

lab_eeg_partial

October 11, 2021

0.0.1 Student Name: Danrong Li

0.0.2 Student Net Id: dl4111

1 Lab: Source Localization for EEG

EEG or [Electroencephalography](#) is a powerful tool for neuroscientists in understanding brain activity. In EEG, a patient wears a headset with electrodes that measures voltages at a number of points on the scalp. These voltages arise from ionic currents within the brain. A common *inverse problem* is to estimate the which parts of the brain caused the measured response. Source localization is useful in understanding which parts of the brain are involved in certain tasks. A key challenge in this inverse problem is that the number of unknowns (possible locations in the brain) is much larger than the number of measurements. In this lab, we will use LASSO regression on a real EEG dataset to overcome this problem and determine the brain region that is active under an auditory stimulus.

In addition to the concepts in the [prostate LASSO demo](#) you will learn to:

- * Represent responses of multi-channel time-series data, such as EEG, using linear models
- * Perform LASSO and Ridge regression
- * Select the regularization level via cross-validation
- * Visually compare the sparsity between the solutions

We first download standard packages.

```
[1]: import numpy as np
import matplotlib.pyplot as plt
import pickle

from sklearn.linear_model import Lasso, Ridge, ElasticNet
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

1.1 Load the Data

The data in this lab is taken from one of the sample datasets in the [MNE website](#). The sample data is a recording from one subject who experienced some auditory stimulus on the left ear.

The raw data is very large (1.5G) and also requires that you install the `mne` python package. To make this lab easier, I have extracted and processed a small section of the data. The following command will download a `pickle` file `eeg_dat.p` to your local machine. If you do want to create the data yourself, the program to create the data is in this directory in the github repository.

```
[2]: fn_src = 'https://drive.google.com/uc?
      ↪export=download&id=1RzQpKON0cXSMxH2Zz0I4iVMiTgD6ttS1'
      fn_dst = 'eeg_dat.p'

      import os
      from six.moves import urllib

      if os.path.isfile(fn_dst):
          print('File %s is already downloaded' % fn_dst)
      else:
          print('Fetching file %s [53MB]. This may take a minute..' % fn_dst)
          urllib.request.urlretrieve(fn_src, fn_dst)
          print('File %s downloaded' % fn_dst)
```

File eeg_dat.p is already downloaded

Now run the following command which will get the data from the pickle file.

```
[3]: import pickle
      fn = 'eeg_dat.p'
      with open(fn, 'rb') as fp:
          [X,Y] = pickle.load(fp)
```

To understand the data, there are three key variables: * **nt** = number of time steps that we measure data * **nchan** = number of channels (i.e. electrodes) measured in each time step * **ncur** = number of currents in the brain that we want to estimate.

Each current comes from one brain region (called a *voxel*) in either the x, y or z direction. So,

nvoxels = **ncur** / 3

The components of the X and Y matrices are: * **Y[i,k]** = electric field measurement on channel i at time k * **X[i,j]** = sensitivity of channel i to current j.

Using **X.shape** and **Y.shape** compute and print **nt**, **nchan**, **ncur** and **nvoxels**.

```
[4]: X.shape
```

```
[4]: (305, 22494)
```

```
[5]: Y.shape
```

```
[5]: (305, 85)
```

```
[6]: # TODO 1
      nt = Y.shape[1]
      ncur = X.shape[1]
      nchan = X.shape[0]
      nvoxels = int(ncur/3)
```

```
[7]: print("nt = {}".format(nt)+
        "\nncur = {}".format(ncur)+
        "\nnchan = {}".format(nchan)+
        "\nnvoxels = {}".format(nvoxels))
```

```
nt = 85
ncur = 22494
nchan = 305
nvoxels = 7498
```

1.2 Ridge Regression

Our goal is to estimate the currents in the brain from the measurements Y . One simple linear model is:

$$Y[i,k] = \sum_j X[i,j] * W[j,k] + b[k]$$

where $W[j,k]$ is the value of current j at time k and $b[k]$ is a bias. We can solve for the current matrix W via linear regression.

However, there is a problem: * There are $nt \times ncur$ unknowns in W * There are only $nt \times nchan$ measurements in Y .

In this problem, we have:

number of measurements \ll number of unknowns

We need to use regularization in these circumstances. We first try Ridge regression.

First split the data into training and test. Use the `train_test_split` function with `test_size=0.33`.

