Lab 2 - Methods in Linear Regression Problems - Sakshi

February 27, 2022

1 Lab 2: Methods in Linear Regression

1.1 Problems:

1.1.1 Problem 1: Bootstrapping a Confidence Interval

If we don't have a formula for the confidence interval of a statistic, we can often estimate it by sampling from out data set many times, computing the statistic of interest, and then plotting the distribution. This is known as **bootstrapping** the confidence interval, since you're using the data to make estimates about your fits, effectively pulling yourself up by your bootstraps. In this problem, we will see how to boot strap the confidence interval for the β parameters in the linear fit.

Continue with the Lab2 Master file. Lets return to the one variable examples of fitting the sales price to the first floor square footage **1stFlrSF**. Using a for loop, compute β_0 and β_1 1000 times for samples of size $N = 1436$ **with replacement** and store their results in vectors, as in the code below.

[1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import sklearn as sk

import io
import requests

url = "https://raw.githubusercontent.com/tipthederiver/Math-7243-2020/master/
 ↪Datasets/Ames/train.csv"
s = requests.get(url).content
ames = pd.read_csv(io.StringIO(s.decode('utf-8')))
```

[2]:
```
z = ames['GrLivArea'] + ames['BsmtUnfSF']<4000
print("Number of records removed:", len(ames) - sum(z))
data = ames[z]
```

```
Number of records removed: 24
```

[3]:
```
sum(z)
```

[3]: 1436

[4]:
```
data = ames[z]
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1436 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1436 non-null   int64
 1   MSSubClass     1436 non-null   int64
 2   MSZoning       1436 non-null   object
 3   LotFrontage    1178 non-null   float64
 4   LotArea        1436 non-null   int64
 5   Street         1436 non-null   object
 6   Alley          90 non-null     object
 7   LotShape       1436 non-null   object
 8   LandContour    1436 non-null   object
 9   Utilities      1436 non-null   object
 10  LotConfig      1436 non-null   object
 11  LandSlope      1436 non-null   object
 12  Neighborhood   1436 non-null   object
 13  Condition1     1436 non-null   object
 14  Condition2     1436 non-null   object
 15  BldgType       1436 non-null   object
 16  HouseStyle     1436 non-null   object
 17  OverallQual    1436 non-null   int64
 18  OverallCond    1436 non-null   int64
 19  YearBuilt      1436 non-null   int64
 20  YearRemodAdd   1436 non-null   int64
 21  RoofStyle      1436 non-null   object
 22  RoofMatl       1436 non-null   object
 23  Exterior1st    1436 non-null   object
 24  Exterior2nd    1436 non-null   object
 25  MasVnrType     1428 non-null   object
 26  MasVnrArea     1428 non-null   float64
 27  ExterQual      1436 non-null   object
 28  ExterCond      1436 non-null   object
 29  Foundation     1436 non-null   object
 30  BsmtQual       1399 non-null   object
 31  BsmtCond       1399 non-null   object
 32  BsmtExposure   1398 non-null   object
 33  BsmtFinType1   1399 non-null   object
 34  BsmtFinSF1     1436 non-null   int64
 35  BsmtFinType2   1398 non-null   object
 36  BsmtFinSF2     1436 non-null   int64
 37  BsmtUnfSF      1436 non-null   int64
```

