homework5-skel

October 13, 2020

NAME: Nigel Mansell
SECTION: 995
CS 5970: Machine Learning Practices

1 Homework 5: Regularization

1.1 Assignment Overview

Follow the TODOs and read through and understand any provided code.
For all plots, make sure all necessary axes and curves are clearly and accurately labeled. Include figure/plot titles appropriately as well.

1.1.1 Task

For this assignment you will be exploring regularization. Regularization is a powerful tool in machine learning to impose rational constraints on models during the training process to mitigate overfitting to the training set and improve model generalization. By including one or more terms within the cost function to penalize the weights, the learning algorithm will try to fit the data while avoiding certain values for the weights that might overfit the data.

1.1.2 Data set

The BMI data will be utilized. Recall: * *MI* files contain data with the number of action potentials (spikes) for 48 neurons, at mutliple time points, for a single fold. There are 20 folds (20 files), where each fold consists of over 1000 times points (the rows). At each time point, we record the number of activations for each neuron for 20 bins. Therefore, each time point has 48 * 20 = 960 columns.
* *theta* files record the angular position of the shoulder (in column 0) and the elbow (in column 1) for each time point (in radians).
* *dtheta* files record the angular velocity of the shoulder (in column 0) and the elbow (in column 1) for each time point (in radians/sec).
* *torque* files record the torque of the shoulder (in column 0) and the elbow (in column 1) for each time point (N-m).
* *time* files record the actual time stamp of each time point (seconds).

This assignment utilizes code examples and concepts from the lectures on Sept 19 - Oct 1.

1.1.3 Objectives

- Use and understand regularization in regression
- Learn to select hyper-parameters to tune model behavior, specifically:

  - Regularization parameters
  - Training set size

1.1.4 Notes

- Be sure to adequately label all the plots you generate.

1.1.5 General References

- Python Built-in Functions
- Python Data Structures
- Numpy Reference
- Summary of matplotlib
- Pandas DataFrames
- Sci-kit Learn Linear Models
- Sci-kit Learn Ensemble Models
- Sci-kit Learn Metrics
- Sci-kit Learn Model Selection

1.1.6 Hand-In Procedure

- Execute all cells so they are showing correct results
- Notebook:
  - Submit this file (.ipynb) to the Canvas HW5 dropbox
- PDF:
  - File/Print/Print to file -> Produces a copy of the notebook in PDF format
  - Submit the PDF file to the Gradescope HW5 dropbox

[2]:
```
# PROVIDED

import pandas as pd
import numpy as np
import os, re, fnmatch, time
import matplotlib.pyplot as plt
import joblib

from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression, ElasticNet
from sklearn.metrics import make_scorer

FIGWIDTH = 6
FIGHEIGHT = 6
FONTSIZE = 10

plt.rcParams['figure.figsize'] = (FIGWIDTH, FIGHEIGHT)
plt.rcParams['font.size'] = FONTSIZE

plt.rcParams['xtick.labelsize'] = FONTSIZE
plt.rcParams['ytick.labelsize'] = FONTSIZE
```

```
%matplotlib inline
```

2 LOAD DATA

[3]:
```
# PROVIDED

def read_bmi_file_set(directory, filebase):
    '''
    Read a set of CSV files and append them together
    :param directory: The directory in which to scan for the CSV files
    :param filebase: A file specification that potentially includes wildcards
    :returns: A list of Numpy arrays (one for each fold)
    '''

    # The set of files in the directory
    files = fnmatch.filter(os.listdir(directory), filebase)
    files.sort()

    # Create a list of Pandas objects; each from a file in the directory that␣
 ↪matches filebase
    lst = [pd.read_csv(directory + "/" + file, delim_whitespace=True,␣
 ↪header=None).values for file in files]

    # Concatenate the Pandas objects together.  ignore_index is critical here␣
 ↪so that
    # the duplicate row indices are addressed
    return lst
```

[4]:
```
""" TODO
Load the BMI data from all the folds, using read_bmi_file_set()
"""
# may need to adjust the filepath if you are not working on Oscer
dir_name = '/home/nigel/Desktop/mlp/mlp_2020/datasets/bmi/DAT6_08'
#loading data
MI_folds = read_bmi_file_set(dir_name, 'MI_fold*')
theta_folds = read_bmi_file_set(dir_name, 'theta_fold*')
dtheta_folds = read_bmi_file_set(dir_name, 'dtheta_fold*')
torque_folds = read_bmi_file_set(dir_name, 'torque_fold*')
time_folds = read_bmi_file_set(dir_name, 'time_fold*')

alldata_folds = zip(MI_folds, theta_folds, dtheta_folds, torque_folds,␣
 ↪time_folds)

nfolds = len(MI_folds)
nfolds
```

