homework7

October 29, 2019

NAME: Zhi Li
SECTION: 113523595
CS 5970: Machine Learning Practices

1 Homework 7: Model Comparisons

1.1 Assignment Overview

Generally, it's helpful to first read through the entire notebook before writing any code to obtain a sense of the overall program structure before you start coding.

Follow the TODOs and read through and understand any provided code.

1.1.1 Task

For this assignment, you'll be comparing different models after performing holistic cross validation to find the best parameter sets for various sizes of the training data.

1.1.2 Data set

The BMI data will be utilized. Recall:
* *MI* files contain data with the number of activations 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. * *dtheta* files record the angular velocity of the shoulder (in column 0) and the elbow (in column 1) for each time point.
* *torque* files record the torque of the shoulder (in column 0) and the elbow (in column 1) for each time point. * *time* files record the actual time stamp of each time point.

1.1.3 Objectives

- Understanding regularization using **holistic cross validation**
- Training set size sensitivity analysis

- Model selection

1.1.4 Notes

- Do not save work within the ml_practices folder

1.1.5 General References

- Guide to Jupyter
- Python Built-in Functions
- Python Data Structures
- Numpy Reference
- Numpy Cheat Sheet
- Summary of matplotlib
- DataCamp: Matplotlib
- Pandas DataFrames
- Sci-kit Learn Linear Models
- Sci-kit Learn Ensemble Models
- Sci-kit Learn Metrics
- Sci-kit Learn Model Selection
- SciPy Paired t-test for Dependent Samples
- Student's t-test
- Understanding Paired t-tests

[1]:
```
import pandas as pd
import numpy as np
import scipy.stats as stats
import os, re, fnmatch
import pathlib, itertools, time
import matplotlib.pyplot as plt

from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.metrics import explained_variance_score
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.externals import joblib


FIGW = 10
FIGH = 6
FONTSIZE = 12

HOME_DIR = pathlib.Path.home()

plt.rcParams['figure.figsize'] = (FIGW, FIGH)
```

```python
plt.rcParams['font.size'] = FONTSIZE

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

%matplotlib inline
```

[2]:
```python
"""
Display current working directory of this notebook. If you are using
relative paths for your data, then it needs to be relative to the CWD.
"""
pathlib.Path.cwd()
```

[2]: PosixPath('/home/jovyan')

2 LOAD DATA

[3]:
```python
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: File specification potentially including 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 list of Pandas objects;
    # Each from a file in the directory matching the filebase
    lst = [pd.read_csv(directory + "/" + file, delim_whitespace=True).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]:
```python
""" PROVIDED
Load the BMI data from all the folds, using read_bmi_file_set()
"""
# TODO: might need to change directory
dir_name = str(HOME_DIR / 'ml_practices/imports/datasets/bmi/DAT6_08')

MI_folds = read_bmi_file_set(dir_name, 'MI_fold*')
```

```python
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]:
```python
""" PROVIDED
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  (1193, 960) (1193, 2) (1193, 2) (1193, 2) (1193, 1)
FOLD  1  (1104, 960) (1104, 2) (1104, 2) (1104, 2) (1104, 1)
FOLD  2  (1531, 960) (1531, 2) (1531, 2) (1531, 2) (1531, 1)
FOLD  3  (1265, 960) (1265, 2) (1265, 2) (1265, 2) (1265, 1)
FOLD  4  (1498, 960) (1498, 2) (1498, 2) (1498, 2) (1498, 1)
FOLD  5  (1252, 960) (1252, 2) (1252, 2) (1252, 2) (1252, 1)
FOLD  6  (1375, 960) (1375, 2) (1375, 2) (1375, 2) (1375, 1)
FOLD  7  (1130, 960) (1130, 2) (1130, 2) (1130, 2) (1130, 1)
FOLD  8  (1247, 960) (1247, 2) (1247, 2) (1247, 2) (1247, 1)
FOLD  9  (1257, 960) (1257, 2) (1257, 2) (1257, 2) (1257, 1)
FOLD 10  (1265, 960) (1265, 2) (1265, 2) (1265, 2) (1265, 1)
FOLD 11  (1146, 960) (1146, 2) (1146, 2) (1146, 2) (1146, 1)
FOLD 12  (1225, 960) (1225, 2) (1225, 2) (1225, 2) (1225, 1)
FOLD 13  (1238, 960) (1238, 2) (1238, 2) (1238, 2) (1238, 1)
FOLD 14  (1570, 960) (1570, 2) (1570, 2) (1570, 2) (1570, 1)
FOLD 15  (1359, 960) (1359, 2) (1359, 2) (1359, 2) (1359, 1)
FOLD 16  (1579, 960) (1579, 2) (1579, 2) (1579, 2) (1579, 1)
FOLD 17  (1364, 960) (1364, 2) (1364, 2) (1364, 2) (1364, 1)
FOLD 18  (1389, 960) (1389, 2) (1389, 2) (1389, 2) (1389, 1)
FOLD 19  (1289, 960) (1289, 2) (1289, 2) (1289, 2) (1289, 1)
```

3 PARAMETER SET LIST

[6]:
```python
def generate_paramsets(param_lists):
    '''
    Construct the Cartesian product of the parameters
    PARAMS:
        params_lists: dict of lists of values to try for each parameter.
                      keys of the dict are the names of the parameters
                      values are lists of values to try for the
                      corresponding parameter
    RETURNS: a list of dicts that make up the Cartesian product of the
    parameters
    '''
    keys, values = zip(*param_lists.items())
    # Determines cartesian product of parameter values
    combos = itertools.product(*values)
    # Constructs list of dictionaries
    combos_dicts = [dict(zip(keys, vals)) for vals in combos]
    return list(combos_dicts)
```

4 PERFORMANCE EVALUTION

[7]:
```python
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_rads = np.sqrt(mse)
    rmse_degs = rmse_rads * 180 / np.pi
    return mse, rmse_rads, rmse_degs

def score_eval(model, X, y, preds):
    '''
    Compute the model predictions and corresponding scores, for an
    already trained model.
    PARAMS:
        model: model to predict with
        X: input feature data
        y: true output for X
        preds: predicted output for X
    RETURNS: results as a dictionary of numpy arrays
        mse: mean squared error for each column
        rmse_rads: rMSE in radians
        rmse_deg: rMSE in degrees
        evar: explained variance, best is 1.0
```

```
        score: score computed by the models score() method
    '''
    score = model.score(X, y)

    mse, rmse_rads, rmse_degs = mse_rmse(y, preds)
    evar = explained_variance_score(y, preds)

    # Dictionary of numpy arrays. The numpy arrays must
    # be row vectors, where each element is the result
    # for a different output, when using multiple regression.
    # The keys of the dictionary are the name of the performance
    # metric, and the values are the numpy row vectors
    results = {'mse': np.reshape(mse, (1, -1)),
               'rmse_rads': np.reshape(rmse_rads, (1, -1)),
               'rmse_degs': np.reshape(rmse_degs, (1, -1)),
               'evar': np.reshape(evar, (1, -1)),
               'score': np.reshape(score, (1, -1)),
              }
    return results
```

