Regression Week 5: LASSO Assignment 2

In this notebook, you will implement your very own LASSO solver via coordinate descent. You will:

- Write a function to normalize features
- Implement coordinate descent for LASSO
- Explore effects of L1 penalty

If you are doing the assignment with IPython Notebook

An IPython Notebook has been provided below to you for this quiz. This notebook contains the instructions, quiz questions and partially-completed code for you to use as well as some cells to test your code.

What you need to download

If you are using GraphLab Create

- Download the King County House Sales data In SFrame format: kc_house_data.gl.zip (https://eventing.coursera.org/api/redirectStrict/aeFDlHpq3-eiYQz1xXS7waIoiobUqvlOIh9g1PFwWZYSxdodHPH_wdczXjRxfr_syfPRg_feCrvqdh1ACXU6QA.TALd3MrAc1jF8Z3-LJ8ocw.L0-PnFgFwQXhxXmtrFxcTua7LHnXDhhxALLdnTUs5tpe0F9_w6MAJk83I3D0m-X4lwlwJS4repRDAJoo-BIGiUjIsXoN4VInGdBtH2kw02w9GtdEu5jzHgp6wYl580SjfJj04t48PU1dP2_caLejL6Rmg0xJz5qa1XMTizLkmJvUGiQ88jFJUVmw3o2Lfu0GGCR4vTnap-8pvhqNEnVn53CA0x34kOjJq5wGUHWYeihnfbzG7unVO6uGXoxTEqaexIQvueanpBHFcwqfLAS7y3V41sOcwL5S1WWGgtpQbTu5j6L-u8zLgPOu0D3WS-vZ9uYuSf6k286ihPKAA88U4X682Q2ulx6RmchNiYQyFcIQmz88aekTn2SjFEvadSfcK052B8f-gzHMOTsO_yJX0Vgn8h6eEeKaEESyjt3aot9e4yf485miDzvVJoWZ3Cfs)
- Download the companion IPython Notebook: week-5-lasso-assignment-2-blank.ipynb (https://eventing.coursera.org/api/redirectStrict/ZsrxTLRyw2sXT3-TOtltx8MhOyFOOa2kIb71CK653yGkSFH29uMVeN6HAJPJc9-ECmWFxsaqZF8KnzpyyxBUAg.iWf7al4hznorTLGBjdexJQ.nJATGHYd6PS9Ot24I4zG4fflrt-reePSFb1tjowRkhCxuYXs8iXYtyBUYmGtdmxHobMMySOsN4dOw-zt1AmtbX7aMixsPKC04FiigdOiowN0cSoNczQTghuU2f7sloZNO8f6STE3t5PqvCtYe-6HtaR85WCiKBHMovcKGA1-5fCVLqaIbGQRsi8Gvt0dOeh_-s333aSD_Fl2IQAkRh5StMVoxcAzmntjz38vm7ZpaR3fC-

Yvq1sGnRSds7u1xDbeJ07F2d0eY1_vTeAEsUYYHNcLZGPi3NKrjetteaPIz3PHuS8rCefw6uuFJv-bVFjZvMOx7pVBx6lW7VjHsb9DWGqkptZ1fMB6AS4Kx5Zb1B8cAuaQ9oJQmj9X3O9VxymuJUzwlYs6KycS-45Uv_mSur4UxGnl7NffityPQ4GwrdC2XRmQ9c-f2LNTxhWDvI-bkWj984bs9h9X2bJzdZ_XruwmU8xlNBMYQV1DOqUne7A)

- Save both of these files in the same directory (where you are calling IPython notebook from) and unzip the data file.

