Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ЛАБОРАТОРНАЯ РАБОТА №3

«Переобучение и регуляризация»

Студент А. Ю. Омельчук

Преподаватель М. В. Стержанов

Минск 2019

ХОД РАБОТЫ

Задание.

Набор данных ex3data1.mat представляет собой файл формата *.mat (т.е. сохраненного из Matlab). Набор содержит две переменные X (изменения уровня воды) и y (объем воды, вытекающий из дамбы). По переменной X необходимо предсказать y. Данные разделены на три выборки: обучающая выборка (X, y), по которой определяются параметры модели; валидационная выборка (Xval, yval), на которой настраивается коэффициент регуляризации; контрольная выборка (Xtest, ytest), на которой оценивается качество построенной модели.

1. Загрузите данные ex3data1.mat из файла.
2. Постройте график, где по осям откладываются X и y из обучающей выборки.
3. Реализуйте функцию стоимости потерь для линейной регрессии с L2-регуляризацией.
4. Реализуйте функцию градиентного спуска для линейной регрессии с L2-регуляризацией.
5. Постройте модель линейной регрессии с коэффициентом регуляризации 0 и постройте график полученной функции совместно с графиком из пункта 2. Почему регуляризация в данном случае не сработает?
6. Постройте график процесса обучения (learning curves) для обучающей и валидационной выборки. По оси абсцисс откладывается число элементов из обучающей выборки, а по оси ординат - ошибка (значение функции потерь) для обучающей выборки (первая кривая) и валидационной выборки (вторая кривая). Какой вывод можно сделать по построенному графику?
7. Реализуйте функцию добавления p - 1 новых признаков в обучающую выборку (X2, X3, X4, …, Xp).
8. Поскольку в данной задаче будет использован полином высокой степени, то необходимо перед обучением произвести нормализацию признаков.
9. Обучите модель с коэффициентом регуляризации 0 и p = 8.
10. Постройте график модели, совмещенный с обучающей выборкой, а также график процесса обучения. Какой вывод можно сделать в данном случае?

11. Постройте графики из пункта 10 для моделей с коэффициентами регуляризации 1 и 100. Какие выводы можно сделать?
12. С помощью валидационной выборки подберите коэффиент регуляризации, который позволяет достичь наименьшей ошибки. Процесс подбора отразите с помощью графика (графиков).
13. Вычислите ошибку (потерю) на контрольной выборке.

Результат выполнения:

1. Код выгрузки данных из файла представлен ниже:

```
file_path = os.path.join(os.path.dirname(__file__), 'data', 'ex3data1.mat')
dataset = sio.loadmat(file_path)
x_train = dataset["X"]
x_val = dataset["Xval"]
x_test = dataset["Xtest"]

# squeeze the target variables into one-dimensional arrays
y_train = dataset["y"].squeeze()
y_val = dataset["yval"].squeeze()
y_test = dataset["ytest"].squeeze()
```

2. График представлен ниже:

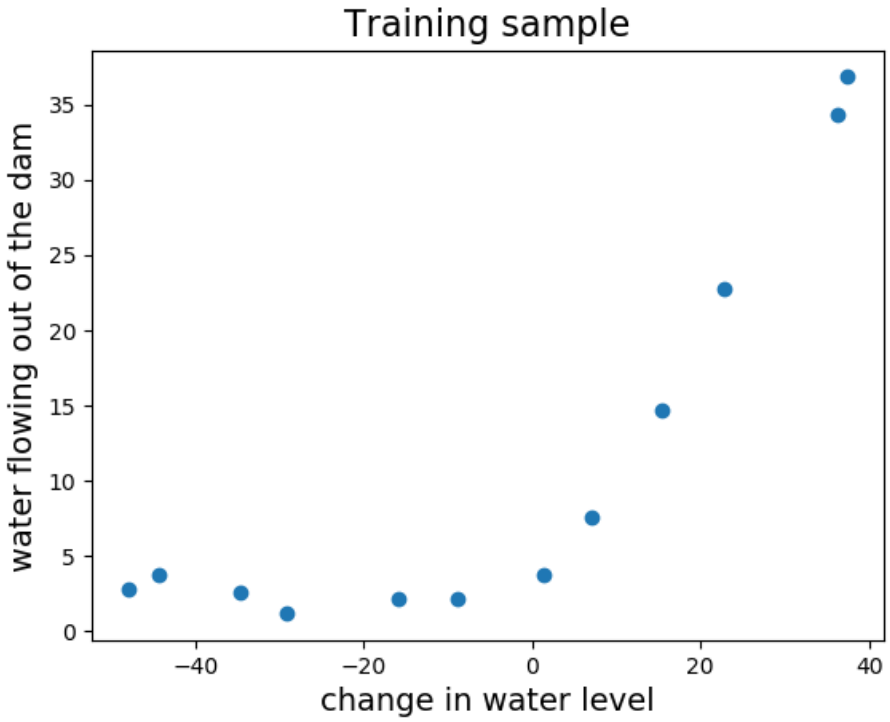

Рисунок 1 – график зависимости объёма воды, вытекающего из дамбы от изменения уровня воды

3. Код функции потерь:

```
def cost_l2(theta, X, y, lamb=0):
    predictions = h0x(X, theta)
    squared_errors = np.sum(np.square(predictions - y))
    regularization = np.sum(lamb * np.square(theta[1:]))
    return (squared_errors + regularization) / (2 * len(y))
```