[4]: 20

[5]:
```
"""
Print out the shape of all the data for each fold
"""
for i, (MI, theta, dtheta, torque, time) in enumerate(alldata_folds):
    print("FOLD %2d " % i, MI.shape, theta.shape,
          dtheta.shape, torque.shape, time.shape)
```

```
FOLD  0  (1194, 960) (1194, 2) (1194, 2) (1194, 2) (1194, 1)
FOLD  1  (1105, 960) (1105, 2) (1105, 2) (1105, 2) (1105, 1)
FOLD  2  (1532, 960) (1532, 2) (1532, 2) (1532, 2) (1532, 1)
FOLD  3  (1266, 960) (1266, 2) (1266, 2) (1266, 2) (1266, 1)
FOLD  4  (1499, 960) (1499, 2) (1499, 2) (1499, 2) (1499, 1)
FOLD  5  (1253, 960) (1253, 2) (1253, 2) (1253, 2) (1253, 1)
FOLD  6  (1376, 960) (1376, 2) (1376, 2) (1376, 2) (1376, 1)
FOLD  7  (1131, 960) (1131, 2) (1131, 2) (1131, 2) (1131, 1)
FOLD  8  (1248, 960) (1248, 2) (1248, 2) (1248, 2) (1248, 1)
FOLD  9  (1258, 960) (1258, 2) (1258, 2) (1258, 2) (1258, 1)
FOLD 10  (1266, 960) (1266, 2) (1266, 2) (1266, 2) (1266, 1)
FOLD 11  (1147, 960) (1147, 2) (1147, 2) (1147, 2) (1147, 1)
FOLD 12  (1226, 960) (1226, 2) (1226, 2) (1226, 2) (1226, 1)
FOLD 13  (1239, 960) (1239, 2) (1239, 2) (1239, 2) (1239, 1)
FOLD 14  (1571, 960) (1571, 2) (1571, 2) (1571, 2) (1571, 1)
FOLD 15  (1360, 960) (1360, 2) (1360, 2) (1360, 2) (1360, 1)
FOLD 16  (1580, 960) (1580, 2) (1580, 2) (1580, 2) (1580, 1)
FOLD 17  (1365, 960) (1365, 2) (1365, 2) (1365, 2) (1365, 1)
FOLD 18  (1390, 960) (1390, 2) (1390, 2) (1390, 2) (1390, 1)
FOLD 19  (1290, 960) (1290, 2) (1290, 2) (1290, 2) (1290, 1)
```

[6]:
```
"""
Print out the first few examples of the theta data
for a few folds
"""
for i, theta in enumerate(theta_folds[:3]):
    print("FOLD %2d" % i)
    print(theta[:5, :])
```

```
FOLD  0
[[0.04303586 1.7163499 ]
 [0.07592855 1.7351056 ]
 [0.13092623 1.728305  ]
 [0.20449087 1.6938625 ]
 [0.28873832 1.6359639 ]]
FOLD  1
[[0.45208906 1.4041636 ]
 [0.43492136 1.4420629 ]
 [0.40924987 1.5011598 ]
```

```
 [0.3782652  1.5710233 ]
 [0.34620736 1.6384762 ]]
FOLD  2
[[0.52770978 1.4720288 ]
 [0.51488065 1.4552057 ]
 [0.49425913 1.4322803 ]
 [0.46639178 1.4069214 ]
 [0.43195224 1.3847791 ]]
```

[7]:
```
"""
Check the data for any NaNs
"""
def anynans(X):
    return np.isnan(X).any()

alldata_folds = zip(MI_folds, theta_folds, dtheta_folds, torque_folds,␣
 ↪time_folds)

for i, (MI, theta, dtheta, torque, time) in enumerate(alldata_folds):
    print("FOLD %2d " % i, anynans(MI), anynans(theta),
          anynans(dtheta), anynans(torque), anynans(time))
```

```
FOLD  0  False False False False False
FOLD  1  False False False False False
FOLD  2  False False False False False
FOLD  3  False False False False False
FOLD  4  False False False False False
FOLD  5  False False False False False
FOLD  6  False False False False False
FOLD  7  False False False False False
FOLD  8  False False False False False
FOLD  9  False False False False False
FOLD 10  False False False False False
FOLD 11  False False False False False
FOLD 12  False False False False False
FOLD 13  False False False False False
FOLD 14  False False False False False
FOLD 15  False False False False False
FOLD 16  False False False False False
FOLD 17  False False False False False
FOLD 18  False False False False False
FOLD 19  False False False False False
```

