Homework 1 Part 2

This is an individual assignment.

Due: Friday, September 23 @ 11:59pm

Question 1 (30 points)

In this question, you will be working with the marathon time predictions dataset. (Same dataset as in HW0).

Carry hyperparameter tuning for training 3 regression models:

- **Model 1**: (multiple) linear regression with Lasso regularizer.
- **Model 2**: polynomial regression with Lasso regularizer.
- **Model 3**: random forest.

For model 1, experiment with the type of regularizer and the regularizer term λ.

For model 2, experiment with polynomial degree, the `interaction_only` argument and the regularizer term λ.

For model 3, experiment with the number of trees and the `criterion`.

1. Build a `GridSearchCV` for each model using `scikit-learn` pipelines.
2. Build a `RandomizedSearchCV` for each model using `scikit-learn` pipelines.
   - Consider an exponential distribution for the regularizer term λ.
   - Consider a uniform distribution for the number of trees.
3. Which set of hyperparmeters worked best for each model?
4. Train the final models and save them as `pickle` files. (See bottom of lecture 5 part 1 for an example.)

Preprocess data (From HW0)

In [203…

```
import pandas as pd
import numpy as np

# Import data
marathon_data = pd.read_csv('MarathonData.csv');
```

```
# Clean data
marathon_data = marathon_data.drop(['id','Marathon','Name','CrossTraining', 'CA
marathon_data = marathon_data.dropna(subset=['Category']);
marathon_data['Wall21'] = pd.to_numeric(marathon_data['Wall21'], errors='coerce
```

In [204…

```
# Get numerical and categorical attributes
m_data_num = marathon_data[['km4week','sp4week','Wall21','MarathonTime']];
m_data_cat = marathon_data[['Category']];
```

In [205…

```
# Custom Transformer
from sklearn.base import BaseEstimator, TransformerMixin

km4week_ix, sp4week_ix = 0, 1

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_km4week_per_sp4week=True):
        self.add_km4week_per_sp4week = add_km4week_per_sp4week
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        if self.add_km4week_per_sp4week:
            km4week_per_sp4week = X[:,km4week_ix] / X[:,sp4week_ix]
            estimate =  X[:,2]*X[:,sp4week_ix] / X[:,km4week_ix] + 2*X[:,2]
        return np.c_[X, km4week_per_sp4week, estimate]
```

In [206…

```
from sklearn.pipeline import Pipeline

num_pipeline = Pipeline([('attribs_adder', CombinedAttributesAdder())])

marathon_num_addons = num_pipeline.fit_transform(m_data_num.to_numpy())
```

In [207…

```
marathon_all = pd.concat([m_data_cat, pd.DataFrame(marathon_num_addons,
                           columns=np.hstack((m_data_num.columns, ['ratio','est
                           index=m_data_num.index)], axis=1)
```

In [208…

```
wall21_cat = pd.cut(marathon_all['Wall21'],
                    bins=[0., 1.4, 1.6, 1.8, np.inf],
                    labels=[1, 2, 3, 4])
```

In [209…

```
from sklearn.model_selection import train_test_split

# Split into testing and training sets
train_set, test_set, income_cat_train, income_cat_test = train_test_split(marat
                                                                          test_
                                                                          shuf1
                                                                          rando
                                                                          strat
```

In [210…

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler

# Build transformation pipeline to encode categorical and numerical
# attributes

num_attribs = list(train_set.columns[1:])
```

```
cat_attribs = ['Category']
num_pipeline = Pipeline([('std_scaler', StandardScaler())])
# Pipeline
full_pipeline = ColumnTransformer([('num', num_pipeline, num_attribs),
                                   ('cat', OneHotEncoder(), cat_attribs)])

train_set_prepared = full_pipeline.fit_transform(train_set)
test_set_prepared = full_pipeline.transform(test_set)
training_set = pd.DataFrame(train_set_prepared,
                            columns=np.hstack((train_set.columns[1:],
                                               ['Cat1','Cat2','Cat3','Cat4','Cat
                            index=train_set.index)