If you are not using GraphLab Create

- Download the King County House Sales data csv file: kc_house_data.csv (https://eventing.coursera.org/api/redirectStrict/jBGLh37KBuF4Y-SOyOHTT7Ctnro9C8_4GdQGbJVhhvDM-2-rRH6k4bjBBYcPDuJ0_hnBcCOXdjLl8eDGXtcqEQ.QneSGBimdKxln2Q3QwY-pg.UQiw6mPlq511Huh5yjxYwGC6r9vw9vfucveNbOP1ANrnIxMGDsuMkcJsQN9t509CTQqTc3XObM5XoTdr_cVERZ_8Xk6aviYSyz3g0M7ldFAXYfD0yr5-CWYE0LW9CRDML0Wh_jHQwRyFOAXJF2EEUc8rt4yxsbX-pli6cdLo5nW4kKL37HHJLvkc8y2hVTN8SDOVVbfti9qRfudEOA_V_oh7yOgarPg2mpRNMKrYxh2ryRGkqBWM4eoDuobsAI14Pv2LSQ6MO-qKD0NTNr3yAYhQQuu_UJodAZFBB3puzB4nj3Lm4yMYsOLzsl9VynAQS5X5UxHWfDlWVp-76KwFSzskaKwUqLjrlzYwJrAzFICygwUBnk_OegRJovGRYKQUYXvSw_UCJOtxVhQYpR83I9JUiMkkwfulEt20p9IFzixwB2HJKmfukX91kOVMavGB)
- Download the King County House Sales training data csv file: kc_house_train_data.csv (https://eventing.coursera.org/api/redirectStrict/xlYlSwp8E1KnL3FJqvGIXJGpaDNmb2Zg2EKhUtZUcriaT_dRthWDCwr6aHtGCXV75EmsgF1OQLIjINbuOKFQ0A.tLI_AVklJfi9GHsPtXSuDw.Df2zayHOE-aofQG8JaOXieYDdvoy5ihTCx2TcNGr6ckX7kkvrAc_E_ehTZpac8iBORsdG0M-OLmOMiEvxDz-7ae8dkl0u9LO4zf9fSHU8Vi4MEj2iMmNpJOAV0ggX8s6yUgPeu9uzIPun8it7X3DoBwh8kH7_HtGJrlL0KZk5Q1FPK9nKbw235xuKLULBEGrODXqpN-DmS_rwh7idGCQNfhJqSA_SeQ2oFohGPzzvST--Xmz8kzI115I-Ncl7M5jte-dD9S5_QFhGEXTO53pco4w3TY-aK-neMi_ZV_OI_fF0aOSkco9NhpZ4Dlm0E87oIvpWAQHbXcE_NVB5WxtovPLWi0nngimhZWHz0H3JUkrv_5G9LspgEXb8zwDZdNb7Qy5r5Dq4TCQCPAG-UvRIu1hC-La2C_5UGYIAfCCR66N5Y2NZPYlNTAV7R7_r1KmmPDpSzXvLqGBTzASO5vzRg)
- Download the King County House Sales testing data csv file: kc_house_test_data.csv (https://eventing.coursera.org/api/redirectStrict/NV1KTqz1b0bAPeedUavx3a35kouUc_996mld4bLTEGC_FIdVgVtv63_SE9IkxLW_iTOF_UK0jK8dJNKbGi9j6Q.qH05wR27TT72LrQ2SvxG3Q.DK-PPQoda3IHoi5V-2C8kCQNtsaVuo5E58K2d3BGhFOqSyj2eo6Bj_zQKAMhjSVkqIsqrHFIpv-m9FtpsXYrNW4Nv6o4oNH41WKcm1SU_IhBKmLVOTTnIsgVdfb7CLlFab54sA6VtXet76Cp_fLQ3NRcEhvNpYJs2HOrjD0RHhXxHW6RgxsNNNOTnxS1OVK-gHQ56iLt0hE4tURYqTlNXPCymOHdcI6P3Q0E0gwkWwzGOYor3XUN_avOiVrOiOB1SsqKZqNGi_yosFn02y1EIwvmmSJcWUWOETPYW0q9wLynvHZVlvscwNJibPaP9PKqCUa56-F7yWhYOReS0sa22-IAAqz8r93w2SXRgNBMB17pZcPfc2kxiidX_h60M2m-

tHrkjvc2OKW50QjU56thvWhDmURYWlEidA7cH65H8Rv1cqiQo4x86NOB2mCor07jCANjtiy COb6HyAWIPJPg-A)

- **IMPORTANT: use the following types for columns when importing the csv files. Otherwise, they may not be imported correctly: [str, str, float, float, float, float, int, str, int, int, int, int, int, int, int, int, str, float, float, float, float]. If your tool of choice requires a dictionary of types for importing csv files (e.g. Pandas), use:**

```
dtype_dict = {'bathrooms':float, 'waterfront':int, 'sqft_above':int, 'sqft_living15':float, 'grade':int, 'yr_renovated':int, 'price':float, 'bedrooms':float, 'zipcode':str, 'long':float, 'sqft_lot15':float, 'sqft_living':float, 'floors':str, 'condition':int, 'lat':float, 'date':str, 'sqft_basement':int, 'yr_built':int, 'id':str, 'sqft_lot':int, 'view':int}
```

Useful resources

You may need to install the software tools or use the free Amazon EC2 machine. Instructions for both options are provided in the reading for Module 1 (Simple Regression).