5 CROSS VALIDATION

[8]:
```
""" TODO
Complete KFoldHolisticCrossValidation implementation
General Procedure:
+ iter over hyper-parameter sets
  1. set hyper-parameters of the model
  2. iter over train set sizes
     a. iter over data set splits/rotations
         i. train the model
        ii. evaluate the model on train, val, and test sets
       iii. record the results
     b. record the results by size
  3. record the results by hyper-parameter set
"""
class KFoldHolisticCrossValidation():
    def __init__(self, model, paramsets, eval_func, opt_metric,
                 maximize_opt_metric=False, trainsizes=[1], rotation_skip=1):
        ''' TODO
        Object for managing and performing cross validation for a given model␣
↪for
        a list of parameter sets and train set sizes. Note, train set size is␣
↪in
        terms of number of folds (not samples)
        PARAMS:
```

```
            model: base ML model

            paramsets: list of dicts of parameter sets to give to the model

            eval_func: function used to evaluate/score the model
                       The eval_func must have the following arguments: model,␣
↪X,
                       ytrue, ypreds and return a dict of numpy arrays with␣
↪shape
                       1-by-n, where n is the number of outputs if using␣
↪multiple
                       regression.
                       template function header: eval_func(model, X, y, preds)
                       template output: {'metrics1':1_by_n_array, ...}

            opt_metric: the optized metric. one of the metric key names␣
↪returned
                        from eval_func to use to pick the best parameter sets

            maximize_opt_metric: True if opt_metric is maximized; False if␣
↪minimized

            trainsizes: list of training set sizes (in number of folds) to try

            rotation_skip: build model and evaluate every ith rotation (1=all␣
↪possible
                            rotations; 2=every other rotation, etc.)
        '''
        # TODO: set the class variables
        self.model = model#TODO
        self.paramsets = paramsets#TODO
        self.trainsizes = trainsizes#TODO
        self.eval_func = eval_func#TODO
        self.opt_metric = opt_metric + '_mean'
        self.maximize_opt_metric = maximize_opt_metric
        self.rotation_skip = rotation_skip

        # Results attributes
        # Full recording of all results for all paramsets, sizes, rotations,
        # and metrics. This is a list of dictionaries for each paramset
        self.results = None
        # Validation summary report of all means and standard deviations for
        # all metrics, for all paramsets, and sizes. This is a 3D s-by-r-by-p
        # numpy array. Where s is the number of sizes, r the number of summary
        # metrics +2, and p is the number of paramsets
        self.report_by_size = None
```

```
        # List of the indices of the best paramset for each size
        self.best_param_inds = None

    def perform_cross_validation(self, all_Xfolds, all_yfolds, trainsize,␣
↪verbose=0):
        ''' TODO: This is where the bulk of the work will be done
        Perform cross validation for a singular train set size and single␣
↪hyper-parameter
        set, by evaluating the model's performance over multiple data set␣
↪rotations all
        of the same size.

        NOTE: This function assumes the hyper-parameters have already been set␣
↪in the model

        PARAMS:
            all_Xfolds: list containing all of the input data folds
            all_yfolds: list containing all of the output data folds
            trainsize: number of folds to use for training
            verbose: flag to display simple debugging information

        RETURNS: train, val, and test set results for all rotations of the data␣
↪sets and
                 the summary (i.e. the averages over all the rotations) of the␣
↪results.
                 results is a dictionary of dictionaries of r-by-n numpy arrays.
↪ Where r
                 is the number of rotations, and n is the number of outputs␣
↪from the model.
                 summary is a dictionary of dictionaries of 1-by-n numpy arrays.
↪

                 General form:
                     results.keys() = ['train', 'val', 'test']

                     results['train'].keys() = ['metric1', 'metric2', ...]

                     results['train']['metric1'] = numpy_array

                     results =
                     {
                         'train':
                                 {
                                     'mse'       : r_by_n_numpy_array,
                                     'rmse_rads': r_by_n_numpy_array,
                                     'rmse_degs': r_by_n_numpy_array,
```

```
                                ...
                            },
                    'val'  : {...},
                    'test' : {...}
                }

                summary =
                {
                    'train':
                            {
                                'mse_mean'       : 1_by_n_numpy_array,
                                'mse_std'        : 1_by_n_numpy_array,
                                'rmse_rads_mean': 1_by_n_numpy_array,
                                'rmse_rads_std' : 1_by_n_numpy_array,
                                ...
                            },
                    'val'  : {...},
                    'test' : {...}
                }

                For example, you can access the MSE results for the␣
↪validation
                set like so:
                    results['train'][metric]
                For example, you can access the summary (i.e. the average␣
↪results
                over all the rotations) for the test set for the rMSE in␣
↪degrees
                like so:
                    summary['test']['rmse_degs_mean']
        '''

        # Verify a valid train set size was provided
        nfolds = len(all_Xfolds)
        if trainsize > nfolds - 2:
            err_msg = "ERROR: KFoldHolisticCrossValidation.
↪perform_cross_validation() - "
            err_msg += "trainsize (%d) cant be more than nfolds (%d) - 2" %␣
↪(trainsize, nfolds)
            raise ValueError(err_msg)

        # Set up results recording for each rotation
        results = {'train': None, 'val': None, 'test': None}
        summary = {'train': {}, 'val': {}, 'test': {}}

        model = self.model
        evaluate = self.eval_func
```

```
        # TODO: Rotate through the data to try different train, val, and test␣
↪sets
        for rotation in range(0, nfolds, self.rotation_skip):
            # TODO: Determine fold indices for train, val, and test set.
            #       The val and tests are each only 1 fold
            trainfolds = (np.arange(0, trainsize)+rotation)%nfolds # TODO
            valfold = (nfolds-2+rotation)%nfolds# TODO
            testfold = (nfolds-1+rotation)%nfolds# TODO

            # TODO: Construct train set by concatenating the individual␣
↪training
            #       folds together (hint: see np.take() and np.concatenate())
            X = np.concatenate([all_Xfolds[trainfold] for trainfold in␣
↪trainfolds])# TODO
            y = np.concatenate([all_yfolds[trainfold] for trainfold in␣
↪trainfolds])# TODO

            # TODO: Construct validation set. Hint: this is always one fold
            Xval = all_Xfolds[valfold]# TODO
            yval = all_yfolds[valfold]# TODO

            # TODO: Construct test set
            Xtest = all_Xfolds[testfold]# TODO
            ytest = all_yfolds[testfold]# TODO

            # DEBUGGING
            if verbose:
                print("TRAIN", X.shape, y.shape, trainfolds)
                print("VAL", Xval.shape, yval.shape, valfold)
                print("TEST", Xtest.shape, ytest.shape, testfold)

            # TODO: Train model using the training set
            model.fit(X,y)

            # TODO: Predict with the model for train, val, and test sets
            preds = model.predict(X)#TODO
            preds_val = model.predict(Xval)#TODO
            preds_test = model.predict(Xtest)#TODO

            # TODO: Evaluate the model for each set
            res_train = evaluate(model, X,y,preds)#TODO
            res_val = evaluate(model, Xval,yval,preds_val)#TODO
            res_test = evaluate(model, Xtest,ytest,preds_test)#TODO
#             print(res_train)

            # Record the train, val, and test set results. These are dicts
```

```
            # of result metrics, returned by the evaluate function
            # TODO: For the first rotation, store the results from evaluating
            #       with the train, val, and tests by setting the values of the␣
↪
            #       appropriate items within the results dict
            if results['train'] is None:
                results['train'] = res_train#TODO
                results['val'] = res_val#TODO
                results['test'] = res_test#TODO
            else:
                # Append the results for each rotation
                for metric in res_train.keys():
                    results['train'][metric] = np.
↪append(results['train'][metric],
                                                     res_train[metric],␣
↪axis=0)
                    results['val'][metric] = np.append(results['val'][metric],
                                                   res_val[metric], axis=0)
                    results['test'][metric] = np.append(results['test'][metric],
                                                    res_test[metric],␣
↪axis=0)
#             print(results)

        # Compute and record the mean and standard deviation for the given size␣
↪for each metric
        for metric in results['train'].keys():
            for stat_set in ['train', 'val', 'test']:
                summary[stat_set][metric+'_mean'] = np.
↪mean(results[stat_set][metric],
                                                        axis=0).reshape(1,␣
↪-1)
                summary[stat_set][metric+'_std'] = np.
↪std(results[stat_set][metric],
                                                      axis=0).reshape(1, -1)

        return results, summary

    def grid_cross_validation(self, all_Xfolds, all_yfolds, verbose=0):
        ''' TODO
        (MAIN PROCEDURE) Perform cross validation for multiple sets of
        parameters and train set sizes. Calls self.perform_cross_validation().
        This is the procedure that executes cross validation for all parameter
        sets and all sizes.

        PARAMS:
            all_Xfolds: all the input data folds (list of folds, as it was
```

```
                    loaded from the files)
            all_yfolds: all the output data folds (list of folds)
            verbose: flag to print out simple debugging information

        RETURNS: best parameter set for each train set size as a list of
                 parameter indices. Additionally, returns self.report_by_size,
                 the 3D array of validation means (overall rotations) for all
                 paramsets, for each metric, for all sizes. The structure of
                 the returned object is a dictionary of the following form:
                 {
                   'report_by_size' : self.report_by_size,
                   'best_param_inds': self.best_param_inds
                 }
        '''
        sizes = self.trainsizes
        paramsets = self.paramsets
        nparamsets = len(paramsets)
        print("nparamsets", nparamsets)

        # Set up all results
        all_results = []

        # Iterate over parameter sets
        for params in paramsets:
            # Set up paramset results
            param_res = []
            param_smry = None

            # Set model parameters
            print("Current paramset\n", params)
            self.model.set_params(**params)

            # Iterate over the different train set sizes
            for size in sizes:
                # TODO: Cross-validation for the current model and train set␣
↪size
                res, smry = self.
↪perform_cross_validation(all_Xfolds,all_yfolds, size,verbose)# TODO

                # Save the results
                param_res.append(res)
                # Save the mean and standard deviation statistics (summary)
                if param_smry is None: param_smry = smry
                else:
                    # For each metric measured, append the summary results
                    for metric in smry['train'].keys():
                        for stat_set in ['train', 'val', 'test']:
```

```
                            stat = smry[stat_set][metric]
                            param_smry[stat_set][metric] = np.
↪append(param_smry[stat_set][metric],
                                                                        stat,␣
↪axis=0)

            # Append the results and summary for the parameter set
            all_results.append({'params':params, 'results':param_res,
                                'summary':param_smry})

        # Generate reports and determine best params for each size
        self.results = all_results
        self.report_by_size = self.get_reports()
        self.best_param_inds = self.get_best_params(self.opt_metric,
                                                    self.maximize_opt_metric)
        return {'report_by_size':self.report_by_size,
                'best_param_inds':self.best_param_inds}

    def get_reports(self):
        ''' PROVIDED
        Get the mean validation summary of all the parameters for each size
        for all metrics. This is used to determine the best parameter set
        for each size

        RETURNS: the report_by_size as a 3D s-by-r-by-p array. Where s is
                 the number of train sizes tried, r is the number of summary
                 metrics evaluated+2, and p is the number of parameter sets.
        '''
        results = self.results
        sizes = np.reshape(self.trainsizes, (1, -1))

        nsizes = sizes.shape[1]
        nparams = len(results)

        # Set up the reports objects
        metrics = list(results[0]['summary']['val'].keys())
        colnames = ['params', 'size'] + metrics
        report_by_size = np.empty((nsizes, len(colnames), nparams),␣
↪dtype=object)

        # Determine the mean val for each parameter set for each size for all␣
↪metrics
        for p, paramset_result in enumerate(results):
            params = paramset_result['params']
            res_val = paramset_result['summary']['val']
```

```python
            # Compute the mean val performance for each train size for each␣
↪metric
            means_by_size = [np.mean(res_val[metric], axis=1) for metric in␣
↪metrics]
            # Include the train set sizes into the report
            means_by_size = np.append(sizes, means_by_size, axis=0)
            # Include the parameter sets into the report
            param_strgs = np.reshape([str(params)]*nsizes, (1, -1))
            means_by_size = np.append(param_strgs, means_by_size, axis=0).T
            # Append the parameter set means into the report
            report_by_size[:,:,p] = means_by_size
        return report_by_size

    def get_best_params(self, opt_metric, maximize_opt_metric):
        ''' PROVIDED (Do read through all the provided code)
        Determines the best parameter set for each train size, based
        on a specific metric.

        PARAMS:
            opt_metric: optimized metric. one of the metrics returned
                        from eval_func, with '_mean' appended for the
                        summary stat. This is the mean metric used to
                        determine the best parameter set for each size

            maximize_opt_metric: True if the max of opt_metric should be
                                 used to determine the best parameters.
                                 False if the min should be used.
        RETURNS: list of best parameter set indicies for each size
        '''
        results = self.results
        report_by_size = self.report_by_size

        metrics = list(results[0]['summary']['val'].keys())

        # Determine best params for each size, for the optimized metric
        best_param_inds = None
        metric_idx = metrics.index(opt_metric)

        if maximize_opt_metric:
            # Add two for the additional cols for params and size
            best_param_inds = np.argmax(report_by_size[:, metric_idx+2, :],␣
↪axis=1)
        else:
            best_param_inds = np.argmin(report_by_size[:, metric_idx+2, :],␣
↪axis=1)
        # Return list of best params indices for each size
        return best_param_inds
```

```
def get_best_params_strings(self):
    ''' PROVIDED
    Generates a list of strings of the best params for each size
    RETURNS: list of strings of the best params for each size
    '''
    best_param_inds = self.best_param_inds
    results = self.results
    return [str(results[p]['params']) for p in best_param_inds]

def get_report_best_params_for_size(self, size):
    ''' PROVIDED
    Get the mean validation summary for the best parameter set
    for a specific size for all metrics.
    PARAMS:
        size: index of desired train set size for the best
              paramset to come from. Size here is the index in
              the trainsizes list, NOT the actual number of folds.
    RETURNS: the best parameter report for the size as an s-by-m
             dataframe. Where each row is for a different size, and
             each column is for a different summary metric.
    '''
    best_param_inds = self.best_param_inds
    report_by_size = self.report_by_size

    bp_index = best_param_inds[size]

    metrics = list(self.results[0]['summary']['val'].keys())
    colnames = ['params', 'size'] + metrics
    report_best_params_for_size = pd.DataFrame(report_by_size[:,:,bp_index],
                                               columns=colnames)
    return report_best_params_for_size

def plot_cv(self, foldsindices, results, summary, metrics, size):
    ''' PROVIDED
    Plotting function for after perform_cross_validation(),
    displaying the train and val set performances for each rotation
    of the training set.

    PARAMS:
        foldsindices: indices of the train sets tried
        results: results from perform_cross_validation()
        summary: mean and standard deviations of the results
        metrics: list of result metrics to plot. Available metrics
                 are the keys in the dict returned by eval_func
        size: train set size
```

```
        RETURNS: the figure and axes handles
        '''
        nmetrics = len(metrics)

        # Initialize figure plots
        fig, axs = plt.subplots(nmetrics, 1, figsize=(12,6))
        fig.subplots_adjust(hspace=.4)
        # When 1 metric is provided, allow the axs to be iterable
        axs = np.array(axs).ravel()

        # Construct each subplot
        for metric, ax in zip(metrics, axs):
            # Compute the mean for multiple outputs
            res_train = np.mean(results['train'][metric], axis=1)
            res_val = np.mean(results['val'][metric], axis=1)
            # Plot
            ax.plot(foldsindices, res_train, label='train')
            ax.plot(foldsindices, res_val, label='val')
            ax.set(ylabel=metric)
        axs[0].legend(loc='upper right')
        axs[0].set(xlabel='Fold Index')
        axs[0].set(title='Performance for Train Set Size ' + str(size))
        return fig, axs

    def plot_param_train_val(self, metrics, paramidx=0, view_test=False):
        ''' PROVIDED
        Plotting function for after grid_cross_validation(),
        displaying the mean (summary) train and val set performances
        for each train set size.

        PARAMS:
            metrics: list of summary metrics to plot. '_mean' or '_std'
                     must be append to the end of the base metric name.
                     These base metric names are the keys in the dict
                     returned by eval_func
            paramidx: parameter set index
            view_test: flag to view the test set results

        RETURNS: the figure and axes handles
        '''
        sizes = self.trainsizes
        results = self.results

        summary = results[paramidx]['summary']
        params = results[paramidx]['params']

        nmetrics = len(metrics)
```

```python
        # Initialize figure plots
        fig, axs = plt.subplots(nmetrics, 1, figsize=(12,6))
        fig.subplots_adjust(hspace=.4)
        # When 1 metric is provided, allow the axs to be iterable
        axs = np.array(axs).ravel()

        # Construct each subplot
        for metric, ax in zip(metrics, axs):
            # Compute the mean for multiple outputs
            res_train = np.mean(summary['train'][metric], axis=1)
            res_val = np.mean(summary['val'][metric], axis=1)
            # Plot
            ax.plot(sizes, res_train, label='train')
            ax.plot(sizes, res_val, label='val')
            if view_test:
                res_test = np.mean(summary['test'][metric], axis=1)
                ax.plot(sizes, res_test, label='test')
            ax.set(ylabel=metric)
        axs[-1].set(xlabel='Train Set Size (# of folds)')
        axs[0].set(title=str(params))
        axs[0].legend(loc='upper right')
        return fig, axs

    def plot_allparams_val(self, metrics):
        ''' PROVIDED
        Plotting function for after grid_cross_validation(), displaying
        mean (summary) validation set performances for each train size
        for all parameter sets for the specified metrics.

        PARAMS:
            metrics: list of summary metrics to plot. '_mean' or '_std'
                     must be append to the end of the base metric name.
                     These base metric names are the keys in the dict
                     returned by eval_func

        RETURNS: the figure and axes handles
        '''
        sizes = self.trainsizes
        results = self.results

        nmetrics = len(metrics)

        # Initialize figure plots
        fig, axs = plt.subplots(nmetrics, 1, figsize=(10,6))
        fig.subplots_adjust(hspace=.4)
        # When 1 metric is provided, allow the axs to be iterable
```

```
        axs = np.array(axs).ravel()

        # Construct each subplot
        for metric, ax in zip(metrics, axs):
            for p, param_results in enumerate(results):
                summary = param_results['summary']
                params = param_results['params']
                # Compute the mean for multiple outputs
                res_val = np.mean(summary['val'][metric], axis=1)
                ax.plot(sizes, res_val, label=str(params))
            ax.set(ylabel=metric)
        axs[-1].set(xlabel='Train Set Size (# of folds)')
        axs[0].set(title='Validation Performance')
        axs[0].legend(bbox_to_anchor=(1.02, 1), loc='upper left',
                      ncol=1, borderaxespad=0., prop={'size': 8})
        return fig, axs

    def plot_best_params_by_size(self):
        ''' PROVIDED
        Plotting function for after grid_cross_validation(), displaying
        mean (summary) train and validation set performances for the best
        parameter set for each train size for the specified metrics.

        RETURNS: the figure and axes handles
        '''
        results = self.results
        metric = self.opt_metric
        best_param_inds = self.best_param_inds
        sizes = np.array(self.trainsizes)

        # Unique set of best params for the legend
        unique_param_sets = np.unique(best_param_inds)
        lgnd_params = [self.paramsets[p] for p in unique_param_sets]

        # Initialize figure
        fig, axs = plt.subplots(2, 1, figsize=(10,6))
        fig.subplots_adjust(hspace=.4)
        # When 1 metric is provided, allow the axs to be iterable
        axs = np.array(axs).ravel()
        set_names = ['train', 'val']

        # Construct each subplot
        for i, (ax, set_name) in enumerate(zip(axs, set_names)):
            for p in unique_param_sets:
                # Obtain indices of sizes this paramset was best for
                param_size_inds = np.where(best_param_inds == p)[0]
                param_sizes = sizes[param_size_inds]
```

```
                # Compute the mean over multiple outputs for each size
                param_summary = results[p]['summary'][set_name]
                metric_scores = np.mean(param_summary[metric][param_size_inds, :
↪], axis=1)
                # Plot the param results for each size it was the best for
                ax.scatter(param_sizes, metric_scores, s=120, marker=(p+2, 1))
                #ax.grid(True)

            set_name += ' Set Performance'
            ax.set(ylabel=metric, title=set_name)

        axs[-1].set(xlabel='Train Set Size (# of folds)')
        axs[0].legend(lgnd_params, bbox_to_anchor=(1.02, 1), loc='upper left',
                      ncol=1, borderaxespad=0., prop={'size': 8})
        return fig, axs
```

6 PERFORM CROSS VALIDATION

Initialize holistic cross validation objects to explore Linear, Ridge, Lasso, and ElasticNet models.

The experiments for the ElasticNet have been provided in a file (hw7_full_crossval.pkl) due to the length of time it takes to run; however, you are welcome to re-run these experiements, for all/various train set sizes, and rotations, using score_eval as the eval_func, and rmse_degs as the metric to optimize. The file can be found in the hw7 folder in the ml_practices directory, along with this notebook.

The inputs for the models are the MI data and the outputs are the torque (you'll provide the shoulder and elbow simulataneouly, as done in the previous HW).

[9]:
```
""" PROVIDED
Holistic Cross Validation Options:
* ridge_alphas: list of alphas to try for the RIDGE model
* lasso_alphas: list of alphas to try for the LASSO model
* en_alphas: list of alphas to try for the ELASTICNET model
* l1_ratios: list of l1_ratios to try for the ELASTICNET model

* trainsizes: list of number of folds to utilize in the train set
* opt_metric: the optimized metric, returned by the eval_func, used
  to select the best parameter sets
* maximize_opt_metric: True if the opt_metric is maximized; False
  otherwise
* skip: the number of folds to skip when rotating through train sets
  of the same size
"""
ridge_alphas = [1, 10, 50, 100, 500, 1000, 10000]
lasso_alphas = [.001, .005, .01, .025, .05, .075, .1]
```

```python
en_alphas = lasso_alphas + [0.5, 1]
l1_ratios = [0.001, .025, .05, .1, .5, 1]

trainsizes = range(1, nfolds-1)
opt_metric = 'rmse_degs'
maximize_opt_metric = False
skip = 1

# True to always run cross validation, false to re-load existing run
# or run cross validation for the first time
force = False
# Tag for the filename to save the experiments to
prefix = "_full"
```

6.1 LINEAR REGRESSION

Ordinary least squares Linear Regression.