test_set = pd.DataFrame(test_set_prepared,
                            columns=np.hstack((test_set.columns[1:],
                                               ['Cat1','Cat2','Cat3','Cat4','Cat
                            index=test_set.index)
# training/test labels
t_train = training_set['MarathonTime'].copy()
t_test = test_set['MarathonTime'].copy()
training_set
```

Out[210]:

	km4week	sp4week	Wall21	MarathonTime	ratio	estimate	Cat1	Cat2	Cat3
69	-0.781538	-0.125938	0.795044	0.956411	-0.737816	-0.119785	0.0	0.0	0.0
49	0.645110	-0.126108	0.045389	0.220521	0.667326	-0.127303	0.0	0.0	0.0
7	1.728739	-0.125604	-1.032240	-1.224978	1.493881	-0.134130	0.0	1.0	0.0
86	-1.705351	-0.126944	1.966380	1.771147	-1.526800	-0.104529	1.0	0.0	0.0
78	-1.311659	-0.126689	1.591552	1.534610	-1.143580	-0.112256	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...
43	-1.147945	-0.125593	-0.048318	0.036548	-1.128551	-0.122928	0.0	0.0	0.0
68	-0.099398	-0.126899	0.560777	0.903847	0.171394	-0.123638	0.0	0.0	0.0
32	0.153969	-0.126187	-0.563705	-0.252552	0.216478	-0.130295	0.0	1.0	0.0
19	0.586641	-0.128988	-1.032240	-0.962160	2.034899	-0.134397	0.0	0.0	0.0
72	-0.364458	-0.126319	1.450992	1.008974	-0.256912	-0.117459	0.0	0.0	0.0

64 rows × 12 columns

In [211…

```
test_set
```

Out[211]:

	km4week	sp4week	Wall21	MarathonTime	ratio	estimate	Cat1	Cat2	Cat3
59	-0.641212	-0.126816	0.654484	0.562184	-0.422223	-0.121768	1.0	0.0	0.0
85	-1.108966	-0.126785	1.966380	1.692301	-0.918156	-0.111560	0.0	1.0	0.0
70	0.988129	-0.126491	0.841898	0.982693	1.148231	-0.123190	0.0	1.0	0.0
13	0.886783	-0.124776	-1.032240	-1.119851	0.479473	-0.133346	1.0	0.0	0.0
14	0.590539	-0.125423	-0.891679	-1.093569	0.404332	-0.132446	0.0	0.0	0.0
6	1.066088	-0.125677	-1.172800	-1.303823	0.915293	-0.134552	1.0	0.0	0.0
39	0.294295	-0.127521	-0.142024	-0.147424	0.832637	-0.128551	0.0	0.0	0.0
50	-0.442417	-0.127442	0.185950	0.246802	-0.039002	-0.125470	0.0	0.0	0.0
37	-0.372254	-0.125990	-0.095171	-0.173706	-0.339567	-0.126495	0.0	1.0	0.0
63	-1.596209	-0.125803	0.279656	0.667311	-1.519286	-0.116457	0.0	0.0	0.0
33	1.978208	-0.125902	-0.235731	-0.252552	1.854559	-0.129805	0.0	0.0	1.0
60	-0.719171	-0.126880	0.607630	0.641029	-0.489850	-0.121846	1.0	0.0	0.0
18	0.890681	-0.125710	-0.844826	-0.988442	0.765010	-0.132539	0.0	0.0	0.0
3	2.956592	-0.125921	-1.453920	-1.645487	2.786312	-0.137037	0.0	1.0	0.0
40	-0.013643	-0.125830	-0.188878	-0.068579	-0.039002	-0.127686	0.0	1.0	0.0
55	-1.354536	-0.125294	0.185950	0.509620	-1.346461	-0.119468	0.0	0.0	0.0
84	-0.243622	-0.126802	1.450992	1.666019	-0.008946	-0.118058	1.0	0.0	0.0

Model 1: (multiple) linear regression with Lasso regularizer

In [212…

```
# Use lasso regularization and experiment only with the lamda
# value. For model 1 experiment with the regularizer term
```

In [213…

```
from sklearn.linear_model import LinearRegression
# Train the Linear Regression Model
lin_reg = LinearRegression()
lin_reg.fit(training_set, t_train)

# Parameters w
w = np.hstack((lin_reg.intercept_,lin_reg.coef_)).reshape(-1,1)
```

In [214…

```
# Make predictions for linear regression model
y_train_lr = lin_reg.predict(training_set)
y_train_lr
```

Out[214]:
```
array([ 0.95641091,  0.22052068, -1.22497799,  1.77114653,  1.53461038,
       -0.06857906, -1.88202284, -1.11985082,  0.66731118,  0.03654812,
       -0.67306032,  0.1416753 , -0.22626982, -0.3051152 , -2.46022231,
       -0.69934211, -1.04100544, -0.83075108,  0.03654812,  1.24551065,
        0.40449324,  1.61345576, -1.69805029,  1.63973756,  1.48204679,
       -1.14613261,  0.87756553, -0.56793314, -0.38396058, -0.54165135,
        1.56089217,  0.562184  ,  1.63973756,  1.0089745 ,  0.74615656,
       -1.25125979, -1.17241441,  0.43077503, -1.38266876,  1.19294706,
       -1.14613261,  0.1153935 , -0.43652417,  0.87756553, -0.3051152 ,
        0.82500194,  0.64102938, -0.19998803, -0.48908776,  0.50962041,
       -1.06728723,  1.29807423, -1.48779593, -1.04100544,  0.06282991,
        0.06282991, -0.41024238,  0.53590221, -0.17370623,  0.03654812,
        0.90384732, -0.25255161, -0.96216005,  1.0089745 ])
```

Using Lasso Regularization to experiment with different lambda values:

In [215…
```python
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.pipeline import Pipeline