```
 38  TotalBsmtSF    1436 non-null   int64
 39  Heating        1436 non-null   object
 40  HeatingQC      1436 non-null   object
 41  CentralAir     1436 non-null   object
 42  Electrical     1435 non-null   object
 43  1stFlrSF       1436 non-null   int64
 44  2ndFlrSF       1436 non-null   int64
 45  LowQualFinSF   1436 non-null   int64
 46  GrLivArea      1436 non-null   int64
 47  BsmtFullBath   1436 non-null   int64
 48  BsmtHalfBath   1436 non-null   int64
 49  FullBath       1436 non-null   int64
 50  HalfBath       1436 non-null   int64
 51  BedroomAbvGr   1436 non-null   int64
 52  KitchenAbvGr   1436 non-null   int64
 53  KitchenQual    1436 non-null   object
 54  TotRmsAbvGrd   1436 non-null   int64
 55  Functional     1436 non-null   object
 56  Fireplaces     1436 non-null   int64
 57  FireplaceQu    747 non-null    object
 58  GarageType     1357 non-null   object
 59  GarageYrBlt    1357 non-null   float64
 60  GarageFinish   1357 non-null   object
 61  GarageCars     1436 non-null   int64
 62  GarageArea     1436 non-null   int64
 63  GarageQual     1357 non-null   object
 64  GarageCond     1357 non-null   object
 65  PavedDrive     1436 non-null   object
 66  WoodDeckSF     1436 non-null   int64
 67  OpenPorchSF    1436 non-null   int64
 68  EnclosedPorch  1436 non-null   int64
 69  3SsnPorch      1436 non-null   int64
 70  ScreenPorch    1436 non-null   int64
 71  PoolArea       1436 non-null   int64
 72  PoolQC         5 non-null      object
 73  Fence          278 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1436 non-null   int64
 76  MoSold         1436 non-null   int64
 77  YrSold         1436 non-null   int64
 78  SaleType       1436 non-null   object
 79  SaleCondition  1436 non-null   object
 80  SalePrice      1436 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 919.9+ KB
```

[5]:
```python
N = 1000

beta0 = []
beta1 = []

for i in range(N):
    train = data.sample(n=1436, replace=True)
    X_train = train.drop(columns=['SalePrice', 'Id'])
    Y_train = train['SalePrice']
    X = np.matrix(X_train['1stFlrSF'])
    Y = np.matrix(Y_train)
    X = X.reshape((-1, 1))
    Y = Y.reshape((-1, 1))
    X = np.append(np.ones(X.shape), X, 1)

    betas = ((X.T * X).I * X.T * Y)
    betas = np.squeeze(np.asarray(betas))
    ## Compute beta0 and beta1, using linear algebra, sklearn, or scipy
    beta0.append(betas[0])
    beta1.append(betas[1])

