Regression Week 4: Ridge Regression Assignment 1

In this assignment, we will run ridge regression multiple times with different L2 penalties to see which one produces the best fit. We will revisit the example of polynomial regression as a means to see the effect of L2 regularization. In particular, we will:

- Use a pre-built implementation of regression to run polynomial regression
- Use matplotlib to visualize polynomial regressions
- Use a pre-built implementation of regression to run polynomial regression, this time with L2 penalty
- Use matplotlib to visualize polynomial regressions under L2 regularization
- Choose best L2 penalty using cross-validation.
- Assess the final fit using test data.

We will continue to use the House data from previous assignments. (In the next programming assignment for this module, you will implement your own ridge regression learning algorithm using gradient descent.)

IMPORTANT: Choice of tools

For the purpose of this assessment, you may choose between GraphLab Create and scikit-learn (with Pandas). You are free to experiment with other tools (e.g. R or Matlab), but they may not produce correct numbers for the quiz questions.

- If you are using GraphLab Create, download the IPython notebook and follow the instructions contained in the notebook.
- If you are using Pandas+scikit-learn combination, follow through the instructions in this reading.

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

- Download the companion IPython Notebook: week-4-ridge-regression-assignment-1-blank.ipynb
- 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
- Download the King County House Sales training data csv file: wk3_kc_house_train_data.csv
- Download the King County House Sales validation data csv file: wk3_kc_house_valid_data.csv
- Download the King County House Sales testing data csv file: wk3_kc_house_test_data.csv
- Download csv file containing randomly shuffled rows of training and validation data combined:wk3_kc_house_train_valid_shuffled.csv
- Download the King County House Sales subset 1 data csv file: wk3_kc_house_set_1_data.csv
- Download the King County House Sales subset 2 data csv file: wk3_kc_house_set_2_data.csv
- Download the King County House Sales subset 3 data csv file: wk3_kc_house_set_3_data.csv
- Download the King County House Sales subset 4 data csv file: wk3_kc_house_set_4_data.csv

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

If you are using GraphLab Create and the companion IPython Notebook

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

If you are using scikit-learn with Pandas

1. Copy and paste an equivalent of ‘polynomial_sframe’ function from Module 3 (Polynomial Regression). This function accepts an array ‘feature’ (of type pandas.Series) and a maximal ‘degree’ and returns an data frame (of type pandas.DataFrame) with the first column equal to ‘feature’ and the remaining columns equal to ‘feature’ to increasing integer powers up to ‘degree’.

2. For the remainder of the assignment we will be working with the house Sales data as in Module 3 (Polynomial Regression). Load in the data and also sort the sales data frame by ‘sqft_living’. When we plot the fitted values we want to join them up in a line and this works best if the variable on the X-axis (which will be ‘sqft_living’) is sorted. For houses with identical square footage, we break the tie by their prices.

```
import pandas as pd


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


sales = pd.read_csv('kc_house_data.csv', dtype=dtype_dict)

sales = sales.sort(['sqft_living','price'])
```

3. Let us revisit the 15th-order polynomial model using the 'sqft_living' input. Generate polynomial features up to degree 15 using `polynomial_sframe()` and fit a model with these features. When fitting the model, use an L2 penalty of 1.5e-5:

```
l2_small_penalty = 1.5e-5
```

Note: When we have so many features and so few data points, the solution can become highly numerically unstable, which can sometimes lead to strange unpredictable results. Thus, rather than using no regularization, we will introduce a tiny amount of regularization (l2_penalty=1.5e-5) to make the solution numerically stable. (In lecture, we discussed the fact that regularization can also help with numerical stability, and here we are seeing a practical example.)