# lambda = 1 (default regularization parameter)
lasso = Lasso()
lasso.fit(training_set, t_train)
# Making predictions
y_train_reg = lasso.predict(training_set)
y_train_reg
```

Out[215]:
```
array([-1.2490009e-16, -1.2490009e-16, -1.2490009e-16, -1.2490009e-16,
       -1.2490009e-16, -1.2490009e-16, -1.2490009e-16, -1.2490009e-16,
       -1.2490009e-16, -1.2490009e-16, -1.2490009e-16, -1.2490009e-16,
       -1.2490009e-16, -1.2490009e-16, -1.2490009e-16, -1.2490009e-16,
       -1.2490009e-16, -1.2490009e-16, -1.2490009e-16, -1.2490009e-16,
       -1.2490009e-16, -1.2490009e-16, -1.2490009e-16, -1.2490009e-16,
       -1.2490009e-16, -1.2490009e-16, -1.2490009e-16, -1.2490009e-16,
       -1.2490009e-16, -1.2490009e-16, -1.2490009e-16, -1.2490009e-16,
       -1.2490009e-16, -1.2490009e-16, -1.2490009e-16, -1.2490009e-16,
       -1.2490009e-16, -1.2490009e-16, -1.2490009e-16, -1.2490009e-16,
       -1.2490009e-16, -1.2490009e-16, -1.2490009e-16, -1.2490009e-16,
       -1.2490009e-16, -1.2490009e-16, -1.2490009e-16, -1.2490009e-16,
       -1.2490009e-16, -1.2490009e-16, -1.2490009e-16, -1.2490009e-16,
       -1.2490009e-16, -1.2490009e-16, -1.2490009e-16, -1.2490009e-16,
       -1.2490009e-16, -1.2490009e-16, -1.2490009e-16, -1.2490009e-16,
       -1.2490009e-16, -1.2490009e-16, -1.2490009e-16, -1.2490009e-16])
```

In [216…
```python
# lambda = 0.01
lasso1 = Lasso(alpha=0.01)
lasso1.fit(training_set, t_train)
# Making predictions
y_train_reg1 = lasso1.predict(training_set)
y_train_reg1
```

Out[216]:
```
array([ 0.9468468 ,  0.21831547, -1.21272821,  1.75343506,  1.51926428,
       -0.06789327, -1.86320262, -1.10865231,  0.66063806,  0.03618264,
       -0.66632972,  0.14025854, -0.22400712, -0.30206405, -2.43562009,
       -0.69234869, -1.03059538, -0.82244357,  0.03618264,  1.23305554,
        0.4004483 ,  1.5973212 , -1.68106978,  1.62334018,  1.46722632,
       -1.13467129,  0.86878987, -0.56225381, -0.38012098, -0.53623484,
        1.54528325,  0.55656216,  1.62334018,  0.99888475,  0.73869499,
       -1.23874719, -1.16069026,  0.42646728, -1.36884207,  1.18101759,
       -1.13467129,  0.11423957, -0.43215893,  0.86878987, -0.30206405,
        0.81675192,  0.63461909, -0.19798815, -0.48419688,  0.50452421,
       -1.05661436,  1.28509349, -1.47291797, -1.03059538,  0.06220161,
        0.06220161, -0.40613995,  0.53054318, -0.17196917,  0.03618264,
        0.89480885, -0.2500261 , -0.95253845,  0.99888475])
```

In [217…
```python
# lambda = 0.001
lasso2 = Lasso(alpha=0.001)
lasso2.fit(training_set, t_train)
# Making predictions
y_train_reg2 = lasso2.predict(training_set)
y_train_reg2
```

Out[217]:
```
array([ 0.95534661,  0.22015986, -1.2236265 ,  1.76958903,  1.53316698,
       -0.06826022, -1.88018887, -1.11884627,  0.66622078,  0.03706893,
       -0.67235406,  0.14110419, -0.22590106, -0.30460385, -2.45761715,
       -0.69858832, -1.03986902, -0.82995569,  0.03663763,  1.24411957,
        0.40383892,  1.61218344, -1.69627456,  1.63830008,  1.48069845,
       -1.14496291,  0.87640856, -0.56749542, -0.38350269, -0.54110432,
        1.55944045,  0.56132293,  1.63798641,  1.00765831,  0.7450412 ,
       -1.24997839, -1.17139322,  0.43054369, -1.38138496,  1.19184708,
       -1.14488449,  0.11534043, -0.43616726,  0.87715353, -0.30468227,
        0.82448896,  0.63986889, -0.19990204, -0.48855737,  0.50901123,
       -1.06606407,  1.29694097, -1.48616519, -1.03986902,  0.06283269,
        0.0628719 , -0.41001141,  0.53528471, -0.17390303,  0.03644159,
        0.90268204, -0.25256662, -0.96128385,  1.00836406])
```

In [218…
```python
# lambda = 0.0001
lasso3 = Lasso(alpha=0.0001)
lasso3.fit(training_set, t_train)
# Making predictions
y_train_reg3 = lasso3.predict(training_set)
y_train_reg3
```

Out[218]:
```
array([ 0.95632258,  0.22050606, -1.22488904,  1.77092363,  1.5344529 ,
       -0.06858118, -1.88189641, -1.11978153,  0.66724984,  0.03654612,
       -0.67302241,  0.14165848, -0.22627068, -0.30510701, -2.46004165,
       -0.69929607, -1.04092778, -0.83069374,  0.03649507,  1.24535947,
        0.40446625,  1.614254  , -1.69793198,  1.6396098 ,  1.4819001 ,
       -1.14605734,  0.87747055, -0.56790057, -0.38393048, -0.54161702,
        1.56077539,  0.56207799,  1.63958786,  1.008844  ,  0.74609457,
       -1.251174  , -1.17233073,  0.43074309, -1.38259325,  1.19286261,
       -1.14604885,  0.11535921, -0.43649579,  0.87748704, -0.30508954,
        0.82493824,  0.64097242, -0.19998937, -0.48904508,  0.50957618,
       -1.06720604,  1.29795546, -1.48768882, -1.04094427,  0.06280952,
        0.06282077, -0.41022573,  0.53583371, -0.17369568,  0.03650763,
        0.90377545, -0.25254084, -0.9621155 ,  1.0088924 ])
```

1. Build a `GridSearchCV` for each model using `scikit-learn` pipelines.

In [219…
```python
from sklearn.model_selection import GridSearchCV
```