3 REGULARIZED REGRESSION

[191]:
```
""" TODO
Evaluate the training performance of an already trained model
"""
def mse_rmse(trues, preds):
    '''
    Compute MSE and rMSE for each column separately.
    '''
    mse = np.sum(np.square(trues - preds), axis=0) / trues.shape[0]
    rmse = np.sqrt(mse)
    return mse, rmse

def predict_score_eval(model, X, y):
    '''
    Compute the model predictions and cooresponding scores.
    PARAMS:
        X: feature data
        y: cooresponding output
    RETURNS:
        mse: mean squared error for each column
        rmse: rMSE in radians
        score: score computed by the models score() method
        preds: predictions of the model from X
    '''

    # TODO: follow similar example to HW4 to complete implementation
    #calling predict and score, also calling mse_rmse
    preds = model.predict(X)
    score = model.score(X,y)
    mse, rmse = mse_rmse(y, preds)

    return mse, rmse, score, preds


""" TODO
Create scoring function object for gridsearch

This represents a more general way of creating a scoring mechanism than
what was discussed in the lectures.

GridSearchCV: https://scikit-learn.org/stable/modules/generated/sklearn.
 ↪model_selection.GridSearchCV.html
make_scorer: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.
 ↪make_scorer.html

"""
```

```
def rmse_scorer(trues, preds):
    '''
    Compute rMSE
    '''

    #cammse mse_rmse
    mse, rmse = mse_rmse(trues, preds)
    return rmse

# Make the scoring function for GridSearch
rmse_scoring = make_scorer(rmse_scorer, greater_is_better=False)
```

[192]:
```
"""
Construct training set to obtain best model and testing set for
evaluation. The model will focus on predicting the shoulder torque.
"""
# Extract fold indices for the training and testing sets
trainset_fold_inds = [5, 6]
testset_fold_inds = [8, 9]

# Combine the folds into singular numpy arrays
# Training set
MI_trainset = [MI_folds[f] for f in trainset_fold_inds]
torque_trainset = [torque_folds[f] for f in trainset_fold_inds]
time_trainset = [time_folds[f] for f in trainset_fold_inds]

X = np.concatenate(MI_trainset, axis=0)
y = np.concatenate(torque_trainset, axis=0)[:,0]
time = np.concatenate(time_trainset, axis=0)

# Testing set
MI_testset = [MI_folds[f] for f in testset_fold_inds]
torque_testset = [torque_folds[f] for f in testset_fold_inds]
time_testset = [time_folds[f] for f in testset_fold_inds]

Xtest = np.concatenate(MI_testset, axis=0)
ytest = np.concatenate(torque_testset, axis=0)[:,0]
time_test = np.concatenate(time_testset, axis=0)
```

[193]:
```
X.shape, y.shape, Xtest.shape, ytest.shape
```

[193]:
```
((2629, 960), (2629,), (2506, 960), (2506,))
```

3.1 Linear Model

[194]:
```
""" TODO
Construct and train the default linear model using the training set.
Display the Training and Testing rMSEs.
You can use the rmse_scorer for this.
"""

#creating linear model, fitting it, and getting preds for rmse_scorer for output
linear = LinearRegression().fit(X,y)
preds = linear.predict(X)
print('Training rMSE ', rmse_scorer(y, preds))
#this preds is for testing set
preds = linear.predict(Xtest)
print('Testing rMSE ', rmse_scorer(ytest, preds))
```

```
Training rMSE  0.06272460011342579
Testing rMSE  0.10205029259987247
```

3.2 Grid Search and ElasticNet Model

[12]:
```
""" TODO
Specify the parameter search grid as a dictionary, and display it
"""
alphas = np.logspace(-10, 9, base=2, num=9, endpoint=True)
l1_ratios = np.arange(0, 1.2, .2)
max_iters = [1e4]
nalphas = len(alphas)
nl1_ratios = len(l1_ratios)

#creating a dictionary for gridsearch
param_grid = [{'alpha':alphas, 'l1_ratio':l1_ratios, 'max_iter':max_iters}]
param_grid
```

