MS&E 349: Homework 4

Group 3

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
```

```
/Users/chih-hsuankao/.pyenv/versions/anaconda3-2019.03/lib/py
thon3.7/site-packages/scipy/__init__.py:137: UserWarning: Num
Py 1.16.5 or above is required for this version of SciPy (det
ected version 1.16.2)
  UserWarning)
```

Predict monthly S&P 500 index returns with the following financial variables:

In [3]:

```
df = pd.read_csv('ret_predictors.csv')
df['yyyymm'] = pd.to_datetime(df['yyyymm'], format='%Y%m', errors='coerce'
).dropna()

# shift the targeting y (last column) by 1
df['ret'] = df['ret (S&P 500 annualized log return including dividends)'].
shift(periods=-1)

df
```

Out[3]:

	Unnamed: 0	yyyymm	b/m	tbl	AAA	BAA	lty	ntis	
0	1	1927-01-01	0.443706	0.0323	0.0466	0.0561	0.0351	0.050834	0.0
1	2	1927-02-01	0.428501	0.0329	0.0467	0.0559	0.0347	0.051682	0.0
2	3	1927-03-01	0.469765	0.0320	0.0462	0.0554	0.0331	0.046370	0.0
3	4	1927-04-01	0.456754	0.0339	0.0458	0.0548	0.0333	0.050518	0.0

4	5	1927-05-01	0.434783	0.0333	0.0457	0.0550	0.0327	0.055279	0.0
5	6	1927-06-01	0.452385	0.0307	0.0458	0.0555	0.0334	0.058826	0.0
6	7	1927-07-01	0.414553	0.0296	0.0460	0.0555	0.0333	0.059754	0.0
7	8	1927-08-01	0.396227	0.0270	0.0456	0.0548	0.0329	0.054526	0.0
8	9	1927-09-01	0.380586	0.0268	0.0454	0.0542	0.0330	0.094617	0.0
9	10	1927-10-01	0.413801	0.0308	0.0451	0.0538	0.0325	0.094370	0.0
10	11	1927-11-01	0.379396	0.0304	0.0449	0.0535	0.0320	0.082270	0.0
11	12	1927-12-01	0.374689	0.0317	0.0446	0.0532	0.0316	0.076474	0.0
12	13	1928-01-01	0.378670	0.0331	0.0446	0.0535	0.0321	0.062605	0.0
13	14	1928-02-01	0.386077	0.0333	0.0446	0.0533	0.0318	0.055172	0.0
14	15	1928-03-01	0.363255	0.0327	0.0446	0.0532	0.0317	0.054364	0.0
15	16	1928-04-01	0.368095	0.0362	0.0446	0.0533	0.0319	0.049372	0.0
16	17	1928-05-01	0.354397	0.0390	0.0449	0.0542	0.0327	0.047187	0.0
17	18	1928-06-01	0.370300	0.0392	0.0457	0.0555	0.0326	0.050298	0.0
18	19	1928-07-01	0.360648	0.0412	0.0461	0.0558	0.0344	0.059380	0.0
19	20	1928-08-01	0.324030	0.0436	0.0464	0.0561	0.0341	0.057398	0.0
20	21	1928-09-01	0.328166	0.0457	0.0461	0.0559	0.0346	0.027979	0.0
21	22	1928-10-01	0.308931	0.0470	0.0461	0.0558	0.0336	0.034018	0.0
22	23	1928-11-01	0.265526	0.0426	0.0458	0.0555	0.0338	0.038372	0.0
23	24	1928-12-01	0.259667	0.0426	0.0461	0.0560	0.0340	0.063068	0.0
24	25	1929-01-01	0.245347	0.0466	0.0462	0.0563	0.0349	0.078448	0.0
25	26	1929-02-01	0.245424	0.0439	0.0466	0.0566	0.0363	0.071782	0.0
26	27	1929-03-01	0.272300	0.0460	0.0470	0.0579	0.0377	0.079803	0.0
27	28	1929-04-01	0.263397	0.0480	0.0469	0.0580	0.0358	0.099320	0.0
28	29	1929-05-01	0.282775	0.0509	0.0470	0.0580	0.0373	0.117985	0.0