[10]:
```python
""" TODO
LinearRegression

Execute cross validation procedure for all sizes for the
LinearRegression model using grid_cross_validation().
The parameter list for the LinearRegression model is a
list with just an empty dictionary [{}]
"""
lnr_fullcvfname = "hw7" + prefix + "_linear_crossval.pkl"

model = LinearRegression()
lnr_crossval = KFoldHolisticCrossValidation(model, [{}], score_eval,
                                            opt_metric, maximize_opt_metric,
                                            trainsizes, skip)

lnr_crossval_report = None
if force or (not os.path.exists(lnr_fullcvfname)):
    # TODO: Execute cross validation procedure for all parameters and sizes
    lnr_crossval_report = lnr_crossval.grid_cross_validation(MI_folds,␣
↪torque_folds, verbose=0)# TODO
    # TODO: Save the cross validation object, use joblib.dump()
    joblib.dump(lnr_crossval, lnr_fullcvfname)

else:
    # Re-load saved crossval object instead of re-running
    lnr_crossval = joblib.load(lnr_fullcvfname)
    lnr_crossval_report = {'report_by_size': lnr_crossval.report_by_size,
                          'best_param_inds': lnr_crossval.best_param_inds}
```

```
lnr_crossval.model, lnr_crossval.rotation_skip, lnr_crossval.trainsizes
```

[10]:
```
(LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
          normalize=False), 1, range(1, 19))
```

[11]:
```
lnr_crossval
```

[11]:
```
<__main__.KFoldHolisticCrossValidation at 0x7fa97a49cf60>
```

6.2 RIDGE

$\min_w ||y - w^T X||_2^2 + \alpha ||w||_2^2$

α: amount of L_2 regularization to apply. Larger α greater penalize the model for larger weights

w: the weights from the model

X: feature or input data

y: true outputs

[12]:
```
""" TODO
RIDGE

Initialize a KFoldHolisticCrossValidation object that uses RIDGE
as the model, and the provided r_allparamsets

Execute cross validation procedure for all sizes for the Ridge
model using grid_cross_validation()
"""
r_fullcvfname = "hw7" + prefix + "_ridge_crossval.pkl"

r_param_lists = {'alpha':ridge_alphas, 'max_iter':[1e4]}
r_allparamsets = generate_paramsets(r_param_lists)
print(pd.DataFrame(r_allparamsets))

model = Ridge()
# TODO: Initialize a KFoldHolisticCrossValidation object using Ridge
r_crossval = KFoldHolisticCrossValidation(model, r_allparamsets, score_eval,
                                          opt_metric, maximize_opt_metric,
                                          trainsizes, skip)
# TODO

r_crossval_report = None
if force or (not os.path.exists(r_fullcvfname)):
    # TODO: Execute cross validation for all parameters and sizes
```

```
    r_crossval_report = r_crossval.grid_cross_validation(MI_folds,␣
↪torque_folds, verbose=0)# TODO
    # TODO: Save the cross validation object
    joblib.dump(r_crossval, r_fullcvfname)


else:
    # Re-load saved crossval object instead of re-running
    r_crossval = joblib.load(r_fullcvfname)
    r_crossval_report = {'report_by_size' : r_crossval.report_by_size,
                         'best_param_inds': r_crossval.best_param_inds}

r_crossval.model, r_crossval.rotation_skip, r_crossval.trainsizes
```

```
   alpha  max_iter
0      1   10000.0
1     10   10000.0
2     50   10000.0
3    100   10000.0
4    500   10000.0
5   1000   10000.0
6  10000   10000.0
```

[12]:
```
(Ridge(alpha=10000, copy_X=True, fit_intercept=True, max_iter=10000.0,
    normalize=False, random_state=None, solver='auto', tol=0.001),
 1,
 range(1, 19))
```

6.3 LASSO

$\min_w \frac{1}{2N}||y - w^T X||_2^2 + \alpha||w||_1$

N: the number of samples

[13]:
```
""" TODO
LASSO

Initialize a KFoldHolisticCrossValidation object that uses LASSO
as the model, and the provided l_allparamsets

Execute cross validation procedure for all sizes for the Lasso
model using grid_cross_validation()
"""
l_fullcvfname = "hw7" + prefix + "_lasso_crossval.pkl"

l_param_lists = {'alpha':lasso_alphas, 'max_iter':[1e4]}
l_allparamsets = generate_paramsets(l_param_lists)
```

```python
print(pd.DataFrame(l_allparamsets))

model = Lasso()
# TODO: Initialize a KFoldHolisticCrossValidation object using Lasso
l_crossval = KFoldHolisticCrossValidation(model, l_allparamsets, score_eval,
                                          opt_metric, maximize_opt_metric,
                                          trainsizes, skip)# TODO


l_crossval_report = None
if force or (not os.path.exists(l_fullcvfname)):
    # TODO: Execute cross validation for all parameters and sizes
    l_crossval_report = l_crossval.grid_cross_validation(MI_folds,␣
↪torque_folds, verbose=0)# TODO
    # TODO: Save the cross validation object
    joblib.dump(l_crossval, l_fullcvfname)

else:
    # Re-load saved crossval object instead of re-running
    l_crossval = joblib.load(l_fullcvfname)
    l_crossval_report = {'report_by_size' : l_crossval.report_by_size,
                         'best_param_inds': l_crossval.best_param_inds}

l_crossval.model, l_crossval.rotation_skip, l_crossval.trainsizes
```

```
   alpha  max_iter
0  0.001   10000.0
1  0.005   10000.0
2  0.010   10000.0
3  0.025   10000.0
4  0.050   10000.0
5  0.075   10000.0
6  0.100   10000.0
```

[13]:

```
(Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=10000.0,
    normalize=False, positive=False, precompute=False, random_state=None,
    selection='cyclic', tol=0.0001, warm_start=False), 1, range(1, 19))
```

6.4 ELASTICNET

$\min_w \frac{1}{2N}||y - w^T X||_2^2 + \alpha L_1 ||w||_1 + \frac{1}{2}\alpha(1 - L_1)||w||_2^2$

L_1: the L_1 ratio

[14]:

```python
""" TODO
ELASTICNET

Initialize a KFoldHolisticCrossValidation object that uses ELASTICNET
```

```
as the model, and the provided allparamsets

Execute cross validation procedure for all sizes for the ELASTICNET
model using grid_cross_validation()

Re-load the existing experiment
"""
fullcvfname = "hw7" + prefix + "_crossval.pkl"

param_lists = {'alpha':en_alphas, 'l1_ratio':l1_ratios, 'max_iter':[1e4]}
allparamsets = generate_paramsets(param_lists)
nparamsets = len(allparamsets)
print(pd.DataFrame(allparamsets))

model = ElasticNet()
crossval = KFoldHolisticCrossValidation(model, allparamsets, score_eval,
                                        opt_metric, maximize_opt_metric,
                                        trainsizes, skip)

crossval_report = None
if force or (not os.path.exists(fullcvfname)):
    # Execute cross validation for all parameters and sizes
    crossval_report = crossval.grid_cross_validation(MI_folds,
                                                     torque_folds,
                                                     verbose=0)
    # Save the cross validation object
    joblib.dump(crossval, fullcvfname)
else:
    # TODO: Re-load saved crossval object. Use joblib.load()
    crossval = joblib.load(fullcvfname)# TODO
    crossval_report = {'report_by_size' : crossval.report_by_size,
                       'best_param_inds': crossval.best_param_inds}

crossval.model, crossval.rotation_skip, crossval.trainsizes
```

```
    alpha  l1_ratio  max_iter
0   0.001     0.001   10000.0
1   0.001     0.025   10000.0
2   0.001     0.050   10000.0
3   0.001     0.100   10000.0
4   0.001     0.500   10000.0
5   0.001     1.000   10000.0
6   0.005     0.001   10000.0
7   0.005     0.025   10000.0
8   0.005     0.050   10000.0
9   0.005     0.100   10000.0
10  0.005     0.500   10000.0
```

```
11  0.005     1.000   10000.0
12  0.010     0.001   10000.0
13  0.010     0.025   10000.0
14  0.010     0.050   10000.0
15  0.010     0.100   10000.0
16  0.010     0.500   10000.0
17  0.010     1.000   10000.0
18  0.025     0.001   10000.0
19  0.025     0.025   10000.0
20  0.025     0.050   10000.0
21  0.025     0.100   10000.0
22  0.025     0.500   10000.0
23  0.025     1.000   10000.0
24  0.050     0.001   10000.0
25  0.050     0.025   10000.0
26  0.050     0.050   10000.0
27  0.050     0.100   10000.0
28  0.050     0.500   10000.0
29  0.050     1.000   10000.0
30  0.075     0.001   10000.0
31  0.075     0.025   10000.0
32  0.075     0.050   10000.0
33  0.075     0.100   10000.0
34  0.075     0.500   10000.0
35  0.075     1.000   10000.0
36  0.100     0.001   10000.0
37  0.100     0.025   10000.0
38  0.100     0.050   10000.0
39  0.100     0.100   10000.0
40  0.100     0.500   10000.0
41  0.100     1.000   10000.0
42  0.500     0.001   10000.0
43  0.500     0.025   10000.0
44  0.500     0.050   10000.0
45  0.500     0.100   10000.0
46  0.500     0.500   10000.0
47  0.500     1.000   10000.0
48  1.000     0.001   10000.0
49  1.000     0.025   10000.0
50  1.000     0.050   10000.0
51  1.000     0.100   10000.0
52  1.000     0.500   10000.0
53  1.000     1.000   10000.0
```

[14]:

```
(ElasticNet(alpha=1, copy_X=True, fit_intercept=True, l1_ratio=1,
           max_iter=10000.0, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False),
```

```
 1,
 range(1, 19))
```

7 RESULTS

7.0.1 Understand the result output structure

[15]:
```
""" PROVIDED
List KFoldHolisticCrossValidation Attributes
"""
dir(crossval)
```

[15]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'best_param_inds',
 'eval_func',
 'get_best_params',
 'get_best_params_strings',
 'get_report_best_params_for_size',
 'get_reports',
 'grid_cross_validation',
```

```
 'maximize_opt_metric',
 'model',
 'opt_metric',
 'paramsets',
 'perform_cross_validation',
 'plot_allparams_val',
 'plot_best_params_by_size',
 'plot_cv',
 'plot_param_train_val',
 'report_by_size',
 'results',
 'rotation_skip',
 'trainsizes']
```

[16]:
```
""" PROVIDED
Results attribute is a list of dictionaries. Each element, or dictionary
corresponds to the results for a single parameter set
"""
len(crossval.results), crossval.results[0].keys()
```

[16]:
```
(54, dict_keys(['params', 'results', 'summary']))
```

[17]:
```
""" PROVIDED
* crossval.results[0]['results'] is a list of dictionaries with the results
  for each size for the parameter set at index 0
* crossval.results[1]['summary'] is a dictionary of summary results for the
  train, val, and test sets for the parameter set at index 1
"""
len(crossval.results[0]['results']), crossval.results[1]['summary'].keys()
```

[17]:
```
(18, dict_keys(['train', 'val', 'test']))
```

[18]:
```
""" PROVIDED
* crossval.results[0]['results'][2] is a dictionary with the results
  for the train size at index 2 for the parameter set at index 0
* crossval.results[1]['summary']['val'] is a dictionary of summary (over the
  sizes) results for the val set for the parameter set at index 1, for all
  metrics
"""
crossval.results[0]['results'][2].keys(), crossval.results[1]['summary']['val'].
 ↪keys()
```

[18]:
```
(dict_keys(['train', 'val', 'test']),
 dict_keys(['mse_mean', 'mse_std', 'rmse_rads_mean', 'rmse_rads_std',
'rmse_degs_mean', 'rmse_degs_std', 'evar_mean', 'evar_std', 'score_mean',
'score_std']))
```

[19]:
```
""" PROVIDED
* crossval.results[0]['results'][2]['train'] is a dictionary of all results for
  the train set for the parameter set at index 0, the size at index 2, for all
  metrics
* crossval.results[1]['summary']['val']['mse_mean'] is a numpy array of␣
↪averages
  for the val set for the parameter set at index 1, for the mse. The averages␣
↪are
  computed over the sizes
"""
crossval.results[0]['results'][2]['train'].keys(), crossval.
↪results[1]['summary']['val']['mse_mean'].shape
```

[19]:
```
(dict_keys(['mse', 'rmse_rads', 'rmse_degs', 'evar', 'score']), (18, 2))
```

[20]:
```
""" PROVIDED
* crossval.results[0]['results'][2]['train']['mse'] is a dictionary of all
  results for the train set for the parameter set at index 0, the size at
  index 2, for the mse, for all rotations (there are 20 rotations when skip=1)
"""
crossval.results[0]['results'][2]['train']['mse'].shape
```

[20]:
```
(20, 2)
```

7.0.2 Best Parameters for Each Size

[21]:
```
""" PROVIDED
Results options:
* size_idx: index of the size from the list of train sizes to examine results
* metrics: list of summary (average) metrics to examine results
"""
# index 7 corresponds to train size 8
size_idx = 7
metrics = ['rmse_degs_mean', 'evar_mean']
```

[22]:
```
""" PROVIDED
Display the lists of the best parameter sets for each size for all
the models, expect the Linear model (as it has only one parameter set)
"""
print("Best Parameter Sets For Each Train Set Size")

print("RIDGE")
r_best_param_info = pd.DataFrame((r_crossval.trainsizes,
                                  r_crossval.best_param_inds,
                                  r_crossval.get_best_params_strings()),
                                  index=['train_size','param_index','paramset'])
```

```python
print(r_best_param_info.T)


print("LASSO")
l_best_param_info = pd.DataFrame((l_crossval.trainsizes,
                                  l_crossval.best_param_inds,
                                  l_crossval.get_best_params_strings()),
                                  index=['train_size','param_index','paramset'])
print(l_best_param_info.T)

print("ELASTICNET")
best_param_info = pd.DataFrame((crossval.trainsizes,
                                crossval.best_param_inds,
                                crossval.get_best_params_strings()),
                                index=['train_size', 'param_index', 'paramset'])
print(best_param_info.T)
```