[12]:
```
[{'alpha': array([9.76562500e-04, 5.06577951e-03, 2.62780130e-02,
1.36313467e-01,
        7.07106781e-01, 3.66801617e+00, 1.90273138e+01, 9.87014928e+01,
        5.12000000e+02]),
  'l1_ratio': array([0. , 0.2, 0.4, 0.6, 0.8, 1. ]),
  'max_iter': [10000.0]}]
```

[13]:
```
""" TODO
Perform the GridSearch using an ElasticNet model and the parameter grid
constructed above. Use 10 cross validation folds, rmse_scoring for
the scoring function, return_train_score=True, iid=False,
and set the verbosity to 1. Use n_jobs = -1, to make use of all available
processors. Execute the grid search using the training data.
"""
```

```
gridsearchfname = "hw5_gridsearch.pkl"
search = None
if os.path.exists(gridsearchfname):
    # Gridsearch was already completed and saved: use this instead
    search = joblib.load(gridsearchfname)
else:
    # Perform the grid search

    #calling elasticnet then calling gridseachcv
    model = ElasticNet()
    search = GridSearchCV(model, param_grid, cv = 10, scoring=rmse_scoring,
                          return_train_score=True, iid=False, verbose=1,␣
 ↪n_jobs=-1)

    # TODO: fit the search to the data
    #fitting search
    search.fit(X,y)
    joblib.dump(search, "hw5_gridsearch.pkl")
```

```
Fitting 10 folds for each of 54 candidates, totalling 540 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  4.2min
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed: 19.0min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed: 37.9min
[Parallel(n_jobs=-1)]: Done 540 out of 540 | elapsed: 43.6min finished
/home/nigel/.local/lib/python3.8/site-
packages/sklearn/model_selection/_search.py:847: FutureWarning: The parameter
'iid' is deprecated in 0.22 and will be removed in 0.24.
  warnings.warn(
/home/nigel/.local/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:529: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 11.830944698368521, tolerance: 0.00975858493662234
  model = cd_fast.enet_coordinate_descent(
```

[35]:
```
""" TODO
Get and display the best parameter set

Note: see the best_params_ property of the GridSearchCV object
"""

#getting best parameter set and displaying it
best_param_set = search.best_params_
print(best_param_set)
```

```
{'alpha': 0.7071067811865476, 'l1_ratio': 0.0, 'max_iter': 10000.0}
```

[25]:
```
""" TODO
Get and fit the best estimator to the training data

Note: see the best_estimator_ property of the GridSearchCV object
"""

#getting best estimator and fitting it
best_estimator = search.best_estimator_
best_estimator.fit(X, y)
```

```
ElasticNet(alpha=0.7071067811865476, l1_ratio=0.0, max_iter=10000.0)
```

```
/home/nigel/.local/lib/python3.8/site-
packages/sklearn/linear_model/_coordinate_descent.py:529: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 11.830944698368521, tolerance: 0.00975858493662234
  model = cd_fast.enet_coordinate_descent(
```