4. Функция градиентного спуска:

```
def gradient_l2(theta, X, y, lamb):
    predictions = h0x(X, theta)
    gradient = np.dot(X.transpose(), (predictions - y))
    regularization = lamb * theta
    regularization[0] = 0  # because formula for 0 member is different
    return (gradient + regularization) / len(y)
```

5. Графики приведены ниже:

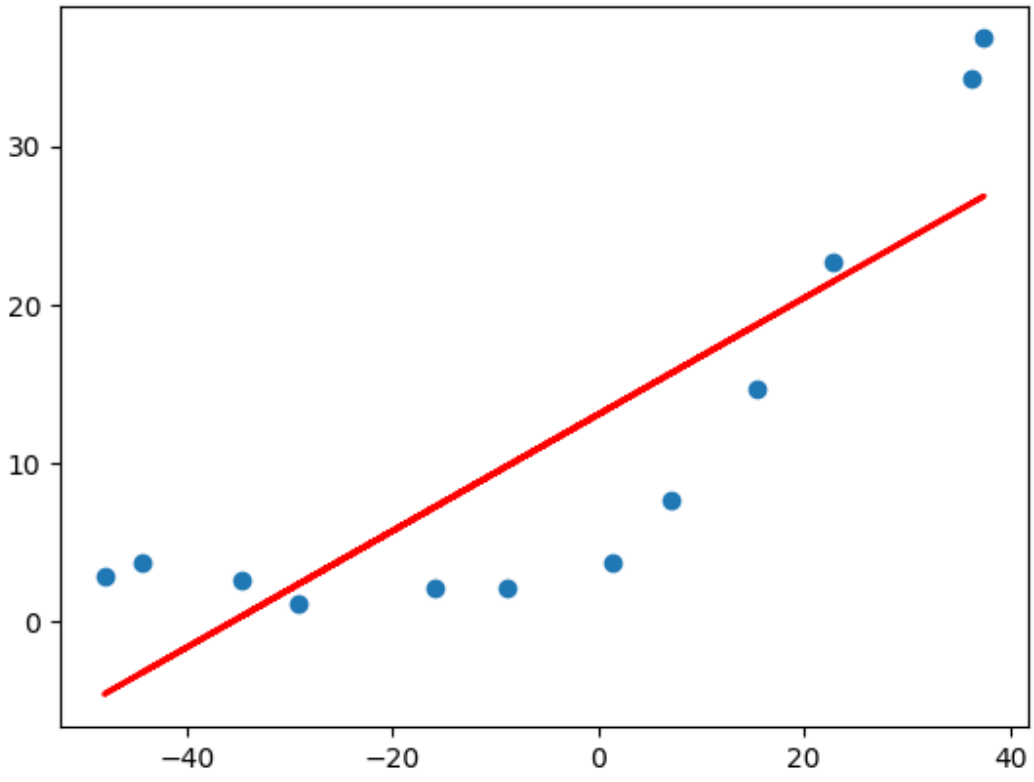

Рисунок 2 – график полученной функции

Регуляризация в данном случае не сработает, потому что она равна нулю и функция линейна. В общем виде можно увидеть, что когда мы добавляем регуляризацию, то любую полиномиальную функцию таким образом мы

стараемся как бы «выпрямить». Но если пытаться применять регуляризацию к линейной функции, то это абсолютно ничего не даст. В данном случае можно было бы обучать и с lambda=1000, но результат был бы абсолютно такой же. Так же на графике явно выражен high bias, и использование регуляризации здесь также не поможет.

6. График приведён ниже:

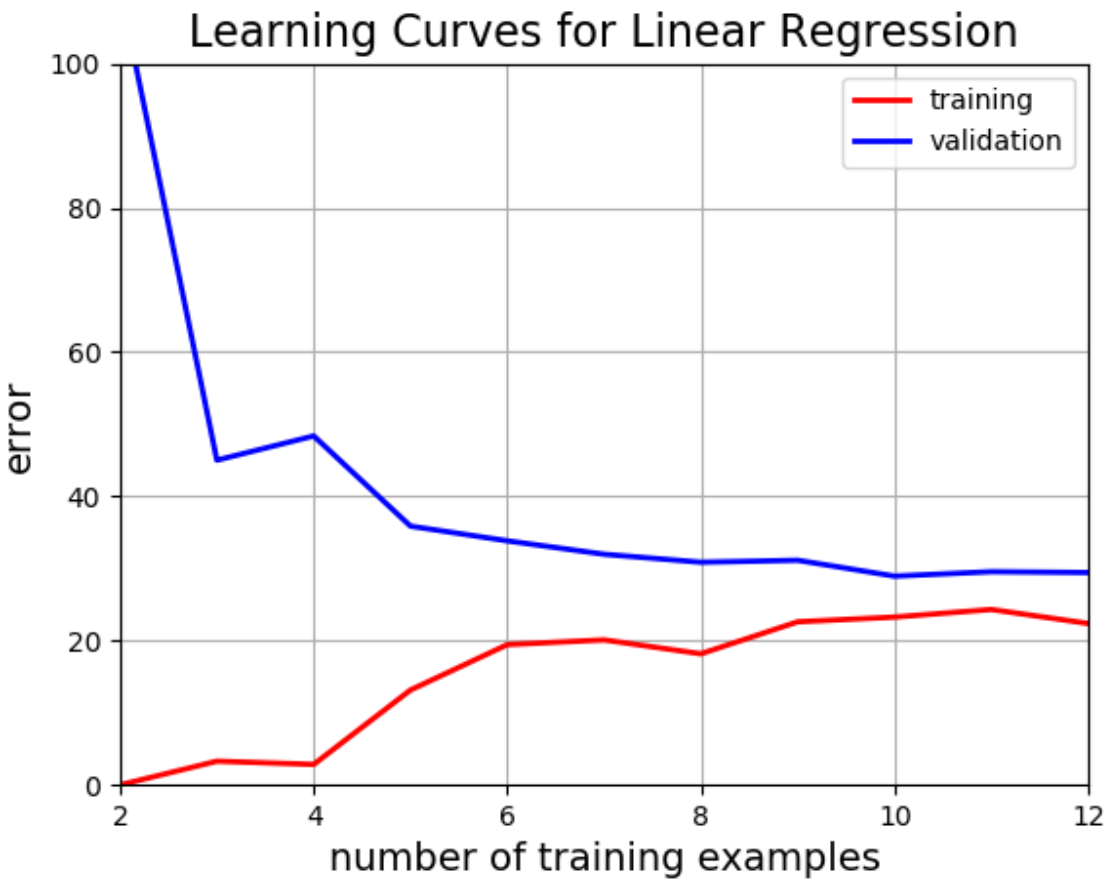

Рисунок 3 – график зависимости ошибки на валидационном и тренировочном сете в зависимости от количества примеров

На графике очевиден high bias problem (underfitting). Выбранная hypothesis (линейная функция) не может подойти данной выборке. Один из вариантов улучшения – усложнить функцию, добавив полиномы высшей степени.

7. Код реализации:

```
def polynom(x, degree):
    X_poly = np.zeros(shape=(len(x), degree))
```

```
    for i in range(0, degree):
        X_poly[:, i] = x.squeeze() ** (i + 1);
    return X_poly

x_train_poly = polynom(x_train, 8)
x_val_poly = polynom(x_val, 8)
x_test_poly = polynom(x_test, 8)
```

8. Код реализации:

```
train_means = x_train_poly.mean(axis=0)
train_std = np.std(x_train_poly, axis=0, ddof=1)

x_train_poly = (x_train_poly - train_means) / train_std
x_val_poly = (x_val_poly - train_means) / train_std
x_test_poly = (x_test_poly - train_means) / train_std

X_train_poly = np.hstack((np.ones((len(x_train_poly), 1)), x_train_poly))
X_val_poly = np.hstack((np.ones((len(x_val_poly), 1)), x_val_poly))
X_test_poly = np.hstack((np.ones((len(x_test_poly), 1)), x_test_poly))
```

9. Код реализации:

```
theta = opt.fmin_cg(cost_l2, np.zeros(X_train_poly.shape[1]), gradient_l2, (X_train_poly,
                                y_train, 0), disp=False)
```

10. Графики приведены ниже:

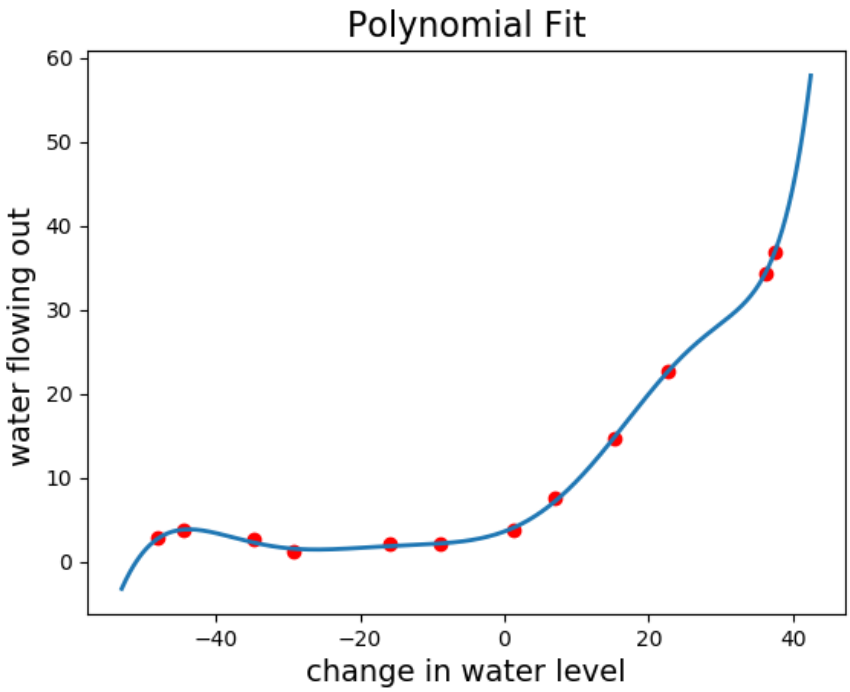

Рисунок 4 – график полученной функции совмещенный с исходными данными (lambda=0)

