Regression Week 5: Feature Selection and LASSO (Interpretation)

In this notebook, 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 notebook, you will implement your own LASSO solver, using coordinate descent.

Fire up graphlab create

In [85]:

```
import graphlab
```

Load in house sales data

Dataset is from house sales in King County, the region where the city of Seattle, WA is located.

In [86]:

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

Create new features

As in Week 2, we consider features that are some transformations of inputs.

In [87]:

```
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']

# In the dataset, 'floors' was defined with type string,
# so we'll convert them to float, before creating a new feature.
sales['floors'] = sales['floors'].astype(float)
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.

Learn regression weights with L1 penalty

Let us fit a model with all the features available, plus the features we just created above.

In [88]:

```
all_features = ['bedrooms', 'bedrooms_square',
```

```
              'bathrooms',
              'sqft_living', 'sqft_living_sqrt',
              'sqft_lot', 'sqft_lot_sqrt',
              'floors', 'floors_square',
              'waterfront', 'view', 'condition', 'grade',
              'sqft_above',
              'sqft_basement',
              'yr_built', 'yr_renovated']
```

Applying L1 penalty requires adding an extra parameter (`l1_penalty`) to the linear regression call in GraphLab Create. (Other tools may have separate implementations of LASSO.) Note that it's important to set `l2_penalty=0` to ensure we don't introduce an additional L2 penalty.

In [89]:

```
model_all = graphlab.linear_regression.create(sales, target='price',
features=all_features,
                                              validation_set=None,
l2_penalty=0., l1_penalty=1e10)
```

```
PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 21613
PROGRESS: Number of features          : 17
PROGRESS: Number of unpacked features : 17
PROGRESS: Number of coefficients    : 18
PROGRESS: Starting Accelerated Gradient (FISTA)
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+-----------+--------------+-------------
-------+---------------+
PROGRESS: | Iteration | Passes   | Step size | Elapsed Time | Training-
max_error | Training-rmse |
PROGRESS: +-----------+----------+-----------+--------------+-------------
-------+---------------+
PROGRESS: Tuning step size. First iteration could take longer than
subsequent iterations.
PROGRESS: | 1         | 2        | 0.000002  | 0.252015     |
6962915.603493     | 426631.749026 |
PROGRESS: | 2         | 3        | 0.000002  | 0.274016     |
6843144.200219     | 392488.929838 |
PROGRESS: | 3         | 4        | 0.000002  | 0.295017     |
6831900.032123     | 385340.166783 |
PROGRESS: | 4         | 5        | 0.000002  | 0.317018     |
6847166.848958     | 384842.383767 |
PROGRESS: | 5         | 6        | 0.000002  | 0.339020     |
6869667.895833     | 385998.458623 |
PROGRESS: | 6         | 7        | 0.000002  | 0.361021     |
6847177.773672     | 380824.455891 |
PROGRESS: +-----------+----------+-----------+--------------+-------------
-------+---------------+
PROGRESS: TERMINATED: Iteration limit reached.
PROGRESS: This model may not be optimal. To improve it, consider
increasing `max_iterations`.
```

Find what features had non-zero weight.

In [90]:
```
print('number of nonzeros = %d' % (model_all.coefficients['value']).nnz())
```

```
number of nonzeros = 6
```

In [104]:
```
print(model_all.coefficients.print_rows(num_rows=model_all.num_coefficient
s))
```

name	index	value
(intercept)	None	274873.05595
bedrooms	None	0.0
bedrooms_square	None	0.0
bathrooms	None	8468.53108691
sqft_living	None	24.4207209824
sqft_living_sqrt	None	350.060553386
sqft_lot	None	0.0
sqft_lot_sqrt	None	0.0
floors	None	0.0
floors_square	None	0.0
waterfront	None	0.0
view	None	0.0
condition	None	0.0
grade	None	842.068034898
sqft_above	None	20.0247224171
sqft_basement	None	0.0
yr_built	None	0.0
yr_renovated	None	0.0

```
[18 rows x 3 columns]