With the L2 penalty specified above, fit the model and print out the learned weights. Add "alpha=l2_small_penalty" and "normalize=True" to the parameter list of linear_model.Ridge:

```
from sklearn import linear_model

import numpy as np
```

```
poly15_data = polynomial_sframe(sales['sqft_living'], 15) # use equivalent of `polyno
mial_sframe`

model = linear_model.Ridge(alpha=l2_small_penalty, normalize=True)

model.fit(poly15_data, sales['price'])
```

4. ***Quiz Question: What's the learned value for the coefficient of feature power_1?***

Observe Overfitting

5. Recall from Module 3 (Polynomial Regression) that the polynomial fit of degree 15 changed wildly whenever the data changed. In particular, when we split the sales data into four subsets and fit the model of degree 15, the result came out to be very different for each subset. The model had a high variance. We will see in a moment that ridge regression reduces such variance. But first, we must reproduce the experiment we did in Module 3.

6. For this section, please download the provided csv files for each subset and load them with the given list of types:

```
# dtype_dict same as above

set_1 = pd.read_csv('wk3_kc_house_set_1_data.csv', dtype=dtype_dict)

set_2 = pd.read_csv('wk3_kc_house_set_2_data.csv', dtype=dtype_dict)

set_3 = pd.read_csv('wk3_kc_house_set_3_data.csv', dtype=dtype_dict)

set_4 = pd.read_csv('wk3_kc_house_set_4_data.csv', dtype=dtype_dict)
```

7. Just as we did in Module 3 (Polynomial Regression), fit a 15th degree polynomial on each of the 4 sets, plot the results and view the weights for the four models. This time, set

```
l2_small_penalty=1e-9
```

and use this value for the L2 penalty. Make sure to add "alpha=l2_small_penalty" and "normalize=True" to the parameter list of linear_model.Ridge.

The four curves should differ from one another a lot, as should the coefficients you learned.

8. ***Quiz Question: For the models learned in each of these training sets, what are the smallest and largest values you learned for the coefficient of feature power_1?*** (For the purpose of answering this question, negative numbers are considered "smaller" than positive numbers. So -5 is smaller than -3, and -3 is smaller than 5 and so forth.)

Ridge regression comes to rescue

9. Generally, whenever we see weights change so much in response to change in data, we believe the variance of our estimate to be large. Ridge regression aims to address this issue by penalizing "large" weights. (The weights looked quite small, but they are not that small because 'sqft_living' input is in the order of thousands.)

10. Fit a 15th-order polynomial model on set_1, set_2, set_3, and set_4, this time with a large L2 penalty. Make sure to add "alpha=l2_large_penalty" and "normalize=True" to the parameter list, where the value of l2_large_penalty is given by

```
l2_large_penalty=1.23e2
```

These curves should vary a lot less, now that you introduced regularization.

11. ***QUIZ QUESTION: For the models learned with regularization in each of these training sets, what are the smallest and largest values you learned for the coefficient of feature power_1?*** (For the purpose of answering this question, negative numbers are considered "smaller" than positive numbers. So -5 is smaller than -3, and -3 is smaller than 5 and so forth.)

Selecting an L2 penalty via cross-validation

12. Just like the polynomial degree, the L2 penalty is a "magic" parameter we need to select. We could use the validation set approach as we did in the last module, but that approach has a major disadvantage: it leaves fewer observations available for training. **Cross-validation** seeks to overcome this issue by using all of the training set in a smart way.

We will implement a kind of cross-validation called k-fold cross-validation. The method gets its name because it involves dividing the training set into k segments of roughly equal size. Similar to the validation set method, we measure the validation error with one of the segments designated as the validation set. The major difference is that we repeat the process k times as follows:

- Set aside segment 0 as the validation set, and fit a model on rest of data, and evalutate it on this validation set
- Set aside segment 1 as the validation set, and fit a model on rest of data, and evalutate it on this validation set
- ...
- Set aside segment k-1 as the validation set, and fit a model on rest of data, and evalutate it on this validation set

After this process, we compute the average of the k validation errors, and use it as an estimate of the generalization error. Notice that all observations are used for both training and validation, as we iterate over segments of data.

13. To estimate the generalization error well, it is crucial to shuffle the training data before dividing them into segments. We reserve 10% of the data as the test set and randomly shuffle the remainder. Le'ts call the shuffled data 'train_valid_shuffled'.