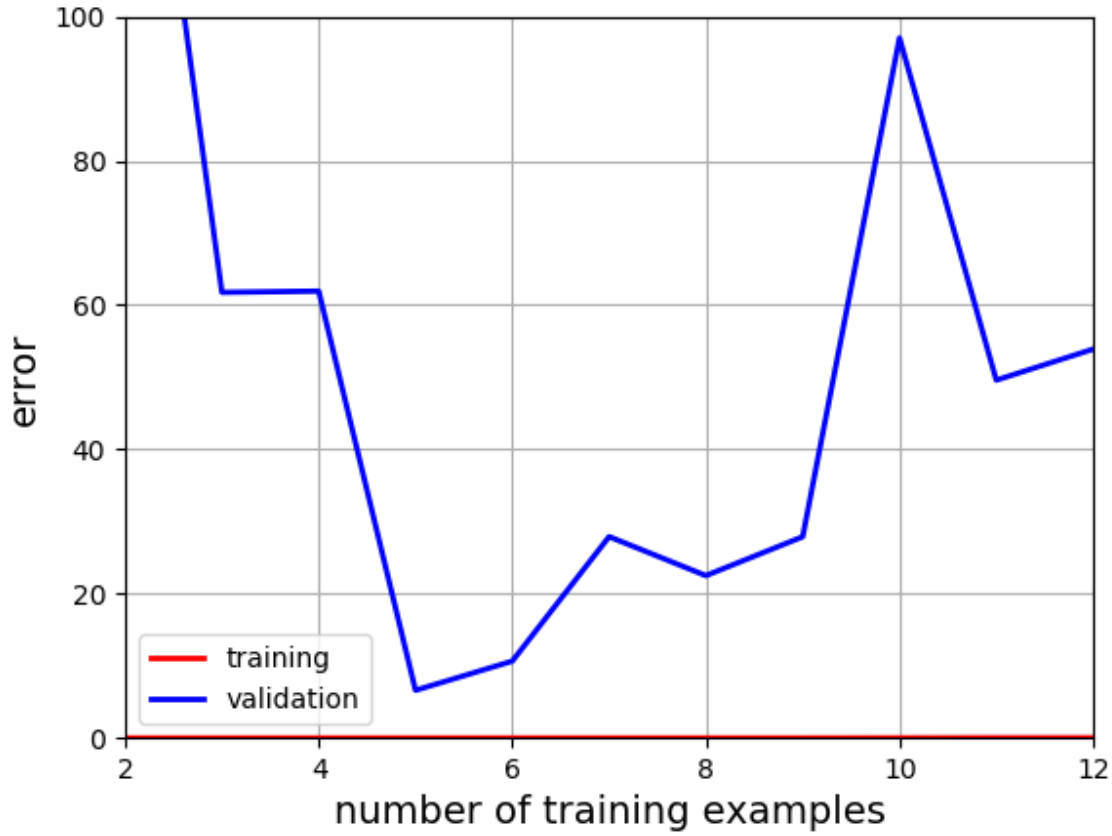

Рисунок 5 – график зависимости ошибки на валидационном и тренировочном сете в зависимости от количества примеров (lambda=0)

На первом графике видно, что полученная функция идеально копирует все данные тренировочного сета (потому что параметр регуляризации=0). Однако, на втором графике можно увидеть, что ошибка на валидационном сете ведёт себя абсолютно непредсказуемо и очень сильно варьируется, что вполне ожидаемо. Т. е. в текущей реализации мы идеально сможем предсказать все тренировочные семплы (что доказывает красная линия на втором графике, которая постоянно равна нулю), однако на реальных или валидационных данных результат будет просто неадекватным, и, скорее всего, не будет представлять ничего общего с ожидаемым результатом (ситуация, когда модель переобучена). Т. е. очевиден overfitting. В общем случае, эта проблема решается увеличением коэффициента регуляризации и добавлением дополнительных данных в тренировочный список. Альтернативный вариант – уменьшить полиномиальную степень hypothesis, что уменьшит вычисления.

11. Графики приведены ниже:

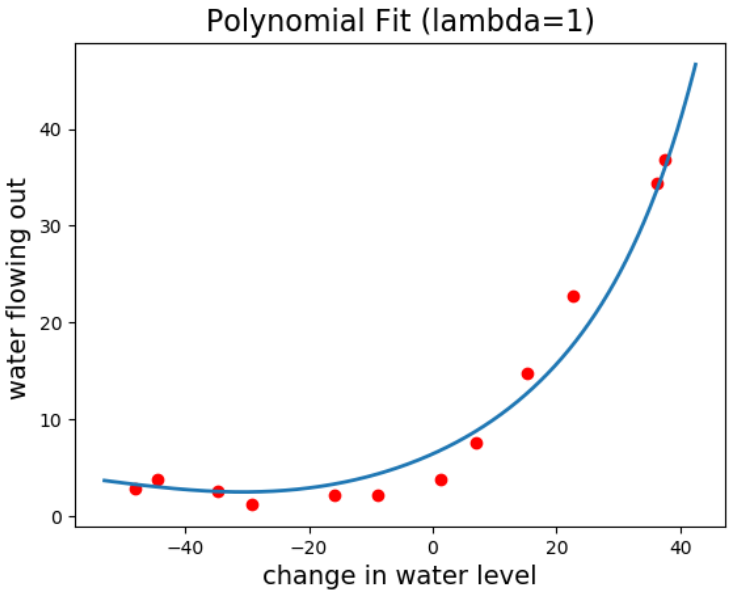

Рисунок 6 – график полученной функции совмещенный с исходными данными (lambda=1)