```python
pipe = Pipeline([('lasso_reg', Lasso())]);

# Grid of paramater values for the hyperparameters

param_grid_l_lasso = {'lasso_reg__alpha':[0.00001,0.0001,0.00011,
                                          0.001,0.002,0.003,
                                          0.01, 0.1, 0.5, 1]}
param_grid_l_lasso
```

Out[219]:

```
{'lasso_reg__alpha': [1e-05,
  0.0001,
  0.00011,
  0.001,
  0.002,
  0.003,
  0.01,
  0.1,
  0.5,
  1]}
```

In [220…

```python
grid_search_l_lasso = GridSearchCV(pipe,
                                   param_grid=param_grid_l_lasso,
                                  cv=10,
                                  refit=True);
grid_search_l_lasso
```

Out[220]:

```
GridSearchCV(cv=10, estimator=Pipeline(steps=[('lasso_reg', Lasso())]),
             param_grid={'lasso_reg__alpha': [1e-05, 0.0001, 0.00011, 0.001,
                                              0.002, 0.003, 0.01, 0.1, 0.5,
                                              1]})
```

In [221…

```python
grid_search_l_lasso.fit(training_set, t_train);
```

In [222…

```python
# Check best set of hyperparameters based on training data
grid_search_l_lasso.best_params_
```

Out[222]:

```
{'lasso_reg__alpha': 0.00011}
```

In [223…

```python
# Access final estimator to automatically fill alpha parameter
# with the best value found during hyperparamter tuning
# (lambda=0.00011)
final_model_lin_lasso = grid_search_l_lasso.best_estimator_
grid_search_l_lasso.best_score_
```

Out[223]:

```
0.9999999870492134
```

1. Build a `RandomizedSearchCV` for each model using `scikit-learn` pipelines.

- Consider an exponential distribution for the regularizer term λ.
- Consider a uniform distribution for the number of trees.

In [224…

