HW2_STAT760

February 8, 2018

1 HW 2: Apply Linear Regression to Construct a Prostate Cancer Model

Blanca Miller
STAT 760
02/06/2018
Prostrate Data Info: http://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.info.txt
Data Set: http://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data
Given a training set of prostate cancer linear data, estimate the model parameters using subset selection.

1.1 STEPS

1. import data
2. break data into two groups: train and test
3. break the training set into two matrices X : design matrix (add column of 1s at beginning) y : vector of responses
4. convert the X and y data frames into numpy arrays
5. standardize the predictors to have unit variance
6. estimate weights
7. calculate RSS for training set for all possible models
8. Graph k by RSS

- Fxn evaluates RSS for given X, y, beta
- Fxn trains models given list of columns
- Fxn generates all possible list of columns
- Fxn plot dictionary

1.2 Imports

```
In [1]: import numpy as np
        import pandas as pd
        import itertools
        import matplotlib.pyplot as plt
```

1.3 Load Data

```
In [2]: data = pd.read_csv('prostate.data', delimiter='\t')
```

1.4 Functions

```
In [3]: def evaluateRSS(df, predictor_subset, beta):

            # generate predictor df
            predictor_df = df[predictor_subset]

            # generate response df
            response_df = df["lpsa"]

            # convert predictor df to ndarray
            predictor_matrix = predictor_df.as_matrix()

            # normalize predictors
            predictor_matrix = (predictor_matrix - np.mean(predictor_matrix, axis=0))/np.std(pre

            # adds constant term 1s to predictor matrix
            predictor_matrix = np.c_[np.ones(len(predictor_matrix)), predictor_matrix]

            # convert response df to ndarray
            response_matrix = response_df.as_matrix()

            # compute error
            e = response_matrix - np.dot(predictor_matrix, beta)

            # return error
            return np.dot(e,e)
```

```
In [4]: def train(df, predictors_subset):

            # generate predictor df
            predictor_df = df[predictors_subset]

            # generate response df
            response_df = df["lpsa"]

            # convert predictor df to ndarray
            predictor_matrix = predictor_df.as_matrix()

            # normalize predictors
            predictor_matrix = (predictor_matrix - np.mean(predictor_matrix, axis=0))/np.std(pre

            # adds constant term 1s to predictor matrix
            predictor_matrix = np.c_[np.ones(len(predictor_matrix)), predictor_matrix]

            # convert response df to ndarray
            response_matrix = response_df.as_matrix()

            # fit model by pinv:
```

```
        # calculate psuedo-inverse
        predictor_inverse = np.linalg.pinv(predictor_matrix)

        # multiply pseudo-inverse by response matrix
        beta = np.dot(predictor_inverse, response_matrix)

        # return weights
        return beta
```

```
In [5]: def generate_columns(predictors):
            # return all possible subset of predictors - n choose k
            n = len(predictors)

            subsets = []

            for k in range(1, n+1):
                x = list(itertools.combinations(predictors, k))
                x = [list(y) for y in x]
                subsets += x
            return subsets
```

```
In [6]: def plot(rss_dict):
            # plots subset by rss
            fig = plt.figure()
            ax = fig.add_subplot(1, 1, 1)
            x = rss_dict[:,0]
            y = rss_dict[:,1]
            ax.scatter(x,y)
            plt.show()
```

1.5 Train Models

```
In [7]: predictors = list(data.columns.values[1:9])
```

```
In [8]: train_data = data[data.train == 'T']
        test_data = data[data.train == 'F']
```

```
In [9]: predictor_subsets = generate_columns(predictors)
        rss_values = {k:[] for k in range(1,9)}
```

```
In [10]: for p in predictor_subsets:
             beta = train(train_data, p)
             rss = evaluateRSS(train_data, p, beta)
             rss_values[len(p)].append(rss)
```

```
In [11]: rss_values
```

```
Out[11]: {1: [44.52858265645385,
           73.613540185775335,
```

```
 91.292039020755482,
 89.624912082459488,
 66.42240272124414,
 73.239391316218757,
 84.991790459378052,
 76.953236677263263],
2: [37.091845632561338,
 44.495642164067419,
 39.992304351212333,
 42.312584301379623,
 44.467567998531834,
 44.424078149255507,
 43.423103787681718,
 73.028097216638173,
 73.305459530268365,
 51.714246792470377,
 56.768034339535888,
 63.059181884423303,
 57.175054366940252,
 87.197409584605992,
 64.044028773554828,
 71.208857866843857,
 83.827432099917743,
 75.824639480492948,
 55.044014020601956,
 64.137962336059871,
 78.888337266541214,
 69.582295055713956,
 64.088802847943214,
 63.294934863258128,
 62.362702781956301,
 71.749951063174379,
 70.603509520730938,
 76.950883206566914],
3: [36.817229416000025,
 36.01516710570592,
 34.907748856567864,
 37.089787447798514,
 36.658492427702328,
 35.434033239986206,
 39.802309202295788,
 42.244606971046714,
 44.43777353564316,
 44.415174658518076,
 43.423016786725107,
 35.96948006886921,
 39.970852869380721,
 39.896830217190455,
```

```
 38.545461464804418,
 41.263782402004438,
 42.267033549979999,
 41.869161533703867,
 44.301168014954449,
 42.562229605787095,
 43.099943647805276,
 72.84486383586156,
 51.548299233644642,
 56.716510713484887,
 62.80914942486244,
 57.06730471494464,
 48.601324135493357,
 55.074650324588212,
 62.835263443329488,
 56.467668934282329,
 49.909275078140865,
 48.10158389631021,
 47.417428583140726,
 54.630336735488498,
 53.570614437348148,
 57.101450639781923,
 54.774354550932962,
 63.867968488321203,
 78.755678004387065,
 69.525536805544945,
 62.20614337508249,
 62.311130030324009,
 61.19660255290141,
 70.572267707969075,
 69.400798784891393,
 75.774282975691193,
 52.778138763661516,
 52.786771795161172,
 51.530655271190874,
 63.206685901646033,
 61.856715635150863,
 69.54806242205423,
 62.448644696209527,
 62.052388626434976,
 62.173911688862894,
 70.571758668498077],
4: [35.464238001473191,
 34.712129622912379,
 36.8142026323154,
 36.049948120539554,
 34.815976423396407,
 32.81499474881555,
```

35.997877720345642,
35.667124547041951,
34.256787963135594,
34.273694721437316,
34.606688894416209,
34.074151863327096,
36.604275512383893,
34.746685114030711,
35.328554300945314,
35.762472367207536,
39.778736027296191,
39.594656768563958,
38.084320496054282,
41.203154292512664,
42.223637529440829,
41.849759194967582,
44.298548108223493,
42.546423826417332,
43.078113811846613,
35.357266631281469,
35.945795880621787,
35.493291436340279,
39.893073071050566,
38.238086420070047,
37.996532935990317,
40.999543720120002,
39.623312100382449,
41.739241310987772,
42.070406824624442,
48.59828096432507,
55.073232862732894,
62.482519006110948,
56.209941872824999,
49.839777901032285,
47.995087329407106,
47.397248632534811,
54.525230142741847,
53.525939875523683,
56.924276004445218,
46.706447196632695,
45.651696333928612,
44.692117214812356,
53.365903236442762,
52.153701516246095,
56.438289966912919,
47.663872765972833,
47.325932341640254,
47.096972715960064,

53.443536473783666,
52.663323877186585,
52.786647025948696,
51.526859567513917,
63.156138242952366,
61.818447405520246,
69.463894400533974,
61.432734534698604,
60.870167682747606,
61.168085348682801,
69.391009945230607,
51.77603907379109,
51.150469708901412,
51.49219322086384,
61.852731001239853,
61.865063596479814],
5: [32.28443746269631,
35.44384366134495,
34.704357000907613,
33.18849557681235,
34.083344003155155,
34.170208530567379,
33.644414586126189,
35.943402221663973,
33.916491474706113,
34.802458627723276,
32.322836795582703,
32.636848329725531,
32.069447332331798,
35.665535924859874,
33.835525242944549,
34.03813696596756,
33.649790928897154,
32.21753338431634,
34.057477254089768,
34.54863253165265,
35.159657902704765,
35.672790024714857,
35.13063768535882,
39.594193686581214,
37.694470290319927,
37.719335854691501,
40.990429981934092,
39.622267422641208,
41.688483685677724,
42.069512088885361,
35.207768745361847,
33.953537981525031,

35.273261065222151,
37.571189956527554,
39.392187991739952,
46.695898087816417,
45.284698737418957,
44.472357035694529,
53.09304326377768,
51.950949986443959,
56.110164622303316,
47.574583175153386,
47.30869098440597,
47.000748312092156,
53.339162003636915,
45.062977653634263,
44.535176559361354,
44.530921666671041,
52.104485796334416,
47.007278081596525,
51.77559675345443,
51.14604362963302,
51.491974557770831,
61.805746825693397,
60.842900645348436,
51.113054423721309],
6: [31.812830659773233,
31.805316901775825,
31.195594615243188,
34.689273250613915,
32.6037506583923,
33.14254428478425,
33.086783141763263,
31.572711025970815,
33.642782767538563,
33.868572946006104,
31.914263328598985,
30.539778129147358,
32.002775193927519,
33.53348157772524,
32.157943183770286,
34.86285919024521,
33.442089040181685,
35.011567475223572,
37.23986333205216,
39.386829537710604,
33.635367125035629,
44.723960117252872,
44.324641253846785,
44.176795979394399,

```
  51.826870621487188,
  46.917990256961076,
  44.377460083432084,
  51.112633574394316],
 7: [30.958629941880631,
  29.437300317417076,
  31.194688345903845,
  32.519818480573058,
  31.570706017488391,
  30.414990170034599,
  33.26543290315886,
  44.03662151270499],
 8: [29.426384459908398]}
```

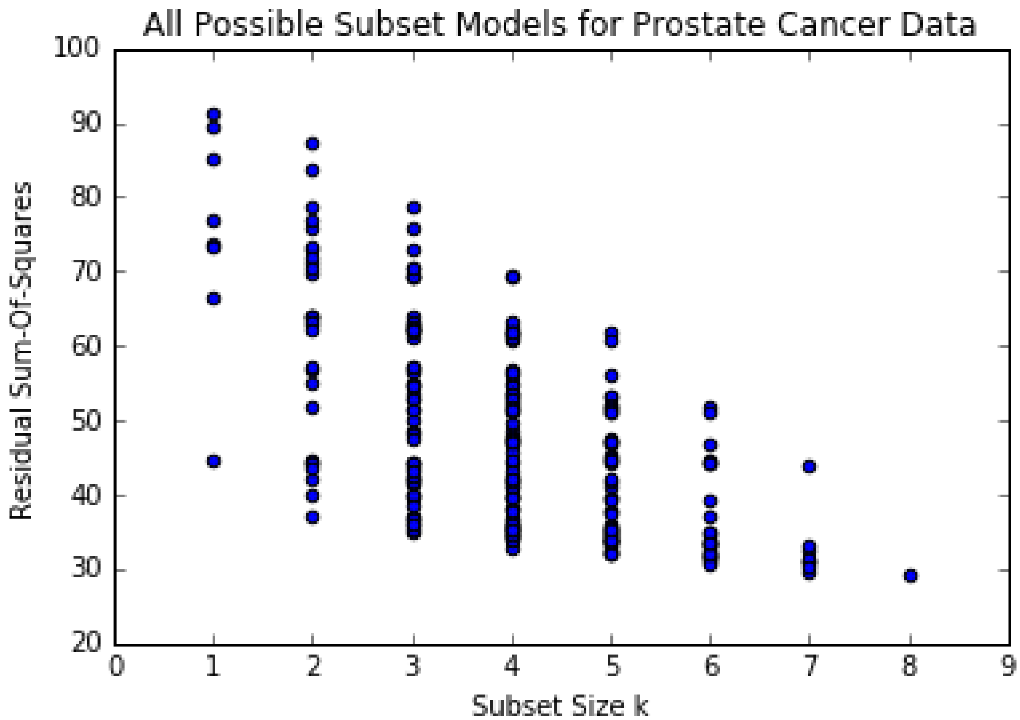
1.6 Plot RSS for Subsets

In [12]:
```
# set x values according to the key and corresponding length of its dictionary
x = [j for j in range(1,9) for k in range(len(rss_values[j]))]

# set y values to k subsets(1-8)
y = []
for k in rss_values.keys():
    y += rss_values[k]

# graph scatter plot
plt.scatter(x,y)
plt.title("All Possible Subset Models for Prostate Cancer Data")
plt.xlabel("Subset Size k")
plt.ylabel("Residual Sum-Of-Squares")
```

Out[12]:
```
<matplotlib.text.Text at 0x7f71a9387450>
```


All Possible Subset Models for Prostate Cancer Data
Residual Sum-Of-Squares
Subset Size k
100
90
80
70
60
50
40
30
20
0
1
2
3
4
5
6
7
8
9