Lecture 2: Linear methods for prediction

Alan Lee

Department of Statistics
STATS 769 Lecture 2

July 23, 2015

Outline

Today's agenda

In this lecture we discuss linear methods for prediction, although many of the concepts covered will be applicable to other methods. Some of the ideas should be familiar from other courses, particularly STATS 330/762. Today we will cover

- Linear models
- Least squares
- Prediction error
- Cross-validation and bootstrap estimates of prediction error
- Subset selection techniques

We will use the California housing data as a running example.

Basic prediction setup

y : The output (response variable, target), the quantity to be predicted, assumed numeric for today's lecture;

$x_1, \ldots, x_k$: The inputs (explanatory variables, features) the variables used for prediction;

$\hat{f}$: the predictor, the mathematical function used to combine the inputs to produce the prediction.

We assume that there is a relationship of the form

$$y = f(x_1, \ldots, x_k) + \text{error}$$

that connects the inputs to the outputs, but f is unknown. The errors are assumed to be unpredictable. We select our predictor $\hat{f}$ from some class of predictors. In today's lecture we assume this class is the class of *linear functions*.

Linear predictors

This is the simplest prediction method, but is often surprisingly competitive with the more sophisticated methods we will start to discuss next week. Linear predictors are linear functions of the form

$$f(x_1, \ldots, x_k) = b_0 + b_1 x_1 + \cdots + b_k x_k$$

so that the predicted output is a linear combination of the inputs.

We need to choose the best predictor in this class. we do this by choosing the b's to minimize the sum of squared errors

$$(y - b_0 - b_1 x_1 - \cdots - b_k x_k)^2.$$

Choosing the b's

Suppose we have rectangular set of data where columns correspond to variables and rows to cases, which we can represent by

$$(x_{i1}, \ldots, x_{ik}, y_i), \ i = 1, 2, \ldots, n$$

where the subscript i indexes the cases, $x_{i1}, \ldots, x_{ik}$ are the values of the inputs for the ith case, and y_i is the output for the ith case. Our problem is to minimize the sum of squared errors

$$\sum_{i=1}^{n} (y_i - b_0 - b_1 x_{i1} - \cdots - b_k x_{ik})^2$$

as a function of the coefficients $b_0, b_1, \ldots, b_k$. This is a standard mathematical problem and very robust computer algorithms exist for solving it. See e.g. STATS 310 and STATS 760 for more details.

Computing the predictor

If $\hat{b}_0$, $\hat{b}_1$,..., $\hat{b}_k$ are the minimizing values, then the predictor for new data $x_1, \ldots, x_k$ is

$$\hat{y} = \hat{b}_0 + \hat{b}_1 x_1 + \cdots + \hat{b}_k x_k.$$

The $\hat{b}$'s (and many other things) are computed by the R function `lm`. A generic function call (for $k = 4$) is

```
my.model <- lm(y~x1+x2+x3+x4, data=data.df)
```

where the data is in a data frame `data.df`, containing inputs `x1`, `x2`, `x3`, `x4` and output `y`. The coefficients $\hat{b}_0, \hat{b}_1, \ldots, \hat{b}_k$ are computed by

```
coefficients(my.model)
```

The California data

The California dataset contains information on 20,640 California "block groups". We will use 10,000 for this example, holding the rest back for future use. The variables are

medval	:	The median value of houses in the block group (the output);
medinc	:	The median income of residents of the block group;
medage	:	The median age of residents of the block group;
rooms	:	The total number of rooms in houses in the block group;
bedrooms	:	The total number of bedrooms in houses in the block group;
pop	:	The population of the block group;
house	:	The number of households in the block group;
lat	:	The latitude of the block group;
long	:	The longitude of the block group.

The 10,000 block groups are in the data frame `california.df`.

Fitting the linear model

We want to develop a predictor to predict the log of the median value. To fit the model (using log(medval) as the output and the other variables as inputs), and calculate the coefficients, type

```
> california <- lm(log(medval)~ medinc + medage + rooms +
      bedrooms + pop + house + lat + long, data = california.df)
> coefficients(california)
  (Intercept)        medinc        medage         rooms      bedrooms
-1.092281e+01  1.792134e-01  3.362244e-03 -3.611447e-05  5.207509e-04
          pop         house           lat          long
-1.557328e-04  1.872739e-04 -2.724780e-01 -2.665619e-01
```

