Лабораторная работа №9: «Рекомендательные системы»

Набор данных **ex9_movies.mat** представляет собой файл формата *.mat (т.е. сохраненного из Matlab). Набор содержит две матрицы Y и R - рейтинг 1682 фильмов среди 943 пользователей. Значение Rij может быть равно 0 или 1 в зависимости от того оценил ли пользователь j фильм i. Матрица Y содержит числа от 1 до 5 - оценки в баллах пользователей, выставленные фильмам.

Задание.

1. Загрузите данные **ex9_movies.mat** из файла.
2. Выберите число признаков фильмов (n) для реализации алгоритма коллаборативной фильтрации.
3. Реализуйте функцию стоимости для алгоритма.
4. Реализуйте функцию вычисления градиентов.
5. При реализации используйте векторизацию для ускорения процесса обучения.
6. Добавьте L2-регуляризацию в модель.
7. Обучите модель с помощью градиентного спуска или других методов оптимизации.
8. Добавьте несколько оценок фильмов от себя. Файл **movie_ids.txt** содержит индексы каждого из фильмов.
9. Сделайте рекомендации для себя. Совпали ли они с реальностью?
10. Также обучите модель с помощью сингулярного разложения матриц. Отличаются ли полученные результаты?
11. Ответы на вопросы представьте в виде отчета.

Реализация:

In[1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pnd
```

1.1 Загрузите набор данных ex9_movies.mat из файла.

In[2]:

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

1.2 Постройте график загруженных данных в виде диаграммы рассеяния

In[3]:

```
NUM_FEATURES = 15
```

1.3 Реализуйте функцию стоимости для алгоритма

CollaborativeFiltering.cost_func

Функция стоимости:

$$J(x^{(i)}, ..., x^{(n_m)}, \Theta^{(I)}, ..., \Theta^{(n_u)}) = \frac{1}{2}\sum((\Theta^{(j)})^T x^{(i)} - y^{(i,\ j)})^2 + (\frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\Theta_k^{(j)})^2) + (\frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2)$$

Вычисление градиента:

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

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

In[4]:

```
class CollaborativeFiltering:
    def __init__(self, num_features=NUM_FEATURES, gradi-
ent_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
        regular-
ized_cost = cost + (self.reg_lambda / 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_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]class CollaborativeFiltering:
    def __init__(self, num_features=NUM_FEATURES, gradi-
ent_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)
```

1.4 Реализуйте функцию вычисления градиентов

CollaborativeFiltering.gradient_descent

1.5 При реализации используйте векторизацию для ускорения процесса обучения

Все функции, реализованные в классе CollaborativeFiltering используют векторизацию