```
Best Parameter Sets For Each Train Set Size
RIDGE
   train_size param_index                                paramset
0           1           5  {'alpha': 1000, 'max_iter': 10000.0}
1           2           5  {'alpha': 1000, 'max_iter': 10000.0}
2           3           5  {'alpha': 1000, 'max_iter': 10000.0}
3           4           5  {'alpha': 1000, 'max_iter': 10000.0}
4           5           5  {'alpha': 1000, 'max_iter': 10000.0}
5           6           5  {'alpha': 1000, 'max_iter': 10000.0}
6           7           5  {'alpha': 1000, 'max_iter': 10000.0}
7           8           5  {'alpha': 1000, 'max_iter': 10000.0}
8           9           5  {'alpha': 1000, 'max_iter': 10000.0}
9          10           5  {'alpha': 1000, 'max_iter': 10000.0}
10         11           5  {'alpha': 1000, 'max_iter': 10000.0}
11         12           5  {'alpha': 1000, 'max_iter': 10000.0}
12         13           5  {'alpha': 1000, 'max_iter': 10000.0}
13         14           5  {'alpha': 1000, 'max_iter': 10000.0}
14         15           5  {'alpha': 1000, 'max_iter': 10000.0}
15         16           5  {'alpha': 1000, 'max_iter': 10000.0}
16         17           5  {'alpha': 1000, 'max_iter': 10000.0}
17         18           5  {'alpha': 1000, 'max_iter': 10000.0}
LASSO
   train_size param_index                                 paramset
0           1           0  {'alpha': 0.001, 'max_iter': 10000.0}
1           2           0  {'alpha': 0.001, 'max_iter': 10000.0}
2           3           0  {'alpha': 0.001, 'max_iter': 10000.0}
3           4           0  {'alpha': 0.001, 'max_iter': 10000.0}
4           5           0  {'alpha': 0.001, 'max_iter': 10000.0}
5           6           0  {'alpha': 0.001, 'max_iter': 10000.0}
6           7           0  {'alpha': 0.001, 'max_iter': 10000.0}
```

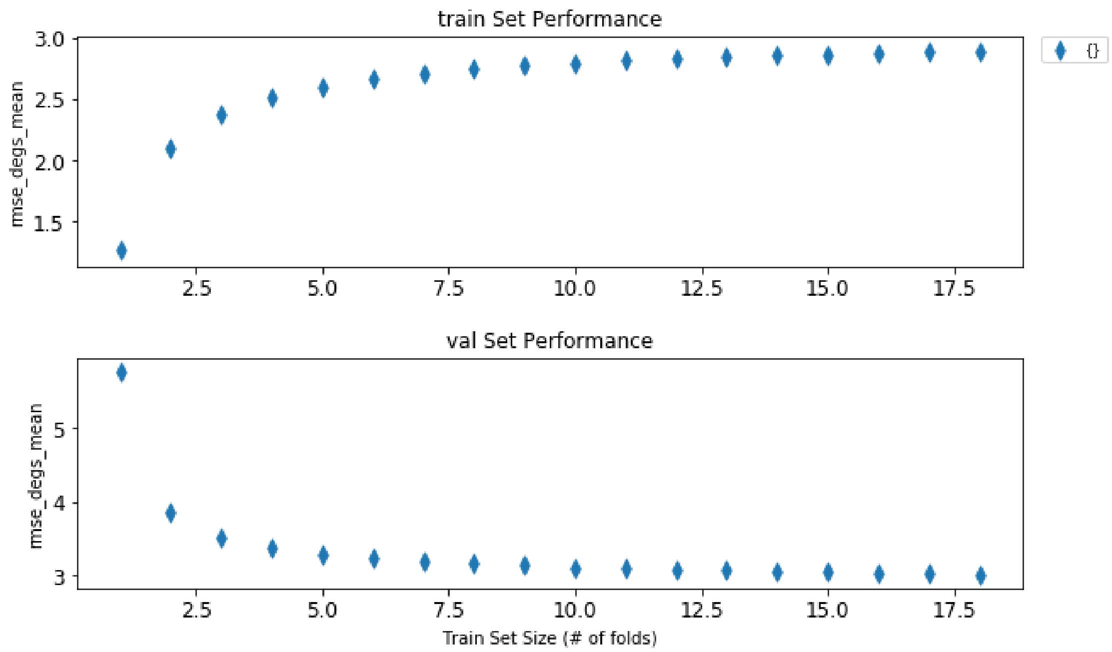
```
7          8          0  {'alpha': 0.001, 'max_iter': 10000.0}
8          9          0  {'alpha': 0.001, 'max_iter': 10000.0}
9         10          0  {'alpha': 0.001, 'max_iter': 10000.0}
10        11          0  {'alpha': 0.001, 'max_iter': 10000.0}
11        12          0  {'alpha': 0.001, 'max_iter': 10000.0}
12        13          0  {'alpha': 0.001, 'max_iter': 10000.0}
13        14          0  {'alpha': 0.001, 'max_iter': 10000.0}
14        15          0  {'alpha': 0.001, 'max_iter': 10000.0}
15        16          0  {'alpha': 0.001, 'max_iter': 10000.0}
16        17          0  {'alpha': 0.001, 'max_iter': 10000.0}
17        18          0  {'alpha': 0.001, 'max_iter': 10000.0}
ELASTICNET
   train_size param_index                                          paramset
0           1          42  {'alpha': 0.5, 'l1_ratio': 0.001, 'max_iter': …
1           2          42  {'alpha': 0.5, 'l1_ratio': 0.001, 'max_iter': …
2           3          42  {'alpha': 0.5, 'l1_ratio': 0.001, 'max_iter': …
3           4          36  {'alpha': 0.1, 'l1_ratio': 0.001, 'max_iter': …
4           5          36  {'alpha': 0.1, 'l1_ratio': 0.001, 'max_iter': …
5           6          36  {'alpha': 0.1, 'l1_ratio': 0.001, 'max_iter': …
6           7          36  {'alpha': 0.1, 'l1_ratio': 0.001, 'max_iter': …
7           8          36  {'alpha': 0.1, 'l1_ratio': 0.001, 'max_iter': …
8           9          36  {'alpha': 0.1, 'l1_ratio': 0.001, 'max_iter': …
9          10          36  {'alpha': 0.1, 'l1_ratio': 0.001, 'max_iter': …
10         11          36  {'alpha': 0.1, 'l1_ratio': 0.001, 'max_iter': …
11         12          36  {'alpha': 0.1, 'l1_ratio': 0.001, 'max_iter': …
12         13          30  {'alpha': 0.075, 'l1_ratio': 0.001, 'max_iter'…
13         14          30  {'alpha': 0.075, 'l1_ratio': 0.001, 'max_iter'…
14         15          30  {'alpha': 0.075, 'l1_ratio': 0.001, 'max_iter'…
15         16          30  {'alpha': 0.075, 'l1_ratio': 0.001, 'max_iter'…
16         17          30  {'alpha': 0.075, 'l1_ratio': 0.001, 'max_iter'…
17         18          30  {'alpha': 0.075, 'l1_ratio': 0.001, 'max_iter'…
```

7.0.3 Plot Best Parameters for Each Size

[23]:
```python
""" PROVIDED
LINEAR REGRESSION
Plot the mean (summary) train and validation set performances for
each train size for the optimized metric. Use plot_best_params_by_size()

Note: for LinearRegression, there is only one parameter set.
"""
lnr_crossval.plot_best_params_by_size()
```

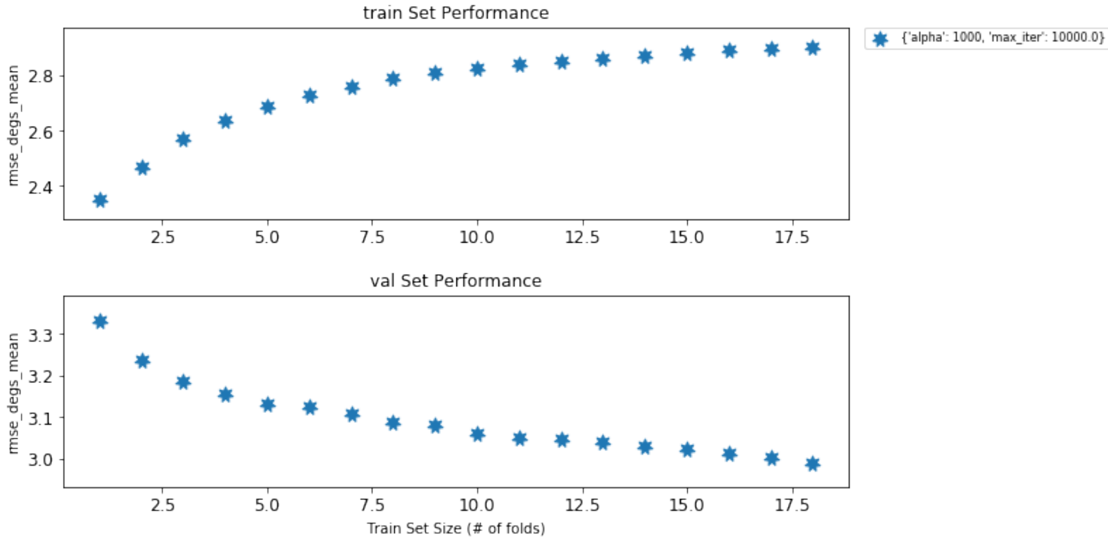
[23]:
```
(<Figure size 720x432 with 2 Axes>,
 array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fa9789d86d8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fa978a7cba8>],
```

```
        dtype=object))
```



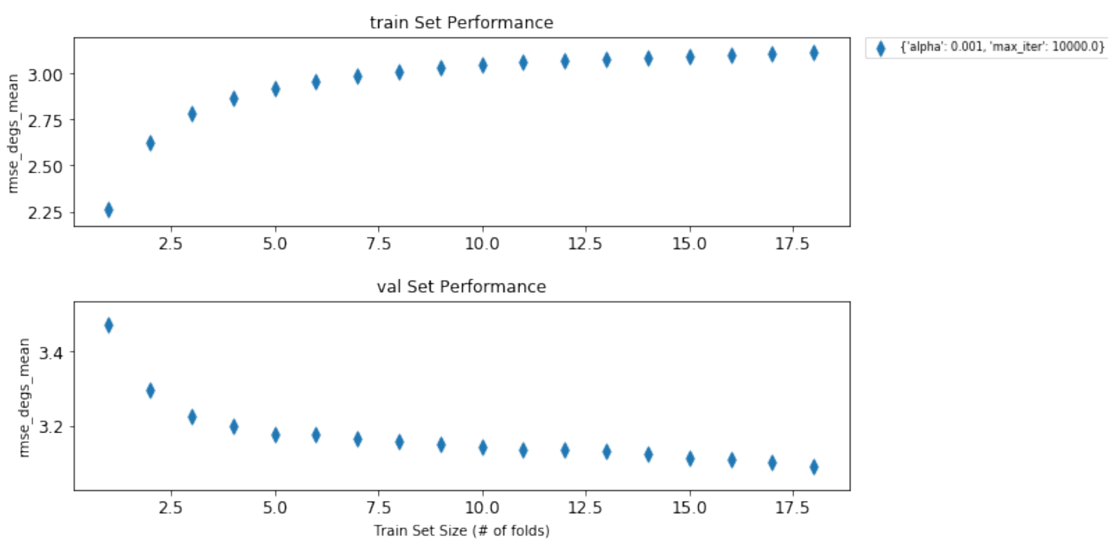
[24]:
```
""" TODO
RIDGE
Plot the mean (summary) train and validation set performances for
the best parameter set for each train size for the optimized
metrics. Use plot_best_params_by_size()
"""
r_crossval.plot_best_params_by_size()
```

[24]:
```
(<Figure size 720x432 with 2 Axes>,
 array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fa97a03ac88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fa97a1124a8>],
       dtype=object))
```



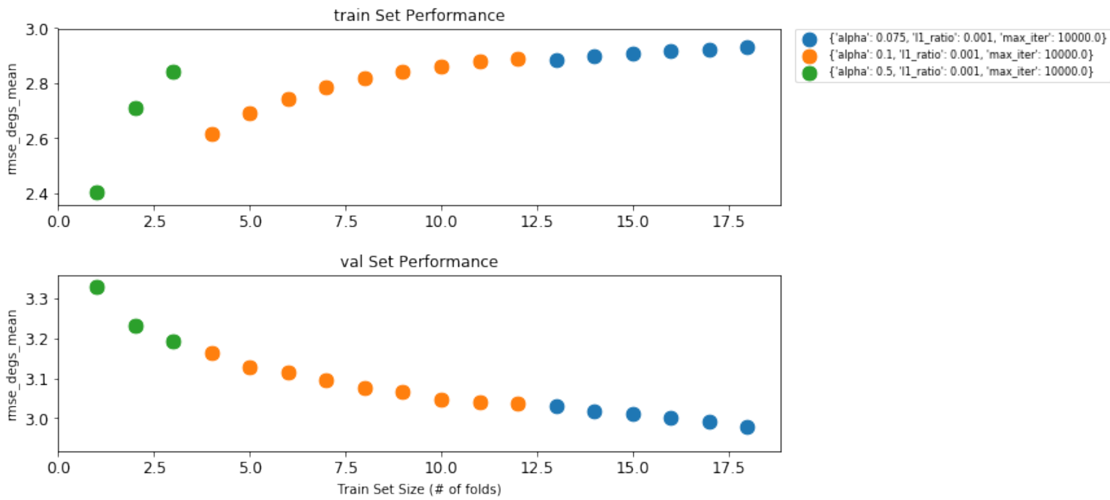
[25]:
```
""" TODO
LASSO
Plot the mean (summary) train and validation set performances for
the best parameter set for each train size for the optimized
metrics. Use plot_best_params_by_size()
"""

l_crossval.plot_best_params_by_size()
```

[25]:
```
(<Figure size 720x432 with 2 Axes>,
 array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fa97a20f860>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fa97a464828>],
       dtype=object))
```



[26]:
```
""" TODO
ELASTICNET
Plot the mean (summary) train and validation set performances for
the best parameter set for each train size for the optimized
metrics. Use plot_best_params_by_size()
"""
crossval.plot_best_params_by_size()
```

[26]:
```
(<Figure size 720x432 with 2 Axes>,
 array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fa97a6b1358>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fa97a78e0b8>],
       dtype=object))
```

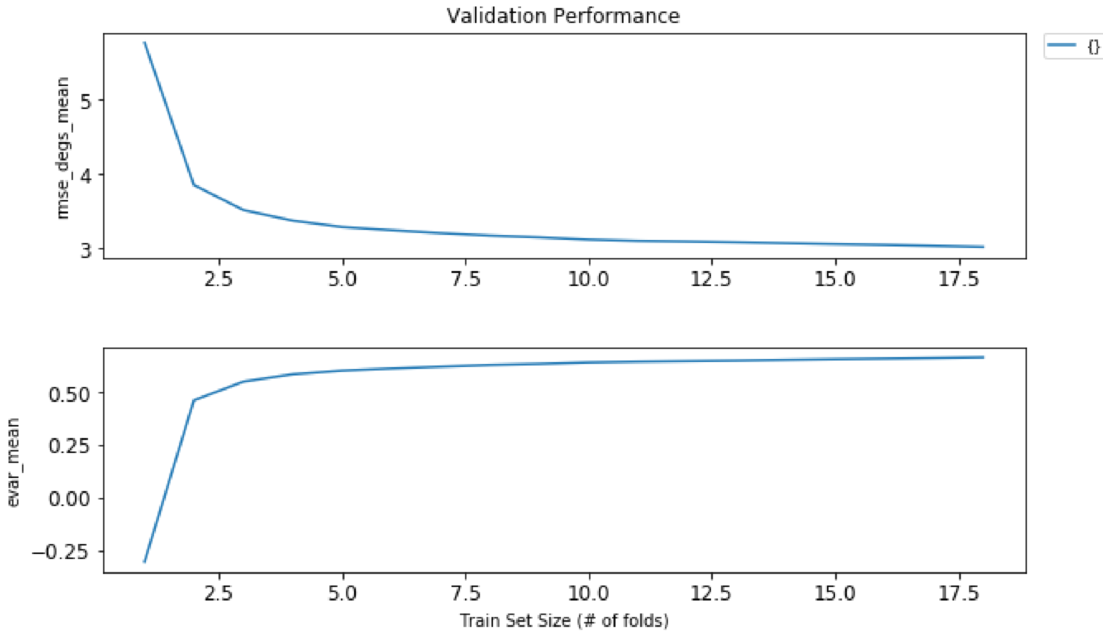


7.0.4 Plot Validation for All Parameter Sets for Each Size

[27]:
```
""" TODO
LINEAR REGRESSION
Plot the validation results for all parameter sets over all train
sizes, for the specified metrics, rmse_degs_mean and evar_mean
(this variable is declared above). Use plot_allparams_val()
"""

lnr_crossval.plot_allparams_val(metrics)
```

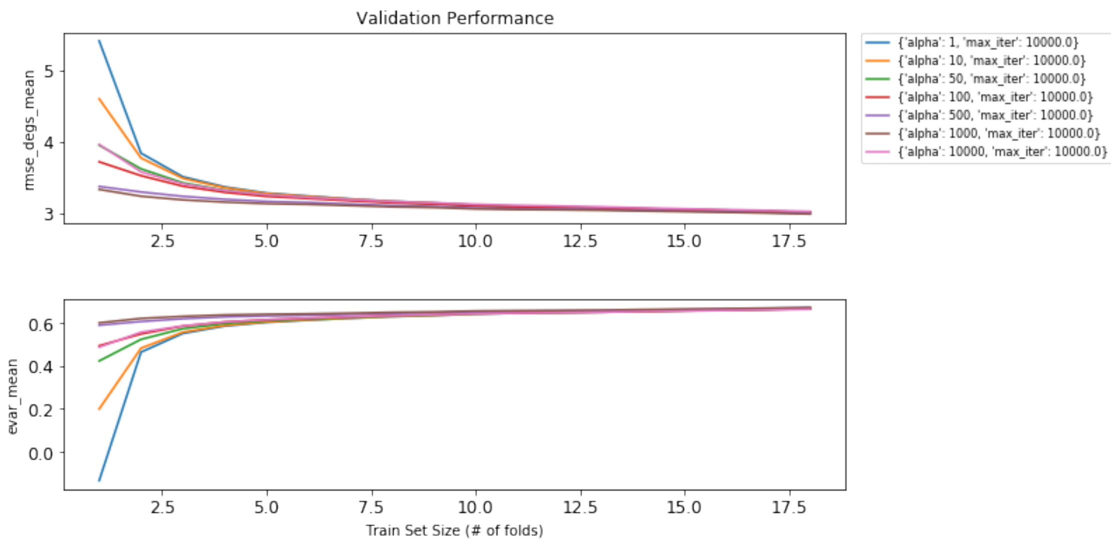
[27]:
```
(<Figure size 720x432 with 2 Axes>,
 array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fa97a7a9860>,
```