29	30	1929-06-01	0.253581	0.0480	0.0477	0.0594	0.0367	0.116196	0.0

...	...	...	...	...	...	...	...	...	
1062	1063	2015-07-01	0.308953	0.0003	0.0415	0.0520	0.0263	-0.008070	0.0
1063	1064	2015-08-01	0.330671	0.0007	0.0404	0.0519	0.0264	-0.009535	0.0
1064	1065	2015-09-01	0.335612	0.0002	0.0407	0.0534	0.0253	-0.012923	0.0
1065	1066	2015-10-01	0.309414	0.0002	0.0395	0.0534	0.0259	-0.016208	0.0
1066	1067	2015-11-01	0.308429	0.0012	0.0406	0.0546	0.0265	-0.017810	0.0
1067	1068	2015-12-01	0.313649	0.0023	0.0397	0.0546	0.0268	-0.021611	0.0
1068	1069	2016-01-01	0.331911	0.0026	0.0400	0.0545	0.0236	-0.020262	0.0
1069	1070	2016-02-01	0.330902	0.0031	0.0396	0.0534	0.0217	-0.024023	0.0
1070	1071	2016-03-01	0.327955	0.0029	0.0382	0.0513	0.0218	-0.022999	0.0
1071	1072	2016-04-01	0.326321	0.0023	0.0362	0.0479	0.0223	-0.023554	0.0
1072	1073	2016-05-01	0.326072	0.0027	0.0365	0.0468	0.0219	-0.027005	0.0
1073	1074	2016-06-01	0.323475	0.0027	0.0350	0.0453	0.0179	-0.028683	0.0
1074	1075	2016-07-01	0.314661	0.0030	0.0328	0.0422	0.0175	-0.031666	0.0
1075	1076	2016-08-01	0.315197	0.0030	0.0332	0.0424	0.0186	-0.030725	0.0
1076	1077	2016-09-01	0.316794	0.0029	0.0341	0.0431	0.0196	-0.032610	0.0
1077	1078	2016-10-01	0.319688	0.0033	0.0351	0.0438	0.0220	-0.028997	0.0
1078	1079	2016-11-01	0.303286	0.0045	0.0386	0.0471	0.0267	-0.027361	0.0
1079	1080	2016-12-01	0.293479	0.0051	0.0406	0.0483	0.0272	-0.025012	0.0
1080	1081	2017-01-01	0.291980	0.0051	0.0392	0.0466	0.0278	-0.022562	0.0
1081	1082	2017-02-01	0.278678	0.0052	0.0395	0.0464	0.0270	-0.018621	0.0
1082	1083	2017-03-01	0.281599	0.0074	0.0401	0.0468	0.0274	-0.016151	0.0
1083	1084	2017-04-01	0.277870	0.0080	0.0387	0.0457	0.0265	-0.015497	0.0
1084	1085	2017-05-01	0.276969	0.0089	0.0385	0.0455	0.0256	-0.010100	0.0
1085	1086	2017-06-01	0.272545	0.0098	0.0368	0.0437	0.0258	-0.009702	0.0
1086	1087	2017-07-01	0.265804	0.0107	0.0370	0.0439	0.0262	-0.013104	0.0

1087	1088	2017-08-01	0.265114	0.0101	0.0363	0.0431	0.0242	-0.012138	0.0
1088	1089	2017-09-01	0.259706	0.0103	0.0363	0.0430	0.0259	-0.011027	0.0
1089	1090	2017-10-01	0.248906	0.0107	0.0360	0.0432	0.0261	-0.012358	0.0
1090	1091	2017-11-01	0.239727	0.0123	0.0357	0.0427	0.0260	-0.012243	0.0
1091	1092	2017-12-01	0.235393	0.0132	0.0351	0.0422	0.0254	-0.019946	0.0

1092 rows × 19 columns

In this problem, we are going to use the data from January 1927 to January 1985 as the training set, the data from February 1985 to January 1997 as the validation set and and the data from January 1997 to Nov 2017 as the test set.

Report the MSE and R-squared for the training, validation and test data for each of the following methods and briefly interpret the results.

In [4]:

```python
df = df.set_index(df['yyyymm'])

df_train = df['1927-01-01':'1985-02-01']
df_validation = df['1985-02-01':'1997-01-01']
df_test = df['1997-01-01':].dropna() # drop row with na

print('Train Dataset:',df_train.shape)
print('Validation Dataset:',df_validation.shape)
print('Test Dataset:',df_test.shape)
```

```
Train Dataset: (698, 19)
Validation Dataset: (144, 19)
Test Dataset: (251, 19)
```

In [5]:

```python
X_train = df_train.loc[:, 'b/m':'d/e'].to_numpy()
y_train = df_train['ret'].to_numpy()

X_validation = df_validation.loc[:, 'b/m':'d/e'].to_numpy()
y_validation = df_validation['ret'].to_numpy()

X_test = df_test.loc[:, 'b/m':'d/e'].to_numpy()
y_test = df_test['ret'].to_numpy()

print('Train X shape:',X_train.shape)
print('Train Y shape:',y_train.shape)
print('Validation X shape:',X_validation.shape)
print('Validation Y shape:',y_validation.shape)
print('Test X shape:',X_test.shape)
print('Test Y shape:',y_test.shape)
```

```
Train X shape: (698, 15)
Train Y shape: (698,)
Validation X shape: (144, 15)
```

```
Validation X shape: (144, 15)
Validation Y shape: (144,)
Test X shape: (251, 15)
Test Y shape: (251,)
```

1. Linear Model

In [6]:

```
regr = linear_model.LinearRegression()
```

In [7]:

```
regr.fit(X_train, y_train)
print('Coefficients: \n', regr.coef_)
```

```
Coefficients:
 [ 6.14445366e-02 -4.46310302e+04  8.93883395e-01  1.50118456
e-01
 -1.31223497e+00 -5.90874643e-02  5.35571580e+05 -7.59587409e
-01
 -1.15216619e-01  1.61071656e-01  2.34259229e-01  1.67941016e
+06
  8.62247957e-02 -1.67941027e+06 -1.67941031e+06]
```

In [8]:

```
regr_pred_train = regr.predict(X_train)
regr_pred_validation = regr.predict(X_validation)
regr_pred_test = regr.predict(X_test)
```

In [9]:

```
print("========Train========")
print('MSE: %.6f' % mean_squared_error(y_train, regr_pred_train))
print('R_squared: %.6f' % r2_score(y_train, regr_pred_train))
print("======Validation======")
print('MSE: %.6f' % mean_squared_error(y_validation, regr_pred_validation
))
print('R_squared: %.6f' % r2_score(y_validation, regr_pred_validation))
print("=========Test=========")
print('MSE: %.6f' % mean_squared_error(y_test, regr_pred_test))
print('R_squared: %.6f' % r2_score(y_test, regr_pred_test))
```

```
========Train========
MSE: 0.003427
R_squared: 0.051918
======Validation======
MSE: 0.002209
R_squared: -0.280197
=========Test=========
MSE: 0.002390
R_squared: -0.300502
```

2. Penalized Linear Model

In [10]:

```
from sklearn.linear_model import Lasso, LassoCV #Q2.1
from sklearn.linear_model import Ridge, RidgeCV #Q2.2
from sklearn.linear_model import ElasticNet, ElasticNetCV #Q2.3
```

Let ρ = 0 (i.e. Lasso), find the optimal λ from the “cross validation” set and report the MSE and R^2 in the training, cross validation and test sets.

In [11]:

```
lassocv = LassoCV(alphas = None, cv = 10, max_iter = 100000, normalize = T
rue)
lassocv.fit(X_validation, y_validation) # cross-validation

lasso = Lasso(max_iter = 10000, normalize = True)
lasso.set_params(alpha=lassocv.alpha_)
lasso.fit(X_train, y_train)
```

Out[11]:

```
Lasso(alpha=0.0006030459741131299, copy_X=True, fit_intercept
=True,
   max_iter=10000, normalize=True, positive=False, precompute
=False,
   random_state=None, selection='cyclic', tol=0.0001, warm_st
art=False)
```

In [12]:

```
print('Optimal lambda from cross validation is ', lassocv.alpha_)
```

```
Optimal lambda from cross validation is  0.000603045974113129
9
```

In [13]:

```
lasso_pred_train = lasso.predict(X_train)
lasso_pred_validation = lasso.predict(X_validation)
lasso_pred_test = lasso.predict(X_test)
```

In [14]:

```
print("========Train========")
print('MSE: %.6f' % mean_squared_error(y_train, lasso_pred_train))
print('R_squared: %.6f' % r2_score(y_train, lasso_pred_train))
print("======Validation======")
print('MSE: %.6f' % mean_squared_error(y_validation, lasso_pred_validation
))
print('R_squared: %.6f' % r2_score(y_validation, lasso_pred_validation))
print("=========Test=========")
print('MSE: %.6f' % mean_squared_error(y_test, lasso_pred_test))
print('R_squared: %.6f' % r2_score(y_test, lasso_pred_test))
```

```
========Train========
MSE: 0.003615
R_squared: 0.000000
======Validation======
MSE: 0.001758
```

```
R_squared: -0.019107
=========Test=========
MSE: 0.001838
R_squared: -0.000172
```

Let ρ = 1 (i.e. Ridge Regression), find the optimal λ and report the MSE and R^2 as part 1.

In [15]:

```
ridgecv = RidgeCV(normalize = True)
ridgecv.fit(X_validation, y_validation)
```

Out[15]:

```
RidgeCV(alphas=array([ 0.1,  1. , 10. ]), cv=None, fit_interc
ept=True,
    gcv_mode=None, normalize=True, scoring=None, store_cv_val
ues=False)
```

In [16]:

```
print('Optimal lambda from cross validation is ', ridgecv.alpha_)
```

```
Optimal lambda from cross validation is  10.0
```

In [17]:

```
ridge = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge.fit(X_train, y_train)

ridge_pred_train = ridge.predict(X_train)
ridge_pred_validation = ridge.predict(X_validation)
ridge_pred_test = ridge.predict(X_test)
```

In [18]:

```
print("========Train========")
print('MSE: %.6f' % mean_squared_error(y_train, ridge_pred_train))
print('R_squared: %.6f' % r2_score(y_train, ridge_pred_train))
print("======Validation======")
print('MSE: %.6f' % mean_squared_error(y_validation, ridge_pred_validation
))
print('R_squared: %.6f' % r2_score(y_validation, ridge_pred_validation))
print("=========Test=========")
print('MSE: %.6f' % mean_squared_error(y_test, ridge_pred_test))
print('R_squared: %.6f' % r2_score(y_test, ridge_pred_test))
```

```
========Train========
MSE: 0.003584
R_squared: 0.008494
======Validation======
MSE: 0.001767
R_squared: -0.024088
=========Test=========
MSE: 0.001846
R_squared: -0.004782
```

Find the optimal λ and ρ $(0 \leq \rho \leq 1)$ and report the estimation errors as part 1.

In [19]:

```
elasticcv = ElasticNetCV(normalize = True)
elasticcv.fit(X_validation, y_validation)
```

```
/Users/chih-hsuankao/.pyenv/versions/anaconda3-2019.03/lib/py
thon3.7/site-packages/sklearn/model_selection/_split.py:2053:
FutureWarning: You should specify a value for 'cv' instead of
relying on the default value. The default value will change f
rom 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)
```

Out[19]:

```
ElasticNetCV(alphas=None, copy_X=True, cv='warn', eps=0.001,
       fit_intercept=True, l1_ratio=0.5, max_iter=1000, n_alp
has=100,
       n_jobs=None, normalize=True, positive=False, precomput
e='auto',
       random_state=None, selection='cyclic', tol=0.0001, ver
bose=0)
```

In [20]:

```
print('Optimal lambda from cross validation is ', elasticcv.alpha_)
print('Optimal rho from cross validation is ', elasticcv.l1_ratio_)
```

```
Optimal lambda from cross validation is  0.001206091948226258
6
Optimal rho from cross validation is  0.5
```

In [21]:

```
elastic = ElasticNet(alpha = elasticcv.alpha_,  l1_ratio = elasticcv.l1_ra
tio_, normalize = True)
elastic.fit(X_train, y_train)

elastic_pred_train = elastic.predict(X_train)
elastic_pred_validation = elastic.predict(X_validation)
elastic_pred_test = elastic.predict(X_test)
```

In [22]:

```
print("========Train========")
print('MSE: %.6f' % mean_squared_error(y_train, elastic_pred_train))
print('R_squared: %.6f' % r2_score(y_train, elastic_pred_train))
print("======Validation======")
print('MSE: %.6f' % mean_squared_error(y_validation, elastic_pred_validati
on))
print('R_squared: %.6f' % r2_score(y_validation, elastic_pred_validation))
print("=========Test=========")
print('MSE: %.6f' % mean_squared_error(y_test, elastic_pred_test))
print('R_squared: %.6f' % r2_score(y_test, elastic_pred_test))
```

```
========Train========
MSE: 0.003615
R_squared: 0.000000
```

```
======Validation======
MSE: 0.001758
R_squared: -0.019107
=========Test=========
MSE: 0.001838
R_squared: -0.000172
```

3. Principle Component Regression

In [51]:

```
from sklearn import model_selection
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
```

In [52]:

```
pd_X_train = df_train.loc[:, 'b/m':'d/e']
pd_y_train = df_train.loc[:, 'ret (S&P 500 annualized log return including
dividends)']

pd_X_validation = df_validation.loc[:, 'b/m':'d/e']
pd_y_validation = df_validation.loc[:, 'ret (S&P 500 annualized log return
including dividends)']

pd_X_test = df_test.loc[:, 'b/m':'d/e']
pd_y_test = df_test.loc[:, 'ret (S&P 500 annualized log return including d
ividends)']

pd_X_train = df_train.loc[:, 'b/m':'d/e']
pd_y_train = df_train['ret']

pd_X_validation = df_validation.loc[:, 'b/m':'d/e']
pd_y_validation = df_validation['ret']

pd_X_test = df_test.loc[:, 'b/m':'d/e']
pd_y_test = df_test['ret']
```

In [53]:

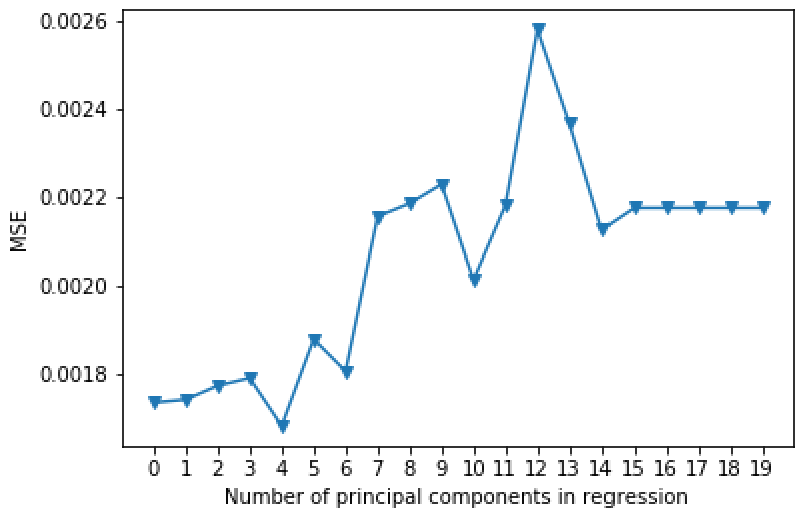
```
pca = PCA()
regr = linear_model.LinearRegression()
X_reduced_validation = pca.fit_transform(scale(pd_X_validation))
n = len(X_reduced_validation)

# 10-fold CV, with shuffle
kf_10 = model_selection.KFold(n_splits=10, shuffle=True, random_state=1)

mse = []

# Calculate MSE with only the intercept (no principal components in regres
sion)
score = -1*model_selection.cross_val_score(regr, np.ones((n,1)),
                                           pd_y_validation.ravel(),
                                           cv = kf_10,
                                           scoring='neg_mean_squared_erro
r').mean()
mse.append(score)
```