```
[8]: # TODO 2
Xtr,Xts,Ytr,Yts = train_test_split(X,Y,test_size=0.33, random_state = 3)
```

Use the `Ridge` regression object in `sklearn` to fit the model on the training data. Use a regularization, `alpha=1`.

```
[9]: # TODO 3
regr = Ridge(alpha=1)
regr.fit(Xtr,Ytr)
Yhat_train = regr.predict(Xtr)
Yhat_test = regr.predict(Xts)
```

Predict the values Y on both the training and test data. Use the `r2_score` method to measure the R^2 value on both the training and test. You will see that R^2 value is large for the training data, it is very low for the test data. This suggests that even with regularization, the model is over-fitting the data.

```
[10]: # TODO 4
rsq_tr = r2_score(Ytr,Yhat_train)
rsq_ts = r2_score(Yts,Yhat_test)
```

```
[11]: print("R^2 value for train: {}".format(rsq_tr))
      print("R^2 value for test: {}".format(rsq_ts))
```

R² value for train: 0.5999491212694172

R² value for test: 0.09981805129823948

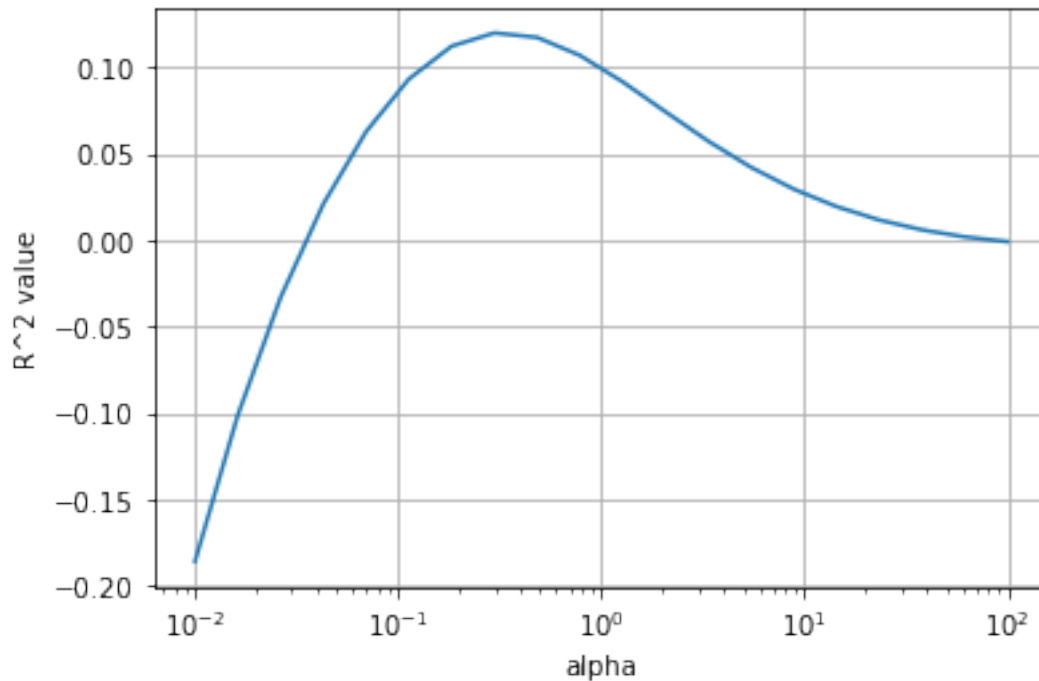
Next, try to see if we can get a better R² score using different values of **alpha**. Use cross-validation to measure the test R² for 20 alpha values logarithmically spaced from 10⁻² to 10² (use `np.logspace()`). You can use regular cross-validation. You do not need to do K-fold.

