Министерство образования Республики Беларусь

Учреждение образования

“Белорусский государственный университет информатики и радиоэлектроники”

Факультет компьютерных систем и сетей

Специальность “Обработка больших объемов информации”

Лабораторная работа №3

**Переобучение и регуляризация**

Выполнил:

магистрант гр. 858641 Кальман В.А.

Проверил:

Стержанов М. В.

Минск 2019

**Данные.**

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.**

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

**import** **matplotlib.pyplot** **as** **plt**

**from** **scipy** **import** io

**from** **sklearn.preprocessing** **import** PolynomialFeatures

%matplotlib inline

ex3data1 = io.loadmat('Data/Lab 3/ex3data1.mat')

Xtrain, ytrain = ex3data1['X'], ex3data1['y'].flatten()

Xval, yval = ex3data1['Xval'], ex3data1['yval'].flatten()

Xtest, ytest = ex3data1['Xtest'], ex3data1['ytest'].flatten()

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

plt.figure(figsize=(16, 10))

plt.scatter(Xtrain, ytrain, cmap='viridis')

plt.xlabel('X', size=16)

plt.ylabel('y', size=16)

plt.show()

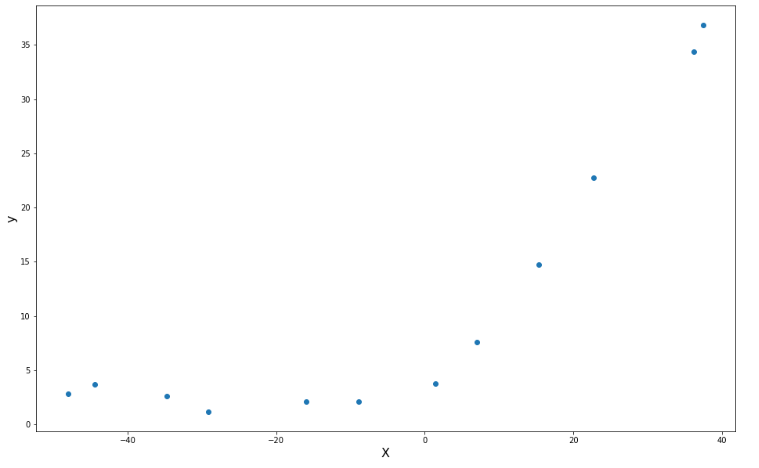


Рисунок 1 – график зависимости объёма воды, вытекающего из дамбы от изменения уровня воды

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

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

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. 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?**

X\_linspace = np.expand\_dims(np.linspace(-40, 40, 100), axis=1)

y\_linspace = np.array([trained\_model(x\_i) **for** x\_i **in** X\_linspace])

plt.figure(figsize=(16, 10))

plt.plot(X\_linspace.flatten(), y\_linspace, color='r')

plt.scatter(Xtrain, ytrain, cmap='viridis')

plt.xlabel('X', size=16)

plt.ylabel('y', size=16)

plt.show()

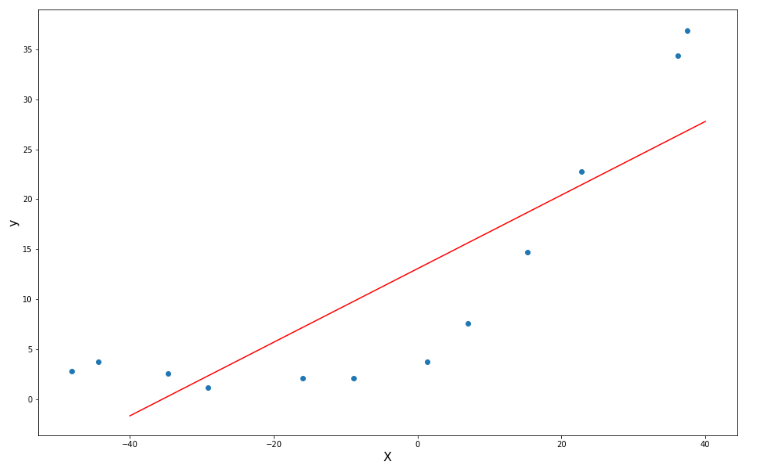


Рисунок 2 – график полученной функции

Регуляризация не работает, так как из-за лямбда равного нулю занулится слагаемое, которое отвечает за регуляризацию и в лоссе и в градиенте, соответственно.