Making predictions

In the example, we held out 10,640 observations. We will use these data to make some predictions. The new data is in the data frame `newCalifornia.df`. To predict the log median income for these 10,640 block groups, we type

```
> predictions <- predict(california, newdata = newCalifornia.df)
> head(predictions)
       1        2        3        4        5        6
12.97626 13.05747 12.82574 12.29892 12.27339 12.27443
> log(newCalifornia.df$medval[1:6])
[1] 13.02276 12.78968 12.77167 12.50507 12.60887 12.39421
```

Prediction error

How well have we predicted the house prices? In other words, what sort of error do we expect? One measure of this error is the **expected squared error**, the average squared error over a large number of future predictions. If we have some new data (a "test set") where we know the actual output (as we did for the California data) , we can calulate the average squared error directly. For the California data we get

```
> actuals <- log(newCalifornia.df$medval)
> mean((predictions - actuals)^2)
[1] 0.1177588
```

Thus, our estimate of the expected squared error is about 0.118. Note that this is measured on the log scale. This estimate is known as the **test set estimate** or the **out-of-sample** estimate.

Why not use the original data?

Could we use the training set (the data used to fit the model) to estimate the expected squared error? This is called the **in-sample** estimate or the **apparent error**. To do this, we could type

```
> mean((predict(california) - log(california.df$medval))^2)
[1] 0.1135258
```

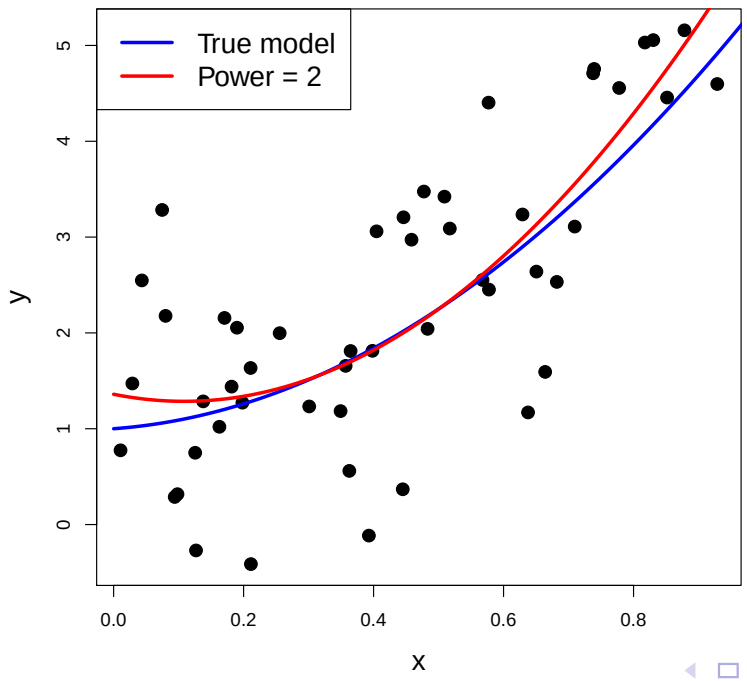
The in-sample estimate is a bit smaller, which is typical: in most cases it **underestimates** the expected squared error:

Example: polynomial regression

Suppose we have some data (50 data points) which follow a quadratic model

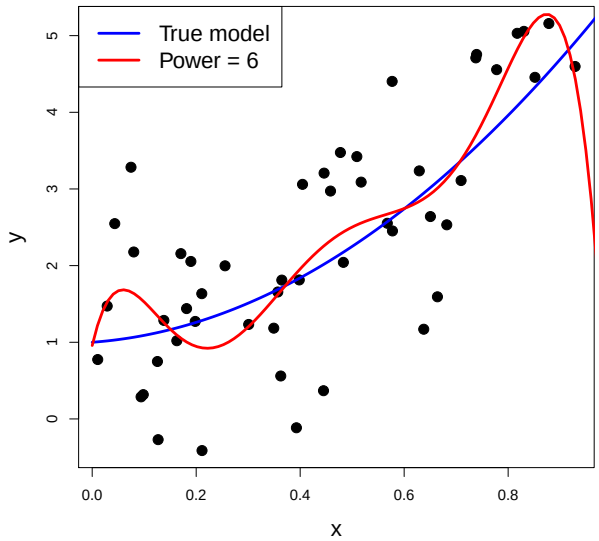
$$Y = 1 + 0.5x + 4x^2 + N(0, 1)$$

where the x's are uniform on [0,1].