```
# Calculate MSE using CV for the top 19 principle components, adding one c
omponent at the time.
for i in np.arange(1, 20):
    score = -1*model_selection.cross_val_score(regr,
                                               X_reduced_validation[:,:i],
                                               pd_y_validation.ravel(),
                                               cv=kf_10,
                                               scoring='neg_mean_squared_e
rror').mean()
    mse.append(score)
```

In [54]:

```
plt.plot(np.array(mse), '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.xlim(xmin=-1)
plt.xticks(np.arange(0, 20, step=1))
plt.show()
```

```
/Users/chih-hsuankao/.pyenv/versions/anaconda3-2019.03/lib/py
thon3.7/site-packages/matplotlib/axes/_base.py:3215: Matplotl
ibDeprecationWarning:
The `xmin` argument was deprecated in Matplotlib 3.0 and will
be removed in 3.2. Use `left` instead.
  alternative='`left`', obj_type='argument')
```



From here, we found that the lowest cross-validation error occurs when num_components=4 are used.

In [55]:

```
# Train pcr model on training data
pcr = linear_model.LinearRegression()
X_reduced_train = pca.fit_transform(scale(pd_X_train))
pcr.fit(X_reduced_train[:,:4], pd_y_train)

X_reduced_test = pca.fit_transform(scale(pd_X_test))[:,:4]
X_reduced_validation = pca.fit_transform(scale(pd_X_validation))[:,:4]

# print(X_reduced_train.shape)
```

```
# print(X_reduced_test.shape)
# print(X_reduced_validation.shape)
```

In [57]:

```
pcr_pred_train = pcr.predict(X_reduced_train[:,:4])
pcr_pred_validation = pcr.predict(X_reduced_validation)
pcr_pred_test = pcr.predict(X_reduced_test)
```

In [58]:

```
print("========Train========")
print('MSE: %.6f' % mean_squared_error(pd_y_train, pcr_pred_train))
print('R_squared: %.6f' % r2_score(pd_y_train, pcr_pred_train))
print("======Validation======")
print('MSE: %.6f' % mean_squared_error(pd_y_validation, pcr_pred_validatio
n))
print('R_squared: %.6f' % r2_score(pd_y_validation, pcr_pred_validation))
print("=========Test=========")
print('MSE: %.6f' % mean_squared_error(pd_y_test, pcr_pred_test))
print('R_squared: %.6f' % r2_score(pd_y_test, pcr_pred_test))
```

```
========Train========
MSE: 0.003555
R_squared: 0.016550
======Validation======
MSE: 0.001738
R_squared: -0.007024
=========Test=========
MSE: 0.001878
R_squared: -0.022258
```

4. Partial Least Squares

In [59]:

```
from sklearn.cross_decomposition import PLSRegression, PLSSVD
```

In [60]:

```
n = len(pd_X_validation)

# 10-fold CV, with shuffle
kf_10 = model_selection.KFold(n_splits=10, shuffle=True, random_state=1)

mse = []

for i in np.arange(1, 15):
    pls = PLSRegression(n_components=i)
    score = model_selection.cross_val_score(pls,
                                            scale(pd_X_validation),
                                            pd_y_validation,
                                            cv = kf_10,
                                            scoring='neg_mean_squared_erro
r').mean()
    mse.append(-score)

# Plot results
```

```
plt.plot(np.arange(1, 15), np.array(mse), '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.xlim(xmin=-1)
```

```
/Users/chih-hsuankao/.pyenv/versions/anaconda3-2019.03/lib/py
thon3.7/site-packages/matplotlib/axes/_base.py:3215: Matplotl
ibDeprecationWarning:
The `xmin` argument was deprecated in Matplotlib 3.0 and will
be removed in 3.2. Use `left` instead.
  alternative='`left`', obj_type='argument')
```

Out[60]:

```
(-1, 14.65)
```

From here, we found that the low cross-validation errors occur when around K=5 partial least squares dimensions are used.

In [61]:

```
pls = PLSRegression(n_components=5)
pls.fit(scale(pd_X_train), pd_y_train)

print("========Train========")
print('MSE: %.6f' % mean_squared_error(pd_y_train, pls.predict(scale(pd_X_
train))))
print('R_squared: %.6f' % r2_score(pd_y_train, pls.predict(scale(pd_X_trai
n))))
print("======Validation======")
print('MSE: %.6f' % mean_squared_error(pd_y_validation, pls.predict(scale(
pd_X_validation))))
print('R_squared: %.6f' % r2_score(pd_y_validation, pls.predict(scale(pd_X
_validation))))
print("=========Test=========")
print('MSE: %.6f' % mean_squared_error(pd_y_test, pls.predict(scale(pd_X_t
est))))
print('R_squared: %.6f' % r2_score(pd_y_test, pls.predict(scale(pd_X_test
))))
```

```
========Train========
MSE: 0.003481
R_squared: 0.037093
```

```
======validation======
MSE: 0.001723
R_squared: 0.001596
=========Test=========
MSE: 0.002182
R_squared: -0.187290
```

5. Regression Tree

In [62]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor
```

In [63]:

```
model = DecisionTreeRegressor()

gs = GridSearchCV(model,
                  param_grid = {'max_depth': range(1, 10),
                                'max_leaf_nodes': range(2, 100, 4)},
                  cv = 5, #(default) 5-fold cross validation
                  n_jobs = 1,
                  scoring = 'neg_mean_squared_error')

gs.fit(X_validation, y_validation)

print(gs.best_params_,'correpond to L and K respectively')
#print(-gs.best_score_)
```

```
{'max_depth': 2, 'max_leaf_nodes': 34} correpond to L and K r
espectively
```

```
/Users/chih-hsuankao/.pyenv/versions/anaconda3-2019.03/lib/py
thon3.7/site-packages/sklearn/model_selection/_search.py:841:
DeprecationWarning: The default of the `iid` parameter will c
hange from True to False in version 0.22 and will be removed
in 0.24. This will change numeric results when test-set sizes
are unequal.
  DeprecationWarning)
```

In [64]:

```
dtr = DecisionTreeRegressor(max_depth = 2,
                            max_leaf_nodes = 34) #TBD
```

In [65]:

```
dtr.fit(X_train, y_train)
```

Out[65]:

```
DecisionTreeRegressor(criterion='mse', max_depth=2, max_featu
res=None,
           max_leaf_nodes=34, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=None, splitter='best')
```

In [66]:

```
dtr_pred_train = dtr.predict(X_train)
dtr_pred_validation = dtr.predict(X_validation)
dtr_pred_test = dtr.predict(X_test)
```

In [67]:

```
print("========Train========")
print('MSE: %.6f' % mean_squared_error(y_train, dtr_pred_train))
print('R_squared: %.6f' % r2_score(y_train, dtr_pred_train))
print("======Validation======")
print('MSE: %.6f' % mean_squared_error(y_validation, dtr_pred_validation))
print('R_squared: %.6f' % r2_score(y_validation, dtr_pred_validation))
print("=========Test=========")
print('MSE: %.6f' % mean_squared_error(y_test, dtr_pred_test))
print('R_squared: %.6f' % r2_score(y_test, dtr_pred_test))
```

```
========Train========
MSE: 0.002774
R_squared: 0.232629
======Validation======
MSE: 0.001756
R_squared: -0.017647
=========Test=========
MSE: 0.003811
R_squared: -1.074157
```

6. Boosted Regression Tree

In [68]:

```
from sklearn.ensemble import GradientBoostingRegressor
```

In [69]:

```
model = GradientBoostingRegressor()

gs = GridSearchCV(model,
                  param_grid = {'max_depth':range(1, 15, 1),
                                'learning_rate': [0.0001, 0.001, 0.01, 0.1
, 0.2],
                                'n_estimators': range(1, 150, 2)},
                  n_jobs = 4, # run in parallel, adjust this with #CPU/#GP
U
                  scoring = 'neg_mean_squared_error')

gs.fit(X_validation, y_validation)

print(gs.best_params_,'correpond to v, L and B respectively')
#print(-gs.best_score_)
```

```
/Users/chih-hsuankao/.pyenv/versions/anaconda3-2019.03/lib/py
thon3.7/site-packages/sklearn/model_selection/_split.py:2053:
FutureWarning: You should specify a value for 'cv' instead of
relying on the default value. The default value will change f
rom 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)
```

```
{'learning_rate': 0.001, 'max_depth': 7, 'n_estimators': 1} c
orrepond to v, L and B respectively
```