beta0 = np.array(beta0)
beta1 = np.array(beta1)
```

[6]:
```python
beta0.shape
```

[6]: (1000,)

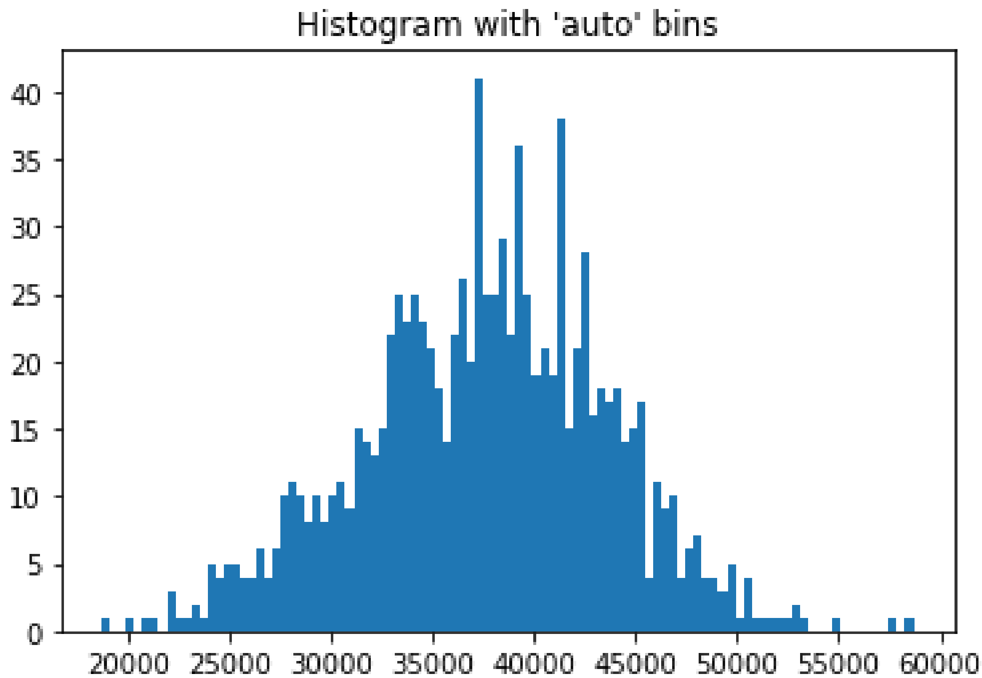
[7]:
```python
beta1.shape
```

[7]: (1000,)

Turn in

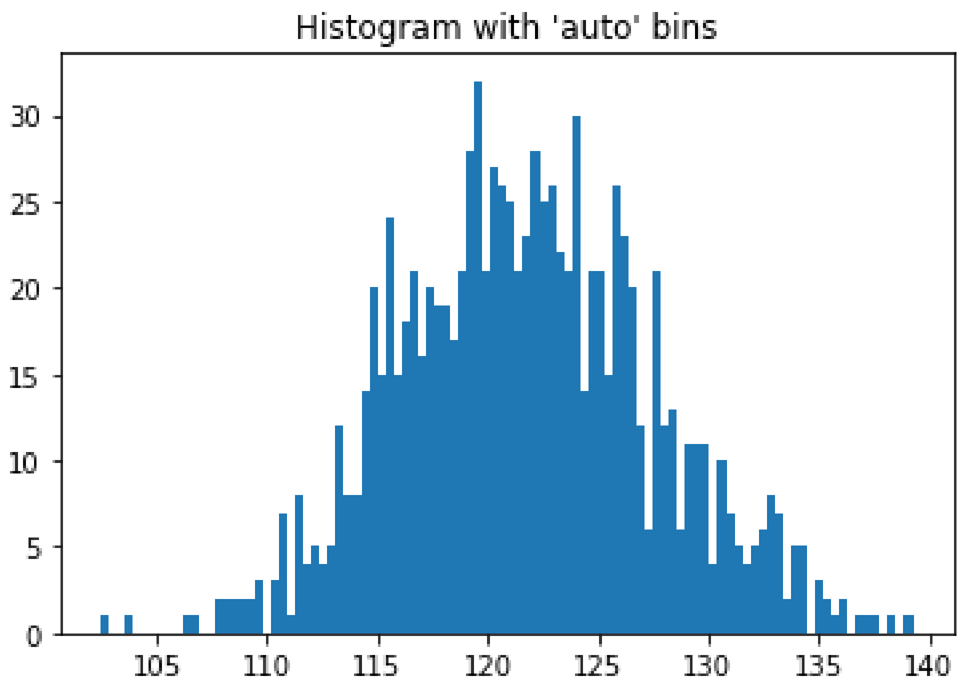
1. Plot a histogram of β_0 and β_1.
2. Using `beta0.sort()`, sort the values and find the interval containing the middle 950 values. This is the bootstrap 95% confidence interval.
3. Using the formulas from (Section 4.Statistics for ML. Sec4StatisticsML.pdf page 17.), compute the confidence interval. Remember that here you use all of the training data. Compare your results.

[8]:
```python
import matplotlib.pyplot as plt
_ = plt.hist(beta0, bins=100)
plt.title("Histogram with 'auto' bins")
plt.show()
```



[9]:

```
import matplotlib.pyplot as plt
_ = plt.hist(beta1, bins=100)
plt.title("Histogram with 'auto' bins")
plt.show()
```



[10]:
```python
beta0.sort()
beta1.sort()
```

[11]:
```python
print(f"Confidence Interval of beta0 from sampling is ({beta0[24]},␣
↪{beta0[974]})")
```

```
Confidence Interval of beta0 from sampling is (24801.726054759485,
48925.08748767334)
```

[12]:
```python
print(f"Confidence Interval of beta1 from sampling is ({beta1[24]},␣
↪{beta1[974]})")
```

```
Confidence Interval of beta1 from sampling is (110.86180704656837,
133.18660144765556)
```

[13]:
```python
X = np.array(data['1stFlrSF'])
Y = np.array(data['SalePrice'])
```

[14]:
```python
X = X.reshape((-1, 1))
Y = Y.reshape((-1, 1))
X = np.append(np.ones(X.shape), X, 1)
```

[15]:
```python
betas = np.linalg.inv(X.T @ X) @ X.T @ Y
```

[16]:
```python
n, d = X.shape
```

[17]:
```python
std = (1 / (n - d)) * (Y - X @ betas).T @ (Y - X @ betas)
```

[18]:
```python
std[0]
```

[18]:
```
array([3.2508792e+09])
```

[19]:
```python
var = std[0, 0]
```

[20]:
```python
covar = var * np.linalg.inv(X.T @ X)
```

[21]:
```python
sd_beta0, sd_beta1 = np.sqrt(covar[0, 0]), np.sqrt(covar[1, 1])
```

[22]:
```python
print(f"95% Confidence Interval for beta0 is \
({betas[0, 0] - 2 * sd_beta0}, {betas[0, 0] + 2 * sd_beta0})")
```