Under/over fitting

Suppose we fit a 6 degree polynomial (shown as a red solid line). This is a linear model with variables x, x^2, ..., x^6. The 6 degree polynomial will not predict new data well.



The in-sample error estimate is 0.803, but the out-of sample error (based on a test set of 200) is 3.768. The true expected squared error is about 4.30.

Methods for estimating PE

We discuss three:

- The test set estimate (requires a new data set with inputs and outputs)
- Cross validation (does not require a new data set, but uses the test set estimate)
- The Bootstrap (uses repeated bootstrap samples)

The test set estimate

As we have seen, this is simple but requires a new data set containing both inputs and outputs.

Would we be better combining the test and training sets and using the combined set to fit the model? A bigger sample size will give a better fit i.e. less prediction error.

We would then have to use another method (such as CV or bootstrap) to actually estimate the prediction error.

Cross-validation

In cross validation, we split the data randomly into 10 parts, use one part for the test set and the remaining 9 parts for the training set, and estimate the PE with the test set estimator.

We repeat this process using a different tenth as the test set each time. We average the resulting 10 PE estimates to get a final estimate. We can repeat for different random splits and average.

This is (10-fold) cross-validation. (Could also split into 5 parts - 5-fold CV).

Cross-validation in R

There are several options. In the R330 package, there is a function `cross.val` that will calculate a cross-validation estimate of prediction error. In the package `bootstrap`, there is a function `crossval` that can be used with linear models (as well as many others). The package `caret` which we will look at next week, is even more comprehensive.
For the California data, we have

```
> cross.val(california, nfold = 10, nrep = 20)
Cross-validated estimate of root
mean square prediction error =  0.3383909
> 0.3383909^2
[1] 0.1145084
```

Compare with in-sample error of 0.1135 and a test set error of 0.1177. (Since the sample is so big, there is not much difference between these.)

The Bootstrap

The basic idea here is to mimic new data by resampling the training set, i.e. taking repeated bootstrap samples from the training set.

Suppose the training set contains n cases. To take a bootstrap sample, we take a random sample of size n with replacement from the training set. Thus, some of the cases in the training set will be duplicated in the bootstrap sample, and others won't be included at all.

The probability a given case won't be included in the bootstrap sample is $(1 - \frac{1}{n})^n$ which for large n is approximately $e^{-1} = 0.368$ (The probability that a case *will* be included is thus about 0.632.)

Bootstrap approaches

A possible bootstrap approach might be to fit the model to a bootstrap sample, and use the original sample as a test set. This doesn't work too well as the test and training sets are too similar.

An alternative is to use the bootstrap sample as both test and training set. This really will be an underestimate. If we take the difference between the two estimates (called the optimism), we get a (possibly too small) estimate of the amount by which the in-sample error underestimates PE.

We could average the optimisms over B bootstrap samples and add the result to the in-sample error to get a corrected PE estimate. This turns out to work quite well.

The .632 estimator

The percentage of the original data that appears in a bootstrap sample is about 63.2%, so if we use the bootstrap sample as a training set and the original sample as a test set there will be considerable overlap.

A bit of mathematics shows that the "in-sample + optimism" estimator involves, for each data point (x, y) in the original data, averaging the squared errors $(y - \hat{f}_b(x))^2$ over the bootstrap samples. (Here $\hat{f}_b$ is the predictor calculated from the bth bootstrap sample). If the bth sample contains (x, y), this error will be on average too small. Thus, to avoid this, we only average over the "out-of-bag" samples, those not containg (x, y). This results in the "0.632 estimate" of PE.

Summary

Let $PE(\text{test}, \text{training})$ denote the estimate of prediction error obtained by calculating the test set error when using the data set "test" as the test set and the data set "training" as the training set. The different estimates of PE are

- In-sample: $PE = PE(\text{training}, \text{training}) = \overline{err}$
- Test set estimate $PE = PE(\text{test}, \text{training})$
- $\overline{\text{err}}$ + opt: $PE = \overline{err} + \overline{PE(data, bs) - PE(bs, bs)}$
  bs=bootstrap sample
- 0.632: Modified $\overline{\text{err}}$ + opt

