Regression Week 5: LASSO Assignment 1

In this assignment, you will use LASSO to select features, building on a pre-implemented solver for LASSO (using GraphLab Create, though you can use other solvers). You will:

- Run LASSO with different L1 penalties.
- Choose best L1 penalty using a validation set.
- Choose best L1 penalty using a validation set, with additional constraint on the size of subset.

In the second assignment, you will implement your own LASSO solver, using coordinate 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-5-lasso-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 using scikit-learn with Pandas:

- 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

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:

The instructions may apply to other tools, but the set of parameters are specific to scikit-learn.

0. Load the sales dataset using Pandas:

```
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':str, '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_lot':int, 'view':int}


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

1. Create new features by performing following transformation on inputs:

```
from math import log, sqrt

sales['sqft_living_sqrt'] = sales['sqft_living'].apply(sqrt)

sales['sqft_lot_sqrt'] = sales['sqft_lot'].apply(sqrt)
```

```
sales['bedrooms_square'] = sales['bedrooms']*sales['bedrooms']
sales['floors_square'] = sales['floors']*sales['floors']
```

- Squaring bedrooms will increase the separation between not many bedrooms (e.g. 1) and lots of bedrooms (e.g. 4) since 1^2 = 1 but 4^2 = 16. Consequently this variable will mostly affect houses with many bedrooms.
- On the other hand, taking square root of sqft_living will decrease the separation between big house and small house. The owner may not be exactly twice as happy for getting a house that is twice as big.

2. Using the entire house dataset, learn regression weights using an L1 penalty of 5e2. Make sure to add "normalize=True" when creating the Lasso object. Refer to the following code snippet:

```
from sklearn import linear_model  # using scikit-learn


model_all = linear_model.Lasso(alpha=5e2, normalize=True) # set parameters
model_all.fit(sales[all_features], sales['price']) # learn weights
```

3. Quiz Question: Which features have been chosen by LASSO, i.e. which features were assigned nonzero weights?

4. To find a good L1 penalty, we will explore multiple values using a validation set. Let us do three way split into train, validation, and test sets. Download the provided csv files containing training, validation and test sets.

```
testing = pd.read_csv('wk3_kc_house_test_data.csv', dtype=dtype_dict)
training = pd.read_csv('wk3_kc_house_train_data.csv', dtype=dtype_dict)
validation = pd.read_csv('wk3_kc_house_valid_data.csv', dtype=dtype_dict)
```

Make sure to create the 4 features as we did in #1:

```
testing['sqft_living_sqrt'] = testing['sqft_living'].apply(sqrt)
testing['sqft_lot_sqrt'] = testing['sqft_lot'].apply(sqrt)
testing['bedrooms_square'] = testing['bedrooms']*testing['bedrooms']
testing['floors_square'] = testing['floors']*testing['floors']


training['sqft_living_sqrt'] = training['sqft_living'].apply(sqrt)
training['sqft_lot_sqrt'] = training['sqft_lot'].apply(sqrt)
```

```
training['bedrooms_square'] = training['bedrooms']*training['bedrooms']
training['floors_square'] = training['floors']*training['floors']

validation['sqft_living_sqrt'] = validation['sqft_living'].apply(sqrt)
validation['sqft_lot_sqrt'] = validation['sqft_lot'].apply(sqrt)
validation['bedrooms_square'] = validation['bedrooms']*validation['bedrooms']
validation['floors_square'] = validation['floors']*validation['floors']
```

5. Now for each l1_penalty in [10^1, 10^1.5, 10^2, 10^2.5, ..., 10^7] (to get this in Python, type np.logspace(1, 7, num=13).)

- Learn a model on TRAINING data using the specified l1_penalty. Make sure to specify normalize=True in the constructor:

```
model = linear_model.Lasso(alpha=l1_penalty, normalize=True)
```

- Compute the RSS on VALIDATION for the current model (print or save the RSS)

Report which L1 penalty produced the lower RSS on VALIDATION.

6. Quiz Question: Which was the best value for the l1_penalty, i.e. which value of l1_penalty produced the lowest RSS on VALIDATION data?

7. Now that you have selected an L1 penalty, compute the RSS on TEST data for the model with the best L1 penalty.

8. Quiz Question: Using the best L1 penalty, how many nonzero weights do you have? Count the number of nonzero coefficients first, and add 1 if the intercept is also nonzero. A succinct way to do this is

```
np.count_nonzero(model.coef_) + np.count_nonzero(model.intercept_)
```

where 'model' is an instance of linear_model.Lasso.

9. What if we absolutely wanted to limit ourselves to, say, 7 features? This may be important if we want to derive "a rule of thumb" --- an interpretable model that has only a few features in them.

You are going to implement a simple, two phase procedure to achieve this goal:

- Explore a large range of ‘l1_penalty’ values to find a narrow region of ‘l1_penalty’ values where models are likely to have the desired number of non-zero weights.

- Further explore the narrow region you found to find a good value for ‘l1_penalty’ that achieves the desired sparsity. Here, we will again use a validation set to choose the best value for ‘l1_penalty’.

10. Assign 7 to the variable ‘max_nonzeros’.

11. Exploring large range of l1_penalty

For l1_penalty in np.logspace(1, 4, num=20):

- Fit a regression model with a given l1_penalty on TRAIN data. Add "alpha=l1_penalty" and "normalize=True" to the parameter list.

```
model = linear_model.Lasso(alpha=l1_penalty, normalize=True)
```

- Extract the weights of the model and count the number of nonzeros. Take account of the intercept as we did in #8, adding 1 whenever the intercept is nonzero. Save the number of nonzeros to a list.

12. Out of this large range, we want to find the two ends of our desired narrow range of l1_penalty. At one end, we will have l1_penalty values that have too few non-zeros, and at the other end, we will have an l1_penalty that has too many non-zeros.

More formally, find:

- The largest l1_penalty that has more non-zeros than ‘max_nonzeros’ (if we pick a penalty smaller than this value, we will definitely have too many non-zero weights)Store this value in the variable ‘l1_penalty_min’ (we will use it later)
- The smallest l1_penalty that has fewer non-zeros than ‘max_nonzeros’ (if we pick a penalty larger than this value, we will definitely have too few non-zero weights)Store this value in the variable ‘l1_penalty_max’ (we will use it later)

Hint: there are many ways to do this, e.g.:

- Programmatically within the loop above
- Creating a list with the number of non-zeros for each value of l1_penalty and inspecting it to find the appropriate boundaries.

13. Quiz Question: What values did you find for l1_penalty_min and l1_penalty_max?

14. Exploring narrower range of l1_penalty

We now explore the region of l1_penalty we found: between ‘l1_penalty_min’ and ‘l1_penalty_max’. We look for the L1 penalty in this range that produces exactly the right number of nonzeros and also minimizes RSS on the VALIDATION set.

For l1_penalty in np.linspace(l1_penalty_min,l1_penalty_max,20):

- Fit a regression model with a given l1_penalty on TRAIN data. As before, use "alpha=l1_penalty" and "normalize=True".
- Measure the RSS of the learned model on the VALIDATION set

Find the model that the lowest RSS on the VALIDATION set and has sparsity equal to ‘max_nonzeros’. (Again, take account of the intercept when counting the number of nonzeros.)

15. Quiz Question: What value of l1_penalty in our narrow range has the lowest RSS on the VALIDATION set and has sparsity equal to ‘max_nonzeros’?

16. Quiz Question: What features in this model have non-zero coefficients?