Regression Week 4: Ridge Regression Assignment 2

In this assignment, you will implement ridge regression via gradient descent. You will:

- Convert an SFrame into a Numpy array (if applicable)
- Write a Numpy function to compute the derivative of the regression weights with respect to a single feature
- Write gradient descent function to compute the regression weights given an initial weight vector, step size, tolerance, and L2 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/jpH4cQ-glKEjDUcckIS68p3WB6RcZQkMwxmFmQE5eosp7btWxAeJYQaficHGv0lIHDc5InFPMc6QkgkU8C1Dwg.88DU4wl-ckKWaOFboNSo4w.E6WxQ26a7YgB-ZqgIYfR68aAvPSNt_dHvEmgpxuFriqvgxocGxswHHCFzj6GrNzpsh8nx4Bxxwk_oK-eeKYNTSYCFykqh6LUjyOw4mWC_83-2UxeYhL7mnJ1LCCzNNiOSukXwMeKzc5VrDcPu5k0Gvk6fqeA9eAV9cSsOfZ0NCuAr2CGbrAmNkjuwtuB_vis9hJlwuezy5UCGccnsVhaTaH_nSTcWGTeYt5b95ZXPzUEyMa3ibKQ3FawovFozth0OQeWbAIA2ZQcsQOg9-k9Tcd7aJb_Mu7RFwJKq2x01ZYstJlvKAiKoc-R_7IGIBvrrBc_Oakwv3VjvJT3RW7YGdUrs3ci7G_F4_F67CeTMhu5IFDmx2rEyk-TLFqeGYflOARBBc0grQjnoX0WAuGmtNfrpolQQVDfFlwWb0xRTWbx6QRSTAVvNj5zL6GC0Fdd)
- Download the companion IPython Notebook: week-4-ridge-regression-assignment-2

- Download the companion IPython Notebook: week-4-ridge-regression-assignment-2-blank.ipynb (https://eventing.coursera.org/api/redirectStrict/JHlCz1iw4gzAbgikZozqqLh4cqZU0CX9QZrr0CWSIqE9ms473oJbu8qKUKu6DVUyFLGspZaJN4b4hGstzS9NNQ.WJfkoCVSuz7poo2dIyUSKw.j7wYEQr19b4TFUaOXD5jvMokaw690CsSGL_y3eyofS00jX0IOyR_T8N3dkmChrZ3E1Jbv0YMK3y2gjOJ0aLwsUYjSzCtltc6kDpW6--BZ--XsnsM7Z7WAE6EMYFtuChLhn4guO7oCi8KnE87zZHQDmry1lMxO8gTMD05K80VIMn9kq32Y7Ktq6D_jPbssudl3F5QNnQTP7r1Rtsd_6qYT1q5rscmPAVMgufko-Jy94-84NXe5m5pGZyIf-1v49RVPrh7C-kpLUHoV17ZlpKyHrzDsf9WAbpDthbhCipGcUDWEXJDAHHFFQmHTgrpH0wOK7RYvlzM5oJfAgvgMMtvMXlYEWKSZdToDON2RESXI4cNatkwx6tkW6EbgEdyXvEZEGNsafkVGU-y5cD2XiIKHW_mwQh76RNl22cPeGnrN8VuVaI0V_jImtptNfj4wKV5slvSNKuXioahZOvzCUrcL3JBGAQuDsfspV77OEHp2gxl0A4WAfuqHPbS7OJuE1NC)
- 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/AT56PhhtTANQGa29D07DYhwwk6pYTRvFfCTaNR9lNKxuCQfdSI1yb-GPIeagGdM3Np1QcG49IiACZg3xv7o2pQ.mcGjuvbVvrVaBukSLIxvaw.Ub-Y1IhjIUdis2pBWQO05-VQCeBjmq7ffIohAM4SoLiJ-gYtXU0v__DpuAdkCEsoSo7CWaWW2e4JlUnYl_U6PddLEtFrDBLTYSg3ZUpSffl4mJM37CvvlAGJt2DAI5XivUBTppjPqBihcoGZFXVAzawLtLV-1kYBZdIc1yVs08ukLLxMQBQQgv-Ol9w3nSr0oYiyk0L5khtCihfbJNNc0PZHs_zwR5P2ktWJQyLKZSvX5EgPE94x7sa6u2ebqGi-rf3kNmWBP_zlxOckYDpysIeIRUP0UteOGiz6WxPIq6XMeL7xONAztN8mGiB1t126xmGIXP8XfRD_gLldCtnvbqplOZ_B5cS7EZ5zP1dU_r63gI4EkidiBMZyEiQgfj_6eWqdIuLMu7WH2jZ3y-WdUVOb6SopkqE5BXOopyp_rLpiG0OSHYy7LssWtSviCdmx)
- Download the King County House Sales training data csv file: kc_house_train_data.csv (https://eventing.coursera.org/api/redirectStrict/qer2qGa3wrl0bXj5TWqlsVdgX21vL838sUFlWlOQi_K0leKg3yjh0--zdJku3Ivj2tZ3P6Fnfd2O5EIH9lwVQw.r31oT9PsKm-5J8XAdtPk4A.6oo3KxhnEjxaY23s4_OdRkXFsD-chAQkY30o9z7-jnNh8N--JpUOO7K9Z5oZBitRzyhogw50JoWUO-f9A8u6pFOAYzdf_Ksid7VJynnQTs48kp6CP9Z7sqnwQVG5A0SF3NUsqiQsNL63Wh_V_mQZzFs1YvN35AT_vjfeSlnhr1BBXEmCFAuJu41RWTSFb3AMP-BwMC30lLNu5CkSAEQF831vNJ2uz_Y0hPKGDv9EvxKy-ucxE5uCFUbVOyw7F2PLlIBZdR3yOSFTD1OQi2PUYiZkg7Qta8heJK81hHeWgyvkqBSg5wUhwPqIXgZYIQhbwIhPsL39mly4yh7LoUxRsF011wbwhUFADZ10HXp2iCFmLclvbvKt96uJXgqPXPeoLwOlkfb5gEHh_AsUvpsPOu_jiWUBqiS95J4Kw9czbldPDYmUCRXZ31sLmZSetwhw)
- Download the King County House Sales testing data csv file: kc_house_test_data.csv (https://eventing.coursera.org/api/redirectStrict/U4alCpn3Rv_B1q6ef2mdKGGY6-31YNCiaLulSG2Ky6KWk84Qptm3Ay_8F_cM035-lBbBZNhNwQfF7Uc0B_dkGw.-J5LsaZSQn1p4ddvncPMbA.lwrT3UsSO0HZPfK1hzajq6l1pT8ZBnQNmd_GE2PlBK-

