Statistical Methods for ML Unimi

Experimental Project: Housing prices

Anton Shchedro

July 2020

Abstract

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

1 Introduction

This project is based on finding house price (medianHouseValue) based on several features of house:

- *longitude*.
- *latitude*.
- *housingMedianAge* — Median age of a house.
- *totalRooms* — Total number of rooms.
- *totalBedrooms* — Total number of bedrooms.
- *population* — Total number of people residing.
- *households* — Total number of households, a group of people residing within a home unit.
- *medianIncome* — Median income for households (measured in tens of thousands of US Dollars).
- *oceanProximity* — Location of the house w.r.t ocean/sea.

After making algorithm based on Ridge Regression (also known as L2 regularization) it's necessary to study the dependence of the Cross-Validated risk estimate on the parameter α of ridge regression. Then repeat Cross-validation while using PCA.

All work is made in *Jupyter Notebook* for easy check of results of algorithm, while using python libraries *pandas* for data manipulation, *numpy* for matrix and linear algebra and *matplotlib.pyplot* for graphs.

2 Preparation of data

Before using data in algorithm it's necessary to check if data is ready to use, in other words that data has no missing (*NaN*) values and if data has some categorical values, they have to be transformed.

To load data:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('cal-housing.csv')
```

2.1 Categorical values

Categorical values are values that mostly described by words or letters or other not numerical symbols. They may be ordinal if I can easily put them in some order (as example: *sad*, *content*, *happy*), otherwise they are nominal (as example: *Ford*, *Volvo* and other car manufacture).

In case of ordinal value I can transform them into range of numbers starting from zero in same order as our data.

In case of nominal data I can use dummy coding (also knows as one-hot encoding [2]): make for each possible categorical value new column, fill column of particular value with 1 when data record have same categorical value, zero otherwise.

Example of dummy coding [5]:

Nationality	C1	C2	C3
French	0	0	0
Italian	1	0	0
German	0	1	0
Other	0	0	1

In this project exists only one feature with categorical values — *oceanProximity*. This feature have this variables: *NEAR BAY*, *<1H OCEAN*, *INLAND*, *NEAR OCEAN*, *ISLAND* This variables looks like ordinal data, but may be not clear the difference between some of them (*NEAR BAY*, *<1H OCEAN*, *NEAR OCEAN*), so one of possible transformation as ordinal data:

- INLAND → 0
- <1H OCEAN → 1
- NEAR BAY → 2
- NEAR OCEAN → 3
- ISLAND → 4

This transformation is made by function:

```
def ordDataTransform(dt):
    data = dt.copy()
    data = data.replace('INLAND',0)
    data = data.replace('<1H_OCEAN',1)
    data = data.replace('NEAR_BAY',2)
    data = data.replace('NEAR_OCEAN',3)
    data = data.replace('ISLAND',4)
    return data
```

OceanProximity also can be transformed as nominal data by using:

```
import pandas as pd
def nomDataTransform(data):
    return pd.get_dummies(data,['ocean_proximity'])
```

2.2 Missing values

To find if it's necessary to manage missing (*NaN*) values and in which feature I can find them, I can use:

```
dfNan = df[df.isna().any(axis=1)]
dfNan
```

This will show to us a table of entries with missing values. By checking this table and/or checking columns one by one with `df['column_name'].isnull().sum()` I can find that only one column has missing values: *total_bedrooms*.

I can use different methods to deal with *NaN* values:

- Row elimination. → function: *elimdf*
- Apply to NaN value of mean value of the column. → function: *meandf*
- Apply to NaN value of mode value of the column. → function: *modedf*

I can use row elimination without big risk because rows with *NaN* value are about 1% from all data.