```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import expon

# model
pipe = Pipeline([('lasso_reg', Lasso())]);
```

```
param_grid_r = {'lasso_reg__alpha': expon()}
rand_search_l_lasso = RandomizedSearchCV(pipe,
                                          param_distributions=param_grid_r,
                                         n_iter=100,
                                         cv=10,
                                         refit=True);
rand_search_l_lasso
```

Out[224]:
```
RandomizedSearchCV(cv=10, estimator=Pipeline(steps=[('lasso_reg', Lasso())]),
                   n_iter=100,
                   param_distributions={'lasso_reg__alpha': <scipy.stats._dis
tn_infrastructure.rv_frozen object at 0x7fc1659a7fa0>})
```

In [225…
```
rand_search_l_lasso.fit(training_set, t_train);
```

In [226…
```
# Check best set of hyperparameters based on training data
rand_search_l_lasso.best_params_
```

Out[226]:
```
{'lasso_reg__alpha': 0.007052810874333136}
```

In [227…
```
rand_search_l_lasso.best_estimator_
```

Out[227]:
```
Pipeline(steps=[('lasso_reg', Lasso(alpha=0.007052810874333136))])
```

In [228…
```
rand_search_l_lasso.best_score_
```

Out[228]:
```
0.9999410339126735
```

Model 2: polynomial regression with Lasso regularizer.

For model 2, experiment with polynomial degree, the interaction_only argument and the regularizer term λ .

1. Build a `GridSearchCV` for each model using `scikit-learn` pipelines.

In [229…
```
from sklearn.preprocessing import PolynomialFeatures

# Pipeline to implement polynomial features and then find the
# solution for lasso regression (interaction set to False)
lasso_poly_pipe = Pipeline([('poly_feat',PolynomialFeatures()),
                           ('lasso_reg',Lasso())])

param_grid_poly = {'poly_feat__degree': list(range(1,12)),
              'lasso_reg__alpha': [0.0001,0.001,0.01,0.02,
                                   0.03,0.1,0.2,
                                   0.3,0.4,0.5,1],
                  'lasso_reg__tol':[0.5]}

param_grid_poly
```

Out[229]:
```
{'poly_feat__degree': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
 'lasso_reg__alpha': [0.0001,
  0.001,
  0.01,
  0.02,
  0.03,
  0.1,
  0.2,
  0.3,
  0.4,
  0.5,
  1],
 'lasso_reg__tol': [0.5]}
```

In [230…
```
grid_search_poly = GridSearchCV(lasso_poly_pipe,
                                param_grid=param_grid_poly,
                                cv=10,
                                refit=True)

grid_search_poly
```

Out[230]:
```
GridSearchCV(cv=10,
             estimator=Pipeline(steps=[('poly_feat', PolynomialFeatures()),
                                       ('lasso_reg', Lasso())]),
             param_grid={'lasso_reg__alpha': [0.0001, 0.001, 0.01, 0.02, 0.0
3,
                                              0.1, 0.2, 0.3, 0.4, 0.5, 1],
                         'lasso_reg__tol': [0.5],
                         'poly_feat__degree': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                                               11]})
```

In [231…
```
# Training the parameters
grid_search_poly.fit(training_set, t_train)
```

Out[231]:
```
GridSearchCV(cv=10,
             estimator=Pipeline(steps=[('poly_feat', PolynomialFeatures()),
                                       ('lasso_reg', Lasso())]),
             param_grid={'lasso_reg__alpha': [0.0001, 0.001, 0.01, 0.02, 0.0
3,
                                              0.1, 0.2, 0.3, 0.4, 0.5, 1],
                         'lasso_reg__tol': [0.5],
                         'poly_feat__degree': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                                               11]})
```

In [232…
```
grid_search_poly.best_params_
```

Out[232]:
```
{'lasso_reg__alpha': 0.02, 'lasso_reg__tol': 0.5, 'poly_feat__degree': 3}
```

In [233…
```
grid_search_poly.best_estimator_
```

Out[233]:
```
Pipeline(steps=[('poly_feat', PolynomialFeatures(degree=3)),
                ('lasso_reg', Lasso(alpha=0.02, tol=0.5))])
```

In [234…
```
grid_search_poly.best_score_

# Save the GridSearchCV as the final model for polynomial lasso regularization
final_model_poly = grid_search_poly.best_estimator_;
```

In [235…
```
# Pipeline to implement polynomial features and then find the
# solution for lasso regression (interaction set to True)
```