For the purpose of this assignment, let us download the csv file containing pre-shuffled rows of training and validation sets combined: wk3_kc_house_train_valid_shuffled.csv. In practice, you would shuffle the rows with a dynamically determined random seed.

```
train_valid_shuffled = pd.read_csv('wk3_kc_house_train_valid_shuffled.csv', dtype=dty
pe_dict)

test = pd.read_csv('wk3_kc_house_test_data.csv', dtype=dtype_dict)
```

14. Divide the combined training and validation set into equal segments. Each segment should receive n/k elements, where n is the number of observations in the training set and k is the number of segments. Since the segment 0 starts at index 0 and contains n/k elements, it ends at index (n/k)-1. The segment 1 starts where the segment 0 left off, at index (n/k). With n/k elements, the segment 1 ends at index (n*2/k)-1. Continuing in this fashion, we deduce that the segment i starts at index (n*i/k) and ends at (n*(i+1)/k)-1.

With this pattern in mind, we write a short loop that prints the starting and ending indices of each segment, just to make sure you are getting the splits right.

```
n = len(train_valid_shuffled)

k = 10 # 10-fold cross-validation


for i in xrange(k):

    start = (n*i)/k

    end = (n*(i+1))/k-1

    print i, (start, end)
```

Let us familiarize ourselves with array slicing with Pandas. To extract a continuous slice from a DataFrame, use colon in square brackets. For instance, the following cell extracts rows 0 to 9 of train_valid_shuffled. Notice that the first index (0) is included in the slice but the last index (10) is omitted.

```
train_valid_shuffled[0:10] # select rows 0 to 9
```

If the observations are grouped into 10 segments, the segment i is given by

```
start = (n*i)/10

end = (n*(i+1))/10

train_valid_shuffled[start:end+1]
```

Meanwhile, to choose the remainder of the data that's not part of the segment i, we select two slices (0:start) and (end+1:n) and paste them together.

```
train_valid_shuffled[0:start].append(train_valid_shuffled[end+1:n])
```

15. Now we are ready to implement k-fold cross-validation. Write a function that computes k validation errors by designating each of the k segments as the validation set. It accepts as parameters (i) k, (ii) l2_penalty, (iii) dataframe containing input features (e.g. poly15_data) and (iv) column of output values (e.g. price). The function returns the average validation error using k segments as validation sets. We shall assume that the input dataframe does not contain the output column.

For each i in [0, 1, ... k-1]:

- Compute starting and ending indices of segment i and call 'start' and 'end'
- Form validation set by taking a slice (start:end+1) from the data.
- Form training set by appending slice (end+1:n) to the end of slice (0:start).
- Train a linear model using training set just formed, with a given l2_penalty
- Compute validation error (RSS) using validation set just formed

e.g. in Python:

```
def k_fold_cross_validation(k, l2_penalty, data, output):

    ...

    return [average validation error]
```

16. Once we have a function to compute the average validation error for a model, we can write a loop to find the model that minimizes the average validation error. Write a loop that does the following:

- We will again be aiming to fit a 15th-order polynomial model using the sqft_living input

- For each l2_penalty in [10^3, 10^3.5, 10^4, 10^4.5, ..., 10^9] (to get this in Python, you can use this Numpy function: np.logspace(3, 9, num=13).): Run 10-fold cross-validation with l2_penalty.
- Report which L2 penalty produced the lowest average validation error.

Note: since the degree of the polynomial is now fixed to 15, to make things faster, you should generate polynomial features in advance and re-use them throughout the loop. Make sure to use train_valid_shuffled when generating polynomial features!

17. Quiz Question: What is the best value for the L2 penalty according to 10-fold validation?

18. Once you found the best value for the L2 penalty using cross-validation, it is important to retrain a final model on all of the training data using this value of l2_penalty. This way, your final model will be trained on the entire dataset.

19. Quiz Question: Using the best L2 penalty found above, train a model using all training data. What is the RSS on the TEST data of the model you learn with this L2 penalty?