```
def elimdf(data):
    eldf = data.dropna()
    return eldf

def meandf(data):
    meandf = data.fillna(round(df.mean()))
    return meandf

def modedf(data):
    mode = df['total_bedrooms'].mode()
    modedf = data.copy()
    modedf['total_bedrooms'] = (data['total_bedrooms']
                                .fillna(mode[0]))
    return modedf
```

2.3 Splitting the data and other different functions

Function *dataPreparetionColumn* is used to transform data with method chosen for dealing with *categorical* values and *NaN* values, which are described above. Also this function can normalize data if it's needed (by using function *normalization*). Resulting data is shuffled.

Function *splitData* used to randomly split data to training and test set by 80% and 20% respectively.

CrossSplitTrain used to take i fold from k for Cross-Validation, but I choose data by using pre-shuffled data with *dataPreparetionColumn* and taking by index from $(n * (i-1))$ to $(n * i)$ where $n := length(data)/k$.

SplitAnswer separate predicted label from our data. It also can add some integer as new feature to every data point if necessary (as example: add 1 to easily calculate w_0 for bias). This function returns 2 tables: X - our *pandas* table of labels that can be transformed to $m \times d$ design matrix with `to_numpy()` of pandas library, y - correct label also can be transformed to $m \times 1$ matrix with `to_numpy()`.

3 Ridge Regression

Ridge regression is a modification of Linear regression to deal overfitting problem that may occur by adding a regularization parameter α. As Linear regression, Ridge regression used to resolve regression problem by finding model that predict numerical values based on one or multiple features.

3.1 Loss Function

Loss (cost) Function for Ridge Regression is: $||Xw - y||^2 + \alpha * ||w||^2$ [3], where α is regularization parameter. This loss function is used to find optimal vector w with gradient decent, but I'll use closed form solution.

3.2 Algorithm

To find optimal vector w it's possible to use closed form solution of Ridge regression with this formula: $\hat{w}_\alpha = (\alpha I + X^T X)^{-1} X^T y$, where $\hat{w}_\alpha$ is optimal vector with regularization parameter α as hyperparameter [3].

```
def rr(dfX, dfY, alpha):
    #create identity matrix
    ind = np.identity(dfX.T.shape[0])

    inv = np.linalg.inv(np.dot(alpha, ind) +
        np.dot(dfX.T.to_numpy(), dfX.to_numpy()))
    a = np.dot(dfX.T.to_numpy(), dfY.to_numpy())
    w = np.dot(inv, a)
    return w
```

4 Study the dependence of the cross-validated risk estimate on the parameter α of ridge regression.

By Cross-Validation I can check how changes our expected risk with a change of hyperparameters. In case of ridge regression using closed form solution, only one hyperparameter exists: regularization parameter α. For Cross-Validation to calculate error I'll use quadratic loss: $l(y, \hat{y}) = (y - \hat{y})^2$ [2] where $\hat{y} = Xw$.

```
def cv_error(x,y,w):
    a = y - np.dot(x,w)
    cost = np.dot(a.T,a)
    return cost
```

4.1 Choosing methods for data preparation

I can choose method to deal with categorical and *NaN* values by applying different versions of these methods to our data-set, taking same parameter α and using external *K-fold* cross-validation. In code below and results with $\alpha = 1e-3$ [1].

Code:

```
modelCVData = df.copy()
modelCVData = modelCVData.sample(frac=1).
                    reset_index(drop=True)

def choose_method(catDataMeth, nanMeth, alpha, i, k):
    data = nanMeth(catDataMeth(modelCVData))
    test, train = crossSplitTrain(data, i, k)
```

```
    dfX,dfY = splitAnswer(train,'median_house_value',
                                                    True)
    testX,testY = splitAnswer(test,'median_house_value',
                                                    True)
    w = rr(dfX,dfY,alpha)

    error = cv_error(testX.to_numpy(),
                        testY.to_numpy(),w)
    return error
```

Then for each method (in different cell for each method) use this code (placing functions for each method):

```
error = 0
k = 5
for i in range(1,k+1): #5 folds
    error += (k/len(df))*choose_method(
        categoricalFunction,NanFunction,1e-3,i,k)
print(error/k)
```

Results:

	ordinal	nominal
elim. row	4767969480.73	4686246221.19
mean	4826612043.44	4743135286.17
mode	4829632135.13	4745361978.81

I can see that best method (with less error) is transformation of nominal data for categorical values and row elimination for missing values. These methods are chose as principal methods.

4.2 Dependence of the CV on the parameter α

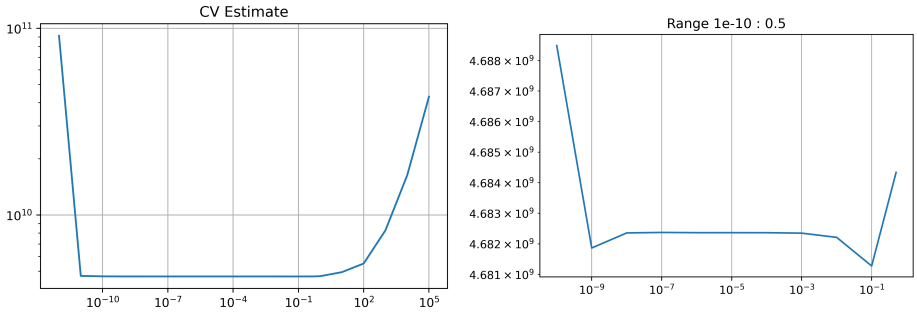
In this subsection I will check α in range from 1e-12 to 1e+5, so in case of 1e-12 α can be think of being approximated to 0, and 1e+5 is big enough to have big error.

I will use *K-fold* cross-validation [1] with this code:

```
data = dataPreparetionColumn(df,nomDataTransform,elimdf,
                                True,'median_house_value')
alphaRange = [10**i for i in range(-12,5)] + [0.5]
kMax = 10 #divide data for 10 folds
estimateCV = []
for alpha in alphaRange:
    costAlpha = 0
    for k in range(1,kMax+1):
        test,train = crossSplitTrain(data,k,kMax)
```

```
            dfX,dfY = splitAnswer(train,
                                  'median_house_value',True,1)
            testX,testY = splitAnswer(test,
                                  'median_house_value',True,1)
            w = rr(dfX,dfY,alpha)
            costAlpha += (kMax/len(df)) * cv_error(
                    testX.to_numpy(),testY.to_numpy(),w)
        estimateCV.append((1/kMax)*costAlpha)
plt.plot(alphaRange,estimateCV)
plt.xscale('log')
plt.yscale('log')
```

Resulting plot:



Resulting Cross-Validated estimates of parameters α is 0.1, but it can show other values (as example $1e-9$), because values in range from $1e-10$ to 0.5 have approximately the same values and the lowest one may be different depends on the different combinations of shuffled data.

5 PCA

Some labels may be correlated with each other. I can use PCA to resolve this problem and to try to improve our algorithm.

To do this I'll use this pseudocode [4]:

Because I have $m >> d$ I can transform pseudocode in following pseudocode:

With resulting code:

```
def pca(data, n):
    #calc of cov matrix and eigenvectors and eigenvalues
    A = np.cov(data, rowvar=False)
    eva, evec = np.linalg.eigh(A)
    #order
    i = np.argsort(eva)[::-1]
    evec = evec[:,i]
```

Input: Matrix of m examples $X \in R^{m,d}$
number of components n
if $m > d$ **then**
$A = X^T X$ Let $u_1, ..., u_n$ be the eigenvectors of A with largest eigenvalues
else
$B = XX^T$;
Let $v_1, ..., v_n$ be the eigenvectors of B with largest eigenvalues for $i = 1, ..., n$ set $u_i = \frac{1}{||X^T v_i||} X^T v_i$
Output: $u_1, ..., u_n$

Input: *data*, number of components n
A = covariance matrix of X (with *numpy cov*)
find eigenvectors and eigenvalues of A (with *numpy linalg.eigh(A)*)
place eigenvalues in decreasing order
U = matrix of $u_1 ... u_n$
compression matrix $W = U^T$
Output: $data_{pca} = data * W$

```
    eva = eva[i]
    #choose n
    evec = evec[:,:n]
    #pca data
    D = np.dot(data, evec)
    return D
```

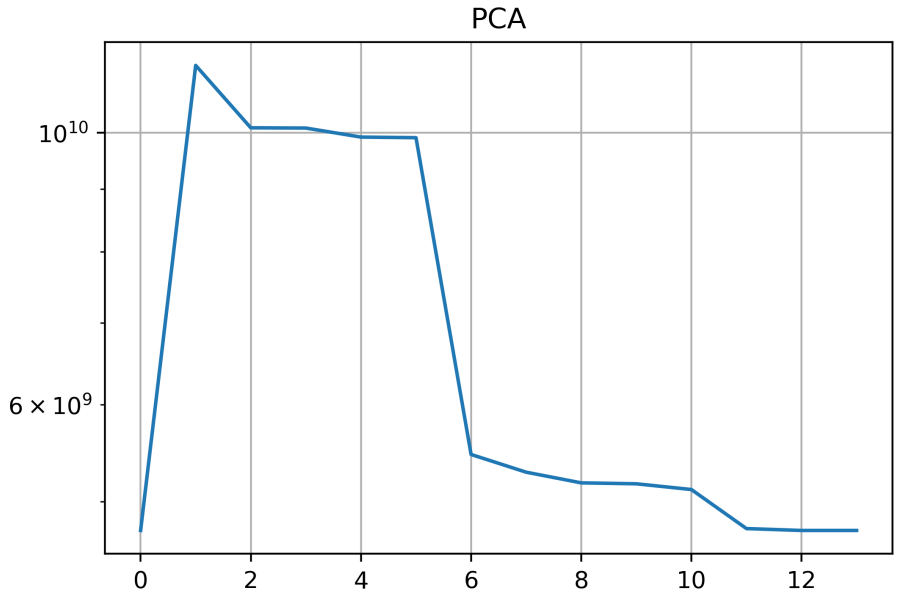
Preparation of data with PCA is used in this way:

```
data = dataPreparetionColumn(df,nomDataTransform,elimdf
                             ,True,'median_house_value')
dataX,dataY = splitAnswer(data,
                             'median_house_value',False)
pca_data = pca(dataX,n)
pca_data = pd.DataFrame(pca_data)
pca_data.insert(1, 'median_house_value', dataY, True)
```

where n is number of components needed.

It's possible to use cross-validation to find optimal number of components. To do it it's necessary to fix regularization parameter α. In a case below α is fixed to 0.1.

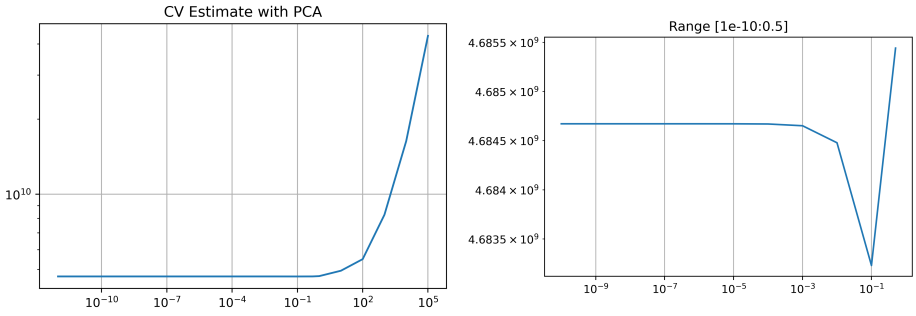
In case of 0 numbers of components were not used PCA algorithm, and this value used for reference of cross-validation estimator with and without PCA. This cross-validation shows best result both with number of components of 12, 13 and without PCA, depending on data shuffling. From this moment all future work with data will be used with PCA parameter 12, because it's a lowest acceptable parameter for PCA.



5.1 Study the dependence of the cross-validated risk estimate on the parameter α of ridge regression after using of PCA

Now it's possible to use cross-validation to check how change risk estimator with different parameters α, as it's been done previously, but with PCA. I will set it to 12 as it was mentioned before. I also will use the same range for α as it was done before.

The results are same, best parameter α is 0.1, but results are more stable (does not change so often as without PCA):



6 Choosing regularization parameter α with Nested Cross-Validation

Nested cross-validation is a version of cross-validation with internal and external cross-validation [1]. It's used with this pseudocode:

Input: *data*, range of α
kFold = 5 used for external CV lfold = 10 used for internal CV
for *k from 0 to kFold* **do**
  test = fold k of data
  train = data - test
  for *l from 0 to lFold* **do**
    intTest = fold l of train
    intTrain = train - intTest
    for α *in range of* α **do**
      train ridge regression with intTrain and α
      error = calculate MSE of ridge regression on intTest
    end
  end
  calculate CV estimate for each α and internal folds
  find α_{best} with best CV estimate
  train ridge regression with train and α_{best}
  error = calculate MSE of ridge regression on test
end
choose α_{best} from each kFold with lowest error
Output: best α

Realization in *python*:

```
data = dataPreparetionColumn(df,nomDataTransform,elimdf,
                                    True,'median_house_value')
dataX,dataY = splitAnswer(data,'median_house_value',
                                                      False)
pca_data = pca(dataX,12)
pca_data = pd.DataFrame(pca_data)
pca_data.insert(1, 'median_house_value', dataY, True)
alphaRange = [10**i for i in range(-12,5)] + [0.5]
kMax = 5 #divide data for 5 external cv folds
lMax = 10 #divide data for 10 internal cv folds
alphaParam = []
costParam = []
for k in range(1,kMax+1):
    test,train = crossSplitTrain(pca_data,k,kMax)
    intCVEstimate = []
    for l in range(1,lMax+1):
```

