|  |  |  |  |
| --- | --- | --- | --- |
| Министерство образования Республики Беларусь  Учреждение образования  Белорусский Государственный Университет Информатики и Радиоэлектроники | | | |
| Факультет | Компьютерных сетей и систем | | |
| Кафедра | Информатики  Дисциплина: Конструирование те технологии электронных вычислительных средств | | |
|  |  | | |
| **лабораторная работа №3**  по курсу Машинное обучение  на тему  **Переобучение и регуляризация** | | | |
| Магистрант:  гр. 858341  Кудин Н.И. | |  | Проверил:  Стержанов М.В. |
| Минск, 2019 | | | |

**Постановка задачи**

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.

**Ход работы**

1. Загрузим данные ex3data1.mat из файла в память с помощью scipy.io

Xval, yval – валидационные выборки

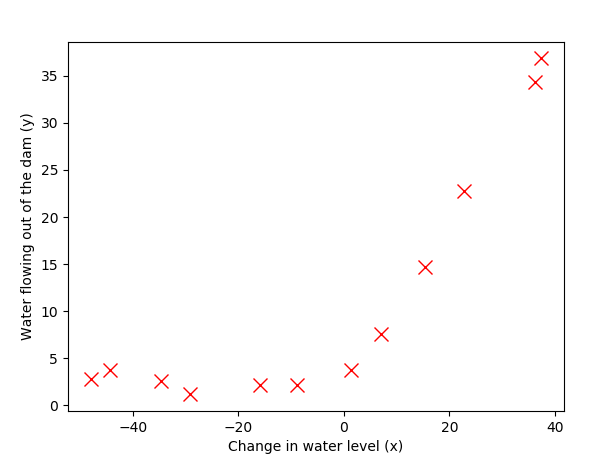
Xtest, ytest – контрольные выборки

*mat = scipy.io.loadmat('ex3data1.mat')*

*X = mat["X"]  
y = mat["y"]**Xval = mat["Xval"]  
yval = mat["yval"]  
Xtest = mat["Xtest"]  
ytest = mat["ytest"]  
  
m = X.shape[0]*

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

*plt.plot(X, y, 'rx', markersize=10, linewidth=1.5)  
plt.xlabel('Change in water level (x)')  
plt.ylabel('Water flowing out of the dam (y)')  
plt.show()*



1. Реализуем функцию стоимости потерь для линейной регрессии с L2-регуляризацией и затем выведем результаты. (linearRegCostFunction.py)

*theta = np.array([[1] , [1]])  
J, grad = lrcf.linearRegCostFunction(X\_padded, y, theta, 1, True)*

*print('Gradient at theta = [1 ; 1]: [{:f}; {:f}] \n'.format(grad[0], grad[1]))*

Функция:

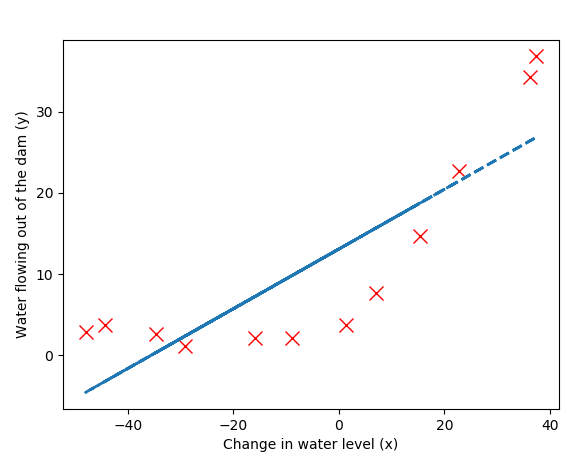
*def linearRegCostFunction(X, y, theta, lambda\_val, return\_grad=False):  
  
 # Initialize some useful values  
 m = len(y) # number of training examples  
  
 # force to be 2D vector  
 theta = np.reshape(theta, (-1,y.shape[1]))  
  
 J = 0  
 grad = np.zeros(theta.shape)  
  