[36]:
```
""" TODO
Get and display the first few lines of results from the gridsearch

Note: see the cv_results_ property of GridSearchCV. And, remember
that this dict can be converted to a DataFrame
"""

#getting the data, setting it to a dataframe, the calling head
stats = search.cv_results_
df = pd.DataFrame(stats)
display(df.head())
```

```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  param_alpha  \
0     112.077216      2.738757         0.003394        0.000516  0.000976562
1       1.678367      0.089537         0.003532        0.001295  0.000976562
2       1.349998      0.090252         0.003167        0.000281  0.000976562
3       1.050393      0.073994         0.003432        0.000408  0.000976562
4       0.911781      0.079315         0.003416        0.000759  0.000976562

  param_l1_ratio param_max_iter  \
0              0          10000
1            0.2          10000
2            0.4          10000
3            0.6          10000
4            0.8          10000

                                              params  split0_test_score  \
0  {'alpha': 0.0009765625, 'l1_ratio': 0.0, 'max_…          -0.106149
1  {'alpha': 0.0009765625, 'l1_ratio': 0.2, 'max_…          -0.093914
2  {'alpha': 0.0009765625, 'l1_ratio': 0.4, 'max_…          -0.087069
```

```
3  {'alpha': 0.0009765625, 'l1_ratio': 0.60000000…          -0.082867
4  {'alpha': 0.0009765625, 'l1_ratio': 0.8, 'max_…          -0.079891

   split1_test_score  …  split2_train_score  split3_train_score  \
0          -0.092582  …           -0.057463           -0.057721
1          -0.079920  …           -0.059808           -0.060140
2          -0.074864  …           -0.062590           -0.063353
3          -0.072308  …           -0.065075           -0.065841
4          -0.071309  …           -0.067201           -0.067922

   split4_train_score  split5_train_score  split6_train_score  \
0           -0.062261           -0.062204           -0.061885
1           -0.064302           -0.064424           -0.063895
2           -0.066961           -0.067099           -0.066373
3           -0.069335           -0.069451           -0.068842
4           -0.071483           -0.071575           -0.071204

   split7_train_score  split8_train_score  split9_train_score  \
0           -0.062173           -0.057877           -0.058921
1           -0.064362           -0.060180           -0.061418
2           -0.067051           -0.062913           -0.064562
3           -0.069466           -0.065173           -0.067235
4           -0.071770           -0.067200           -0.069694

   mean_train_score  std_train_score
0         -0.060292         0.001981
1         -0.062535         0.001864
2         -0.065350         0.001749
3         -0.067823         0.001751
4         -0.070058         0.001828

[5 rows x 33 columns]
```

[56]:
```
""" TODO
Extract and negate the mean_train_score
"""
#extracting main_train_score and negating the negatives
mean_train_score = np.array(-stats['mean_train_score'])
```

[58]:
```
""" TODO
Extract and negate the mean_test_score

Note: although scikit-learn refers to this as a "test score," it is actually
a validation score.  Remember, you are not allowed to look at the test set
performance across a grid of parameter choices (only look at the one test score
for the hyper parameter set that you select).
"""
```

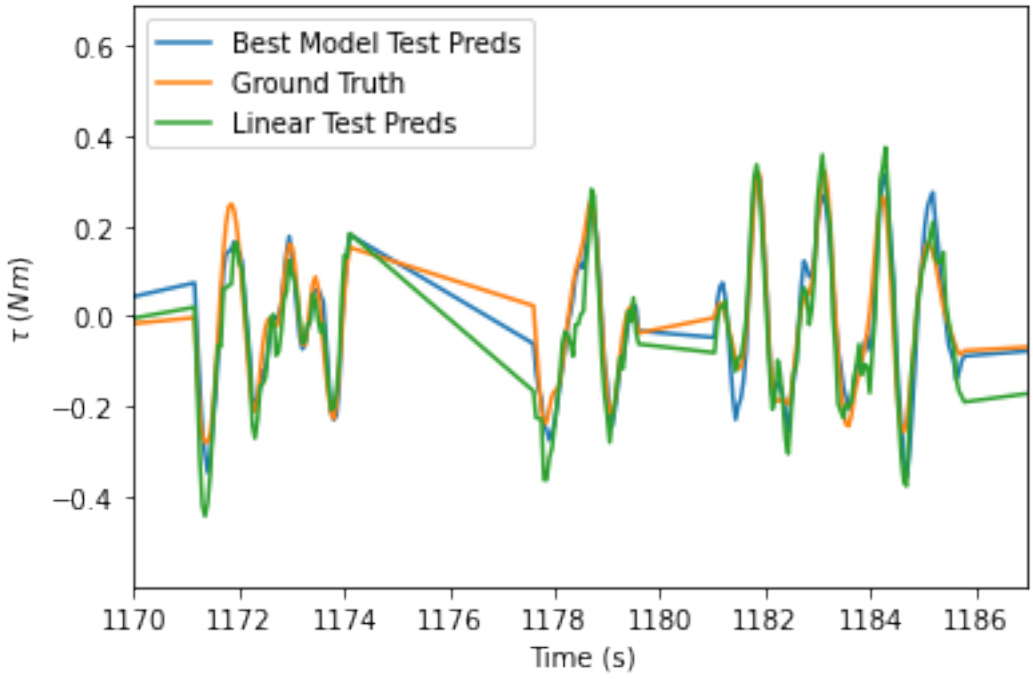
```
#extracting the mean_test_score and negating the negatives
mean_test_score = np.array(-stats['mean_test_score'])
```