```
95% Confidence Interval for beta0 is (27472.0543237304, 47659.45229912873)
```

[23]:
```python
betas[1, 0]
```

[23]:
```
121.75626859147481
```

```
[24]: print(f"95% Confidence Interval for beta1 is \
({betas[1, 0] - 2 * sd_beta1}, {betas[1, 0] + 2 * sd_beta1})")
```

```
95% Confidence Interval for beta1 is (113.35578720752032, 130.15674997542928)
```

Problem 2: Linear Methods on High Dimensional Data

Perform ridge regression and lasso regression on the MRI Slices dataset on blackboard. You should follow the **Loading the Viewing MRI Slices** notebook, eventually loading all slices into Python as a data matrix, with all picture dimensions flattened. The text and code for that process has been reproduced below.

We want to fit the MRI Slices data to the **Normalized Whole-brain Volume (nWBV)** in the labels data.

Turn in:

1. Given the train-test split with seed random_state=255, what is the best α value for pure Ridge Regression? Justify your answer.
2. Given the train-test split with seed random_state=255, what is the best λ value for pure Lasso Regression? Justify your answer.
3. (Bonus) What is the best (α, λ) value for elastic net regression?

You may set the downsample rate to higher you are unable to compute the linear model.

random_state= 255 will fix the random set. See Wiki for a quick explanation. https://en.wikipedia.org/wiki/Random_seed Or https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/ for some more details.

1.1.2 Load MRI All Files

To load all of the files into an array we need to be able to search through the directory. Luckily, this is easy to do using the labels file, since each file name is stored there. We just need to loop through the **Filename** column in the `labels` dataset and load them into an array one by one. There are 702 files in total.

With the array there are two ways we can load them in: First, we can load them into a $609 \times 176 \times 176$ array, which is the best option if we care about the 2D structure. However for algorithms like linear regression that can not see the 2D structure, we may want to flatten the images to a 609×30976 array (note that $30976 = 176 \times 176$). Its easy enough two switch back and forth between the two array structures later. We will start with the flattened array.

```
[25]: # Problem 2
import numpy as np
from numpy import arange
import io
from PIL import Image
import os
import pandas as pd
!pip install opencv-python
import cv2
from sklearn.linear_model import Ridge
```

```python
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
import matplotlib
import io
from PIL import Image
!pip install path
import path


file_dir = "./"
labels = pd.read_csv(file_dir + "labels.csv")
file_dir += "MRI_Images/"
display(labels)
```

```
Requirement already satisfied: opencv-python in
/Users/saint1729/opt/anaconda3/lib/python3.9/site-packages (4.5.5.62)
Requirement already satisfied: numpy>=1.19.3 in
/Users/saint1729/opt/anaconda3/lib/python3.9/site-packages (from opencv-python)
(1.20.3)
Requirement already satisfied: path in
/Users/saint1729/opt/anaconda3/lib/python3.9/site-packages (16.0.0)
```

```
     Unnamed: 0               Filename           ID M/F Hand  Age  Educ  \
0             0   OAS1_0001_MR1_55.png  OAS1_0001_MR1   F    R   74     2
1             1  OAS1_0001_MR1_120.png  OAS1_0001_MR1   F    R   74     2
2             2  OAS1_0001_MR1_180.png  OAS1_0001_MR1   F    R   74     2
3             3   OAS1_0002_MR1_55.png  OAS1_0002_MR1   F    R   55     4
4             4  OAS1_0002_MR1_120.png  OAS1_0002_MR1   F    R   55     4
..          …                    …            …  ..  …   …    …
604         604  OAS1_0449_MR1_120.png  OAS1_0449_MR1   F    R   71     3
605         605  OAS1_0449_MR1_180.png  OAS1_0449_MR1   F    R   71     3
606         606   OAS1_0456_MR1_55.png  OAS1_0456_MR1   M    R   61     5
607         607  OAS1_0456_MR1_120.png  OAS1_0456_MR1   M    R   61     5
608         608  OAS1_0456_MR1_180.png  OAS1_0456_MR1   M    R   61     5