```
lasso_poly_pipe = Pipeline([('poly_feat',PolynomialFeatures()),
                            ('lasso_reg',Lasso())]);

param_grid_poly2 = {'poly_feat__degree': list(range(1,12)),
              'lasso_reg__alpha': [0.0001,0.001,0.01,0.02,
                                   0.03,0.1,0.2,
                                   0.3,0.4,0.5,1],
                  'lasso_reg__tol':[0.5],
                  'poly_feat__interaction_only':[True]}

param_grid_poly2
```

Out[235]:
```
{'poly_feat__degree': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
 'lasso_reg__alpha': [0.0001,
  0.001,
  0.01,
  0.02,
  0.03,
  0.1,
  0.2,
  0.3,
  0.4,
  0.5,
  1],
 'lasso_reg__tol': [0.5],
 'poly_feat__interaction_only': [True]}
```

In [236…
```
grid_search_poly2 = GridSearchCV(lasso_poly_pipe,
                               param_grid=param_grid_poly2,
                               cv=10,
                               refit=True)

grid_search_poly2
```

Out[236]:
```
GridSearchCV(cv=10,
             estimator=Pipeline(steps=[('poly_feat', PolynomialFeatures()),
                                       ('lasso_reg', Lasso())]),
             param_grid={'lasso_reg__alpha': [0.0001, 0.001, 0.01, 0.02, 0.0
3,
                                              0.1, 0.2, 0.3, 0.4, 0.5, 1],
                         'lasso_reg__tol': [0.5],
                         'poly_feat__degree': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                                               11],
                         'poly_feat__interaction_only': [True]})
```

In [237…
```
# Training the parameters
grid_search_poly2.fit(training_set, t_train);
```

In [238…
```
grid_search_poly2.best_params_
```

Out[238]:
```
{'lasso_reg__alpha': 0.02,
 'lasso_reg__tol': 0.5,
 'poly_feat__degree': 2,
 'poly_feat__interaction_only': True}
```

In [239…
```
grid_search_poly2.best_estimator_
```

Out[239]:
```
Pipeline(steps=[('poly_feat', PolynomialFeatures(interaction_only=True)),
                ('lasso_reg', Lasso(alpha=0.02, tol=0.5))])
```

In [240…
```
grid_search_poly2.best_score_
```

Out[240]:
```
0.8973647939100733
```

In [241…
```
# Access final estimator to automatically fill alpha parameter
# with the best value found during hyperparamter tuning
final_model_gs_poly = grid_search_poly.best_estimator_
```

1. Build a `RandomizedSearchCV` for each model using `scikit-learn` pipelines.
   - Consider an exponential distribution for the regularizer term λ.
   - Consider a uniform distribution for the number of trees.

In [242…
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import expon

# model
pipe = Pipeline([('poly_feat',PolynomialFeatures()),
                           ('lasso_reg',Lasso())])

param_grid_r = {
              'lasso_reg__alpha': [0.0001,0.001,0.01,0.02,
                                   0.03,0.1,0.2,
                                   0.3,0.4,0.5,1],
                  'lasso_reg__tol':[0.5],
               'poly_feat__degree': list(range(1,12))}
rand_search_poly_lasso = RandomizedSearchCV(pipe,
                                         param_distributions=param_grid_r,
                                        n_iter=100,
                                        cv=10,
                                        refit=True);
rand_search_poly_lasso
```

Out[242]:
```
RandomizedSearchCV(cv=10,
                   estimator=Pipeline(steps=[('poly_feat',
                                              PolynomialFeatures()),
                                             ('lasso_reg', Lasso())]),
                   n_iter=100,
                   param_distributions={'lasso_reg__alpha': [0.0001, 0.001,
                                                              0.01, 0.02, 0.0
3,
                                                              0.1, 0.2, 0.3,
0.4,
                                                              0.5, 1],
                                        'lasso_reg__tol': [0.5],
                                        'poly_feat__degree': [1, 2, 3, 4, 5,
6,
                                                               7, 8, 9, 10,
                                                               11]})
```

In [243…
```
rand_search_poly_lasso.fit(training_set, t_train);
```

In [244…
```
rand_search_poly_lasso.best_params_
```

Out[244]:
```
{'poly_feat__degree': 3, 'lasso_reg__tol': 0.5, 'lasso_reg__alpha': 0.02}
```

In [245…
```
rand_search_poly_lasso.best_estimator_
```

Out[245]:
```
Pipeline(steps=[('poly_feat', PolynomialFeatures(degree=3)),
                ('lasso_reg', Lasso(alpha=0.02, tol=0.5))])
```

In [246…
```
rand_search_poly_lasso.best_score_
```

Out[246]:
```
0.9023738686642325
```

- Results GridSearchCV: When iteraction_only is false, the best polynomial degree M was 3 and the best regularizer term λ was 0.02 When the iteraction_only was set to True, the best polynomial degree M was 2 and the best regularizer term λ was 0.02
- Results RandomizedSeachCV: best degree M = 3 and the best polynomial degree λ=0.02

Model 3: random forest.

For model 3, experiment with the number of trees and the criterion.