[42]:
```
""" TODO
Display the Training and Testing rMSEs for the best estimator.
You can use rmse_scorer for this
"""
# Train rmse

#displaying the rmse with training using best estimator
best_preds = best_estimator.predict(X)
train_rmse = rmse_scorer(y, best_preds)
print('Train rmse ', train_rmse)
# Test rmse (note: this is the proper test set)

#displaying the rmse with testing using best estimator
best_preds = best_estimator.predict(Xtest)
test_rmse = rmse_scorer(ytest, best_preds)
print('Test rmse ',test_rmse)
```

```
Train rmse  0.07884099144651642
Test rmse  0.07954707193202129
```

[83]:
```
""" TODO
Plot the test set predictions for the best model compared with
the ground truth and the test set predictions from the linear model,
for 1170 to 1187 seconds
"""

#plotting the data
xlim = [1170, 1187]
legends = ['Best Model Test Preds', 'Ground Truth', 'Linear Test Preds']
fig, axs = plt.subplots(1)
axs.plot(time_test, best_preds)
axs.plot(time_test, ytest)
axs.plot(time_test, preds)
axs.set_xlabel('Time (s)')
axs.set_ylabel(r'$\tau \;(Nm)$')
axs.legend(legends, loc='upper left')
axs.set_xlim(xlim)
```