```
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fa97a282a58>],
dtype=object))
```



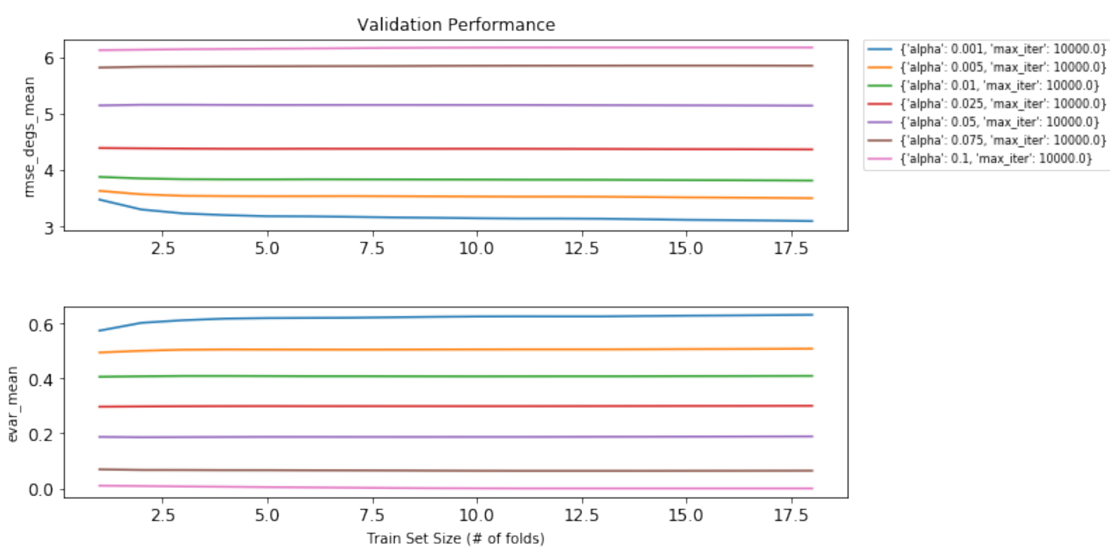
[28]:
```
""" TODO
RIDGE
Plot the validation results for all parameter sets over all train
sizes, for the specified metrics, rmse_degs_mean and evar_mean
(this variable is declared above). Use plot_allparams_val()
"""

r_crossval.plot_allparams_val(metrics)
```

[28]:
```
(<Figure size 720x432 with 2 Axes>,
 array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fa97a55ca58>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fa9789a9c88>],
       dtype=object))
```



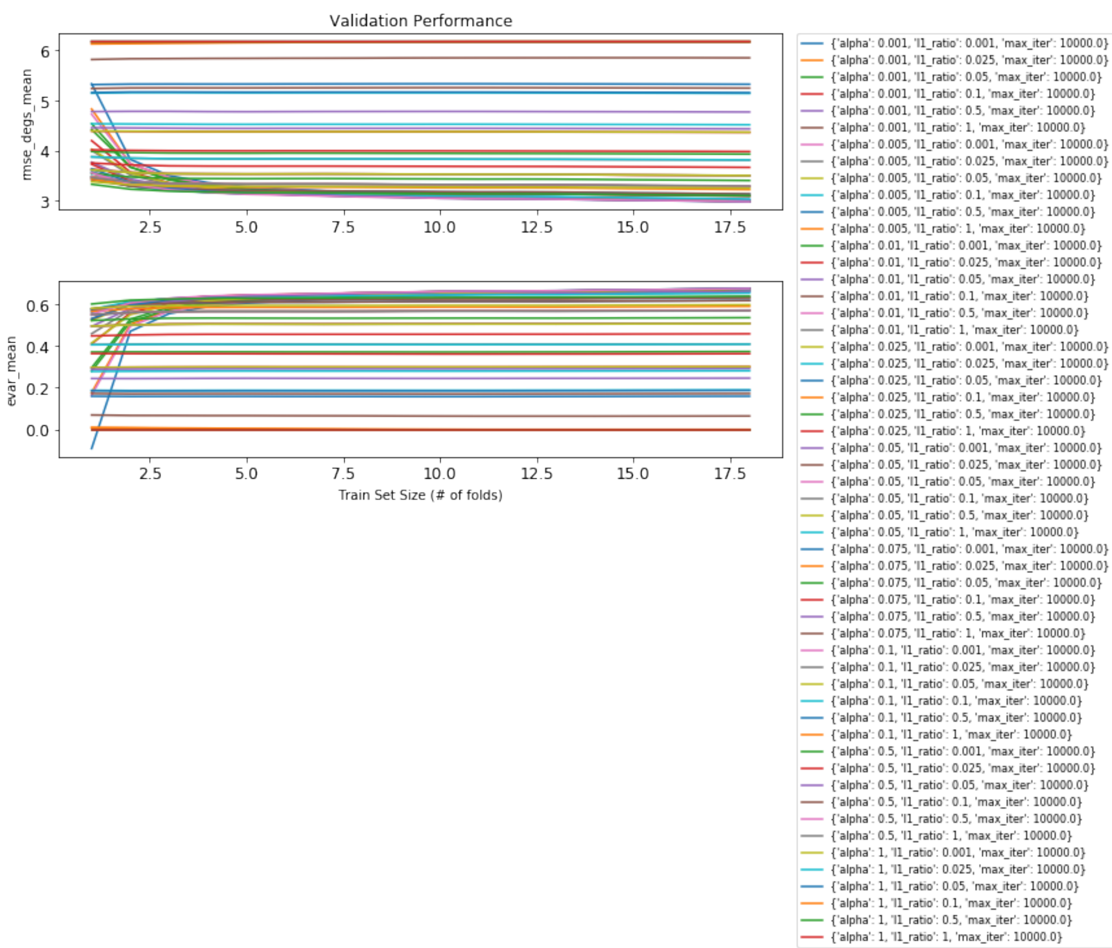
[29]:
```python
""" TODO
LASSO
Plot the validation results for all parameter sets over all train
sizes, for the specified metrics, rmse_degs_mean and evar_mean
(this variable is declared above). Use plot_allparams_val()
"""

l_crossval.plot_allparams_val(metrics)
```

[29]:
```
(<Figure size 720x432 with 2 Axes>,
 array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fa97a026c50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fa97a0c73c8>],
       dtype=object))
```



[30]:
```python
""" TODO
ELASTICNET
Plot the validation results for all parameter sets over all train
sizes, for the specified metrics, rmse_degs_mean and evar_mean
(this variable is declared above). Use plot_allparams_val()
"""

crossval.plot_allparams_val(metrics)
```

[30]:
```
(<Figure size 720x432 with 2 Axes>,
 array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fa97a65e668>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fa96ee24f60>],
       dtype=object))
```



7.0.5 Plot the TRAIN and VAL Set Performances

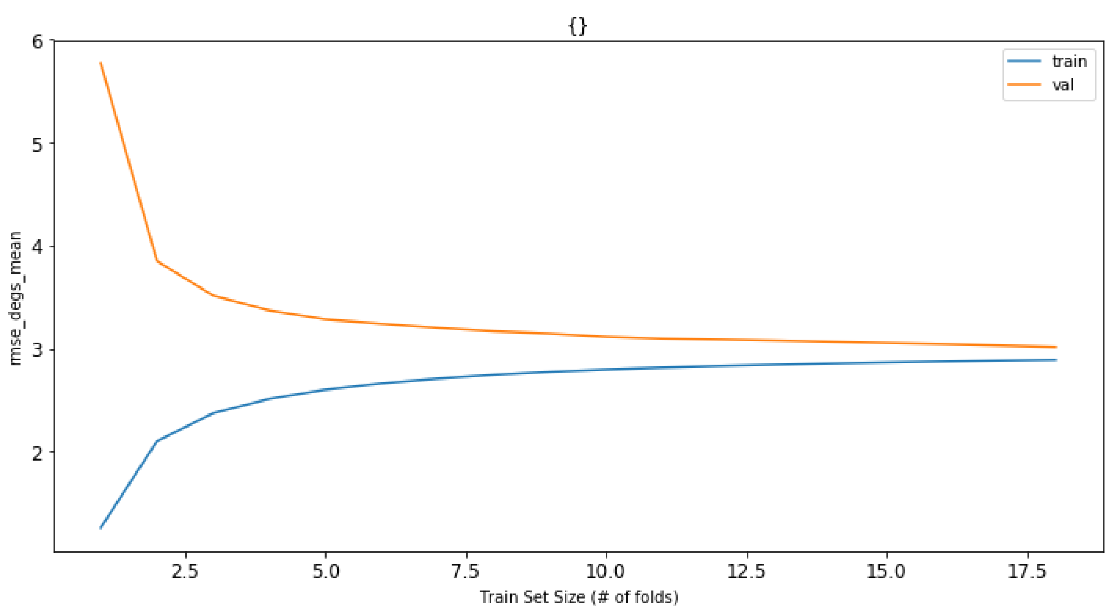
[31]:
```
lnr_crossval.opt_metric
```

[31]: 'rmse_degs_mean'

[32]:
```
""" TODO
LINEAR REGRESSION
For the best parameter set for the train set size at
size_idx=7 (this variable has already been declared above),
plot the TRAIN and VAL set performances using
plot_param_train_val() for just the optimized metric.

Note: there is only one parameter set for the Linear model,
thus paramidx=0
"""
print("Train Set Size", trainsizes[size_idx])

lnr_crossval.plot_param_train_val([lnr_crossval.opt_metric], paramidx= 0)
```

```
Train Set Size 8
```

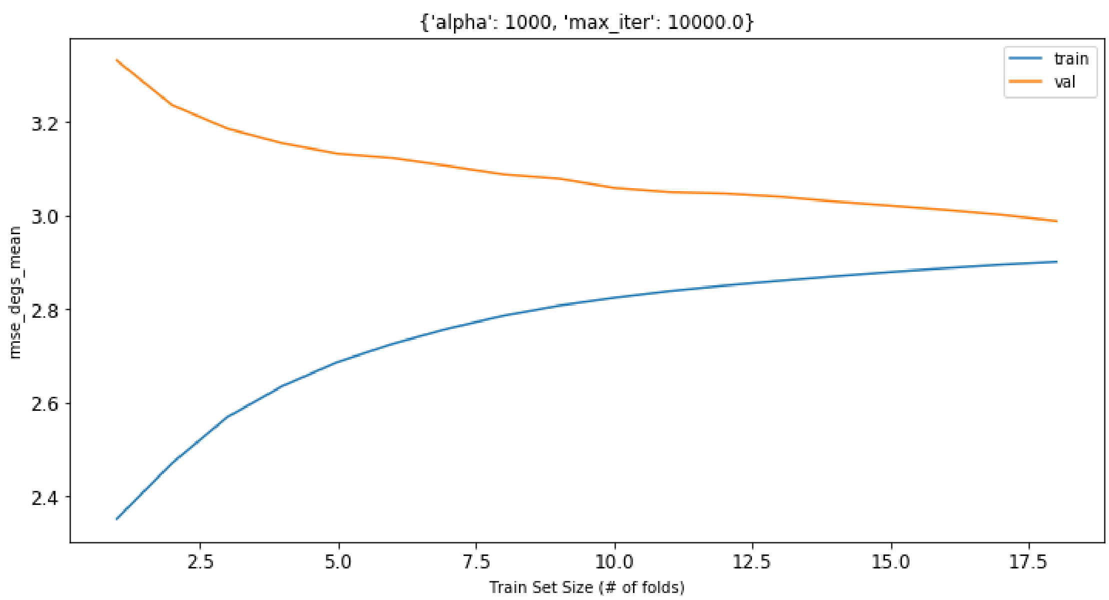
[32]:
```
(<Figure size 864x432 with 1 Axes>,
 array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fa96c13ee48>],
       dtype=object))
```



[33]:
```
""" TODO
RIDGE
For the best parameter set for the train set size at
size_idx=7 (this variable has already been declared above),
plot the TRAIN and VAL set performances using
plot_param_train_val() for just the optimized metric

Use r_crossval.best_param_inds to get the desired parameter
set index
"""
print("Train Set Size", trainsizes[size_idx])
best_param= r_crossval.best_param_inds[size_idx]

r_crossval.plot_param_train_val([r_crossval.opt_metric], paramidx= best_param)
```

```
Train Set Size 8
```

[33]:
```
(<Figure size 864x432 with 1 Axes>,
 array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fa96a6947f0>],
       dtype=object))
```

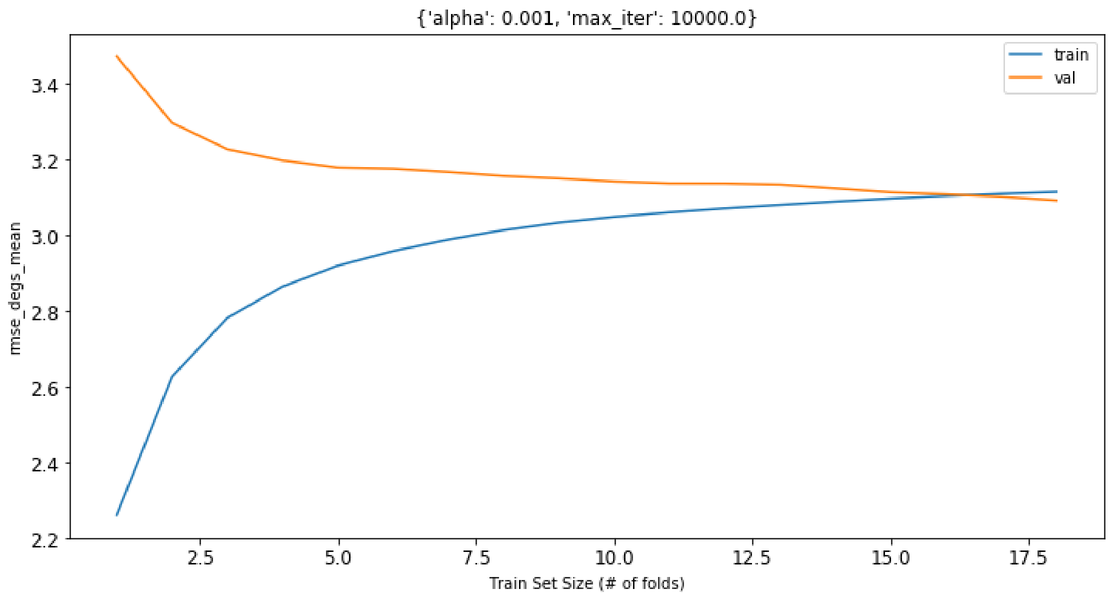


[34]:
```
""" TODO
LASSO
For the best parameter set for the train set size at
size_idx=7 (this variable has already been declared above),
plot the TRAIN and VAL set performances using
plot_param_train_val() for just the optimized metric
```

```python
"""
print("Train Set Size", trainsizes[size_idx])

best_param= l_crossval.best_param_inds[size_idx]

l_crossval.plot_param_train_val([l_crossval.opt_metric], paramidx= best_param)
```