     SES  MMSE  CDR  eTIV   nWBV    ASF  Delay  Slice
0    3.0    29  0.0  1344  0.743  1.306    NaN     55
1    3.0    29  0.0  1344  0.743  1.306    NaN    120
2    3.0    29  0.0  1344  0.743  1.306    NaN    180
3    1.0    29  0.0  1147  0.810  1.531    NaN     55
4    1.0    29  0.0  1147  0.810  1.531    NaN    120
..   …   …   …   …    …    …    …    …
604  4.0    29  0.0  1264  0.818  1.388    NaN    120
605  4.0    29  0.0  1264  0.818  1.388    NaN    180
606  2.0    30  0.0  1637  0.780  1.072    NaN     55
607  2.0    30  0.0  1637  0.780  1.072    NaN    120
```

608 2.0   30 0.0 1637 0.780 1.072   NaN   180

[609 rows x 15 columns]

[26]:
```
DS = 8              # Downsample rate, must be a multiple of 30976

if 30976/DS % 1 > 0:
    print("Downsample rate is not a multiple of 30976")
    DS = 1
    im_size = 30976
else:
    im_size = int(30976/DS)


data = np.zeros([609, im_size])


for i, file_name in enumerate(labels['Filename']):
        img = np.mean(matplotlib.image.imread(file_dir + file_name),axis=2).
 ↪reshape(-1)
        data[i,:] = img[::DS]             # Downsample the image
```

[27]:
```
flatteneddata = []
Y =[]
for i in range(len(labels)):
    filename = labels.loc[i,'Filename']
    y = labels.loc[i,'nWBV']
    data = np.array(cv2.imread(file_dir + filename, 0))
    flattened = data.flatten()
    flatteneddata.append(flattened)
    Y.append(y)
```

[28]:
```
X_train, X_test, y_train, y_test = train_test_split(flatteneddata, Y,␣
 ↪test_size=0.25, random_state=255)
```

[29]:
```
alphas = [0.0001, 0.001, 0.01, 0.1, 1, 1.5, 10, 15, 20, 50, 100, 1000, 10000]
scores_l2 = []

for a in alphas:
    clf = Ridge(alpha = a ,normalize= True).fit(X_train,y_train)
    scores_l2.append(clf.score(X_test,y_test))

scores_l2 = np.array(scores_l2)
```

[30]:
```
print(f"Best alpha = {alphas[scores_l2.argmax()]}")
```

Best alpha = 50

[31]:
```
lmds = [0.00009, 0.0001, 0.0005, 0.001, 0.01]
scores_l1 = []

for lmd in lmds:
    clf = Lasso(alpha=lmd, normalize=True).fit(X_train, y_train)
    scores_l1.append(clf.score(X_test, y_test))

scores_l1 = np.array(scores_l1)
```

[32]:
```
print(f"Best lambda = {lmds[scores_l1.argmax()]}")
```

```
Best lambda = 0.0001
```

[33]:
```
alphas = [0.0001, 0.001, 0.01, 0.1, 1, 1.5, 10, 15, 20, 50, 100, 1000, 10000]
lmds = [0.00009, 0.0001, 0.0005, 0.001, 0.01]
scores_elastic = []

for a in lmds:
    scores = []
    for b in alphas:
        clf = ElasticNet(alpha=a + b, l1_ratio=a / (a + b), normalize=True).
 ↪fit(X_train, y_train)
        scores.append(clf.score(X_test, y_test))
    scores_elastic.append(np.array(scores))

scores_elastic = np.array(scores_elastic)
```

[34]:
```
lmd_argmax, alp_argmax =  np.unravel_index(np.argmax(scores_elastic,␣
 ↪axis=None), scores_elastic.shape)
```

[36]:
```
print(f"Best (alpha, lambda) = ({alphas[alp_argmax]}, {lmds[lmd_argmax]})")
```

```
Best (alpha, lambda) = (0.01, 9e-05)
```

[ ]:
```

```