 # cost function  
 J = ( 1./(2\*m)) \* np.power( (np.dot(X, theta) - y) , 2).sum() + ( float(lambda\_val) / (2\*m)) \* np.power(theta[1:theta.shape[0]],2).sum()  
  
 # regularized gradient  
 grad = (1./m) \* np.dot( X.T, np.dot(X,theta) - y) + ( float(lambda\_val) / m )\*theta  
  
 # unregularize first gradient  
 grad\_no\_regularization = (1./m) \* np.dot( X.T, np.dot(X,theta) - y)  
 grad[0] = grad\_no\_regularization[0]*  
  
 *if return\_grad == True:  
 return J, grad.flatten()  
 elif return\_grad == False:  
 return J*

1. В пункте 3 была реализована функция градиентного спуска для линейной регрессии с L2-регуляризацией. Используем ее передав в нее return\_frad = true и выведем результаты.

*theta = np.array([[1] , [1]])  
J, grad = lrcf.linearRegCostFunction(X\_padded, y, theta, 1, True)  
  
print('Gradient at theta = [1 ; 1]: [{:f}; {:f}] \n'.format(grad[0], grad[1]))*

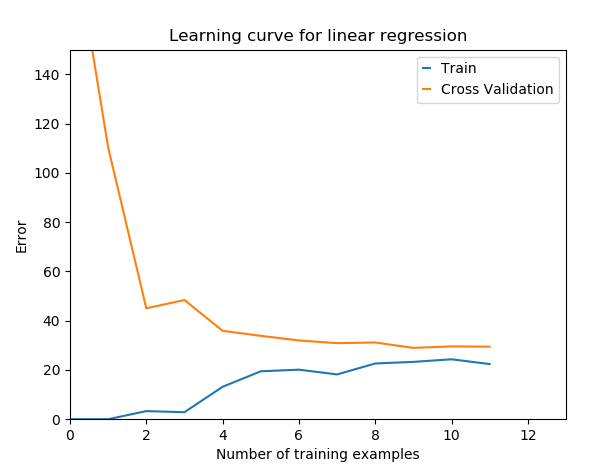
1. Построим модель линейной регрессии с коэффициентом регуляризации 0 и построим график полученной функции совместно с графиком из пункта 2.

*lambda\_val = 0  
theta = tlr.trainLinearReg(X\_padded, y, lambda\_val)  
  
# resets plot   
plt.close()  
  
# Plot fit over the data  
plt.plot(X, y, 'rx', markersize=10, linewidth=1.5)  
plt.xlabel('Change in water level (x)')  
plt.ylabel('Water flowing out of the dam (y)')  
plt.plot(X, np.dot(np.column\_stack((np.ones((m,1)), X)), theta), '--', linewidth=2)  
plt.show()*



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).

*p1, p2 = plt.plot(range(m), error\_train, range(m), error\_val)  
plt.title('Learning curve for linear regression')  
plt.legend((p1, p2), ('Train', 'Cross Validation'), numpoints=1, handlelength=0.5)  
plt.xlabel('Number of training examples')  
plt.ylabel('Error')  
plt.show()  
plt.axis([0, 13, 0, 150])*



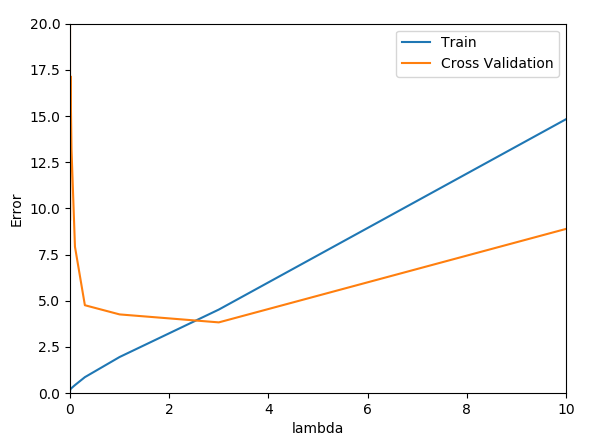
Как видно из графика, количество ошибок резко падает при увеличении числа training examples.

1. Реализуем функцию добавления p - 1 новых признаков в обучающую выборку (X2, X3, X4, …, Xp). Поскольку в данной задаче будет использован полином высокой степени, то необходимо перед обучением произвести нормализацию признаков. Обучим модель с коэффициентом регуляризации 0 и p = 8.

*def polyFeatures(X, p):  
  
 X\_poly = X  
  
 if p >= 2:  
  
 # for each number between column 2 (index 1) and last column  
 for k in range(1,p):  
  
 # add k-th column of polynomial features where k-th column is X.^k  
 X\_poly = np.column\_stack((X\_poly, np.power(X,k+1)))  
   
 return X\_poly*

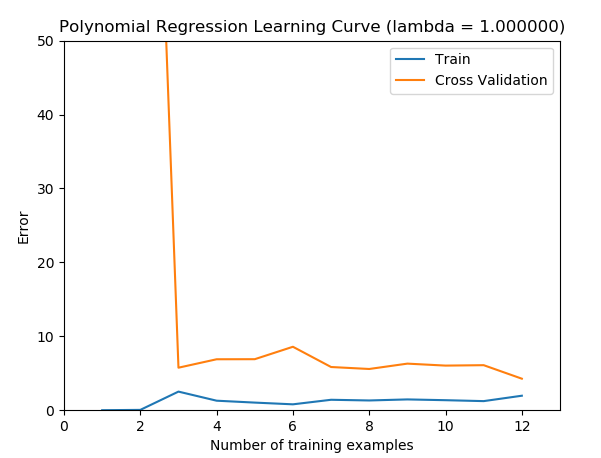
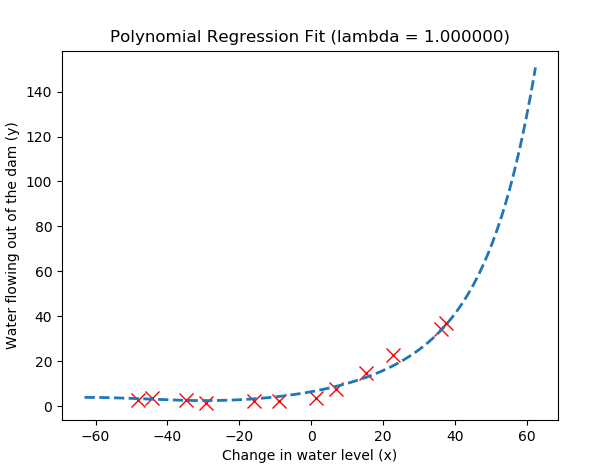
*p = 8;  
  
# Map X onto Polynomial Features and Normalize  
X\_poly = pf.polyFeatures(X, p)  
X\_poly, mu, sigma = fn.featureNormalize(X\_poly) # Normalize  
X\_poly = np.column\_stack((np.ones((m,1)), X\_poly)) # Add Ones  
  
# # Map X\_poly\_test and normalize (using mu and sigma)  
X\_poly\_test = pf.polyFeatures(Xtest, p)  
X\_poly\_test = X\_poly\_test - mu*  
*X\_poly\_test = X\_poly\_test / sigma  
X\_poly\_test = np.column\_stack((np.ones((X\_poly\_test.shape[0],1)), X\_poly\_test)) # Add Ones  
  
# # Map X\_poly\_val and normalize (using mu and sigma)  
X\_poly\_val = pf.polyFeatures(Xval, p)  
X\_poly\_val = X\_poly\_val - mu  
X\_poly\_val = X\_poly\_val / sigma  
X\_poly\_val = np.column\_stack((np.ones((X\_poly\_val.shape[0],1)), X\_poly\_val)) # Add Ones  
  
print('Normalized Training Example 1:')  
print(' {:s} '.format(str(X\_poly[1, :])))*