If you are following the IPython Notebook and/or are new to numpy then you might find the following tutorial helpful: numpy-tutorial.ipynb (https://eventing.coursera.org/api/redirectStrict/qe9MX_3whTMGPOgE6x0yrGvbQKHBiOCAaB82RJKtYYOnGMIdzlXwydFypD-8v7lER9e5rle1Ir6YiLd8EXdUeg.JGRgFhlhvpnYZf4y9UC97A.HhIQLyj3d4fACiXgYB-4u6zYXY1b5HMKszYqaDT2aT19cYvwBsR-Eq4Xu-JVhiUz1M2pHcVF8vZHxEII7zt0DiHkdwh9p9kSaV76ZBLgE2rkZb6nUl1zF0KcL0ZEL5iSZ7-4jGZR58In8DFE-PL0116vfhri94D9lnMrk7vEq3WHTU6qnq6qS1RPrptB_aO_6X71Pm9ZUncqzh83tQTl8N-SuKCErDdpVaSfLTVKXXsy4p1k0yydY-Yp-dw3yj-j_CyLW4ZBqLGRfzSZqLAR8lTEKIkE7ZKHPVZsFwiWXo8QxPqyjqjqPxXeB0QMzBcK8KLzrXIERUO28C6RVbagN4frHchQgvwWQub1QWzjIOfd9y2OLbgCQ7nQiMLyTUGu-4DtTwgtDmlGM3jdTif-Nrnyzy4Ey-9t6LsESPJ2CfSwmQ_vgon4wv2nqgDyxpOV)

If you are using GraphLab Create and the companion IPython Notebook

Open the companion IPython notebook and follow the instructions in the notebook.

If instead you are using other tools to do your homework

You are welcome to write your own code and use any other libraries, like Pandas or R, to help you in the process. If you would like to take this path, follow the instructions below.

1. If you're using SFrame, import GraphLab Create and load in the house data as follows:

```
sales = graphlab.SFrame('kc_house_data.gl/')
```

Otherwise, load all the three csv files.

2. If you're using Python: to do the matrix operations required to perform a coordinate descent we will be using the popular python library 'numpy' which is a computational library specialized for operations on arrays. For students unfamiliar with numpy we have created a numpy tutorial (see useful resources). It is common to import numpy under the name 'np' for short, to do this execute:

```
import numpy as np
```

3. Next, from Module 2 (Multiple Regression), copy and paste the 'get_numpy_data' function (or equivalent) that takes a data set, a list of features (e.g. ['sqft_living', 'bedrooms']), to be used as inputs, and a name of the output (e.g. 'price'). This function returns a 'feature_matrix' (2D array) consisting of first a column of ones followed by columns containing the values of the input features in the data set in the same order as the input list. It also returns an 'output_array' which is an array of the values of the output in the data set (e.g. 'price').

e.g. in Python:

```
def get_numpy_data(data_sframe, features, output):
    ...
    return (feature_matrix, output_array)
```

4. Similarly, copy and paste the 'predict_output' function (or equivalent) from Module 2. This function accepts a 2D array 'feature_matrix' and a 1D array 'weights' and return a 1D array 'predictions'.

e.g. in Python:

```
def predict_output(feature_matrix, weights):
    ...
    return predictions
```

5. In the house dataset, features vary wildly in their relative magnitude: 'sqft_living' is very large overall compared to 'bedrooms', for instance. As a result, weight for 'sqft_living' would be much smaller than weight for 'bedrooms'. This is problematic because "small" weights are dropped first as l1_penalty goes up.

To give equal considerations for all features, we need to normalize features as discussed in the lectures: we divide each feature by its 2-norm so that the transformed feature has norm 1.