**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?**

model = LinearRegression(dim=1)

loss = MSELoss(model, reg\_lambda=0.)

gradient\_descent = get\_gradient\_descent(loss, alpha=0.001, eps=1e-6, validation=(Xval, yval))

result = gradient\_descent(Xtrain, ytrain)

train\_curve = result['loss\_steps']['train']

val\_curve = result['loss\_steps']['val']

plt.figure(figsize=(16, 10))

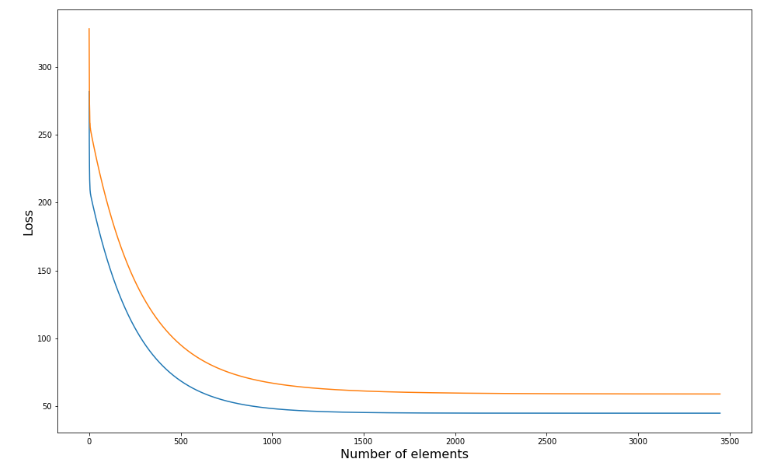
plt.plot(train\_curve)

plt.plot(val\_curve)

plt.ylabel('Loss', size=16)

plt.xlabel('Number of elements', size=16)

plt.show()

 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 predict – усложнить функцию, добавив полиномы высшей степени.

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

**def** add\_polynomial\_features(X, p):

polynomial\_transform = PolynomialFeatures(degree=p)

**return** polynomial\_transform.fit\_transform(X)

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

X\_train\_norm = (Xtrain - Xtrain.mean(axis=0)) / Xtrain.std(axis=0)

X\_val\_norm = (Xval - Xtrain.mean(axis=0)) / Xtrain.std(axis=0)

X\_test\_norm = (Xtest - Xtrain.mean(axis=0)) / Xtrain.std(axis=0)

**9. Обучите модель с коэффициентом регуляризации 0 и p = 8.**

X\_train\_new = add\_polynomial\_features(X\_train\_norm, p=8)

X\_val\_new = add\_polynomial\_features(X\_val\_norm, p=8)

**def** train\_model\_with\_regularization(train, val, reg\_lambda):

X\_train, y\_train = train

model = LinearRegression(dim=X\_train.shape[1])

loss = MSELoss(model, reg\_lambda)

gradient\_descent = get\_gradient\_descent(loss, alpha=0.001, eps=1e-3, validation=val)

**return** gradient\_descent(X\_train, y\_train)

result\_0 = train\_model\_with\_regularization((X\_train\_new, ytrain), (X\_val\_new, yval), reg\_lambda=0.)

result\_0['model']

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

trained\_model = result\_0['model']

X\_linspace = np.expand\_dims(np.linspace(np.min(X\_train\_norm), np.max(X\_train\_norm), 50), axis=1)

X\_linspace\_new = add\_polynomial\_features(X\_linspace, p=8)

y\_linspace = np.array([trained\_model(x\_i) **for** x\_i **in** X\_linspace\_new])