```
Train Set Size 8
```

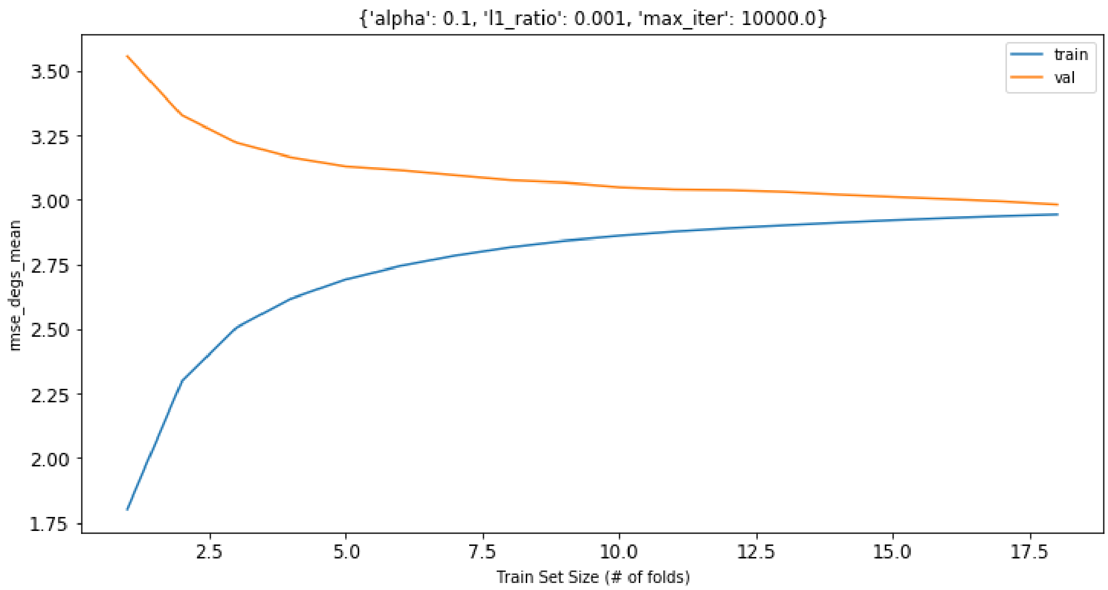
[34]:
```
(<Figure size 864x432 with 1 Axes>,
 array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fa96a694b00>],
       dtype=object))
```



[35]:
```python
"""
ELASTICNET
For the best parameter set for the train set size at
size_idx=7 (this variable has already been declared above),
plot the TRAIN and VAL set performances using
plot_param_train_val() for just the optimized metric
"""
print("Train Set Size", trainsizes[size_idx])
bp_idx = crossval.best_param_inds[size_idx]
crossval.plot_param_train_val([crossval.opt_metric], paramidx=bp_idx)
```

```
Train Set Size 8
```

[35]:
```
(<Figure size 864x432 with 1 Axes>,
 array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fa96a674860>],
       dtype=object))
```



7.0.6 Plot Performance over the Parameter Space

[36]:
```
def plot_param_val_for_size(crossval, metric, alphas, sizeidx=0):
    ''' PROVIDED
    Plotting function for after grid_cross_validation(),
    displaying the mean (summary) train and val set performances
    for each alpha, given the size, for RIDGE and LASSO only

    PARAMS:
        crossval: cross validation object
        metric: summary metric to plot. '_mean' or '_std' must be
                append to the end of the base metric name. These
                base metric names are the keys in the dict returned
                by eval_func
        alphas: list of alpha values
        sizeidx: train size index

    RETURNS: the figure and axes handles
    '''
    sizes = crossval.trainsizes
    results = crossval.results
    best_param_inds = crossval.best_param_inds
```

```
    nalphas = len(alphas)

    nsizes = len(sizes)
    nmetrics = len(metrics)

    # Initialize the matrices for the curve
    Y_train = np.empty((nalphas,))
    Y_val = np.empty((nalphas,))

    # Obtain the mean performance for the curve
    for param_res in results:
        params = param_res['params']
        summary = param_res['summary']

        alpha_idx = alphas.index(params['alpha'])

        # Compute the mean for multiple outputs
        res_train = np.mean(summary['train'][metric][sizeidx, :])
        Y_train[alpha_idx] = res_train

        res_val = np.mean(summary['val'][metric][sizeidx, :])
        Y_val[alpha_idx] = res_val

    # Initialize figure plots
    fig = plt.figure(figsize=(12,2))
    for i, (Y, set_name) in enumerate(zip((Y_train, Y_val),
                                          ('Training', 'Validation'))):
        # Plot
        ax = fig.add_subplot(1, 2, i+1)
        ax.plot(alphas, Y)
        title = "%s Performance, Train Size %d Folds" % (set_name,␣
↪sizes[sizeidx])
        ax.set(title=title)
        ax.set(xlabel=r"$\alpha$", ylabel=metric)
    return fig
```

[60]:
```
def plot_surface(xlist, ylist, Z_train, Z_val, ylabel, zlabel,
                 elev=30, angle=45, title_suffix=""):
    ''' PROVIDED
    Helper plotting function. x-axis is always alpha

    REQUIRES: from mpl_toolkits.mplot3d import Axes3D

    PARAMS:
        xlist: list of x values
        ylist: list of y values
```

```
        Z_train: matrix of performance results from the training set
        Z_val: matrix of performance results from the validation set
        ylabel: y-axis label
        zlabel: z-axis label
        elev: elevation of the 3D plot for the view
        angle: angle in degrees of the 3D plot for the view
        title_suffix: string to append to each subplot title

    RETURNS: the figure and axes handles
    '''
    # Initialize figure
    fig = plt.figure(figsize=(15,5))
    X, Y = np.meshgrid(xlist, ylist)
    for i, (Z, set_name) in enumerate(zip((Z_train, Z_val),
                                          ('Training', 'Validation'))):
        # Plot the surface
        ax = fig.add_subplot(1, 2, i+1, projection='3d')
        surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                               linewidth=0, antialiased=False,
                               vmin=0, vmax= Z.max())
        title = "%s Performance %s" % (set_name, title_suffix)
        ax.view_init(elev=elev, azim=angle)
        ax.set(title=title)
        ax.set(xlabel=r"$\alpha$", ylabel=ylabel, zlabel=zlabel)
        fig.colorbar(surf, ax=ax, shrink=0.5, aspect=10)

    return fig
```

[61]:

```
def plot_param_val_surface_RL(crossval, metric, alphas, elev=30, angle=245):
    ''' PROVIDED
    Plotting function for after grid_cross_validation(),
    displaying the mean (summary) train and val set performances
    for each alpha, for all sizes, for RIDGE and LASSO only

    REQUIRES: from mpl_toolkits.mplot3d import Axes3D

    PARAMS:
        crossval: cross validation object
        metric: summary metric to plot. '_mean' or '_std' must be
                append to the end of the base metric name. These
                base metric names are the keys in the dict returned
                by eval_func
        alphas: list of alpha values
        elev: elevation of the 3D plot for the view
        angle: angle in degrees of the 3D plot for the view

    RETURNS: the figure and axes handles
```

```
    '''
    sizes = crossval.trainsizes
    results = crossval.results
    best_param_inds = crossval.best_param_inds

    nalphas = len(alphas)

    nsizes = len(sizes)
    nmetrics = len(metrics)

    # Initialize the matrices for the surface
    Z_train = np.empty((nsizes, nalphas))
    Z_val = np.empty((nsizes, nalphas))

    # Obtain the mean performance for the surface
    for param_res in results:
        params = param_res['params']
        summary = param_res['summary']

        alpha_idx = alphas.index(params['alpha'])

        # Compute the mean for multiple outputs
        res_train = np.mean(summary['train'][metric], axis=1)
        Z_train[:, alpha_idx] = res_train

        # Compute the mean for multiple outputs
        res_val = np.mean(summary['val'][metric], axis=1)
        Z_val[:, alpha_idx] = res_val

    fig = plot_surface(alphas, sizes, Z_train, Z_val, 'size (# of folds)',
                       metric, elev, angle)
    return fig
```

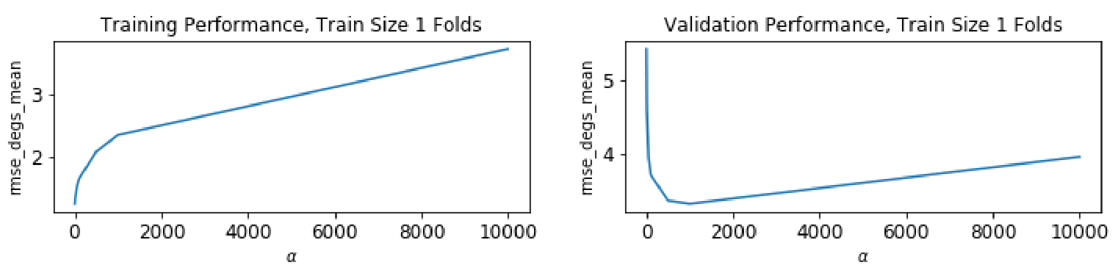
[107]:
```
def plot_param_val_surface_EN(crossval, metric, param_lists,
                              sizeidx=0, elev=35, angle=280):
    ''' PROVIDED
    Plotting function for after grid_cross_validation(),
    displaying the mean (summary) train and val set performances
    for each alpha and l1_ratio, given the size, for the ELASTICNET

    REQUIRES: from mpl_toolkits.mplot3d import Axes3D

    PARAMS:
        crossval: cross validation object
        metric: summary metric to plot. '_mean' or '_std' must be
                append to the end of the base metric name. These
                base metric names are the keys in the dict returned
```

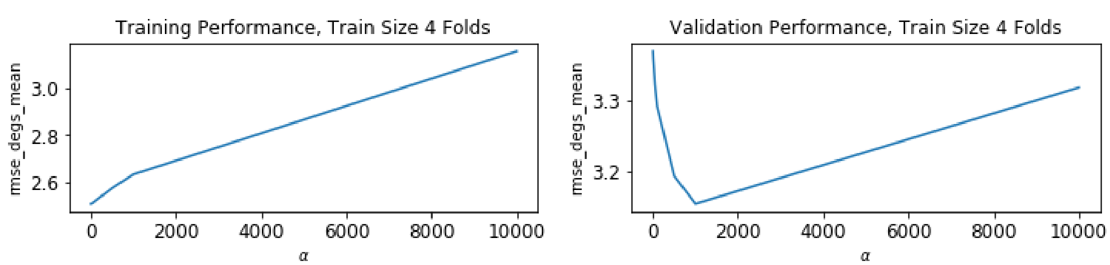
```
            by eval_func
        param_lists: dictionary of the list of alphas and l1_ratios
        sizeidx: train size index
        elev: elevation of the 3D plot for the view
        angle: angle in degrees of the 3D plot for the view

    RETURNS: the figure and axes handles
    '''
    sizes = crossval.trainsizes
    results = crossval.results
    best_param_inds = crossval.best_param_inds

    alphas = list(param_lists['alpha'])
    l1_ratios = list(param_lists['l1_ratio'])

    nalphas = len(alphas)
    nl1_ratios = len(l1_ratios)

    nsizes = len(sizes)
    nmetrics = len(metrics)

    # Initialize the matrices for the surface
    Z_train = np.ones((nl1_ratios, nalphas))*(-9999)
    Z_val = np.ones((nl1_ratios, nalphas))*(-9999)

    # Obtain the mean performance for the surface
    for param_res in results:
        params = param_res['params']
        summary = param_res['summary']

        alpha_idx = alphas.index(params['alpha'])
        l1_idx = l1_ratios.index(params['l1_ratio'])

        # Compute the mean for multiple outputs
        res_train = np.mean(summary['train'][metric][sizeidx, :])

        Z_train[l1_idx, alpha_idx] = res_train

        res_val = np.mean(summary['val'][metric][sizeidx, :])

        Z_val[l1_idx, alpha_idx] = res_val

    Z_train[Z_train<0]=np.nan
    Z_val[Z_val<0]=np.nan
    fig = plot_surface(alphas, l1_ratios, Z_train, Z_val, 'l1_ratio',
                       metric, elev, angle,', Size %d Folds' % sizes[sizeidx])
    return fig
```

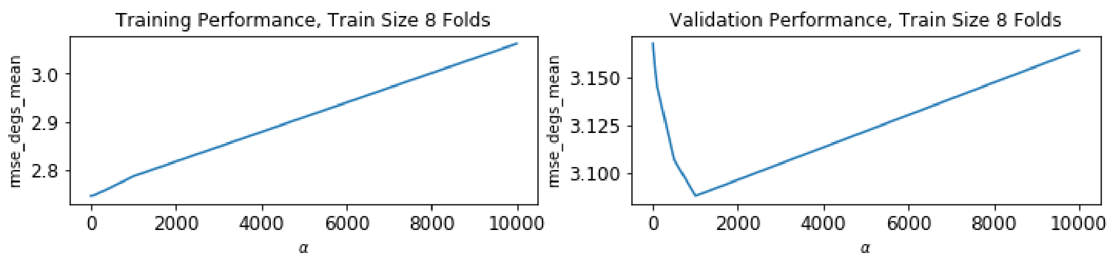
[40]:
```python
""" PROVIDED
List the parameter sets explored for RIDGE
"""
r_crossval.paramsets
```

[40]:
```
[{'alpha': 1, 'max_iter': 10000.0},
 {'alpha': 10, 'max_iter': 10000.0},
 {'alpha': 50, 'max_iter': 10000.0},
 {'alpha': 100, 'max_iter': 10000.0},
 {'alpha': 500, 'max_iter': 10000.0},
 {'alpha': 1000, 'max_iter': 10000.0},
 {'alpha': 10000, 'max_iter': 10000.0}]
```

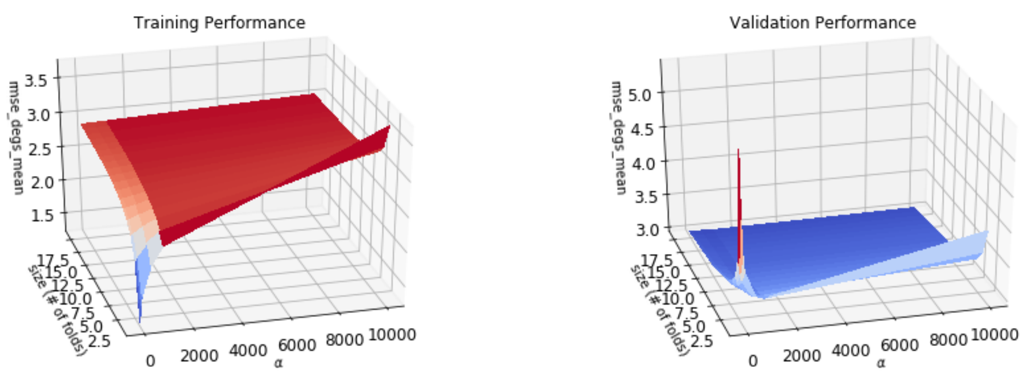
[41]:
```python
""" TODO
Plot the performance versus alpha for the RIDGE model
using plot_param_val_for_size() for size indices 0, 3, and 7,
for the optimized metric (use r_crossval.opt_metric)
"""

fig= plot_param_val_for_size(r_crossval, r_crossval.opt_metric, [each['alpha']␣
↪for each in r_crossval.paramsets], 0)
fig= plot_param_val_for_size(r_crossval, r_crossval.opt_metric, [each['alpha']␣
↪for each in r_crossval.paramsets], 3)
fig= plot_param_val_for_size(r_crossval, r_crossval.opt_metric, [each['alpha']␣
↪for each in r_crossval.paramsets], 7)
```







[42]:
```
""" TODO
RIDGE
Use plot_param_val_surface_RL() to plot the surface of the training
and validation set performance versus alpha and size in the X and Y axes,
using the optimized metric
"""
# Feel free to adjust these to understand the shape of the surface
# Elevation of the plot
elev = 30
# Angle the plot is viewed
angle = 255