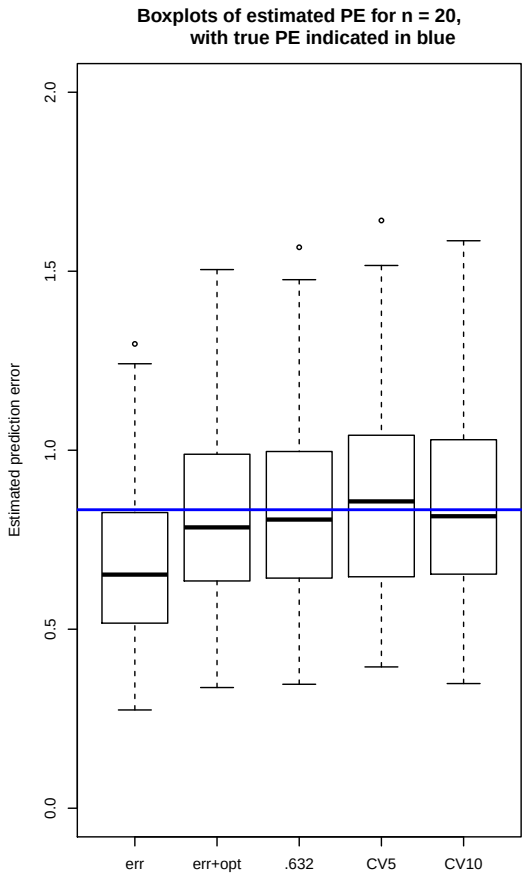
An example

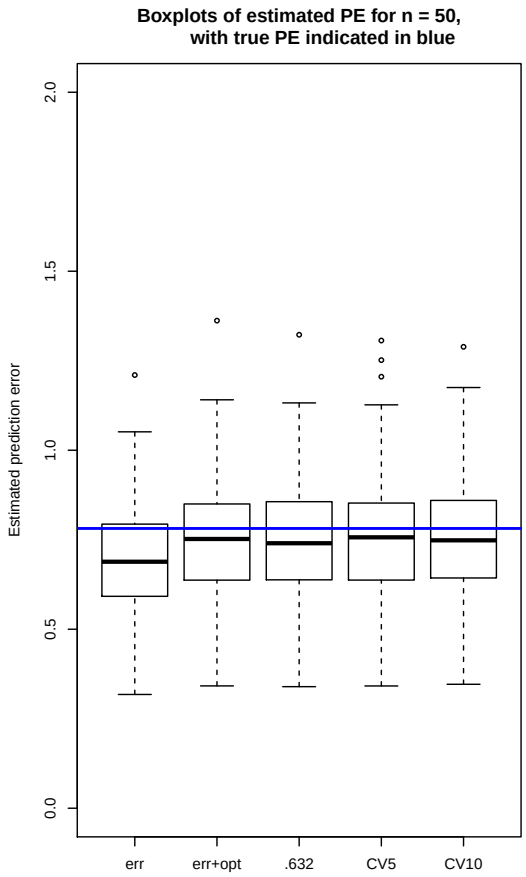
To compare the various estimators, we conducted a small simulation, using data pairs $(x_i, y_i), i = 1, \ldots, n$ sampled from a bivariate normal distribution with means 0, variances 1 and correlation ρ. We want to evaluate the prediction error when using least squares regression as a predictor. Consider the case where ρ is 0.5 and B is 100. We got the following estimates (averaged over 100 replications):

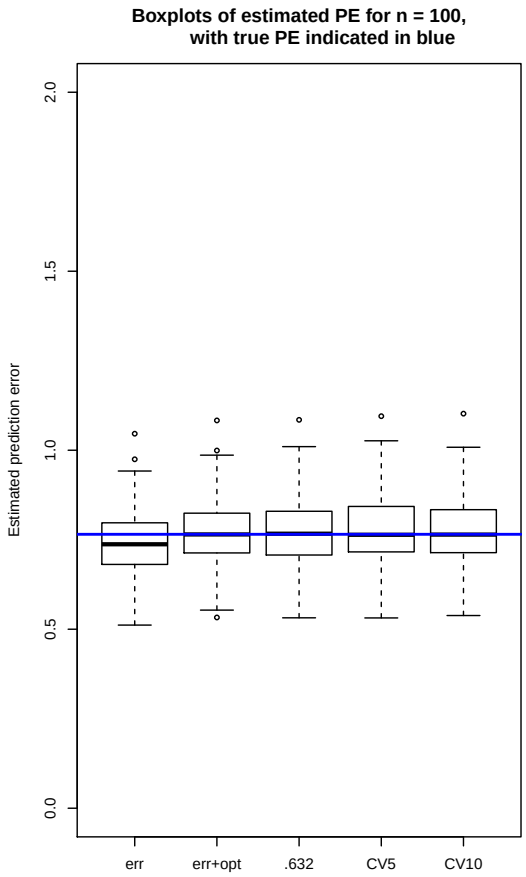
	err	err+opt	.632	CV5	CV10	True PE
$n = 20$	0.675	0.813	0.829	0.860	0.844	0.834
$n = 50$	0.698	0.754	0.755	0.764	0.761	0.781
$n = 100$	0.742	0.771	0.771	0.774	0.773	0.765

The opt+err method looks good. The apparent error $\overline{err}$ underestimates.