```
[12]: # TODO 5
      alphas = np.logspace(-2,2,num=20)
      rsq_cv = []
      for alph in alphas:
          regr = Ridge(alpha=alph)
          regr.fit(Xtr,Ytr)
          Yhat_test = regr.predict(Xts)
          rsq_ts = r2_score(Yts,Yhat_test)
          rsq_cv.append(rsq_ts)
```

Plot the test R² vs. alpha. And print the maximum test R². You should see that the maximum test R² is still not very high.

```
[13]: # TODO 6
      plt.semilogx(alphas,rsq_cv)
      plt.xlabel("alpha")
      plt.ylabel("R^2 value")
      plt.grid()
      maxR2 = max(rsq_cv)
      print("maximum test R^2 = {}".format(maxR2))
```

maximum test R² = 0.12003178046945735



Now, let's take a look at the solution.

- Find the optimal regularization **alpha** from the cross-validation
- Re-fit the model at the optimal **alpha**
- Get the current matrix **W** from the coefficients in the linear model. These are stored in `regr.coef_`. You may need a transpose
- For each current **j** compute $Wrms[j] = \sqrt{\sum_k W[j,k]^2}$ which is root mean squared current.

You will see that the vector **Wrms** is not sparse. This means that the solution that is found with Ridge regression finds currents in all locations.

```
[14]: # TODO 7
      optIndex = np.argmax(rsq_cv)
      optAl = alphas[optIndex]
      regr = Ridge(alpha=optAl)
      regr.fit(Xtr,Ytr)
      W = regr.coef_.T
      Wrms = np.sqrt(np.sum(W**2,axis=1))
      Wrms
```

```
[14]: array([0.26086777, 0.11113737, 0.13657601, ..., 0.0423126 , 0.0499962 ,
            0.07566757])
```

```
[15]: W
```

```
[15]: array([[ 0.0386654 ,  0.04040561,  0.04113065, ...,  0.01378603,
              0.00708401,  0.0011852 ],
            [-0.03267979, -0.03131064, -0.0287227 , ..., -0.00614038,
              -0.00241382,  0.0010929 ],
            [-0.04095378, -0.03857623, -0.03482474, ..., -0.01023514,
              -0.00563579, -0.00121668],
            ...,
            [ 0.00300261,  0.0024171 ,  0.00190474, ...,  0.00113062,
              0.00071006,  0.00052505],
            [-0.00033694,  0.00047848,  0.00111954, ...,  0.00025654,
              0.0005638 ,  0.0007048 ],
            [-0.00244836, -0.00292925, -0.0034843 , ..., -0.00967206,
              -0.00910611, -0.00850428]])
```

1.3 LASSO Regression

We can improve the estimate by imposing sparsity. Biologically, we know that only a limited number of brain regions should be involved in the response to a particular stimuli. As a result, we would expect that the current matrix $W[j,k]$ to be zero for most values j,k . We can impose this constraint using LASSO regularization.

Re-fit the training data using the Lasso model with `alpha=1e-3`. Also set `max_iter=100` and `tol=0.01`. The LASSO solver is much slower, so this may take a minute.

```
[16]: # TODO 8
lasso_regr = Lasso(alpha = 1e-3, max_iter = 100, tol = 0.01)
lasso_regr.fit(Xtr,Ytr)
```

```
[16]: Lasso(alpha=0.001, max_iter=100, tol=0.01)
```

Now, test the model on the test data and measure the R^2 value. You should get a much better fit than with the Ridge regression solution.

```
[17]: # TODO 9
Yhat = lasso_regr.predict(Xts)
rsq = r2_score(Yts,Yhat)
rsq
```

```
[17]: 0.13596199270782092
```

We can now search for the optimal `alpha`. Use cross-validation to find the `alpha` logarithmically space between `alpha=10-3` and `alpha=10-4`. Each fit takes some time, so use only 5 values of `alpha`. Also for each `alpha` store the current matrix. This way, you will not have to re-fit the model.