# TODO: Plot
fig= plot_param_val_surface_RL(r_crossval, r_crossval.opt_metric,␣
↪[each['alpha'] for each in r_crossval.paramsets], elev=elev, angle=angle)
```



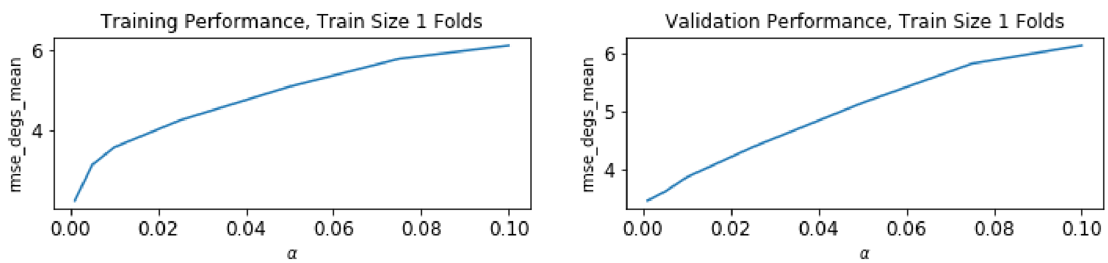
[43]:
```
""" PROVIDED
List the parameter sets explored for LASSO
"""
l_crossval.paramsets
```

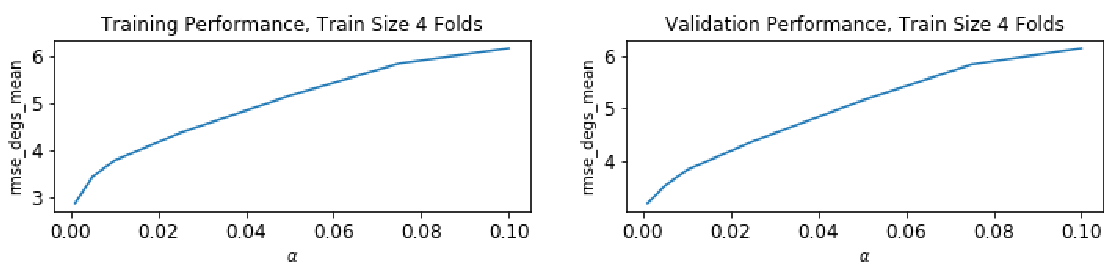
[43]:
```
[{'alpha': 0.001, 'max_iter': 10000.0},
 {'alpha': 0.005, 'max_iter': 10000.0},
```

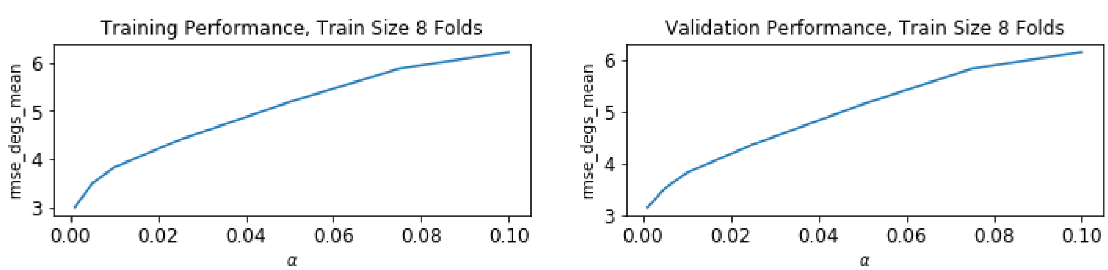
```
 {'alpha': 0.01, 'max_iter': 10000.0},
 {'alpha': 0.025, 'max_iter': 10000.0},
 {'alpha': 0.05, 'max_iter': 10000.0},
 {'alpha': 0.075, 'max_iter': 10000.0},
 {'alpha': 0.1, 'max_iter': 10000.0}]
```

[44]:

```python
""" TODO
Plot the performance versus alpha for the LASSO model
using plot_param_val_for_size() for size indices 0, 3, and 7,
for the optimized metric
"""
fig1= plot_param_val_for_size(l_crossval, l_crossval.opt_metric, [each['alpha'] for each in l_crossval.paramsets], 0)
fig2= plot_param_val_for_size(l_crossval, l_crossval.opt_metric, [each['alpha'] for each in l_crossval.paramsets], 3)
fig3= plot_param_val_for_size(l_crossval, l_crossval.opt_metric, [each['alpha'] for each in l_crossval.paramsets], 7)
```

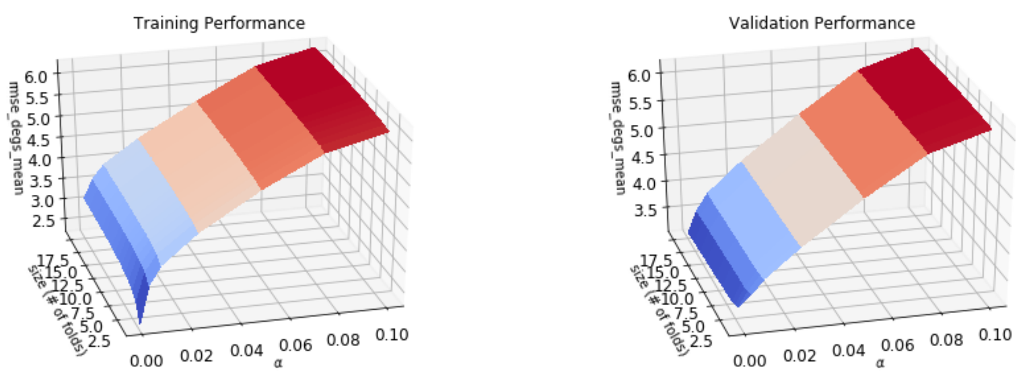






[45]:

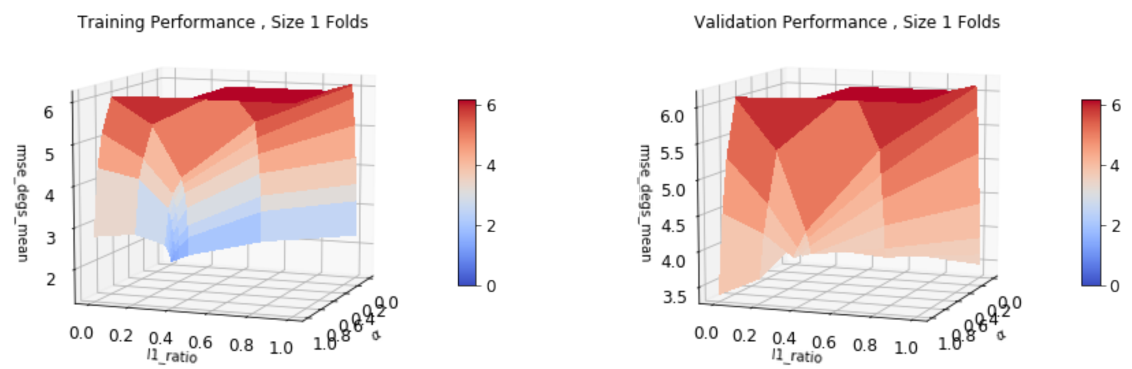
```
""" TODO
LASSO
Use plot_param_val_surface_RL() to plot the surface of the training
and validation set performance versus alpha and size in the X and Y axes,
using the optimized metric
"""
# Feel free to adjust these to understand the shape of the surface
# Elevation of the plot
elev = 30
# Angle the plot is viewed
angle = 255

# TODO: Plot
fig= plot_param_val_surface_RL(l_crossval, l_crossval.opt_metric,␣
↪[each['alpha'] for each in l_crossval.paramsets], elev=elev, angle=angle)
```

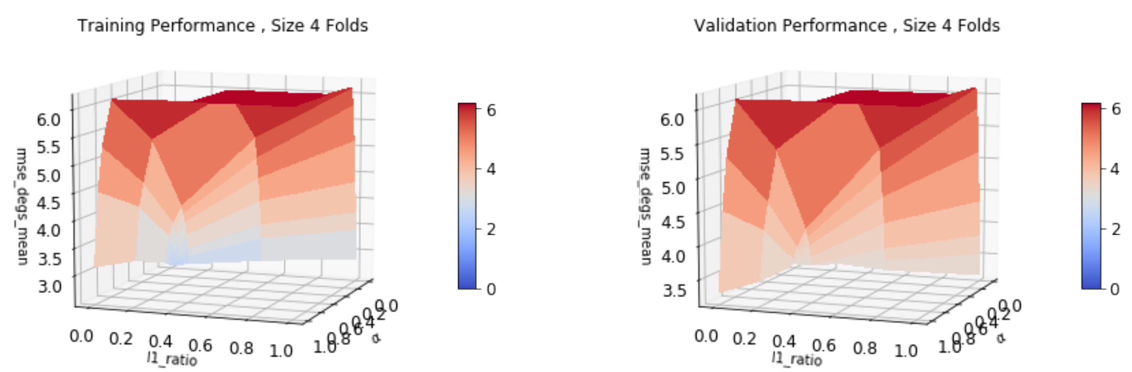


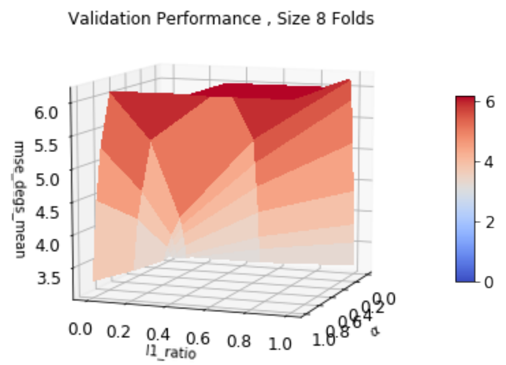
[108]:

```
""" TODO
ELASTICNET
Use plot_param_val_surface() to plot the surface of the training
and validation set performance versus alpha and l1_ratio in the X
and Y axes for the size indices of 0, 3, and 7, for crossval.opt_metric
"""
# Feel free to adjust these to understand the shape of the surface
# Elevation of the plot
elev = 10
# Angle the plot is viewed
angle = 20

crossval_params= {'alpha': sorted(list(set([each['alpha'] for each in crossval.
↪paramsets]))),
```

```
                  'l1_ratio': sorted(list(set([each['l1_ratio'] for each in crossval.paramsets])))
                 }
# TODO: Plot

fig1= plot_param_val_surface_EN(crossval, crossval.opt_metric, crossval_params,sizeidx=0, elev=elev, angle=angle)
fig2= plot_param_val_surface_EN(crossval, crossval.opt_metric, crossval_params,sizeidx=3, elev=elev, angle=angle)

fig3= plot_param_val_surface_EN(crossval, crossval.opt_metric, crossval_params,sizeidx=7, elev=elev, angle=angle)
```







7.0.7 Paired t-tests

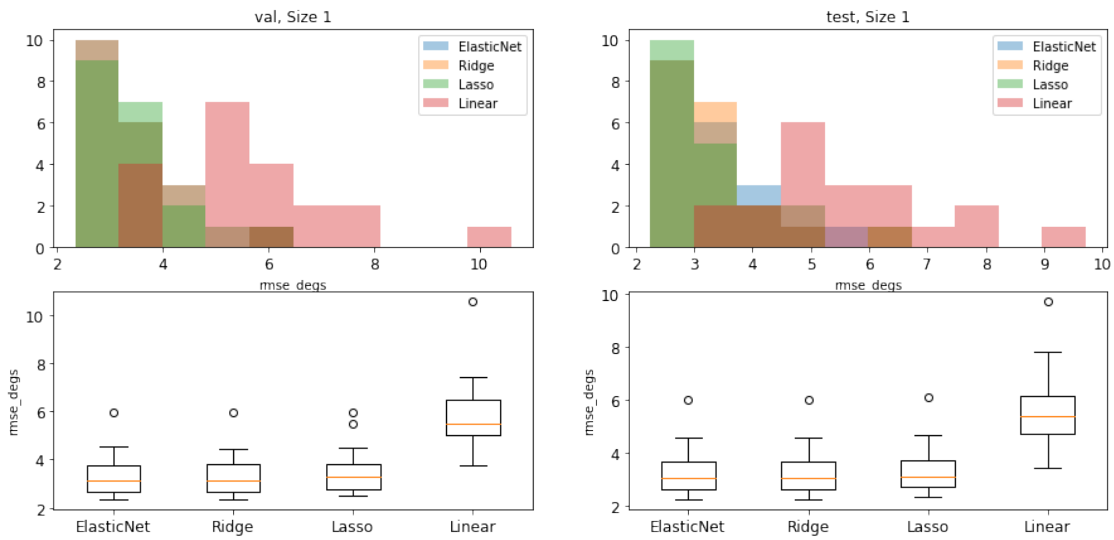
We can use paired t-tests to assess statistical differences between the mean test set performances of the models

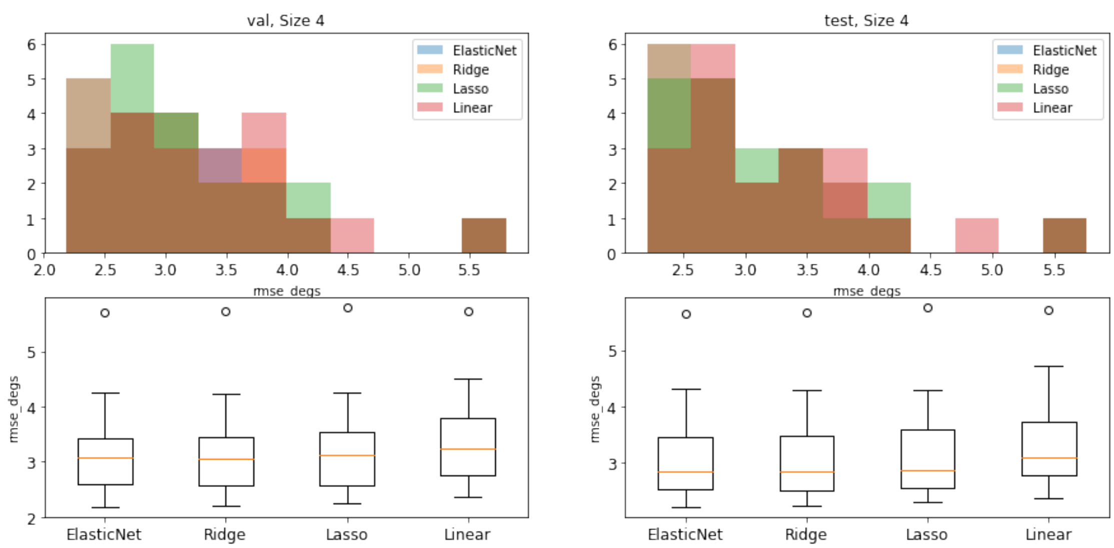
[47]:
```
""" PROVIDED
Obtain all the results for all the models
"""
# LinearRegression
lnr_all_results = lnr_crossval.results

# RIDGE
r_all_results = r_crossval.results

# LASSO
l_all_results = l_crossval.results

# ELASTICNET
all_results = crossval.results
```

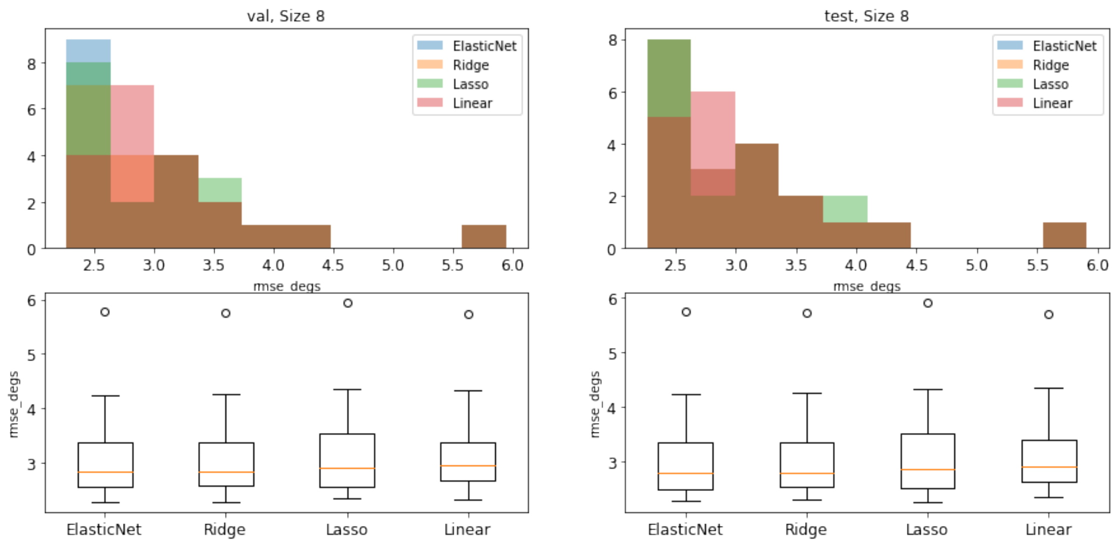
[48]:
```
""" TODO
Complete the plotting code