```
            ctest,ctrain = crossSplitTrain(train,l,lMax)
            dfX,dfY = splitAnswer(ctrain,
                                  'median_house_value',True,1)
            testX,testY = splitAnswer(ctest,
                                  'median_house_value',True,1)
            intCostParam = []
            for alpha in alphaRange:
                w = rr(dfX,dfY,alpha)
                intCostParam.append((lMax/len(train)) *
        cv_error(testX.to_numpy(),testY.to_numpy(),w))
            intCVEstimate.append(intCostParam)
        matrix = np.matrix(intCVEstimate)
        matrix = matrix.sum(axis = 0)/lMax
        dfX,dfY = splitAnswer(train,'median_house_value',
                                                    True,1)
        testX,testY = splitAnswer(test,'median_house_value',
                                                    True,1)
        matrix = matrix.A1
        #alpha = matrix.argmin() + 1
        alpha = alphaRange[matrix.argmin()]
        w = rr(dfX,dfY,alpha)
        cost = (kMax/len(data))*cv_error(testX.to_numpy(),
                                          testY.to_numpy(),w)
        alphaParam.append(alpha)
        costParam.append(cost)
costParam = np.array(costParam)
param = alphaParam[costParam.argmin()]
print("best alpha;   loss")
print(param, '   ', costParam.min())
```

Resulting best parameter α is *0.1*, but this value can be changed because of data shuffling.

7 Check results with Sklearn RidgeCV

It's also possible to use *SKLearn* library for python to solve same problem, and to check if my algorithm is working properly it's possible to check if result of my algorithm and solution made by *sklearn* are similar. To check if ridge parameter α is chose correctly I can use *RidgeCV* from *sklearn.linear_model* and compare their and my α and resulting cross-validation estimate.

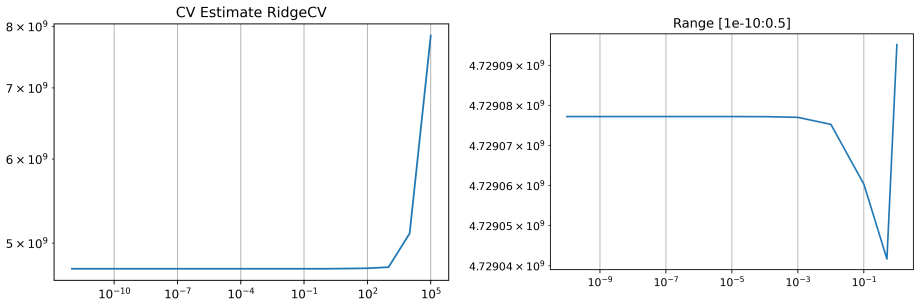
```
from sklearn.linear_model import RidgeCV

data = dataPreparetionColumn(df,nomDataTransform,elimdf,
                             False,'median_house_value')
alphaRange = [10**i for i in range(-12,5)] + [0.5]
```

```
dataX,dataY = splitAnswer(data,'median_house_value',0)
clf = RidgeCV(alphas=alphaRange, store_cv_values =
                                        True).fit(dataX,dataY)
#clf = RidgeCV(alphas=alphaRange, cv = 10).fit(
                                                dataX,dataY)
cvv = clf.cv_values_.sum(axis = 0)/
                                        clf.cv_values_.shape[0]
print(clf.alpha_, '   ', np.min(cvv))
```

Where *clf.alpha_* is best regularization parameter α and *clf.cv_values_* are Cross-Validation estimates of different parameters α.

Resulting Cross-Validation estimates are similar, and best α is not much different (*0.1 in my algorithm and 0.5 in RidgeCV*):



References

[1] Nicolò Cesa-Bianchi. *Statistical Methods for Machine Learning, Lecture Notes: Hyperparameter tuning and risk estimates*. 2020. URL: `http://cesa-bianchi.di.unimi.it/MSA/Notes/crossVal.pdf`.

[2] Nicolò Cesa-Bianchi. *Statistical Methods for Machine Learning, Lecture Notes: Introduction*. 2020. URL: `http://cesa-bianchi.di.unimi.it/MSA/Notes/intro.pdf`.

[3] Nicolò Cesa-Bianchi. *Statistical Methods for Machine Learning, Lecture Notes: Linear prediction*. 2020. URL: `http://cesa-bianchi.di.unimi.it/MSA/Notes/linear.pdf`.

[4] Shai Ben-David Shai Shalev-Shwartz. *Understanding Machine Learning From Theory to Algorithms*. Cambridge University Press, 2014.

[5] *Wikipedia: Categorical variable*. URL: `https://en.wikipedia.org/wiki/Categorical_variable`.