In [70]:

```
brt = GradientBoostingRegressor(max_depth = 7,
                                learning_rate = 0.001,
                                n_estimators = 1) #TBD
brt.fit(X_train, y_train)
```

Out[70]:

```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_ms
e', init=None,
             learning_rate=0.001, loss='ls', max_depth=7,
             max_features=None, max_leaf_nodes=None,
             min_impurity_decrease=0.0, min_impurity_split=No
ne,
             min_samples_leaf=1, min_samples_split=2,
             min_weight_fraction_leaf=0.0, n_estimators=1,
             n_iter_no_change=None, presort='auto', random_st
ate=None,
             subsample=1.0, tol=0.0001, validation_fraction=
0.1, verbose=0,
             warm_start=False)
```

In [71]:

```
brt_pred_train = brt.predict(X_train)
brt_pred_validation = brt.predict(X_validation)
brt_pred_test = brt.predict(X_test)
```

In [72]:

```
print("========Train========")
print('MSE: %.6f' % mean_squared_error(y_train, brt_pred_train))
print('R_squared: %.6f' % r2_score(y_train, brt_pred_train))
print("======Validation======")
print('MSE: %.6f' % mean_squared_error(y_validation, brt_pred_validation))
print('R_squared: %.6f' % r2_score(y_validation, brt_pred_validation))
print("=========Test=========")
print('MSE: %.6f' % mean_squared_error(y_test, brt_pred_test))
print('R_squared: %.6f' % r2_score(y_test, brt_pred_test))
```

```
========Train========
MSE: 0.003612
R_squared: 0.000922
======Validation======
MSE: 0.001758
R_squared: -0.019103
=========Test=========
MSE: 0.001838
R_squared: -0.000265
```

7. Random Forests

In [73]:

```python
from sklearn.ensemble import RandomForestRegressor
```

In [74]:

```python
model = RandomForestRegressor()

gs = GridSearchCV(model,
                  param_grid = {'max_depth':range(1, 15, 1),
                                'n_estimators': range(1, 150, 1)},
                  n_jobs = 4, # run in parallel, adjust this with #CPU/#GP
U
                  scoring = 'neg_mean_squared_error')

gs.fit(X_validation, y_validation)

print(gs.best_params_,'correpond to L and B respectively')
#print(-gs.best_score_)
```

```
/Users/chih-hsuankao/.pyenv/versions/anaconda3-2019.03/lib/py
thon3.7/site-packages/sklearn/model_selection/_split.py:2053:
FutureWarning: You should specify a value for 'cv' instead of
relying on the default value. The default value will change f
rom 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)
```

```
{'max_depth': 5, 'n_estimators': 3} correpond to L and B resp
ectively
```

In [75]:

```python
rfr = RandomForestRegressor(max_depth = 5,
                            n_estimators = 3) #TBD
rfr.fit(X_train, y_train)
```

Out[75]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_de
pth=5,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=Non
e,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=3, n_jo
bs=None,
           oob_score=False, random_state=None, verbose=0, war
m_start=False)
```

In [76]:

```python
rfr_pred_train = rfr.predict(X_train)
rfr_pred_validation = rfr.predict(X_validation)
rfr_pred_test = rfr.predict(X_test)
```

In [77]:

```python
print("========Train========")
print('MSE: %.6f' % mean_squared_error(y_train, rfr_pred_train))
print('R_squared: %.6f' % r2_score(y_train, rfr_pred_train))
print("======Validation======")
print('MSE: %.6f' % mean_squared_error(y_validation, rfr_pred_validation))
print('R_squared: %.6f' % r2_score(y_validation, rfr_pred_validation))
```

```
print("=========Test=========")
print('MSE: %.6f' % mean_squared_error(y_test, rfr_pred_test))
print('R_squared: %.6f' % r2_score(y_test, rfr_pred_test))
```

```
========Train========
MSE: 0.002537
R_squared: 0.298267
======Validation======
MSE: 0.002214
R_squared: -0.283249
=========Test=========

MSE: 0.004861
R_squared: -1.645619
```

In [ ]:

```
```