None
```

Note that a majority of the weights have been set to zero. So by setting an L1 penalty that's large enough, we are performing a subset selection.

QUIZ QUESTION: According to this list of weights, which of the features have been chosen?

Selecting an L1 penalty

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:

- Split our sales data into 2 sets: training and test
- Further split our training data into two sets: train, validation

Be *very* careful that you use seed = 1 to ensure you get the same answer!

In [91]:
```
(training_and_validation, testing) = sales.random_split(.9,seed=1) # 
initial train/test split
(training, validation) = training_and_validation.random_split(0.5, seed=1) 
# split training into train and validate
```

Next, we write a loop that does the following:

- For `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)`.)
  - Fit a regression model with a given `l1_penalty` on TRAIN data. Specify `l1_penalty=l1_penalty` and `l2_penalty=0.` in the parameter list.
  - Compute the RSS on VALIDATION data (here you will want to use `.predict()`) for that `l1_penalty`
- Report which `l1_penalty` produced the lowest RSS on validation data.

When you call `linear_regression.create()` make sure you set `validation_set = None`.
Note: you can turn off the print out of `linear_regression.create()` with `verbose = False`

In [92]:
```
import numpy as np

penalty_rss = []
for l1_penalty in np.logspace(1, 7, num=13):
    model = graphlab.linear_regression.create(training, target='price',
features=all_features,
                                              validation_set=None,
l2_penalty=0.0, l1_penalty=l1_penalty, verbose=False)
    # First get the predictions
    predictions = model.predict(validation)
    # then compute the residuals (since we are squaring it doesn't matter
which order you subtract)
    residuals = validation['price'] - predictions
    # square the residuals and add them up
    residuals_squared = residuals * residuals
    RSS = residuals_squared.sum()
    print("l1_penalty: %s, RSS: $%.6f" % (l1_penalty, RSS))
    penalty_rss.append((l1_penalty, RSS))
```

```
l1_penalty: 10.0, RSS: $625766285142459.875000
l1_penalty: 31.6227766017, RSS: $625766285362394.125000
l1_penalty: 100.0, RSS: $625766286057885.000000
l1_penalty: 316.227766017, RSS: $625766288257224.625000
l1_penalty: 1000.0, RSS: $625766295212186.750000
l1_penalty: 3162.27766017, RSS: $625766317206080.500000
l1_penalty: 10000.0, RSS: $625766386760658.125000
l1_penalty: 31622.7766017, RSS: $625766606749278.500000
l1_penalty: 100000.0, RSS: $625767302791634.125000
l1_penalty: 316227.766017, RSS: $625769507643886.250000
l1_penalty: 1000000.0, RSS: $625776517727024.000000
l1_penalty: 3162277.66017, RSS: $625799062845467.000000
l1_penalty: 10000000.0, RSS: $625883719085425.250000
```

QUIZ QUESTIONS

1 What was the best value for the `l1_penalty`?
2 What is the RSS on TEST data of the model with the best `l1_penalty`?

In [93]:
```
penalty_rss = sorted(penalty_rss, key = lambda x : x[1])
l1_penalty_best = penalty_rss[0][0]
print(l1_penalty_best)
```

```
10.0
```

In [94]:

```
model = graphlab.linear_regression.create(training, target='price',
features=all_features,
                                              validation_set=None,
l2_penalty=0.0, l1_penalty=l1_penalty_best, verbose=False)
# First get the predictions
predictions = model.predict(testing)
# then compute the residuals
residuals = testing['price'] - predictions
# square the residuals and add them up
residuals_squared = residuals * residuals
RSS = residuals_squared.sum()
print("RSS on TEST data: $%.6f" % (RSS))
```

```
RSS on TEST data: $156983602381664.187500
```

QUIZ QUESTION Also, using this value of L1 penalty, how many nonzero weights do you have?

In [95]:

```
print('number of nonzeros = %d' % (model.coefficients['value']).nnz())
```

```
number of nonzeros = 18
```

Limit the number of nonzero weights

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.

In this section, you are going to implement a simple, two phase procedure to achive this goal:

1 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.
2 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`.

In [96]:

```
max_nonzeros = 7
```

Exploring the larger range of values to find a narrow range with the desired sparsity

Let's define a wide range of possible `l1_penalty_values`:

In [97]:

```
l1_penalty_values = np.logspace(8, 10, num=20)
```

Now, implement a loop that search through this space of possible `l1_penalty` values:

- For `l1_penalty` in `np.logspace(8, 10, num=20)`:
  - Fit a regression model with a given `l1_penalty` on TRAIN data. Specify `l1_penalty=l1_penalty` and `l2_penalty=0.` in the parameter list. When you call `linear_regression.create()` make sure you set `validation_set = None`
  - Extract the weights of the model and count the number of nonzeros. Save the number of nonzeros to a list.
    - *Hint: `model['coefficients']['value']` gives you an SArray with the parameters you learned. If you call the method `.nnz()` on it, you will find the*

number of non-zero parameters!

In [98]:
```
penalty_nnz = []

