующем классе:

In [5]:

```
class RecomendationSystem:
    def __init__(self, features_count=10, reg_param=1e-1, learning_rate=1e-1, eps=1e-5,
iteration_count=1e4):
        self.iteration_count = iteration_count
        self.features_count = features_count
        self.learning_rate = learning_rate
        self.reg_param = reg_param
        self.eps = eps

    def _cost_function(self):
        return np.square((np.dot(self.X, self.Theta) - self.Y) * self.R).sum() / 2

    def _cost_function_with_reg(self):
        error = self._cost_function()
        return error + self.reg_param * (np.square(self.X).sum() + np.square(self.Theta).sum()) / 2

    def _gradient_function(self):
        mean_error = (np.dot(self.X, self.Theta) - self.Y) * self.R
        delta_theta = np.dot(self.X.T, mean_error)
        return self.learning_rate * (delta_theta + self.reg_param * self.Theta)

    def _gradient_function_with_reg(self):
        return self._gradient_function() + self.learning_rate * self.reg_param * self.Theta

    def gradient_descent(self):
        iteration = 0
        accuracy_limit_achieved = False

        while not accuracy_limit_achieved and iteration < self.iteration_count:
            theta_gradient = self._gradient_function_with_reg()
            self.Theta -= theta_gradient

            accuracy_limit_achieved = not np.any(np.where(theta_gradient > self.eps))
            iteration += 1

        return self.Theta, accuracy_limit_achieved, iteration

    def _train(self):
        theta, accuracy_limit_achieved, iteration = self.gradient_descent()
        print('Accuracy limit achieved:', accuracy_limit_achieved)
        print('Iterations:', iteration)

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

    def predict(self, user_id, top=10):
        predictions = np.dot(self.X, self.Theta)
        ratings = (self.R[:, user_id] != 1) * predictions[:, user_id]
        return ratings.argsort()[-top:][::-1]
```

3. Реализуйте функцию стоимости для алгоритма.

Функция стоимости включена в **RecomendationSystem** класс под названием **_cost_function**.

4. Реализуйте функцию вычисления градиентов.

Функция вычисления градиентов включена в **RecomendationSystem** класс под названием ___gradient_function.

5. При реализации используйте векторизацию для ускорения процесса обучения.

Все методы **RecomendationSystem** класса работают с векторизированными данными. Другими словами, все математические вычисления, преобразования в классе построены на матрицах и векторах.

6. Добавьте L2-регуляризацию в модель

Функции стоимости и градиента с L2-регуляризацией реализованы в **RecomendationSystem** классе под названиями **_cost_function_with_reg** и **_gradient_function_with_reg**.

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

In [7]:

```
recomendation_system = RecomendationSystem(features_count=features_count)
recomendation_system.fit(Y, R)
```

```
Accuracy limit achieved: True
Iterations: 2
```

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

In [8]:

```
# 72 Mask, The (1984); 73 Maverick (1994); # 250 Fifth Element, The (1997); 257 Men in Black (1997
);
# 204 Back to the Future (1985); 22 Braveheart (1995)
indexes = [72, 73, 250, 257, 204, 22]

my_ratings, presence = np.zeros(Y.shape[0], dtype=int), np.zeros(R.shape[0], dtype=int)
for i in indexes:
    my_ratings[i], presence[i] = 5, 1

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

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

In [10]:

```
movies_list = load_data(DATA_FILE_NAME_2, separator='\n', convert_type=str, encoding='ISO-8859-1')

print('Recomendation system response:')
recomendation_system = RecomendationSystem()
recomendation_system.fit(my_Y, my_R)
predictions = recomendation_system.predict(my_id)

print('\nPredicted movies:')
for movie in movies_list[predictions].flatten():
    print('- ', movie)
```

```
Recomendation system response:
Accuracy limit achieved: True
Iterations: 473

Predicted movies:
-  581 Kalifornia (1993)
-  202 Groundhog Day (1993)
-  232 Young Guns (1988)
-  1100 What Happened Was... (1994)
-  565 Village of the Damned (1995)
```

```
-  1301 Stripes (1981)
-  462 Like Water For Chocolate (Como agua para chocolate) (1992)
-  257 Men in Black (1997)
-  277 Restoration (1995)
-  567 Wes Craven's New Nightmare (1994)
```

Рекомендации совпадают с реальность. Предсказанные фильмы относятся к жанру приключения/фантастика/драма. Фильмы этих жанров были отмечены выскоим рейтингом. Стоит отметить, что в списке фильмов, относящихся к жанрам приключения и фантастика значительно больше.

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

In [12]:

```
class SVDRecomendationSystem(RecomendationSystem):
    def fit(self, Y, R):
        self.Y, self.R = Y, R
        self.X, _, self.Theta = svds(Y.astype('float64'), k=features_count)
        self._train()


print('\nSVD recomendation system response:')
svd_system = SVDRecomendationSystem()
svd_system.fit(my_Y, my_R)
svd_predictions = svd_system.predict(my_id)

print('\nPredicted movies using svd:')
for movie in movies_list[svd_predictions].flatten():
    print('- ', movie)

print('\nPrediction intersections: ', list(set(svd_predictions) & set(predictions)))
```

```
SVD recomendation system response:
Accuracy limit achieved: True
Iterations: 415

Predicted movies using svd:
-  257 Men in Black (1997)
-  187 Godfather: Part II, The (1974)
-  185 Psycho (1960)
-  127 Godfather, The (1972)
-  357 One Flew Over the Cuckoo's Nest (1975)
-  302 L.A. Confidential (1997)
-  28 Apollo 13 (1995)
-  177 Good, The Bad and The Ugly, The (1966)
-  208 Young Frankenstein (1974)
-  521 Deer Hunter, The (1978)

Prediction intersections:  [256]
```

Результаты также совпадают с реальностью. Алогоритм порекоммендовал фильмы жанра приключения/фантастика/драма. Фильмы этих жанров были отмечены выскоим рейтингом по сравнению с другими фильмами. Некоторые из фильмов даже совпали с предыдущими результатами, где модель была обучена с помощью градиентного спуска. Однако стоит отметить, склонность в предсказаниях данной модели к жанру драма - отличных вопрос для дальнейшего исследования.