1. Build a `GridSearchCV` for each model using `scikit-learn` pipelines.

In [247…
```
from sklearn.ensemble import RandomForestRegressor

pipe = Pipeline([('rand_forest',RandomForestRegressor())])

param_grid_rf = {'rand_forest__criterion':["squared_error", "absolute_error"],
                 'rand_forest__n_estimators':[50, 75, 100, 125, 150,
                                              175, 200, 225, 250, 275,
                                              300,325,350,400]}

param_grid_rf
```

Out[247]:
```
{'rand_forest__criterion': ['squared_error', 'absolute_error'],
 'rand_forest__n_estimators': [50,
  75,
  100,
  125,
  150,
  175,
  200,
  225,
  250,
  275,
  300,
  325,
  350,
  400]}
```

In [248…
```
grid_search_forest = GridSearchCV(pipe,
                                  param_grid=param_grid_rf,
                                  cv=5,
                                  refit=True);
grid_search_forest
```

Out[248]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('rand_forest',
                                        RandomForestRegressor())]),
             param_grid={'rand_forest__criterion': ['squared_error',
                                                    'absolute_error'],
                         'rand_forest__n_estimators': [50, 75, 100, 125, 150,
                                                       175, 200, 225, 250, 27
5,
                                                       300, 325, 350, 400]})
```

In [249…
```
grid_search_forest.fit(training_set,t_train);
```

In [250…
```
grid_search_forest.best_params_
```

Out[250]:
```
{'rand_forest__criterion': 'absolute_error', 'rand_forest__n_estimators': 30
0}
```

In [301…
```
grid_search_forest.best_estimator_

# Save GridSearchCV as the final model for forest regression
final_model_forest_reg = grid_search_forest.best_estimator_;
```

In [252…
```
grid_search_forest.best_score_
```

Out[252]:
```
0.9832896989050128
```

1. Build a `RandomizedSearchCV` for each model using `scikit-learn` pipelines.
   - Consider an exponential distribution for the regularizer term λ.
   - Consider a uniform distribution for the number of trees.

In [197…
```
from scipy.stats import uniform
# model
pipe_rf = Pipeline([('rand_forest',RandomForestRegressor())]);

param_grid_rf = {'rand_forest__criterion':["squared_error", "absolute_error"],
                'rand_forest__n_estimators':[(int)(uniform.rvs(10,350))]}
```

In [198…
```
rand_search_rf = RandomizedSearchCV(pipe_rf,
                                    param_distributions=param_grid_rf,
                                    cv=5,
                                    n_iter=1,
                                    refit=True);
rand_search_rf
```

Out[198]:
```
RandomizedSearchCV(cv=5,
                   estimator=Pipeline(steps=[('rand_forest',
                                              RandomForestRegressor())]),
                   n_iter=1,
                   param_distributions={'rand_forest__criterion': ['squared_e
rror',
                                                                   'absolute_
error'],
                                        'rand_forest__n_estimators': [88]})
```

In [199…
```
rand_search_rf.fit(training_set, t_train)
```

Out[199]:
```
RandomizedSearchCV(cv=5,
                   estimator=Pipeline(steps=[('rand_forest',
                                              RandomForestRegressor())]),
                   n_iter=1,
                   param_distributions={'rand_forest__criterion': ['squared_e
rror',
                                                                  'absolute_
error'],
                                        'rand_forest__n_estimators': [88]})
```

In [200…
```
rand_search_rf.best_params_
```

Out[200]:
```
{'rand_forest__n_estimators': 88, 'rand_forest__criterion': 'absolute_error'}
```

In [201…
```
rand_search_rf.best_estimator_
```

Out[201]:
```
Pipeline(steps=[('rand_forest',
                 RandomForestRegressor(criterion='absolute_error',
                                       n_estimators=88))])
```

In [254…
```
rand_search_rf.best_score_
```

Out[254]:
```
0.9818136615212172
```

- Results GridSearchCV: best criteron= 'absolute_error', best number of trees=300
- Results RandomSearchCV: best criterion = 'absolute_error', best number of trees=88

Which set of hyperparmeters worked best for each model?

- Model 1:
  - λ = 0.00011
- Model 2:
  - λ = 0.02
  - M = 3
  - Iteraction_only = False
- Model 3:
  - number of trees = 150
  - criterion = 'absolute error'

4. Train the final models and save them as `pickle` files. (See bottom of lecture 5 part 1 for an example.)

In [268…
```
import joblib