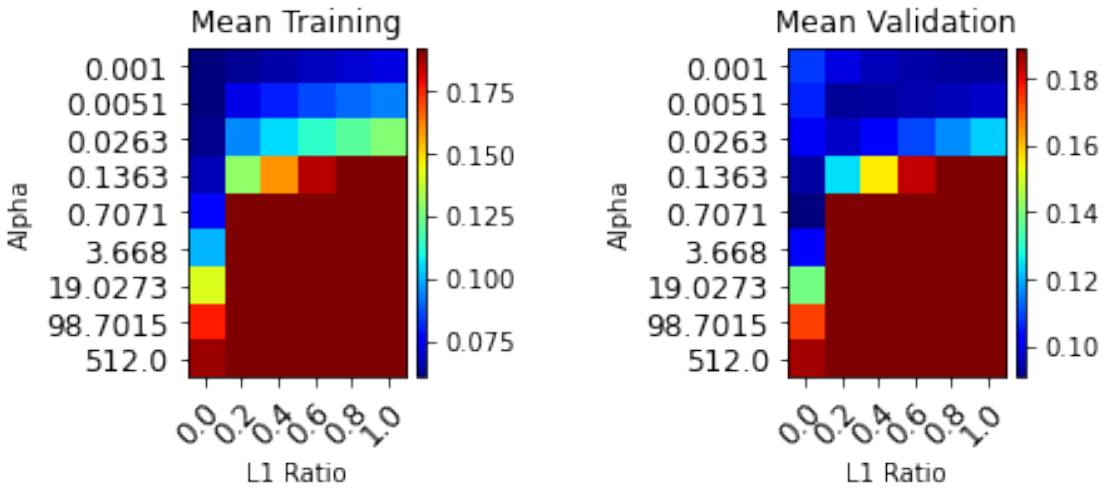
[83]:
```
(1170.0, 1187.0)
```



[189]:
```
""" TODO
Plot the mean training and validation results from the grid search as a
colormap, for the alpha (y-axis) vs the l1 ratio (x-axis). Use two subplots,
subplot(1,2,1) for the training set performance and subplot(1,2,2) for the
validation set performance. You can use imshow or matshow to display colormaps.
Make sure to include appropriate labels, ticks, and colorbars. Use the imshow␣
 ↪function
within matplotlib.pyplot.
"""
#creating a color graph for mean training
from mpl_toolkits.axes_grid1 import make_axes_locatable
fig, axs = plt.subplots(2)
fig.subplots_adjust(wspace=1.5)
axs[0] = plt.subplot(1,2,1)
axs[0].set_title('Mean Training')
axs[0].imshow(mean_train_score.reshape(nalphas, nl1_ratios), cmap='jet')
axs[0].set_xticks(np.arange(nl1_ratios))
axs[0].set_yticks(np.arange(nalphas))
axs[0].set_xticklabels(np.around(l1_ratios,4), fontsize=12)
axs[0].set_yticklabels(np.around(alphas,4), fontsize=12)
plt.setp(axs[0].get_xticklabels(), rotation=45, ha='right',␣
 ↪rotation_mode='anchor')
plt.xlabel('L1 Ratio')
plt.ylabel('Alpha')
```

```
img = axs[0].imshow(mean_train_score.reshape(nalphas, nl1_ratios), cmap='jet')
divider = make_axes_locatable(axs[0])
cax = divider.append_axes("right", size="5%", pad=0.05)
plt.colorbar(img, cax=cax)

#creating a color graph for mean validation
axs[1] = plt.subplot(1,2,2)
axs[1].set_title('Mean Validation')
axs[1].imshow(mean_test_score.reshape(nalphas, nl1_ratios), cmap='jet')
axs[1].set_xticks(np.arange(nl1_ratios))
axs[1].set_yticks(np.arange(nalphas))
axs[1].set_xticklabels(np.around(l1_ratios,4), fontsize=12)
axs[1].set_yticklabels(np.around(alphas,4), fontsize=12)
plt.setp(axs[1].get_xticklabels(), rotation=45, ha='right',␣
 ↪rotation_mode='anchor')
plt.xlabel('L1 Ratio')
plt.ylabel('Alpha')
img = axs[1].imshow(mean_test_score.reshape(nalphas, nl1_ratios), cmap='jet')
divider = make_axes_locatable(axs[1])
cax = divider.append_axes("right", size="5%", pad=0.05)
plt.colorbar(img, cax=cax)
```

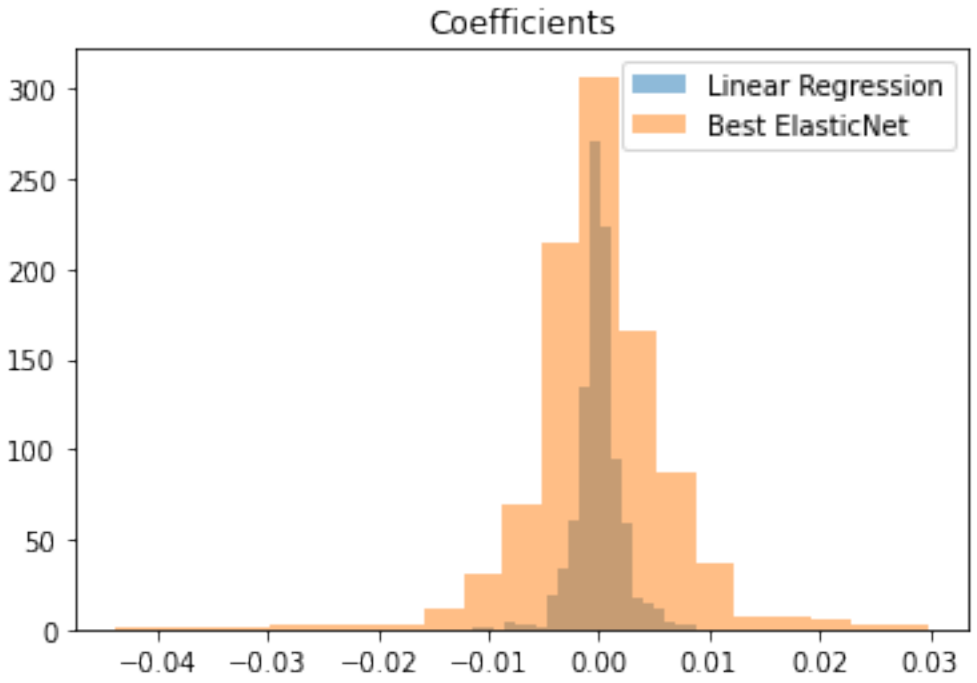
[189]: <matplotlib.colorbar.Colorbar at 0x7fa0a9769460>



[214]:
```
""" TODO
Generate a plot that contains two overlapping histograms:
- Coefficients discovered by LinearRegression
- Coefficients discovered by the best ElasticNet
    (best is relative to the validation performnce)
"""
```

```
#plotting the coefficients for both linear regression and elasticnet
nbins = 21
best_coef = best_estimator.coef_
linear_coef = linear.coef_
plt.title('Coefficients')
plt.hist(best_coef, bins=nbins, alpha=.5, label='Linear Regression')
plt.hist(linear_coef, bins=nbins, alpha=.5, label='Best ElasticNet')
plt.legend()
```

[214]: <matplotlib.legend.Legend at 0x7fa0b7be8e80>



[ ]: