Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Кафедра информатики

Отчет по лабораторной работе:

**Лабораторная работа №3 “Переобучение и регуляризация”**

Выполнил: Карп Александр Игоревич

магистрант кафедры информатики

группа №858641

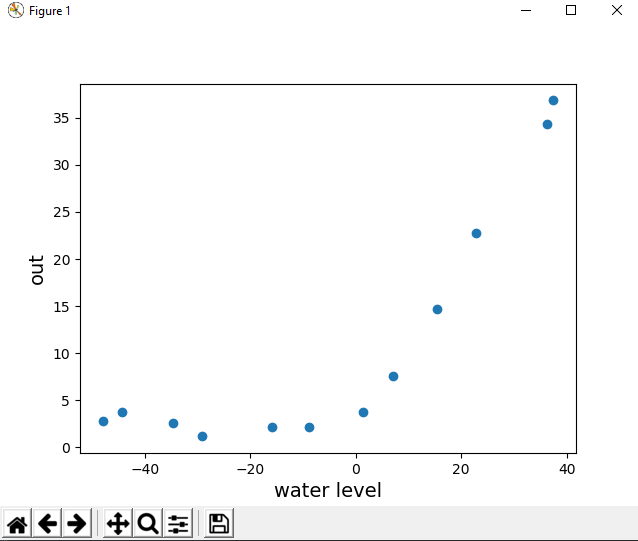
Минск 2019

1. Загрузите данные **ex3data1.mat** из файла.

# 1  
dataset = sio.loadmat("ex3data1.mat")  
  
x\_train = pd.DataFrame(dataset["X"])  
x\_val = pd.DataFrame(dataset["Xval"])  
x\_test = pd.DataFrame(dataset["Xtest"])  
  
y\_train = dataset["y"].squeeze()  
y\_val = dataset["yval"].squeeze()  
y\_test = dataset["ytest"].squeeze()

1. Постройте график, где по осям откладываются X и y из обучающей выборки.

fig, ax = plt.subplots()  
ax.scatter(x\_train, y\_train)  
plt.xlabel("water level", fontsize=14)  
plt.ylabel("out", fontsize=14)  
plt.show()



3) Реализуйте функцию стоимости потерь для линейной регрессии с L2-регуляризацией.

def loss(theta, X, y, lmbd=0):  
 units = np.full((len(X)), 1) # единичный вектор  
 return ((X @ theta - y) \*\* 2 @ units + lmbd \* (np.sum(theta[1:] \*\* 2))) / (len(y))

4) Реализуйте функцию градиентного спуска для линейной регрессии с L2-регуляризацией.

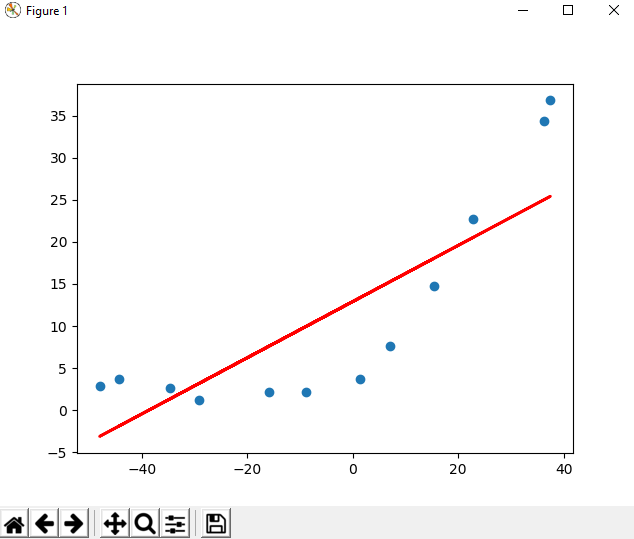
Воспользуемся алгоритмом Бройдена — Флетчера — Гольдфарба — Шанно:

def gradient(theta, X, y, lmbd=0):  
 diff = np.abs(lmbd\*theta)  
 diff[0] = 0  
 return (2 / len(y)) \*( X.T @ (X @ theta - y) + diff )

theta\_bfgs = optimize.fmin\_bfgs(  
 loss,  
 theta.flatten(),  
 gradient,  
 (x\_train\_ones.values, y\_train, 0)  
)

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?

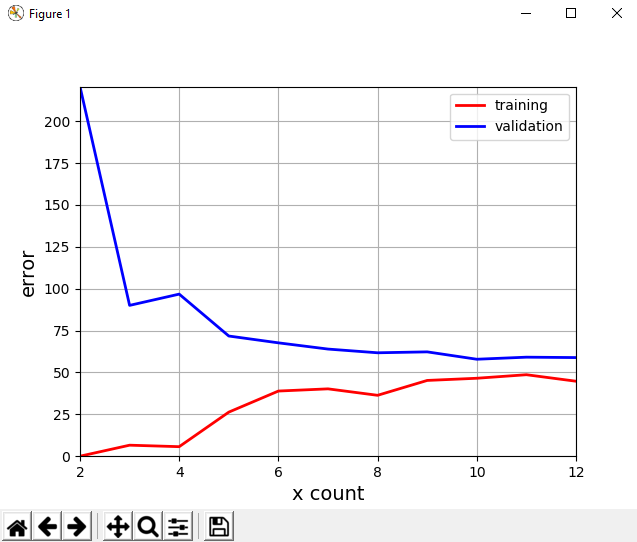
theta\_bfgs = optimize.fmin\_bfgs(  
 loss,  
 theta.flatten(),  
 gradient,  
 (x\_train\_ones.values, y\_train, 0)  
)



Регуляризация в данном случае не сработает в первую очереди из-за того, что наша функция линейна. Во-вторых, будь она нелинейна, регуляризация бы не сработала, так как регуляризационный коэффициент равен нулю.

1. 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?

def learning\_curves\_chart(X\_train, y\_train, X\_val, y\_val, lambda\_=0):  
 m = len(y\_train)  
 train\_err = np.zeros(m)  
 val\_err = np.zeros(m)  
 theta = np.zeros(X\_train.shape[1])  
 for i in range(1, m):  
 theta\_bfgs = optimize.fmin\_bfgs(  
 loss,  
 theta.flatten(),  
 gradient,  
 (X\_train[0:i + 1, :], y\_train[0:i + 1], lambda\_)  
 )  
 train\_err[i] = loss(theta\_bfgs, X\_train[0:i + 1, :], y\_train[0:i + 1])  
 val\_err[i] = loss(theta\_bfgs, 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("x count", fontsize=14)  
 plt.ylabel("error", fontsize=14)  
 plt.legend(["training", "validation"])  
 plt.axis([2, m, 0, max(max(val\_err), max(train\_err))])  
 plt.grid()  
 plt.show()



С ростом количества данных в обучающей выборке ошибка на тренировочном датасете растет, а на валидационном наоборот, уменьшается. Со временем обе ошибка стабилизируются и остаются на одном уровне. Исходные данные, судя по графику из п.2, не могут быть точно апроксимированы линейной функцией, из-за этого мы видим достаточно высокую ошибку.

1. Реализуйте функцию добавления p - 1 новых признаков в обучающую выборку (X2, X3, X4, …, Xp).

def add\_params(X\_original, p=0):  
 X\_copy = X\_original.copy()  
 for i in range(2, p + 1):  
 X\_copy[i] = X\_copy[0] \*\* i  
 return X\_copy

x\_poly =add\_params(x\_train, 8)

1. Поскольку в данной задаче будет использован полином высокой степени, то необходимо перед обучением произвести нормализацию признаков.

def normalize(X\_original):  
 x\_scaled = (X\_original - train\_means) / train\_std  
 df = pd.DataFrame(x\_scaled)  
 return df

train\_means = x\_poly.mean(axis=0)  
train\_std = np.std(x\_poly, axis=0, ddof=1)  
x\_scaled = normalize(x\_poly)

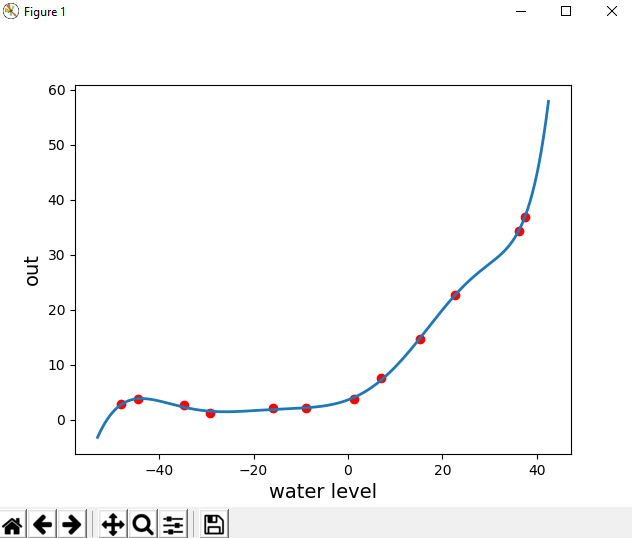
1. Обучите модель с коэффициентом регуляризации 0 и p = 8.

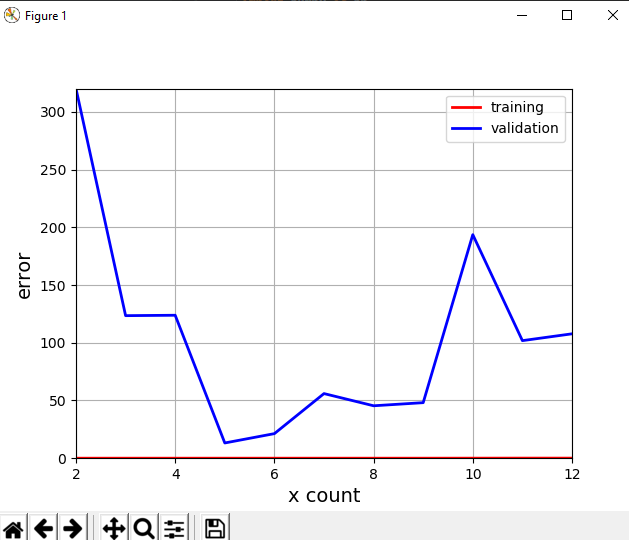
theta = np.zeros(x\_scaled.shape[1])  
theta\_bfgs\_scaled = theta\_bfgs = optimize.fmin\_bfgs(  
 loss,  
 theta.flatten(),  
 gradient,  
 (x\_scaled.values, y\_train, 0)  
)

1. Постройте график модели, совмещенный с обучающей выборкой, а также график процесса обучения. Какой вывод можно сделать в данном случае?

x = pd.DataFrame(np.linspace(min(x\_train.values) - 5, max(x\_train.values) + 5, 1000))  
  
x\_polynom = add\_params(x, 8)  
x\_polynom = normalize(x\_polynom)  
x\_polynom.insert(0, 'Ones', 1)  
  
fig, ax = plt.subplots()  
plt.scatter(x\_train.values, y\_train, color='red')  
plt.plot(x, x\_polynom @ theta\_bfgs\_scaled, linewidth=2)  
plt.xlabel("water level", fontsize=14)  
plt.ylabel("out", fontsize=14)  
plt.show()

val = normalize(add\_params(x\_val, 8))  
val.insert(0, 'Ones', 1)  
learning\_curves\_chart(x\_scaled.values, y\_train, val.values, y\_val, 0)