for l1_penalty in l1_penalty_values:
    model = graphlab.linear_regression.create(training, target='price', 
features=all_features,
                                              validation_set=None, 
l2_penalty=0.0, l1_penalty=l1_penalty, verbose=False)

    penalty_nnz.append((l1_penalty, (model.coefficients['value']).nnz()))
```
In [99]:
```
penalty_nnz
```
Out[99]:
```
[(100000000.0, 18),
 (127427498.57031322, 18),
 (162377673.91887242, 18),
 (206913808.11147901, 18),
 (263665089.87303555, 17),
 (335981828.62837881, 17),
 (428133239.8719396, 17),
 (545559478.11685145, 17),
 (695192796.17755914, 17),
 (885866790.41008317, 16),
 (1128837891.6846883, 15),
 (1438449888.2876658, 15),
 (1832980710.8324375, 13),
 (2335721469.0901213, 12),
 (2976351441.6313128, 10),
 (3792690190.7322536, 6),
 (4832930238.5717525, 5),
 (6158482110.6602545, 3),
 (7847599703.5146227, 1),
 (10000000000.0, 1)]
```

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_nonzero` (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_nonzero` (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.

In [100]:
```
l1_penalty_min = max([t[0] for t in penalty_nnz if t[1] > max_nonzeros])
l1_penalty_max = min([t[0] for t in penalty_nnz if t[1] < max_nonzeros])
```

```
print(l1_penalty_min, l1_penalty_max)
```

```
(2976351441.6313128, 3792690190.7322536)
```

QUIZ QUESTIONS
What values did you find for `l1_penalty_min` and`l1_penalty_max`?

Exploring the narrow range of values to find the solution with the right number of non-zeros that has lowest RSS on the validation set

We will now explore the narrow region of `l1_penalty` values we found:

In [101]:
```
l1_penalty_values = np.linspace(l1_penalty_min,l1_penalty_max,20)
```

- 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. Specify `l1_penalty=l1_penalty` and `l2_penalty=0.` in the parameter list. When you call `linear_regression.create()` make sure you set `validation_set = None`
  - 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_nonzero`.

In [102]:
```
penalty_rss_model = []
for l1_penalty in l1_penalty_values:
    model = graphlab.linear_regression.create(training, target='price',
features=all_features, l2_penalty=0.0,
                                                          l1_penalty=l1_penalty,
validation_set=None, verbose=False)

    model_nnz = (model.coefficients['value']).nnz()

    if model_nnz == max_nonzeros:
         # First get the predictions
        predictions = model.predict(validation)
        # then compute the residuals
        residuals = validation['price'] - predictions
        # square the residuals and add them up
        residuals_squared = residuals * residuals
        RSS = residuals_squared.sum()
        penalty_rss_model.append((l1_penalty, RSS, model))
```

QUIZ QUESTIONS

1. What value of `l1_penalty` in our narrow range has the lowest RSS on the VALIDATION set and has sparsity *equal* to `max_nonzeros`?
2. What features in this model have non-zero coefficients?

In [103]:
```
penalty_rss_model = sorted(penalty_rss_model, key = lambda x : x[1])
l1_penalty_optimum = penalty_rss_model[0][0]
model_optimum = penalty_rss_model[0][2]
print(l1_penalty_optimum)
```

```
print(model_optimum.coefficients.print_rows(num_rows=model_optimum.num_coe
fficients))
```

```
3448968612.16
+------------------+-------+----------------+
|       name       | index |     value      |
+------------------+-------+----------------+
|   (intercept)    |  None | 222253.192544  |
|     bedrooms     |  None | 661.722717782  |
|  bedrooms_square |  None |      0.0       |
|    bathrooms     |  None | 15873.9572593  |
|   sqft_living    |  None | 32.4102214513  |
| sqft_living_sqrt |  None | 690.114773313  |
|     sqft_lot     |  None |      0.0       |
|  sqft_lot_sqrt   |  None |      0.0       |
|      floors      |  None |      0.0       |
|  floors_square   |  None |      0.0       |
|    waterfront    |  None |      0.0       |
|       view       |  None |      0.0       |
|    condition     |  None |      0.0       |
|      grade       |  None | 2899.42026975  |
|    sqft_above    |  None | 30.0115753022  |
|  sqft_basement   |  None |      0.0       |
|     yr_built     |  None |      0.0       |
|   yr_renovated   |  None |      0.0       |
+------------------+-------+----------------+
[18 rows x 3 columns]

None
```