Boxplots of 100 reps







Bootstrap: the California data

The R 330 package has a function `err.boot` that will calculate a bootstrap estimate of PE. The package `bootstrap` has a function `bootpred` that can be used with linear models (as well as many others). Ditto the package `caret` .

```
> err.boot(california, B=200)
$err
[1] 0.1135258

$Err
[1] 0.1143676
```

NB: err is the in-sample estimate, ERR the err + opt estimate.

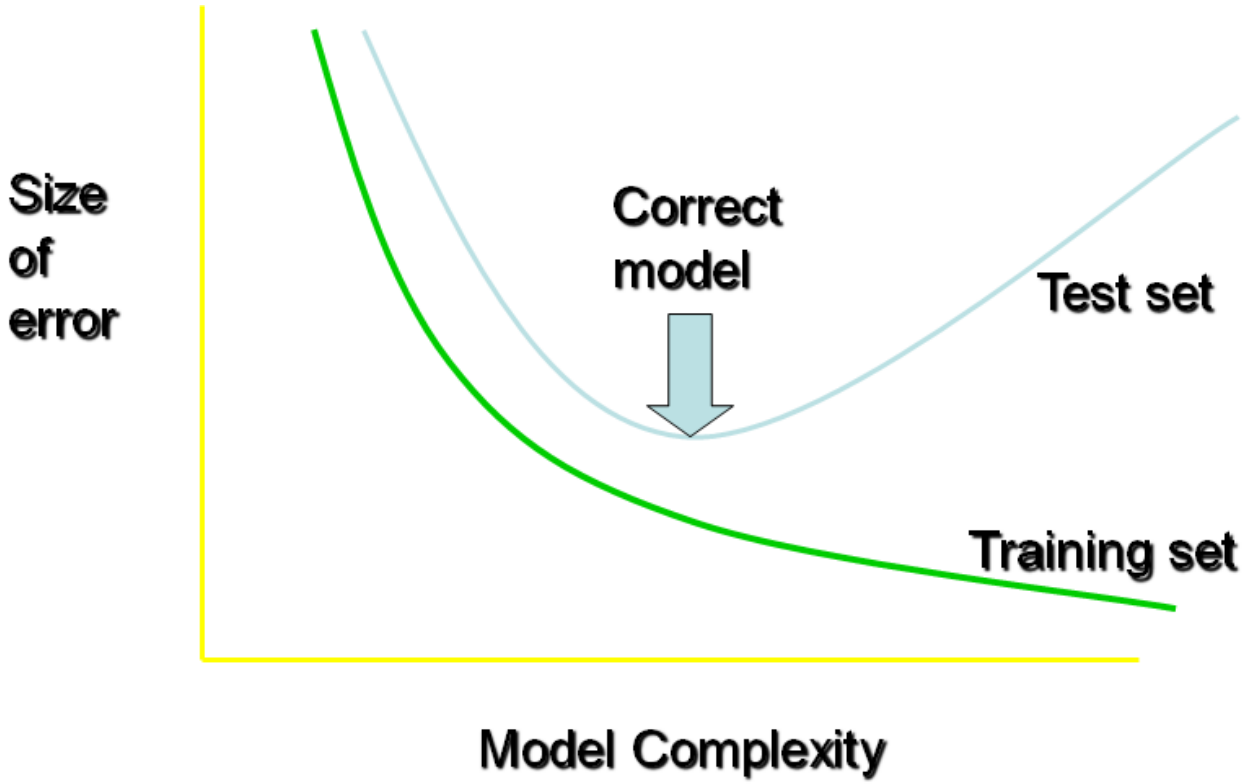
Variable selection

Should we use all the variables available, or just some of them? Of course, we should include all variables we think might be important predictors, identified by conversations with subject matter experts. But using all available variables might overfit and lead to poor predictions, as we saw in the polynomial example.

Model complexity

When fitting models to a training set, we do not want to use a model that has too many variables. Such a model will capture not only the nature of the true function f but will also model the errors, as in the polynomial example. Future data will have the same f but different errors, which will not be well predicted by a too-complex model. Thus, a too-complex model will have a small training set error, but a large PE (test set error, the error expected when predicting future data.)

Model complexity(cont)



As we add more variables, the in-sample error goes down, but after a certain point, the true PE stars to rise.

Methods for variable selection

In linear prediction (and other kinds a well), we might want to identify a subset of variables that predicts well. There are several approaches possible:

- All possible subsets: If the number of variables is not too great, we could examine all possible subsets of variables (If there are k variables there will be 2^k possible subsets).
- We could add variables one at a time, choosing the variable that gives the best improvement in the estimated PE. This is called **forward selection**.
- We could start with all the variables, then delete variables one at a time, choosing the variable that gives the best improvement in the estimated PE. This is called **backward elimination**.
- A combination of the last two: **stepwise regression**.

Measuring PE inprovement

How do we measure the PE improvement? Since there are a lot of comparisons, CV or the bootstrap may be too computationally intensive.

Using the in-sample error is no good, since adding a variable will **always decrease** the in-sample error.

One quick way is to use a penalized form of the in-sample error, the AIC, which is proportional to the in-sample error plus the quantity $2p\hat{\sigma}^2/n$, where p is the number of estimated coefficients in the predictor, and $\hat{\sigma}^2$ is an estimate of the error variance. This is the method used in R.

Example: the California data, forward selection

```
> null.model = lm(log(medval)~1, data=california.df)
> selected = step(null.model, scope = formula(california),
        direction = "forward", trace=0)
> selected

Call:
lm(formula = log(medval) ~ medinc + medage + house + pop + lat +
    long + bedrooms + rooms, data = california.df)

Coefficients:
(Intercept)       medinc       medage        house          pop          lat
 -1.180e+01    1.782e-01    3.261e-03    2.493e-04   -1.725e-04   -2.801e-01
       long     bedrooms        rooms
 -2.762e-01    4.798e-04   -3.186e-05
```

Example: the California data, backward elimination

```
> null.model = lm(log(medval)~1, data=california.df)
> selected = step(california, scope = formula(california),
        direction = "backward", trace=0)
> selected

lm(formula = log(medval) ~ medinc + medage + rooms + bedrooms +
    pop + house + lat + long, data = california.df)

Coefficients:
(Intercept)       medinc       medage        rooms     bedrooms          pop
 -1.180e+01    1.782e-01    3.261e-03   -3.186e-05    4.798e-04   -1.725e-04
      house          lat         long
  2.493e-04   -2.801e-01   -2.762e-01
```

Example: the California data

```
> null.model = lm(log(medval)~1, data=california.df)
> selected = step(california, scope = formula(california),
        direction = "both", trace=0)
> selected

lm(formula = log(medval) ~ medinc + medage + rooms + bedrooms +
    pop + house + lat + long, data = california.df)

Coefficients:
(Intercept)       medinc       medage        rooms     bedrooms
 -1.180e+01    1.782e-01    3.261e-03   -3.186e-05    4.798e-04
       pop        house          lat         long
-1.725e-04   2.493e-04   -2.801e-01   -2.762e-01
```

Example: the California data, all possible subsets

```
> allpossregs(california)  # NB in the R330 package
     rssp sigma2 adjRsq        Cp      AIC      BIC      CV
1 3786.996  0.183  0.434 12114.795 32754.79 32770.67 378.799
2 3625.369  0.176  0.458 10719.011 31359.01 31382.82 362.689
3 2647.389  0.128  0.604  2263.218 22903.22 22934.96 264.871
4 2602.578  0.126  0.611  1877.681 22517.68 22557.36 260.422
5 2430.030  0.118  0.636   387.445 21027.44 21075.05 243.534
6 2399.491  0.116  0.641   125.337 20765.34 20820.88 240.492
7 2390.896  0.116  0.642    53.005 20693.01 20756.49 239.658
8 2385.576  0.116  0.643     9.000 20649.00 20720.42 239.302 <------
  medinc medage rooms bedrooms pop house lat long
1      1      0     0        0   0     0   0    0
2      1      1     0        0   0     0   0    0
3      1      0     0        0   0     0   1    1
4      1      0     0        1   0     0   1    1
5      1      0     0        1   1     0   1    1
6      1      1     0        1   1     0   1    1
7      1      1     1        1   1     0   1    1
8      1      1     1        1   1     1   1    1 <-----
```