0FkkCggJEDMYCd6PQ1bFLxE38-kPZyuDi4jBi0ZOeNbIMFq_03NBCIeK0NYknF7EVw_-T701fQ83cz10wA2PF0Y01MEwznGN-UH9yPBgLgueeMANNLlumhKKFBzq-F5Qj0x94dgyk-Wy7TbeFKm3aku1RRJ-U8Fi_YhoODa4q_B3xADQvvi86DDUf8zuRQD3FVTUTqdqPTjHMITqANSmI2HFYNsBWdUdD7CIZPvQ5cZlzFldFcHJyeuht1xs8IdeW1Pzf0aXaTPpZf6oBvDWMn-X2f2hScP3nJY8AXphaY9VytV1hdgIIawJxb4Ri3AgvSdYeKwJJ-4DCUio90e6pYQp9bYqevGaeP29_2utPaFXu8Gc8rVs16QIzEnphvttePWV5AlWvxswrEFd2WIfh)

- **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.

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/ouOXAjJKBCb8zT6EW3VzoOT3g2VNSvKdcoD4PwWfa9lR46TM0eje6Y0lPdfsrrNAbFBXAHQMxf-c9QkzdD0GWQ.PoBoZOUFED-PW7JUOnz2Lg.iLFmzYLh0KJo_EW95yJ8CWJA1t55ap3iY_5cDL0WXC2OLrWx7z5UYk0iePD7akHZfS7EHn9UZtt1Up2ZLBxzEL0DVI4fyqPyI4eMLM_ub2FiC8jZnq6uhpWN236DaVLrRfDDZtnWeHYV-0dVnGERgfaHJD2CbAy23uWrbcXoz0pp9BsimZuvpixl8RARg6dgYLNpXQouCbpbkBCFUJfka2MUrQPDeKlHRG0Wev0w9J2QFbGshbY9E_IEqEKUvHDgkbS8h6HAjuKFtEyZcyFEgJA9ka7gl-reiK36ez8hO-_e2-SpTRnMG4c2G4Y6MuzBT-HIphMDcJl48eHzTXSWxRKkYryRfxAemksbhcE79Dvcf6soCeec4AXhUXkUztzvy2-f0OAy9WYJVcWCCUDjNC9ux-ljD84s_B81SHTcI9oxE5Z0DoJ_pQ0sj8MTBWJ1)

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

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 gradient 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, copy and paste the 'get_numpy_data' function (or equivalent) that takes a dataframe, 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 alsos return 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. We are now going to move to computing the derivative of the regression cost function. Recall that the cost function is the sum over the data points of the squared difference between an observed output and a predicted output, plus the L2 penalty term.

```
Cost(w)
= SUM[ (prediction - output)^2 ]
+ l2_penalty*(w[0]^2 + w[1]^2 + ... + w[k]^2).
```

Since the derivative of a sum is the sum of the derivatives, we can take the derivative of the

Since the derivative of a sum is the sum of the derivatives, we can take the derivative of the first part (the RSS) as we did in the notebook for the unregularized case in Module 2 and add the derivative of the regularization part. As we saw, the derivative of the RSS with respect to w[i] can be written as:

```
2*SUM[ error*[feature_i] ]
```

The derivative of the regularization term with respect to w[i] is:

```
2*l2_penalty*w[i]
```

Summing both, we get

```
2*SUM[ error*[feature_i] ] + 2*l2_penalty*w[i]
```

That is, the derivative for the weight for feature i is the sum (over data points) of 2 times the product of the error and the feature itself, plus 2*l2_penalty*w[i].

IMPORTANT: We will not regularize the constant. Thus, in the case of the constant, the derivative is just twice the sum of the errors (without the 2*l2_penalty*w[0] term).

Recall that twice the sum of the product of two vectors is just twice the dot product of the two vectors. Therefore the derivative for the weight for feature_i is just two times the dot product between the values of feature_i and the current errors, plus 2*l2_penalty*w[i].

6. With this in mind write the derivative function which computes the derivative of the weight given the value of the feature (over all data points) and the errors (over all data points). To decide when to we are dealing with the constant (so we don't regularize it) we added the extra parameter to the call ‘feature_is_constant’ which you should set to True when computing the derivative of the constant and False otherwise.

e.g. in Python:

```
def feature_derivative_ridge(errors, feature, weight, l2_penalty, feature_is_con
stant):
    ...
    return derivative
```

7. To test your feature derivative function, run the following:

```
(example_features, example_output) = get_numpy_data(sales, ['sqft_living'], 'pri
ce')
my_weights = np.array([1., 10.])
test_predictions = predict_output(example_features, my_weights)
errors = test_predictions - example_output # prediction errors

# next two lines should print the same values
print feature_derivative_ridge(errors, example_features[:,1], my_weights[1], 1,
False)
print np.sum(errors*example_features[:,1])*2+20.
print ''

# next two lines should print the same values
print feature_derivative_ridge(errors, example_features[:,0], my_weights[0], 1,
True)
print np.sum(errors)*2.
```

8. Now we will write a function that performs a gradient descent. The basic premise is simple. Given a starting point we update the current weights by moving in the negative gradient direction. Recall that the gradient is the direction of increase and therefore the negative gradient is the direction of decrease and we're trying to minimize a cost function.

The amount by which we move in the negative gradient direction is called the 'step size'. We stop when we are 'sufficiently close' to the optimum. Unlike in Module 2, this time we will set a maximum number of iterations and take gradient steps until we reach this maximum number. If no maximum number is supplied, the maximum should be set 100 by default. (Use default parameter values in Python.)

With this in mind, write a gradient descent function using your derivative function above. For each step in the gradient descent, we update the weight for each feature before computing our stopping criteria. The function will take the following parameters:

- 2D feature matrix
- array of output values
- initial weights
- step size
- L2 penalty
- maximum number of iterations

To make your job easier, we provide a skeleton in Python:

```
def ridge_regression_gradient_descent(feature_matrix, output, initial_weights, s
tep_size, l2_penalty, max_iterations=100):
    weights = np.array(initial_weights) # make sure it's a numpy array
        #while not reached maximum number of iterations:
        # compute the predictions using your predict_output() function

        # compute the errors as predictions - output
        for i in xrange(len(weights)): # loop over each weight
            # Recall that feature_matrix[:,i] is the feature column associated w
ith weights[i]
            # compute the derivative for weight[i].
            #(Remember: when i=0, you are computing the derivative of the consta
nt!)

            # subtract the step size times the derivative from the current weigh
t
              return weights
```

9. The L2 penalty gets its name because it causes weights to have small L2 norms than otherwise. Let's see how large weights get penalized. Let us consider a simple model with 1 feature.

- features: 'sqft_living'
- output: 'price'

10. Split the dataset into training set and test set. If you are using GraphLab Create, call

```
train_data,test_data = sales.random_split(.8,seed=0)
```

Otherwise, please download the csv files from the download section.

11. Convert the training set and test set using the 'get_numpy_data' function.e.g. in Python:

```
simple_features = ['sqft_living']
my_output = 'price'
(simple_feature_matrix, output) = get_numpy_data(train_data, simple_features, my
_output)
(simple_test_feature_matrix, test_output) = get_numpy_data(test_data, simple_fea
tures, my_output)
```

12. First, let's consider no regularization. Set the L2 penalty to 0.0 and run your ridge regression algorithm to learn the weights of the simple model (described above). Use the following parameters:

- step_size = 1e-12
- max_iterations = 1000
- initial_weights = all zeros

Store the learned weights as

```
simple_weights_0_penalty
```

we'll use them later.

13. Next, let's consider high regularization. Set the L2 penalty to 1e11 and run your ridge regression to learn the weights of the simple model. Use the same parameters as above. Call your weights:

```
simple_weights_0_penalty
```

we'll use them later.

14. If you have access to matplotlib, the following piece of code will plot the two learned models. (The blue line is for the model with no regularization and the red line is for the one with high regularization.)

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(simple_feature_matrix,output,'k.',
        simple_feature_matrix,predict_output(simple_feature_matrix, simple_weigh
ts_0_penalty),'b-',
        simple_feature_matrix,predict_output(simple_feature_matrix, simple_weigh
ts_high_penalty),'r-')
```

If you do not have access to matplotlib, look at each set of coefficients. If you were to plot 'sqft_living' vs the price, which of the two coefficients is the slope and which is the intercept?

15. ***Quiz Question: What is the value of the coefficient for sqft_living that you learned with no regularization, rounded to 1 decimal place? What about the one with high regularization?***

16. ***Quiz Question: Comparing the lines you fit with the with no regularization versus high regularization, which one is steeper?***

17. Compute the RSS on the TEST data for the following three sets of weights:

- The initial weights (all zeros)
- The weights learned with no regularization
- The weights learned with high regularization

18. ***Quiz Question: What are the RSS on the test data for each of the set of weights above (initial, no regularization, high regularization)?***

19. Let us now consider a model with 2 features: ['sqft_living', 'sqft_living_15']. First, create Numpy version of your training and test data with the two features.

e.g. in Python:

```
model_features = ['sqft_living', 'sqft_living15']
my_output = 'price'
(feature_matrix, output) = get_numpy_data(train_data, model_features, my_output)
(test_feature_matrix, test_output) = get_numpy_data(test_data, model_features, m
y_output)
```

20. First, let's consider no regularization. Set the L2 penalty to 0.0 and run your ridge regression algorithm. Use the following parameters:

- initial_weights = all zeros
- step size = 1e-12
- max_iterations = 1000

Call the learned weights

```
multiple_weights_0_penalty
```

21. Next, let's consider high regularization. Set the L2 penalty to 1e11 and run your ridge regression to learn the weights of the simple model. Use the same parameters as above. Call your weights:

```
multiple_weights_high_penalty
```

22. ***Quiz Question: What is the value of the coefficient for 'sqft_living' that you learned with no regularization, rounded to 1 decimal place? What about the one with high regularization?***

23. Compute the RSS on the TEST data for the following three sets of weights:

- The initial weights (all zeros)
- The weights learned with no regularization
- The weights learned with high regularization

24. ***Quiz Question: What are the RSS on the test data for each of the set of weights above (initial, no regularization, high regularization)?***

25. Predict the house price for the 1st house in the test set using the no regularization and high regularization models. (Remember that python starts indexing from 0.)

26. ***Quiz Question: What's the error in predicting the price of the first house in the test set using the weights learned with no regularization? What about with high regularization?***