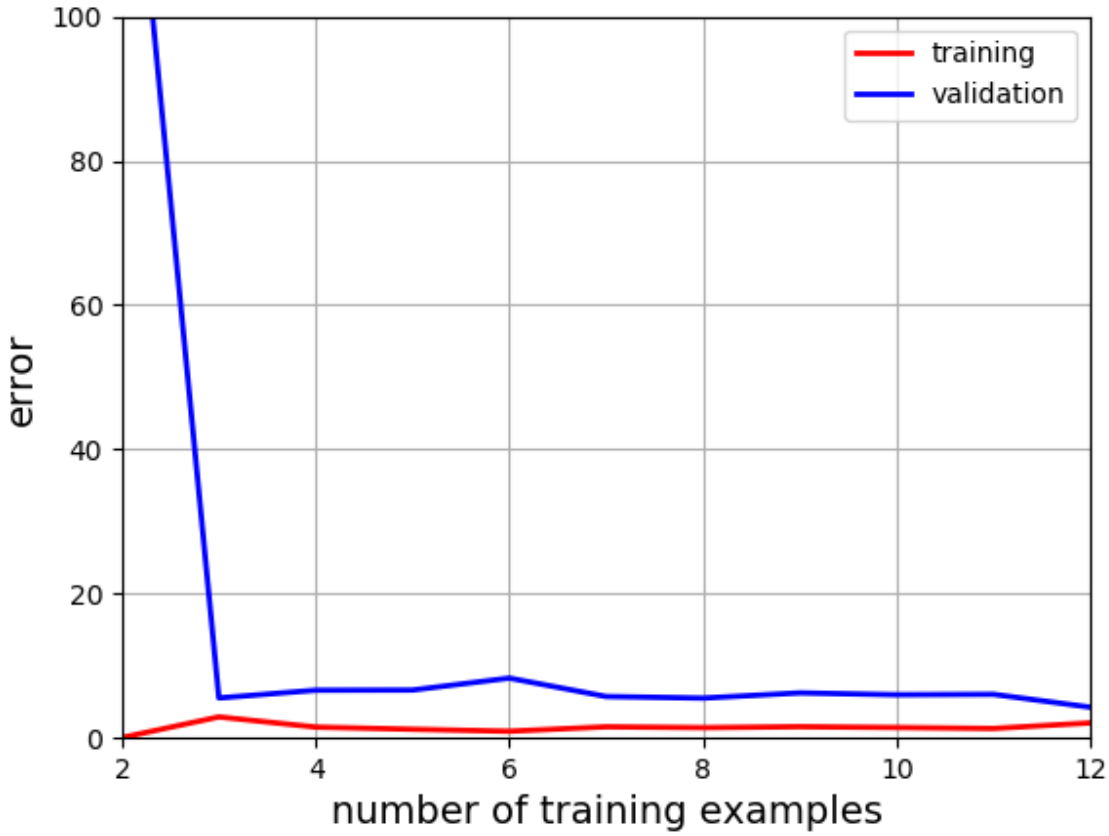

Рисунок 7 – график зависимости ошибки на валидационном и тренировочном сете в зависимости от количества примеров (lambda=1)

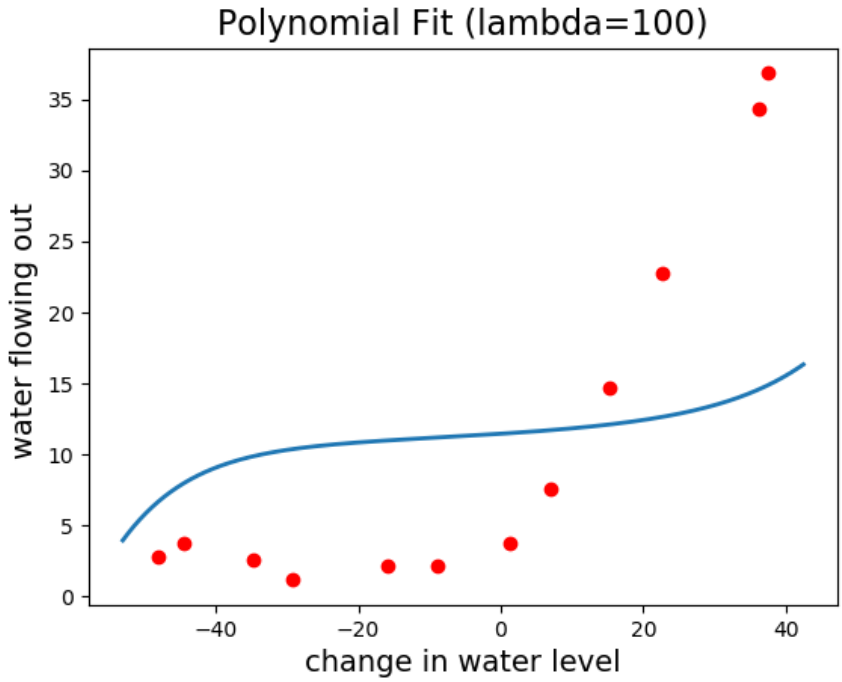

Рисунок 8 – график полученной функции совмещенный с исходными данными (lambda=100)

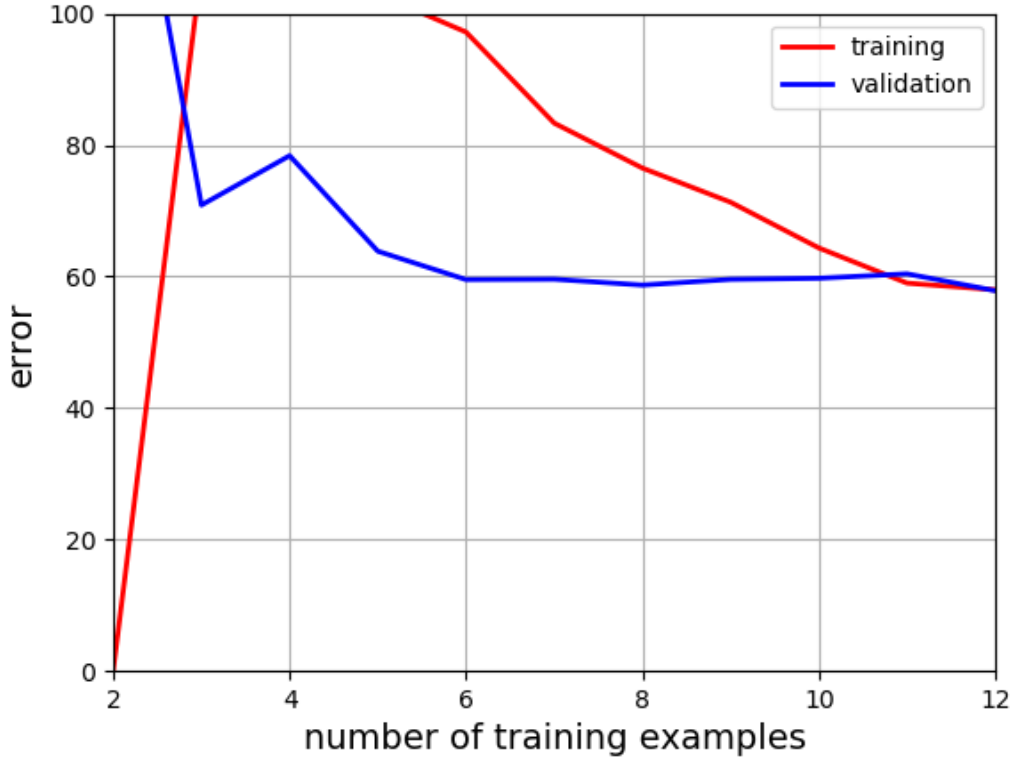

Рисунок 9 – график зависимости ошибки на валидационном и тренировочном сете в зависимости от количества примеров (lambda=100)

На первом графике мы можем наблюдать, как наша искомая функция начала больше выпрямляться и она не так идеально копирует все дата семплы. Это же подтверждает и второй график, когда мы видим, что ошибки на валидационной выборке и тренировочной не равны нулю и в это же время они обе относительно малы (обучение прошло корректно – параметры модели подобраны правильно).

Что же касается второго графика, то здесь мы можем заметить, что на графике кривой с примерами этот график вообще никоим образом не копирует дата семплы, и, более того, он даже не отображает общий вектор, в котором должен двигаться. То есть даже линейная регрессия имеет меньшую ошибку, чем данная модель. Это же и подтверждают высокие ошибки на втором графике, которые хоть и сходятся друг к другу, но крайне высоки, что означает, что данная модель не обучилась на наших данных (недообучение).

12. График приведён ниже:

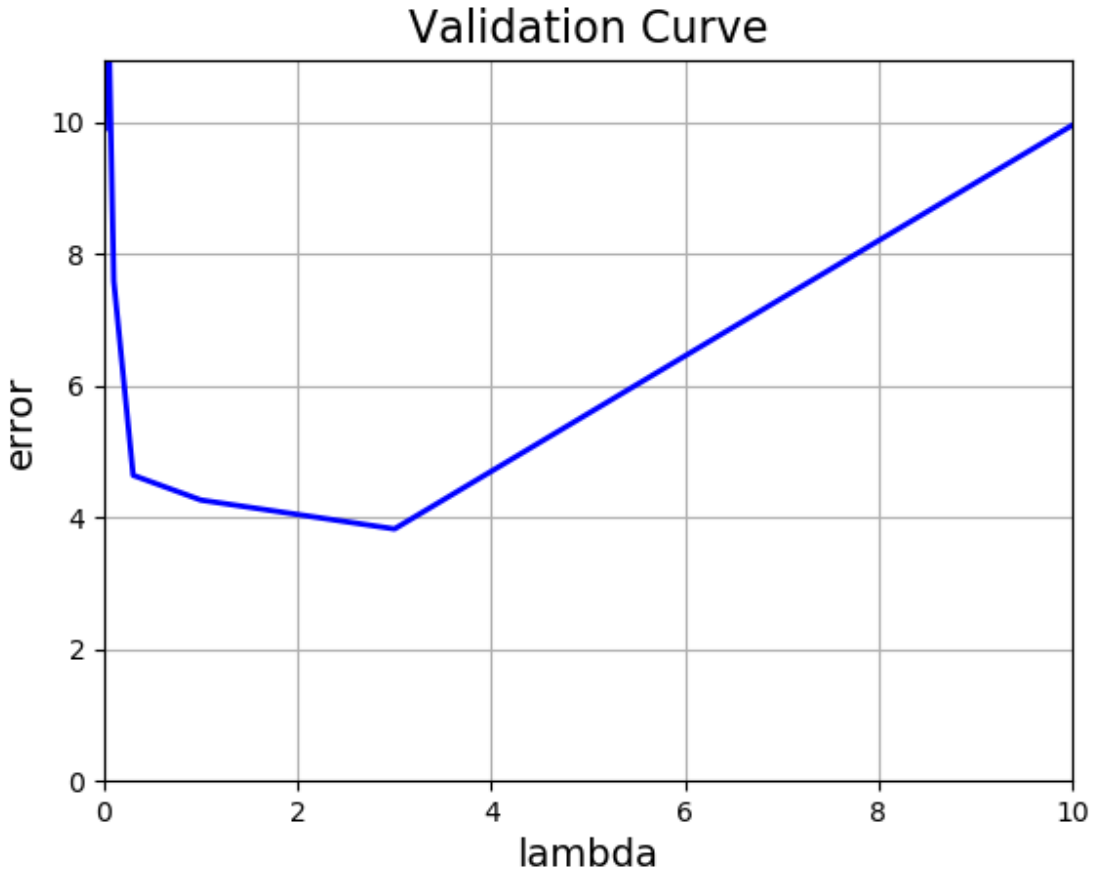

Рисунок 10 – график зависимости ошибки на валидационном сете от параметра lambda

13. Код реализации:

```
    theta = opt.fmin_cg(cost_l2, np.zeros(X_train_poly.shape[1]), gradient_l2, (X_train_poly,
y_train, 3), disp=False)
    test_error = cost_l2(theta, X_test_poly, y_test)
    print("Test Error: ", test_error, "| Regularized Polynomial (lambda=3))")
```

Реультат выполнения:

```
    'Test Error: ', 3.8598878210299112, '| Regularized Polynomial (lambda=3))'
```

Программный код:

```
    import scipy.io as sio
    import matplotlib.pyplot as plt
    import os
    import numpy as np
    from scipy import optimize as opt


    def h0x(X, theta):
        return np.dot(X, theta)


    def cost_l2(theta, X, y, lamb=0):
        predictions = h0x(X, theta)
        squared_errors = np.sum(np.square(predictions - y))
        regularization = np.sum(lamb * np.square(theta[1:]))  # TODO: or theta?
        return (squared_errors + regularization) / (2 * len(y))


    def gradient_l2(theta, X, y, lamb):
        predictions = h0x(X, theta)
        gradient = np.dot(X.transpose(), (predictions - y))
        regularization = lamb * theta
        regularization[0] = 0  # because formula for 0 member is different
        return (gradient + regularization) / len(y)


    def learning_curves_chart(X_train, y_train, X_val, y_val, lambda_):
        m = len(y_train)
        train_err = np.zeros(m)
        val_err = np.zeros(m)
        for i in range(1, m):
            theta = opt.fmin_cg(cost_l2, np.zeros(X_train.shape[1]), gradient_l2, (X_train[0:i
+ 1, :], y_train[0:i + 1], lambda_), disp=False)
            train_err[i] = cost_l2(theta, X_train[0:i + 1, :], y_train[0:i + 1])
            val_err[i] = cost_l2(theta, X_val, y_val)
        plt.plot(range(2, m + 1), train_err[1:], c="r", linewidth=2)
        plt.plot(range(2, m + 1), val_err[1:], c="b", linewidth=2)
        plt.xlabel("number of training examples", fontsize=14)
        plt.ylabel("error", fontsize=14)
        plt.legend(["training", "validation"], loc="best")
        plt.axis([2, m, 0, 100])
```

```
    plt.grid()
    plt.show()


def polynom(x, degree):
    X_poly = np.zeros(shape=(len(x), degree))
    for i in range(0, degree):
        X_poly[:, i] = x.squeeze() ** (i + 1);
    return X_poly


if __name__ == '__main__':
    # 1
    file_path = os.path.join(os.path.dirname(__file__), 'data', 'ex3data1.mat')
    dataset = sio.loadmat(file_path)
    x_train = dataset["X"]
    x_val = dataset["Xval"]
    x_test = dataset["Xtest"]

    # squeeze the target variables into one-dimensional arrays
    y_train = dataset["y"].squeeze()
    y_val = dataset["yval"].squeeze()
    y_test = dataset["ytest"].squeeze()

    # 2
    fig, ax = plt.subplots()
    ax.scatter(x_train, y_train)
    plt.xlabel("change in water level", fontsize=14)
    plt.ylabel("water flowing out of the dam", fontsize=14)
    plt.title("Training sample", fontsize=16)
    plt.show()

    # data preparation
    X_train = np.hstack((np.ones((len(x_train), 1)), x_train))
    theta = np.zeros(X_train.shape[1])

    # 3
    print(cost_l2(theta, X_train, y_train, 0))

    # 4
    print(gradient_l2(theta, X_train, y_train, 0))

    # 5
    theta = opt.fmin_cg(cost_l2, theta, gradient_l2, (X_train, y_train, 0), disp=False)
    print(theta)

    # 6
    h = np.dot(X_train, theta)
    fig, ax = plt.subplots()
    ax.scatter(x_train, y_train)
    ax.plot(X_train[:, 1], h, linewidth=2, color='red')
    plt.show()

    X_val = np.hstack((np.ones((len(x_val), 1)), x_val))
    plt.title("Learning Curves for Linear Regression", fontsize=16)
    learning_curves_chart(X_train, y_train, X_val, y_val, 0)

    # 7
    x_train_poly = polynom(x_train, 8)
```

```
        x_val_poly = polynom(x_val, 8)
        x_test_poly = polynom(x_test, 8)

        # 8
        train_means = x_train_poly.mean(axis=0)
        train_std = np.std(x_train_poly, axis=0, ddof=1)

        x_train_poly = (x_train_poly - train_means) / train_std
        x_val_poly = (x_val_poly - train_means) / train_std
        x_test_poly = (x_test_poly - train_means) / train_std

        X_train_poly = np.hstack((np.ones((len(x_train_poly), 1)), x_train_poly))
        X_val_poly = np.hstack((np.ones((len(x_val_poly), 1)), x_val_poly))
        X_test_poly = np.hstack((np.ones((len(x_test_poly), 1)), x_test_poly))

        # 9
        theta    =    opt.fmin_cg(cost_l2,    np.zeros(X_train_poly.shape[1]),    gradient_l2,
(X_train_poly, y_train, 0), disp=False)
        x = np.linspace(min(x_train) - 5, max(x_train) + 5, 1000)
        x_polynom = polynom(x, 8)
        print(x_polynom)
        x_polynom = (x_polynom - train_means) / train_std
        x_polynom = np.hstack((np.ones((len(x_polynom), 1)), x_polynom))

        # 10
        fig, ax = plt.subplots()
        plt.scatter(x_train, y_train, color='red')
        plt.plot(x, h0x(x_polynom, theta), linewidth=2)
        plt.xlabel("change in water level", fontsize=14)
        plt.ylabel("water flowing out ", fontsize=14)
        plt.title("Polynomial Fit", fontsize=16)
        plt.show()

        learning_curves_chart(X_train_poly, y_train, X_val_poly, y_val, 0)

        # 11
        # lambda = 1
        theta    =    opt.fmin_cg(cost_l2,    np.zeros(X_train_poly.shape[1]),    gradient_l2,
(X_train_poly, y_train, 1), disp=False)
        x = np.linspace(min(x_train) - 5, max(x_train) + 5, 1000)
        x_polynom = polynom(x, 8)
        x_polynom = (x_polynom - train_means) / train_std
        x_polynom = np.hstack((np.ones((len(x_polynom), 1)), x_polynom))
        fig, ax = plt.subplots()
        plt.scatter(x_train, y_train, color='red')
        plt.plot(x, h0x(x_polynom, theta), linewidth=2)
        plt.xlabel("change in water level", fontsize=14)
        plt.ylabel("water flowing out ", fontsize=14)
        plt.title("Polynomial Fit (lambda=1)", fontsize=16)
        plt.show()

        learning_curves_chart(X_train_poly, y_train, X_val_poly, y_val, 1)

        # lambda = 100
        theta    =    opt.fmin_cg(cost_l2,    np.zeros(X_train_poly.shape[1]),    gradient_l2,
(X_train_poly, y_train, 100), disp=False)
        x = np.linspace(min(x_train) - 5, max(x_train) + 5, 1000)
        x_polynom = polynom(x, 8)
        x_polynom = (x_polynom - train_means) / train_std
```

```
        x_polynom = np.hstack((np.ones((len(x_polynom), 1)), x_polynom))
        fig, ax = plt.subplots()
        plt.scatter(x_train, y_train, color='red')
        plt.plot(x, h0x(x_polynom, theta), linewidth=2)
        plt.xlabel("change in water level", fontsize=14)
        plt.ylabel("water flowing out ", fontsize=14)
        plt.title("Polynomial Fit (lambda=100)", fontsize=16)
        plt.show()

        learning_curves_chart(X_train_poly, y_train, X_val_poly, y_val, 100)

        # 12
        lambda_values = [0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10]
        val_err = []
        for lamb in lambda_values:
            theta   =   opt.fmin_cg(cost_l2,   np.zeros(X_train_poly.shape[1]),   gradient_l2,
(X_train_poly, y_train, lamb), disp=False)
            val_err.append(cost_l2(theta, X_val_poly, y_val))
        plt.plot(lambda_values, val_err, c="b", linewidth=2)
        plt.axis([0, len(lambda_values), 0, val_err[-1] + 1])
        plt.grid()
        plt.xlabel("lambda", fontsize=14)
        plt.ylabel("error", fontsize=14)
        plt.title("Validation Curve", fontsize=16)
        plt.show()

        # 13
        theta    =    opt.fmin_cg(cost_l2,    np.zeros(X_train_poly.shape[1]),    gradient_l2,
(X_train_poly, y_train, 3), disp=False)
        test_error = cost_l2(theta, X_test_poly, y_test)
        print("Test Error: ", test_error, "| Regularized Polynomial (lambda=3))")
```