```
[18]: # TODO 10
alphas = np.logspace(-3,-4,num=5)
rsq_cv = []
W_cv = []
```

```

Wrms_cv = []
for alph in alphas:
    lasso_regr = Lasso(alpha = alph, max_iter = 100, tol = 0.01)
    lasso_regr.fit(Xtr,Ytr)
    Yhat = lasso_regr.predict(Xts)
    rsq = r2_score(Yts,Yhat)
    rsq_cv.append(rsq)
    W = lasso_regr.coef_.T
    W_cv.append(W)
    Wrms = np.sqrt(np.sum(W**2,axis=1))
    Wrms_cv.append(Wrms)

```

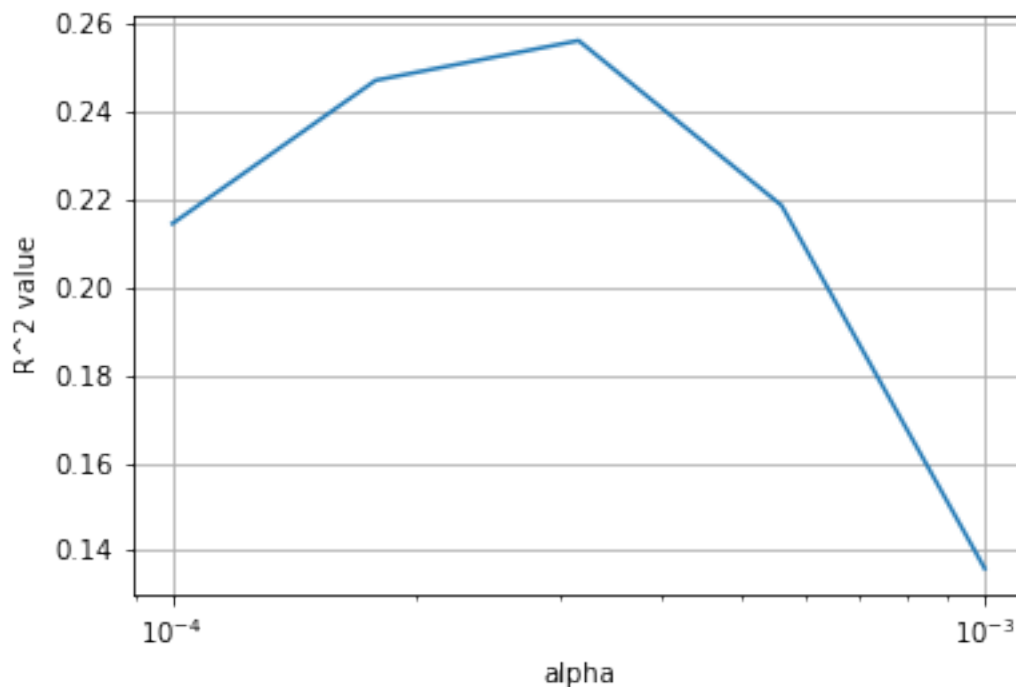
Plot the r^2 value vs. α . Print the optimal r^2 . You should see it is much higher than with the best Ridge Regression case.

```

[19]: # TODO 11
plt.semilogx(alphas,rsq_cv)
plt.xlabel("alpha")
plt.ylabel("R^2 value")
plt.grid()
optIdx = np.argmax(rsq_cv)
maxRsq = rsq_cv[optIdx]
print("optimal R^2 = {}".format(maxRsq))

```

optimal $R^2 = 0.2561946050699821$



Display the current matrix W for the optimal α as you did in the Ridge Regression case. You will see that is much sparser.

```
[20]: # TODO 12
      W[optIdx]
```

```
[20]: array([-0., -0., -0., -0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -0.,
          -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
          -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,  0., -0., -0.,
          -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,  0.,  0.,  0.,
           0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
           0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -0., -0., -0., -0., -0.,
          -0., -0., -0., -0., -0., -0., -0.]
```

```
[21]: Wrms_cv[optIdx]
```

```
[21]: array([0., 0., 0., ..., 0., 0., 0.])
```

1.4 More fun (NOT GRADED)

If you want to more on this lab: * Install the [MNE python package](#). This is an amazing package with many tools for processing EEG data. * In particular, you can use the above results to visualize where in the brain the currents sources are. * You can also improve the fitting with more regularization. For example, we know that the currents will be non-zero in groups: If the current is non-zero for one time, it is likely to non-zero for all time. You can use the Group LASSO method. * You can combine these results to make predictions about what the patient is seeing or hearing or thinking.

```
[ ]:
```