# Train final model for linear regression with lasso regularization

final_model_lin_lasso
joblib.dump(final_model_lin_lasso,'final_model_lin_lasso.pkl')
```

Out[268]:
```
['final_model_lin_lasso.pkl']
```

In [269…
```
# Train final model for polynomial regression with lasso regularization

final_model_poly
joblib.dump(final_model_poly,'final_model_poly.pkl')
```

Out[269]:
```
['final_model_poly.pkl']
```

In [270…
```
# Train final model for random forest regression

final_model_forest_reg
joblib.dump(final_model_forest_reg,'final_model_forest_reg.pkl')
```

Out[270]:
```
['final_model_forest_reg.pkl']
```

Question 2 (17.5 points)

1. Load your trained models and evaluate the performance in the test set.
2. Report the RMSE performance and the 95% CI.
3. Based on these results, which model would you select?

In [302…
```
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from scipy import stats

# Load model for linear regression with lasso regularization
lin_lasso_loaded = joblib.load('final_model_lin_lasso.pkl')

#Predictions
y_train_lin = lin_lasso_loaded.predict(training_set)
y_test_lin = lin_lasso_loaded.predict(test_set)

final_rmse_train_lin = np.sqrt(mean_squared_error(t_train,y_train_lin))
print('RMSE Train: ', final_rmse_train_lin)

final_rmse_test_lin = np.sqrt(mean_squared_error(t_test, y_test_lin))
print('RMSE Test: ', final_rmse_test_lin)
```

```
RMSE Train:  0.00011000000000006289
RMSE Test:  0.00010689240386261003
```

In [303…
```
confidence = 0.95
squared_errors = (t_test - y_test_lin) ** 2
T = stats.t(df=len(squared_errors)-1,
            loc = squared_errors.mean(),
            scale=squared_errors.std(ddof=1)/np.sqrt(len(squared_errors)))
np.sqrt(T.ppf(0.025)), np.sqrt(T.ppf(0.975))
# 95% CI
```

Out[303]:
```
(7.160385460773568e-05, 0.0001331347438213215)
```

In [304…

```
# Load model for polynomial regression with lasso regularization
poly_lasso_loaded = joblib.load('final_model_poly.pkl')

#Predictions
y_train_poly = poly_lasso_loaded.predict(training_set)
y_test_poly = poly_lasso_loaded.predict(test_set)

final_rmse_train_poly = np.sqrt(mean_squared_error(t_train,y_train_poly))
print('RMSE Train: ', final_rmse_train_poly)

final_rmse_test_poly = np.sqrt(mean_squared_error(t_test, y_test_poly))
print('RMSE Test: ', final_rmse_test_poly)
```

```
RMSE Train:  0.19624057311681423
RMSE Test:  0.1990815158421269
```

In [305…

```
confidence = 0.95
squared_errors = (t_test - y_test_poly) ** 2
T = stats.t(df=len(squared_errors)-1,
           loc = squared_errors.mean(),
           scale=squared_errors.std(ddof=1)/np.sqrt(len(squared_errors)))
np.sqrt(T.ppf(0.025)), np.sqrt(T.ppf(0.975))
# 95% CI
```

Out[305]:

```
(0.07546779380266869, 0.2712406901601527)
```

In [306…

```
# Load model for random forest regression
forest_reg_loaded = joblib.load('final_model_forest_reg.pkl')

#Predictions
y_train_fr = forest_reg_loaded.predict(training_set)
y_test_fr = forest_reg_loaded.predict(test_set)

final_rmse_train_fr = np.sqrt(mean_squared_error(t_train,y_train_fr))
print('RMSE Train: ', final_rmse_train_fr)

final_rmse_test_fr = np.sqrt(mean_squared_error(t_test, y_test_fr))
print('RMSE Test: ', final_rmse_test_fr)
```

```
RMSE Train:  0.04872297941457989
RMSE Test:  0.06292372323367616
```

In [307…

```
confidence = 0.95
squared_errors = (t_test - y_test_fr) ** 2
T = stats.t(df=len(squared_errors)-1,
           loc = squared_errors.mean(),
           scale=squared_errors.std(ddof=1)/np.sqrt(len(squared_errors)))
np.sqrt(T.ppf(0.025)), np.sqrt(T.ppf(0.975))

# 95% CI
```

```
/var/folders/1_/2tjx84411dl0n9rvnrjdns8c0000gn/T/ipykernel_1132/1697473024.py:
6: RuntimeWarning: invalid value encountered in sqrt
  np.sqrt(T.ppf(0.025)), np.sqrt(T.ppf(0.975))
```

Out[307]:

```
(nan, 0.0919004147236293)
```

Based on these results, I would choose the Linear regression with Lasso regularizer based on the RMSE score and CI.

Submit Your Solution

Confirm that you've successfully completed the assignment.

Along with the Notebook, include a PDF of the notebook with your solutions.

`add` and `commit` the final version of your work, and `push` your code to your GitHub repository.

Submit the URL of your GitHub Repository as your assignment submission on Canvas.