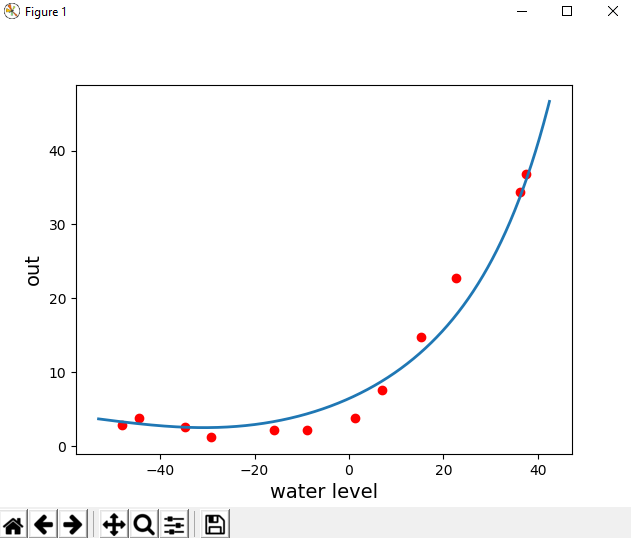


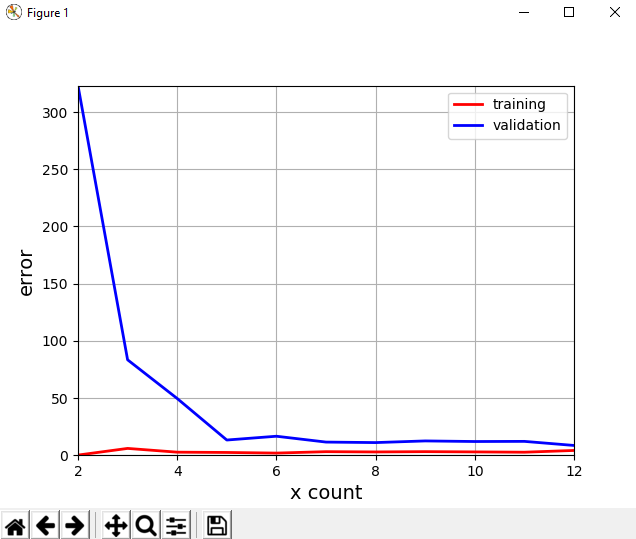


Здесь можно наблюдать, что модель идеально подстроилась под данные из тренировочного сета, но ошибка на валидационном сете непредсказуема, т.к. , возникло переобучение

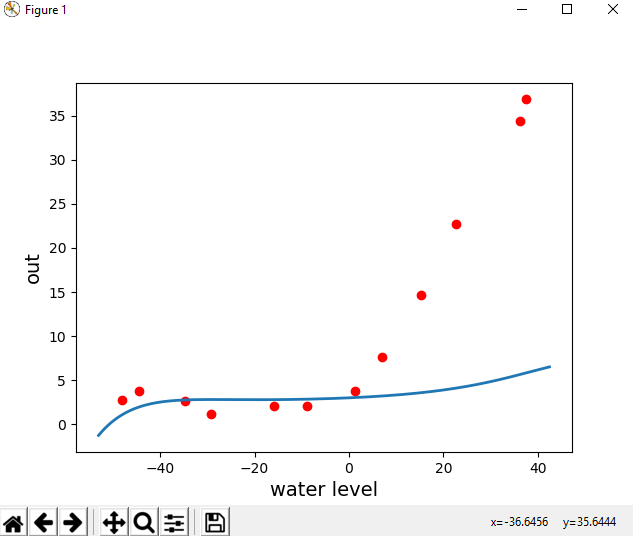
11. Постройте графики из пункта 10 для моделей с коэффициентами регуляризации 1 и 100. Какие выводы можно сделать?

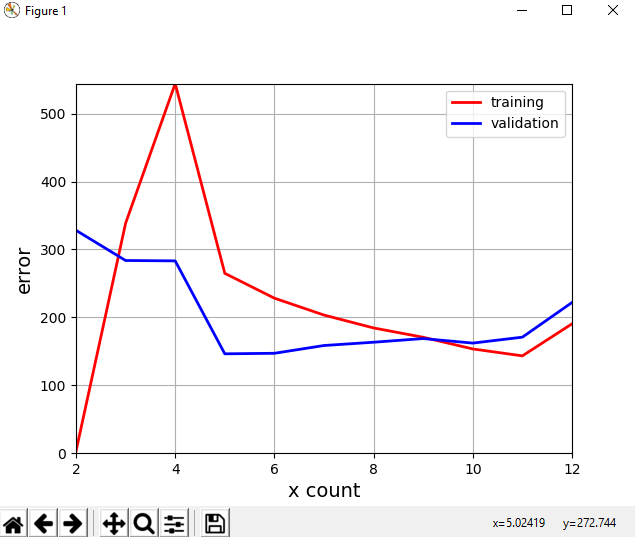
# 11  
theta\_bfgs\_scaled = optimize.fmin\_bfgs(  
 loss,  
 theta.flatten(),  
 gradient,  
 (x\_scaled.values, y\_train, 1)  
)  
  