Plot distributions of the Validation and Test scores from the
best parameter set for each base model for the corresponding
size indices, [0, 3, 7]. The metric of interest is rmse_degs.
These are the distribution of results from each rotation of
the training set
"""
metric = 'rmse_degs'
set_names = ['val', 'test']
nbins = 11
```

```
# Size indices
size_indices = [0, 3, size_idx]

for si in size_indices:
    # Obtain the index of the best parameter set for the size
    # RIDGE
    r_bp_idx = r_crossval.best_param_inds[si]
    # LASSO
    l_bp_idx = l_crossval.best_param_inds[si]
    # ELASTICNET
    bp_idx = crossval.best_param_inds[si]

    # Construct the figure
    fig, axs = plt.subplots(2, 2, figsize=(15,7))
    for i, set_name in enumerate(set_names):
        title = '%s, Size %d' % (set_name, trainsizes[si])

        # LINEAR
        # Note: there's only 1 parameter set for the Linear model
        lnr_res = lnr_all_results[0]['results'][si][set_name]
        lnr_scores = np.mean(lnr_res[metric], axis=1)

        # RIDGE
        # Obtain results for the best parameter set for the size
        ridge_res = r_all_results[r_bp_idx]['results'][si][set_name]
        # Compute the mean of the outputs for each data set rotation
        ridge_scores = np.mean(ridge_res[metric], axis=1)

        # LASSO
        lasso_res = l_all_results[l_bp_idx]['results'][si][set_name]
        lasso_scores = np.mean(lasso_res[metric], axis=1)

        # ELASTICNET
        res = all_results[bp_idx]['results'][si][set_name]
        elastic_scores = np.mean(res[metric], axis=1)

        # Determine the edges for the bins in the histograms
        all_scores = np.concatenate((elastic_scores, ridge_scores,
                                     lasso_scores, lnr_scores))
        mn = np.min(all_scores)
        mx = np.max(all_scores)
        bins = np.linspace(mn, mx, nbins)

        # Histograms
        # TODO: include the hist of the elastic net scores
```

```
        axs[0, i].hist(elastic_scores, bins= bins, alpha= .4)
        axs[0, i].hist(ridge_scores, bins=bins, alpha=.4)
        axs[0, i].hist(lasso_scores, bins=bins, alpha=.4)
        axs[0, i].hist(lnr_scores, bins=bins, alpha=.4)
        axs[0, i].legend(['ElasticNet', 'Ridge', 'Lasso', 'Linear'])
        axs[0, i].set(title=title, xlabel=metric)

        # Boxplots
        axs[1, i].boxplot([elastic_scores, ridge_scores, lasso_scores, lnr_scores])
        axs[1, i].set_xticklabels(['ElasticNet', 'Ridge', 'Lasso', 'Linear'])
        axs[1, i].set(ylabel=metric)
```








```
[49]: """ TODO
Dependent Sample Paired t-test
Two-sided t-test for the null hypothesis that mean of the distribution
of differences between the two test performance distributions is zero
"""
print("Train Set Size", trainsizes[size_idx])

# LINEAR
# Note: there's only 1 parameter set for the LinearRegression model
lnr_res = lnr_crossval.results[0]['results'][size_idx]['test']
lnr_test_res = np.mean(lnr_res[metric], axis=1)

# RIDGE
# Obtain index of best parameters for train size 8
r_bp_idx = r_crossval.best_param_inds[size_idx]
# Obtain all results for the best parameter set for train size 8
ridge_res = r_all_results[r_bp_idx]['results'][size_idx]['test']
# Compute the mean of the outputs for each data set rotation
ridge_test_res = np.mean(ridge_res[metric], axis=1)

# LASSO
l_bp_idx = l_crossval.best_param_inds[size_idx]
lasso_res = l_all_results[l_bp_idx]['results'][size_idx]['test']
lasso_test_res = np.mean(lasso_res[metric], axis=1)

# TODO: ELASTICNET
```

```
bp_idx = crossval.best_param_inds[size_idx]# TODO
net_res = all_results[bp_idx]['results'][size_idx]['test']# TODO
elastic_test_res = np.mean(net_res[metric],axis=1)# TODO
```

Train Set Size 8

[50]:
```
""" TODO
ELASTICNET vs RIDGE
Execute the paired t-test to determine whether to reject the null hypothesis
(i.e. H0) with 95% confidence. H0 is that the mean of the distribution of the
differences between test scores for the best ELASTICNET model and the best␣
↪RIDGE
is zero, when using a training size of 8 (i.e. the size at index 7 of the
trainsizes list). Display the t-statistic, the p-value, and the mean of the
differences (i.e. mean(elastic_test_res - ridge_test_res))

Use stats.ttest_rel(). See the API reference above.
Do the same for all the pairing of models
"""
t, p = stats.ttest_rel(elastic_test_res, ridge_test_res)
print('tscore of ElasricNet and Ridge test is %.4f, p-value is %.4f, the mean␣
↪of difference: %.4f'%(t, p, (elastic_test_res- ridge_test_res).mean()))
```

```
tscore of ElasricNet and Ridge test is -2.5561, p-value is 0.0193, the mean of
difference: -0.0128
```

[51]:
```
""" TODO
ELASTICNET vs LASSO
Execute the paired t-test
"""
t, p = stats.ttest_rel(elastic_test_res, lasso_test_res)
print('tscore of ElasricNet and Lasso test is %.4f, p-value is %.4f, the mean␣
↪of difference: %.4f'%(t, p, (elastic_test_res- lasso_test_res).mean()))
```

```
tscore of ElasricNet and Lasso test is -4.3305, p-value is 0.0004, the mean of
difference: -0.0821
```

[52]:
```
""" TODO
ELASTICNET vs LinearRegression
Execute the paired t-test
"""


t, p = stats.ttest_rel(elastic_test_res, lnr_test_res)
print('tscore of ElasricNet and LinearRegression test is %.4f, p-value is %.4f,␣
↪the mean of difference: %.4f'%(t, p, (elastic_test_res- lnr_test_res).
↪mean()))
```

```
tscore of ElasricNet and LinearRegression test is -4.8955, p-value is 0.0001,
```

```
the mean of difference: -0.0914
```

[53]:
```python
""" TODO
RIDGE vs LASSO
Execute the paired t-test
"""
t, p = stats.ttest_rel(ridge_test_res, lasso_test_res)
print('tscore of Ridge and Lasso test is %.4f, p-value is %.4f, the mean of␣
 ↪difference: %.4f'%(t, p, (ridge_test_res- lasso_test_res).mean()))
```

```
tscore of Ridge and Lasso test is -2.9138, p-value is 0.0089, the mean of
difference: -0.0692
```

[54]:
```python
""" TODO
RIDGE vs LinearRegression
Execute the paired t-test
"""


t, p = stats.ttest_rel(ridge_test_res, lnr_test_res)
print('tscore of Ridge and LinearRegression test is %.4f, p-value is %.5f, the␣
 ↪mean of difference: %.4f'%(t, p, (ridge_test_res- lnr_test_res).mean()))
```

```
tscore of Ridge and LinearRegression test is -5.6123, p-value is 0.00002, the
mean of difference: -0.0786
```

[55]:
```python
""" TODO
LASSO vs LinearRegression
Execute the paired t-test
"""
t, p = stats.ttest_rel(lasso_test_res, lnr_test_res)
print('tscore of Lasso and LinearRegression test is %.4f, p-value is %.4f, the␣
 ↪mean of difference: %.4f'%(t, p, (lasso_test_res- lnr_test_res).mean()))
```

```
tscore of Lasso and LinearRegression test is -0.2556, p-value is 0.8010, the
mean of difference: -0.0094
```

8 DISCUSSION

For each question write 1 to 2 paragraphs of discussion:

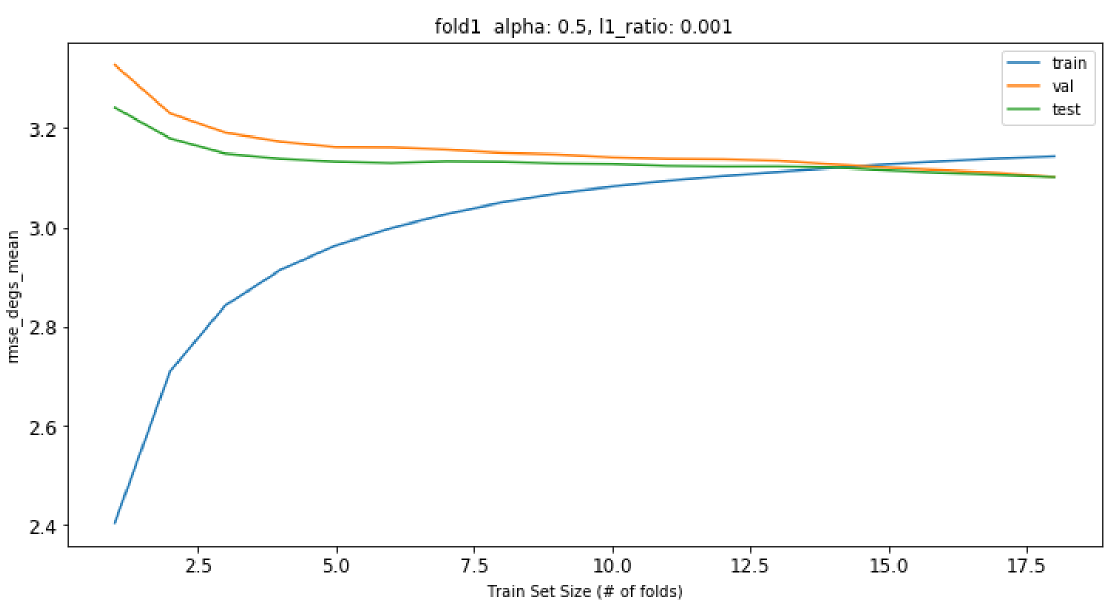
1. Interpret the meaning of the t-test results using 95% confidence. Discuss the statistical meaning as well as the practical interpretation of the results in the context of the data set.
2. For the Elastic Net Model, discuss the differences in the surfaces between the train sizes of 1, 4, and 8 folds, for the training and validation sets.
3. For each of the train set sizes of 1, 4, and 8 folds, which model (Linear, Lasso, Ridge, or ElasticNet) and corresponding parameter set would you select and why? Specify which

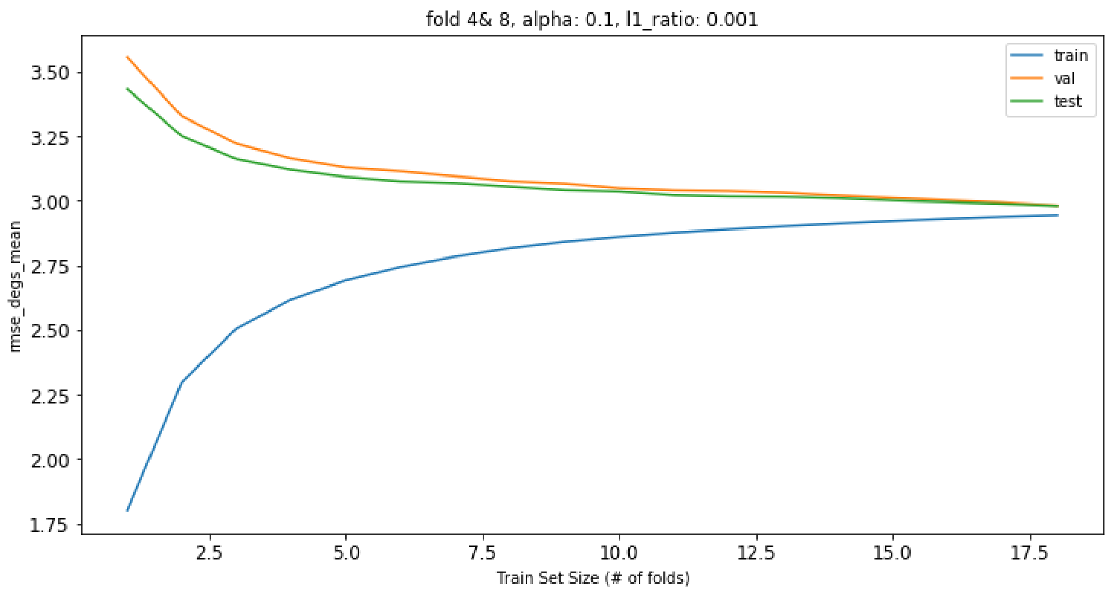
model and parameter set for each size. For each size, use plot_param_train_val() to view the train, val, and test sets of the chosen model(s). Remember, selections should be made based on the validation performance.

p-value	ElasticNet	Ridge	Lasso	LinearRegression
ElasticNet	1	0.0193	0.0004	0.0001
Ridge	0.0193	1	0.0089	2e-5
Lasso	0.0004	0.0089	1	0.8010
LinearRegression	0.0001	2e-5	0.8010	1

[56]:
```
fold_1= crossval.paramsets.index({'alpha':0.5, 'l1_ratio':0.001, 'max_iter':␣
↪10000.0})
fold_other= crossval.paramsets.index({'alpha':0.1, 'l1_ratio':0.001, 'max_iter':
↪ 10000.0})
```

[57]:
```
""" TODO
Discussion question 3 plots
"""
fig1= crossval.plot_param_train_val(['rmse_degs_mean'],fold_1,view_test=True)
plt.title('fold1  alpha: 0.5, l1_ratio: 0.001');
fig1= crossval.plot_param_train_val(['rmse_degs_mean'],fold_other,␣
↪view_test=True)
plt.title('fold 4& 8, alpha: 0.1, l1_ratio: 0.001');
```





1. With regard to question 1, under 95% confidence level, we select out p-values that are less than 0.05. Only Lasso vs. LinearRegression can be thought as under null hypothesis which means there is little difference between the mean of the score. For others, there are more or less significant amount of differences between two models, especially for Ridge vs. LinearRegression.

At a practical side, it comes with over-fitting issues when there is no contraints of weights for LinearRegression. It thus likely to be over-trained, and behaves bad in test samples. Lasso also comes with this problem, because it only applies 1st order constraint over weights rather than second order like Ridge. So you may think Lasso and LinearRegression do not have significant differences. Then, because ElasticNet stays in between Lasso and Eidge, so it has largest difference with LineaRegression, then follows Lasso, Ridge.

2. First of all, for training samples, we could observe relatively low rmse in the low tail as represented by cold color (plot in the same color scale) compared to validation samples in whatever training sizes. In other words, validation has more warm color regions (higher rmse) than training.

Secondly, when increasing training sizes from 1 fold to 4 folds, in the validation process, the least error decreases in the corner region, while only marginal increment of performance from 4 folds to 8 folds from the surface plot. For training process, even the performance drops with training sizes.

3. From the boxplot, barplot provided above to compare four models, ElasticNet is well behaved in validation process, thus the ElasticNet model has certain level of robustness in this prediction problem. I picked it as the benchmark model.

As for the parameter selection, since we have plotted the best parameter with different training sizes. For 1 training fold, the parameter set is {'alpha': 0.5, 'l1_ratio': 0.001}, for training folds of 4 and 8, the optimal parameter set is {'alpha':0.1, 'l1_ratio':0.001 }