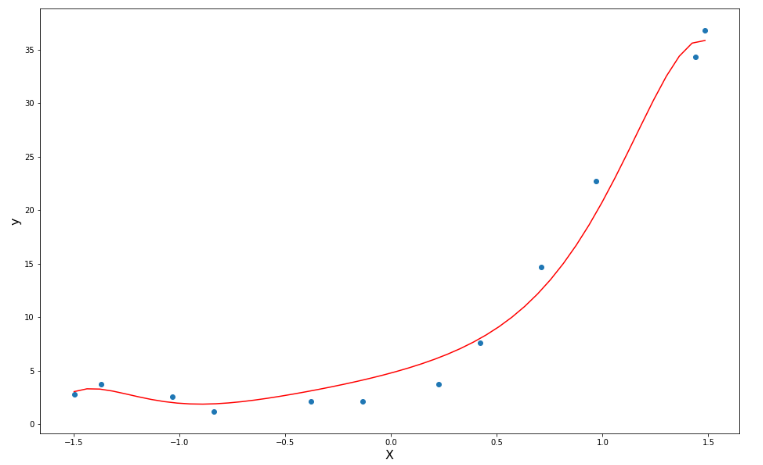
plt.figure(figsize=(16, 10))

plt.plot(X\_linspace.flatten(), y\_linspace, color='r')

plt.scatter(X\_train\_norm, ytrain, cmap='viridis')

plt.xlabel('X', size=16)

plt.ylabel('y', size=16)

plt.show()

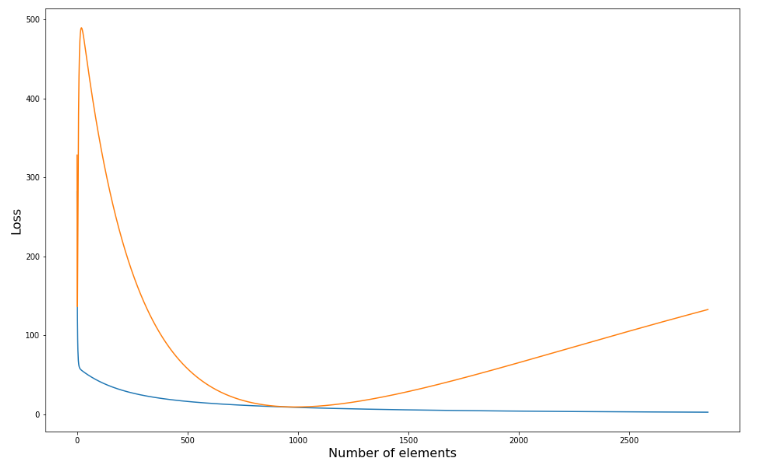
Рисунок 4 – график полученной функции совмещенный с исходными данными (lambda=0)

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

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

trained\_model = result\_1['model']

X\_linspace = np.expand\_dims(np.linspace(np.min(X\_train\_norm), np.max(X\_train\_norm), 50), axis=1)

X\_linspace\_new = add\_polynomial\_features(X\_linspace, p=8)

y\_linspace = np.array([trained\_model(x\_i) **for** x\_i **in** X\_linspace\_new])

plt.figure(figsize=(16, 10))