plt.scatter(x\_train.values, y\_train, color='red')  
plt.plot(x, np.dot(x\_polynom, theta\_bfgs\_scaled), linewidth=2)  
plt.xlabel("water level", fontsize=14)  
plt.ylabel("out", fontsize=14)  
plt.show()  
  
learning\_curves\_chart(x\_scaled.values, y\_train, val.values, y\_val, 1)  
  
theta\_bfgs\_scaled = optimize.fmin\_bfgs(  
 loss,  
 theta.flatten(),  
 gradient,  
 (x\_scaled.values, y\_train, 100)  
)  
plt.scatter(x\_train.values, y\_train, color='red')  
plt.plot(x, np.dot(x\_polynom, theta\_bfgs\_scaled), linewidth=2)  
plt.xlabel("water level", fontsize=14)  
plt.ylabel("out", fontsize=14)  
plt.show()  
learning\_curves\_chart(x\_scaled.values, y\_train, val.values, y\_val, 100)





Как видно из первого графика, регуляризация «сгладила» нашу функцию, она теперь не идет строго по точкам из тренировочного датасета. Как видно из второго графика, регуляризация помогла нам избежать переобучения и ошибка как на тренировочном, так и на валидационном сете минимальна.

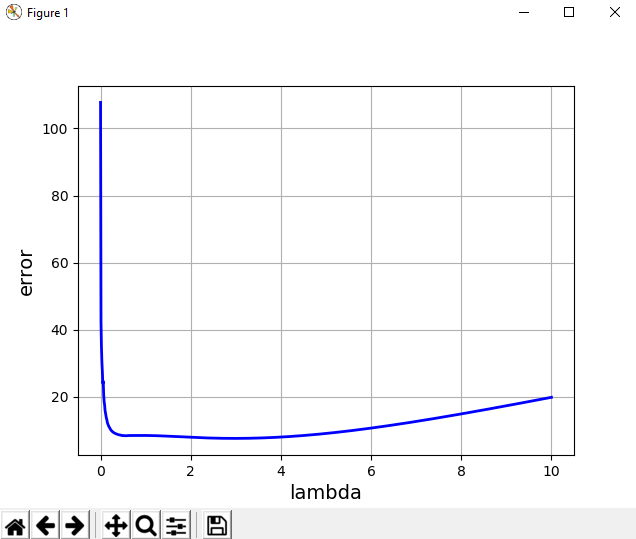




Как видно из графиков, штраф при таком коэффициенте регуляризации оказывается слишком большим, модель не может точно апроксимировать исходные данные

12) С помощью валидационной выборки подберите коэффиент регуляризации, который позволяет достичь наименьшей ошибки. Процесс подбора отразите с помощью графика (графиков).

# 12  
lambda\_values = np.linspace(0,10, 1000)  
val\_err = []  
for lamb in lambda\_values:  
 theta\_bfgs\_scaled = optimize.fmin\_bfgs(  
 loss,  
 theta.flatten(),  
 gradient,  
 (x\_scaled.values, y\_train, lamb)  
 )  
 val\_err.append(loss(theta\_bfgs\_scaled, val, y\_val))  
plt.plot(lambda\_values, val\_err, c="b", linewidth=2)  
plt.grid()  
plt.xlabel("lambda", fontsize=14)  
plt.ylabel("error", fontsize=14)  
plt.show()  
print(lambda\_values[np.argmin(val\_err)])



13) Вычислите ошибку (потерю) на контрольной выборке.

#13  
theta = np.zeros(x\_scaled.shape[1])  
x\_test = normalize(add\_params(x\_test, 8))  
x\_test.insert(0, "Ones", 1)  
theta\_bfgs\_scaled = optimize.fmin\_bfgs(  
 loss,  
 theta.flatten(),  
 gradient,  
 (x\_scaled.values, y\_train, 2.97)  
)  
print(loss(theta\_bfgs\_scaled, x\_test, y\_test))

**Выводы**

В рамках этой лабораторной работы был изучен метод борьбы с переобучением моделей под названием регуляризация. Этот метод довольно прост: в функцию потерь добавляется еще один параметр, равный L2 норме вектора с настраиваемым коэффициентом. Суть этого коэффициента в том, что он штрафует модель за слишком большие вектора, при этом помогая модели справляться с выбросами в тренировочных данных и, тем самым, избегать переобучения.