6. Write a short function called 'normalize_features(feature_matrix)', which normalizes columns of a given feature matrix. The function should return a pair '(normalized_features, norms)', where the second item contains the norms of original features. As discussed in the lectures, we will use these norms to normalize the test data in the same way as we normalized the training data.

e.g. in Python:

```
def normalize_features(features):
    ...
    return (normalized_features, norms)
```

Hint: If you are using Numpy, a handy shorthand for computing 2-norms of columns is

```
norms = np.linalg.norm(X, axis=0)
```

To normalize columns, perform element-wise division.

```
X_normalized = X / norms
```

Review of Coordinate Descent

7. We seek to obtain a sparse set of weights by minimizing the LASSO cost function

```
SUM[ (prediction - output)^2 ] + lambda*( |w[1]| + ... + |w[k]|).
```

(By convention, we do not include w[0] in the L1 penalty term. We never want to push the intercept to zero.)

The absolute value sign makes the cost function non-differentiable, so simple gradient descent is not viable (you would need to implement a method called subgradient descent). Instead, we will use coordinate descent: at each iteration, we will fix all weights but weight i and find the value of weight i that minimizes the objective. That is, we look for

```
argmin_{w[i]} [ SUM[ (prediction - output)^2 ] + lambda*( |w[1]| + ... + |
w[k]|) ]
```

where all weights other than w[i] are held to be constant. We will optimize one w[i] at a time, circling through the weights multiple times.

- Pick a coordinate i
- Compute w[i] that minimizes the LASSO cost function
- Repeat the two steps for all coordinates, multiple times

8. For this assignment, we use cyclical coordinate descent with normalized features, where we cycle through coordinates 0 to (d-1) in order, and assume the features were normalized as discussed above. The formula for optimizing each coordinate is as follows:

```
       ┌ (ro[i] + lambda/2)    if ro[i] < -lambda/2
w[i] = ├ 0                     if -lambda/2 <= ro[i] <= lambda/2
       └ (ro[i] - lambda/2)    if ro[i] > lambda/2
```

where

```
ro[i] = SUM[ [feature_i]*(output - prediction + w[i]*[feature_i]) ].
```

Note that we do not regularize the weight of the constant feature (intercept) w[0], so, for this weight, the update is simply:

```
w[0] = ro[i]
```

Effect of L1 penalty

9. Consider a simple model with 2 features: ‘sqft_living’ and ‘bedrooms’. The output is ‘price’.

- First, run get_numpy_data() (or equivalent) to obtain a feature matrix with 3 columns (constant column added). Use the entire ‘sales’ dataset for now.
- Normalize columns of the feature matrix. Save the norms of original features as ‘norms’.
- Set initial weights to [1,4,1].
- Make predictions with feature matrix and initial weights.
- Compute values of ro[i], where

```
ro[i] = SUM[ [feature_i]*(output - prediction + w[i]*[feature_i]) ]
```

10. ***Quiz Question: Recall that, whenever ro[i] falls between -l1_penalty/2 and l1_penalty/2, the corresponding weight w[i] is sent to zero. Now suppose we were to take one step of coordinate descent on either feature 1 or feature 2. What range of values of l1_penalty <u>would not</u> set w[1] zero, but <u>would</u> set w[2] to zero, if we were to take a step in that coordinate?***

11. ***Quiz Question: What range of values of l1_penalty would set <u>both</u> w[1] and w[2] to zero, if we were to take a step in that coordinate?***

So we can say that ro[i] quantifies the significance of the i-th feature: the larger ro[i] is, the more likely it is for the i-th feature to be retained.

Single Coordinate Descent Step

12. Using the formula above, implement coordinate descent that minimizes the cost function over a single feature i. Note that the intercept (weight 0) is not regularized. The function should accept feature matrix, output, current weights, l1 penalty, and index of feature to optimize over. The function should return new weight for feature i.

e.g. in Python:

```
def lasso_coordinate_descent_step(i, feature_matrix, output, weights, l1_p
enalty):
    # compute prediction
    prediction = ...
    # compute ro[i] = SUM[ [feature_i]*(output - prediction + weight[i]*[f
eature_i]) ]
    ro_i = ...

    if i == 0: # intercept -- do not regularize
        new_weight_i = ro_i
    elif ro_i < -l1_penalty/2.:
        new_weight_i = ...
    elif ro_i > l1_penalty/2.:
        new_weight_i = ...
    else:
        new_weight_i = 0.

    return new_weight_i
```

If you are using Numpy, test your function with the following snippet:

```
# should print 0.425558846691
import math
print lasso_coordinate_descent_step(1, np.array([[3./math.sqrt(13),1./math
.sqrt(10)],
                   [2./math.sqrt(13),3./math.sqrt(10)]]), np.array([1., 1.
]), np.array([1., 4.]), 0.1)
```

Cyclical coordinate descent

13. Now that we have a function that optimizes the cost function over a single coordinate, let us implement cyclical coordinate descent where we optimize coordinates 0, 1, ..., (d-1) in order and repeat.

When do we know to stop? Each time we scan all the coordinates (features) once, we measure the change in weight for each coordinate. If no coordinate changes by more than a specified threshold, we stop.

For each iteration:

- As you loop over features in order and perform coordinate descent, measure how much each coordinate changes.
- After the loop, if the maximum change across all coordinates is falls below the tolerance, stop. Otherwise, go back to the previous step.

Return weights

The function should accept the following parameters:

- Feature matrix
- Output array
- Initial weights
- L1 penalty
- Tolerance

e.g. in Python:

```
def lasso_cyclical_coordinate_descent(feature_matrix, output, initial_weig
hts, l1_penalty, tolerance):
    ...
    return weights
```

14. Let us now go back to the simple model with 2 features: 'sqft_living' and 'bedrooms'. Using 'get_numpy_data' (or equivalent), extract the feature matrix and the output array from from the house dataframe. Then normalize the feature matrix using 'normalized_features()' function.

Using the following parameters, learn the weights on the sales dataset.

- Initial weights = all zeros
- L1 penalty = 1e7
- Tolerance = 1.0

15. ***Quiz Question: What is the RSS of the learned model on the normalized dataset?***

16. ***Quiz Question: Which features had weight zero at convergence?***

Evaluating LASSO fit with more features

17. Let us split the sales dataset into training and test sets. If you are using GraphLab Create, call 'random_split' with .8 ratio and seed=0. Otherwise, please down the corresponding csv files from the downloads section.

18. Create a normalized feature matrix from the TRAINING data with the following set of features.

- bedrooms, bathrooms, sqft_living, sqft_lot, floors, waterfront, view, condition, grade, sqft_above, sqft_basement, yr_built, yr_renovated

Make sure you store the norms for the normalization, since we'll use them later.

19. First, learn the weights with l1_penalty=1e7, on the training data. Initialize weights to all zeros, and set the tolerance=1. Call resulting weights' weights1e7', you will need them later.

20. ***Quiz Question: What features had non-zero weight in this case?***

21. Next, learn the weights with l1_penalty=1e8, on the training data. Initialize weights to all zeros, and set the tolerance=1. Call resulting weights 'weights1e8', you will need them later.

22. ***Quiz Question: What features had non-zero weight in this case?***

23. Finally, learn the weights with l1_penalty=1e4, on the training data. Initialize weights to all zeros, and set the tolerance=5e5. Call resulting weights 'weights1e4', you will need them later. (This case will take quite a bit longer to converge than the others above.)

24. ***Quiz Question: What features had non-zero weight in this case?***

Rescaling learned weights

25. Recall that we normalized our feature matrix, before learning the weights. To use these weights on a test set, we must normalize the test data in the same way. Alternatively, we can rescale the learned weights to include the normalization, so we never have to worry about normalizing the test data:

In this case, we must scale the resulting weights so that we can make predictions with original features:

- Store the norms of the original features to a vector called 'norms':

```
features, norms = normalize_features(features)
```

- Run Lasso on the normalized features and obtain a 'weights' vector
- Compute the weights for the original features by performing element-wise division, i.e.

```
weights_normalized = weights / norms
```

Now, we can apply weights_normalized to the test data, without normalizing it!

26. Create a normalized version of each of the weights learned above. (‘weights1e4’, ‘weights1e7’, ‘weights1e8’). To check your results, if you call ‘normalized_weights1e7’ the normalized version of ‘weights1e7’, then

```
print normalized_weights1e7[3]
```

should print 161.31745624837794.

Evaluating each of the learned models on the test data

27. Let's now evaluate the three models on the test data. Extract the feature matrix and output array from the TEST set. But this time, do NOT normalize the feature matrix. Instead, use the normalized version of weights to make predictions.

Compute the RSS of each of the three normalized weights on the (unnormalized) feature matrix.

28. Quiz Question: Which model performed best on the test data?