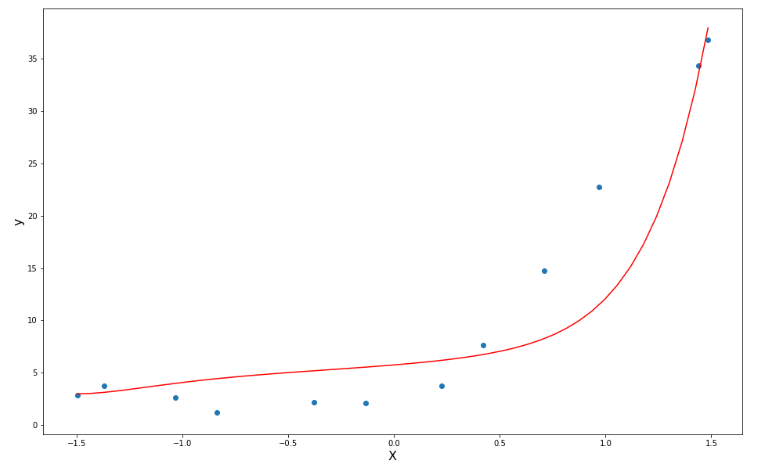
plt.plot(X\_linspace.flatten(), y\_linspace, color='r')

plt.scatter(X\_train\_norm, ytrain, cmap='viridis')

plt.xlabel('X', size=16)

plt.ylabel('y', size=16)

plt.show()

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

train\_curve = result\_1['loss\_steps']['train']

val\_curve = result\_1['loss\_steps']['val']

plt.figure(figsize=(16, 10))

plt.plot(train\_curve)

plt.plot(val\_curve)

plt.ylabel('Loss', size=16)

plt.xlabel('Number of elements', size=16)

plt.show()

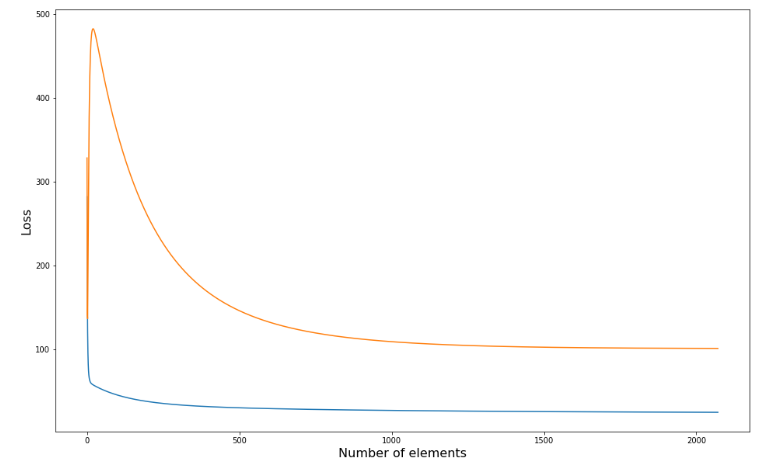


Рисунок 7 – график зависимости ошибки на валидационном и тренировочном сете в зависимости от количества примеров (lambda=1)

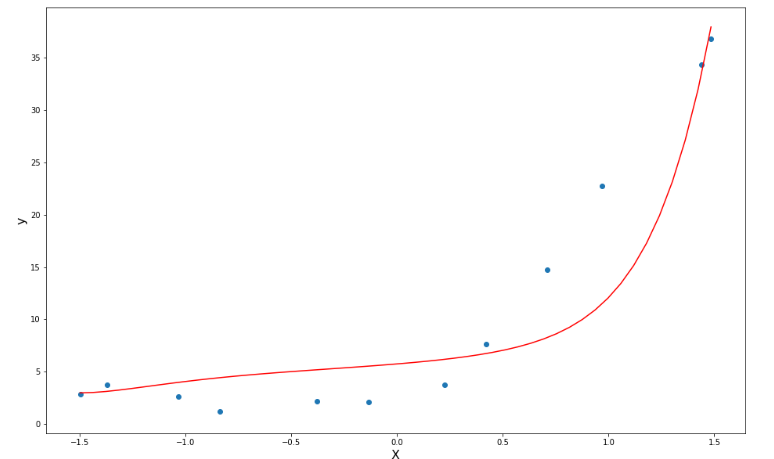


Рисунок 8 – график полученной функции совмещенный с исходными данными (lambda=100)