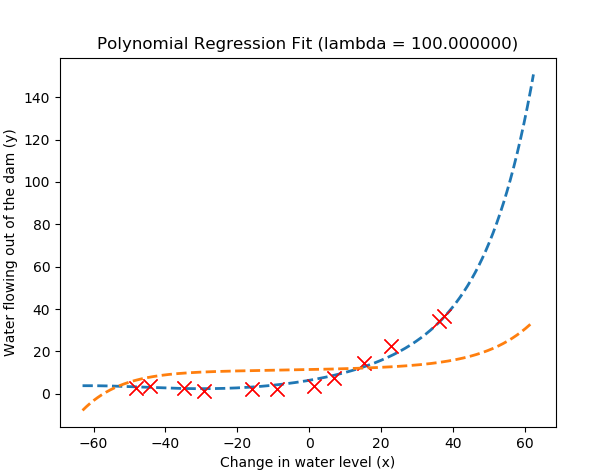
1. Построим график модели, совмещенный с обучающей выборкой, а также график процесса обучения.

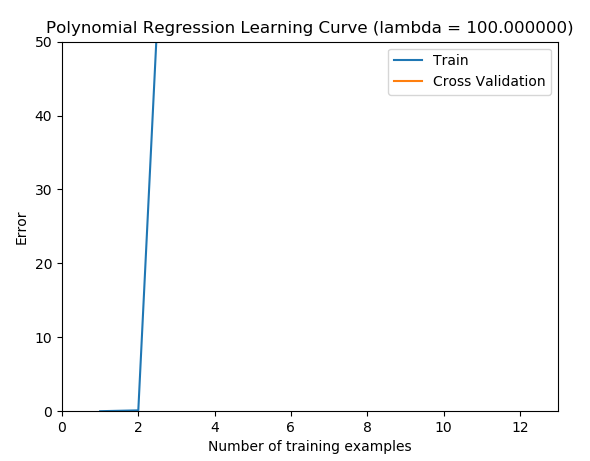


Из графика следует, что при увеличении lambda увеличивается число ошибок.

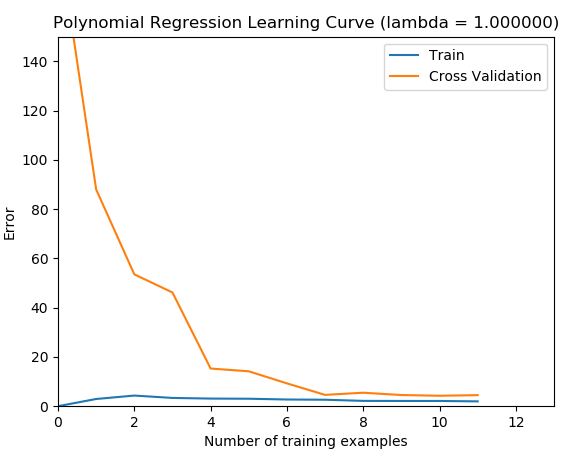
1. Построим график модели, совмещенный с обучающей выборкой, а также график процесса обучения. При построении используем коэффициенты регуляризации 1 и 100.







При lambda = 1 мы наблюдаем гораздо меньшее число ошибок, чем при lambda = 100.

1. С помощью валидационной выборки подберем коэффициент регуляризации, который позволяет достичь наименьшей ошибки. 

Lambda = 1.

1. Вычислим ошибку (потерю) на контрольной выборке.

*theta = tlr.trainLinearReg(X\_poly, y, lambda\_val)  
  
error\_test = lrcf.linearRegCostFunction(X\_poly\_test, ytest, theta, 0)  
print('Test set error: {:f}\n'.format(error\_test))*

error\_test = 2.779947128188635

**Заключение**

Были изучены возможные реализации функции стоимости потерь для линейно регрессии и градиентного спуска для линейной регрессии с L2-регуляризацией. Изучена работа с полиномом высокой степени, а также нормализации признаков для корректной работы алгоритма.

Кроме того, были приобретены навыки работы с такими библиотеками как scipy, matplotlib, numpy.