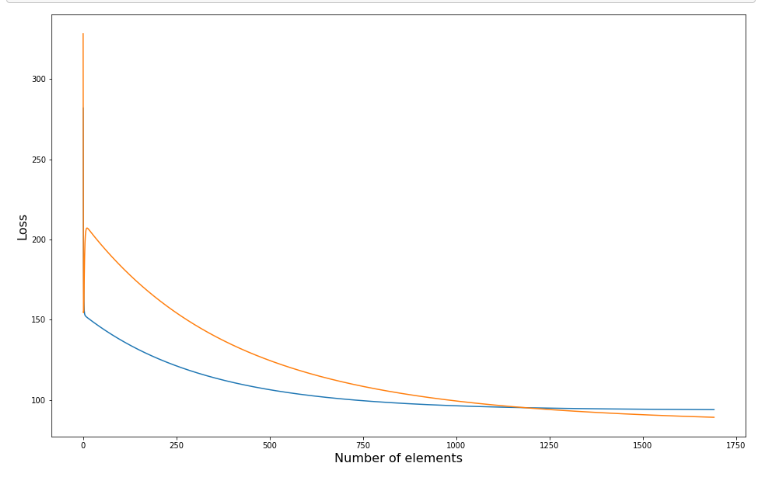


Рисунок 9 – график зависимости ошибки на валидационном и тренировочном сете в зависимости от количества примеров (lambda=100)

Коэффициент регуляризации = 1 – Just Right.

Коэффициент регуляризации = 100 – High Bias.

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

reg\_lambda\_range = np.arange(0., 5., 0.1)

val\_loss\_values = []

**for** reg\_lambda **in** reg\_lambda\_range:

result = train\_model\_with\_regularization((X\_train\_new, ytrain), val=**None**, reg\_lambda=reg\_lambda)

trained\_model = result['model']

val\_loss = MSELoss(trained\_model, reg\_lambda=0)

val\_loss\_values.append(val\_loss(X\_val\_new, yval))

fig = plt.figure(figsize=(16, 10))

plt.scatter(reg\_lambda\_range, val\_loss\_values, cmap='viridis')

plt.xticks(reg\_lambda\_range)

plt.xlabel('Regularization lambda')

plt.ylabel('Validation loss')

plt.show()

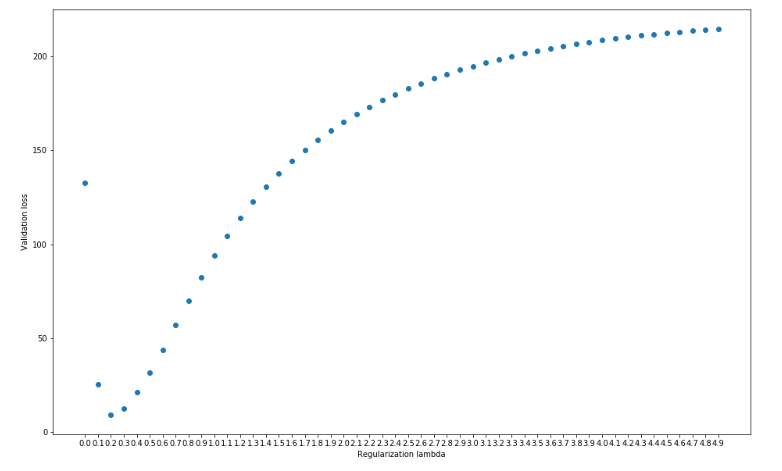


Рисунок 10 – график зависимости ошибки на валидационном сете от параметра lambda

**13. Вычислите ошибку (потерю) на контрольной выборке.**

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))")

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

**Вывод**

Переобучение — негативное явление, возникающее, когда алгоритм обучения вырабатывает предсказания, которые слишком близко или точно соответствуют конкретному набору данных и поэтому не подходят для применения алгоритма к дополнительным данным или будущим наблюдениям Возможные решения при переобучении: Увеличение количества данных в наборе; Уменьшение количества параметров модели; Добавление регуляризации / увеличение коэффициента регуляризации.