Introduction to Machine Learning

Homework 2

Author: Xinyu Liu

In [1]:

```
import pandas as pd
import numpy as np
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn import metrics
import statsmodels.api as sm
from matplotlib import pyplot as plt
from sklearn.ensemble import GradientBoostingRegressor, BaggingRegressor, RandomForestRegressor
from sklearn.linear_model import LinearRegression
```

In [2]:

```
bidenraw=pd.read_csv('nes2008.csv')
bidenraw.describe()
```

Out[2]:

	biden	female	age	educ	dem	rep
count	1807.000000	1807.000000	1807.000000	1807.000000	1807.000000	1807.000000
mean	62.163807	0.552850	47.535141	13.360266	0.431655	0.205313
std	23.462034	0.497337	16.887444	2.440257	0.495444	0.404042
min	0.000000	0.000000	18.000000	0.000000	0.000000	0.000000
25%	50.000000	0.000000	34.000000	12.000000	0.000000	0.000000
50%	60.000000	1.000000	47.000000	13.000000	0.000000	0.000000
75%	85.000000	1.000000	59.500000	16.000000	1.000000	0.000000
max	100.000000	1.000000	93.000000	17.000000	1.000000	1.000000

In [3]:

```
bidenraw.head()
```

Out[3]:

	biden	female	age	educ	dem	rep
0	90	0	19	12	1	0
1	70	1	51	14	1	0
2	60	0	27	14	0	0
3	50	1	43	14	1	0
4	60	1	38	14	0	1

Decision Trees

1. Set up the data and store some things for later use:
    - Set seed
    - Load the data

- Store the total number of features minus the biden feelings in object p
- Set λ (shrinkage/learning rate) range from 0.0001 to 0.04, by 0.001

In [4]:

```
X = bidenraw.drop(columns=['biden'])
y = bidenraw[['biden']]

alphatrainmse = []
alphatestmse = []
for alpha in np.arange(0.0001,0.04,0.001):
##########################################################################################
##
# 3. Write a loop to perform boosting on the training set with 1,000 trees for the pre-defined range of
values of the shrinkage parameter
##########################################################################################
##
    params = {'n_estimators': 1000, 'max_depth': 6, 'min_samples_split': 2,
          'learning_rate': alpha, 'loss': 'ls'}
    clf = GradientBoostingRegressor(**params)
##########################################################################################
##
# 2. Create a training set consisting of 75% of the observations, and a test set with all remaining obs
.
##########################################################################################
##
    X_train, X_test, y_train, y_test = train_test_split(X, np.ravel(y), random_state=0, test_size=0.25)
    result = clf.fit(X_train, y_train)
    y_predtrain = result.predict(X_train)
    y_pred = result.predict(X_test)
    alphatrainmse.append(metrics.mean_squared_error(y_train,y_predtrain))
    alphatestmse.append(metrics.mean_squared_error(y_test,y_pred))
```

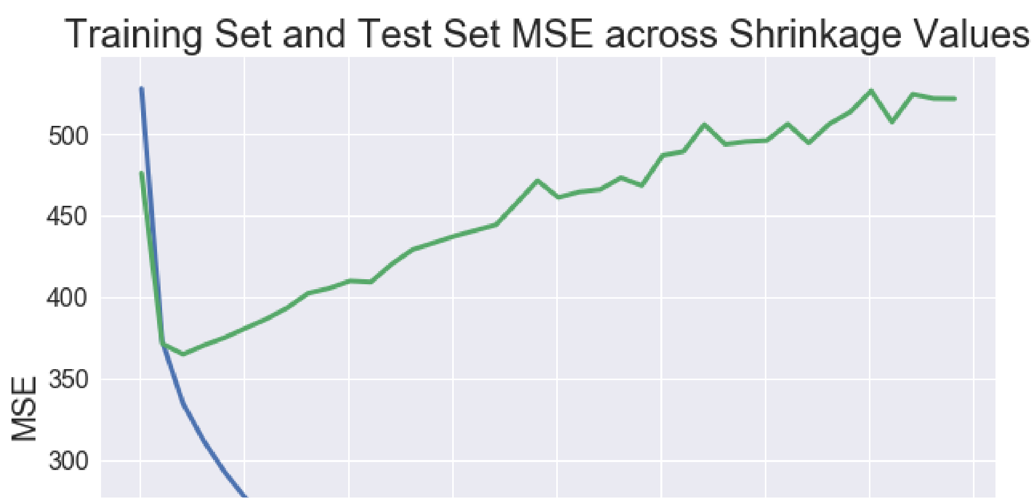
In [5]:

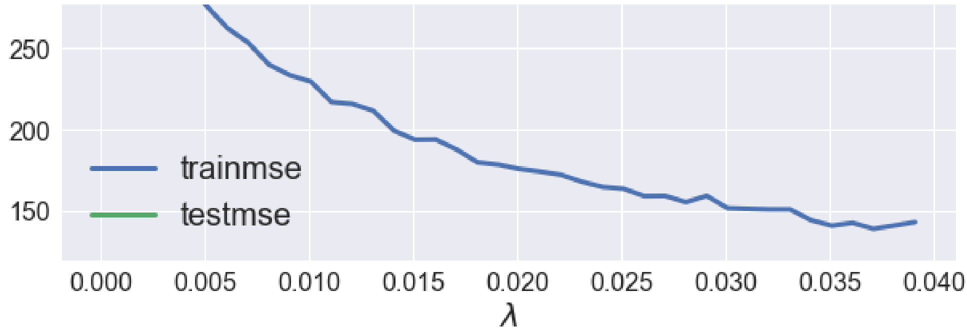
```
##########################################################################################
##
# 3. Plot the training set and test set MSE across shrinkage values
##########################################################################################
##

plt.style.use('seaborn')
f, ax = plt.subplots(figsize = (10,8))
plt.title('Training Set and Test Set MSE across Shrinkage Values', fontsize=24)
plt.plot(np.arange(0.0001,0.04,0.001), alphatrainmse, linewidth=3, label='trainmse')
plt.plot(np.arange(0.0001,0.04,0.001), alphatestmse, linewidth=3, label='testmse')
plt.xlabel(r"$\lambda$", fontsize=20)
plt.ylabel("MSE", fontsize=20)
ax.tick_params(axis='both', which='major', labelsize=16)
ax.legend(ncol = 1,fontsize=20)
```

Out[5]:

<matplotlib.legend.Legend at 0x2005fee3bc8>





In [6]:

```
###################################################################################################
##
# 4. Update the boosting procedure by setting λ equal to 0.01
###################################################################################################
##
params = {'n_estimators': 1000, 'max_depth': 6, 'min_samples_split': 2,
      'learning_rate': 0.01, 'loss': 'ls'}
clf = GradientBoostingRegressor(**params)
result = clf.fit(X_train, y_train)
y_pred = result.predict(X_test)
print(r"The test MSE for lambda=0.01 is {}".format(metrics.mean_squared_error(y_test,y_pred)))
```

The test MSE for lambda=0.01 is 414.0668064143861

In [7]:

```
###################################################################################################
##
# 5. Now apply bagging to the training set. What is the test set MSE for this approach?
###################################################################################################
##
bagging = BaggingRegressor(n_estimators=1000, random_state=0)
result = bagging.fit(X_train, y_train)
y_pred = result.predict(X_test)
print(r"The test MSE for bagging is {}".format(metrics.mean_squared_error(y_test,y_pred)))
```

The test MSE for bagging is 452.4523524693286

In [8]:

```
###################################################################################################
##
# 6.  Now apply random forest to the training set. What is the test set MSE for this approach?
###################################################################################################
##
rdforest = RandomForestRegressor(n_estimators=1000, random_state=0)
result = rdforest.fit(X_train, y_train)
y_pred = result.predict(X_test)
print(r"The test MSE for random forest is {}".format(metrics.mean_squared_error(y_test,y_pred)))
```

The test MSE for random forest is 452.0234364926416

In [9]:

```
###################################################################################################
##
# 7.  Now apply linear regression to the training set. What is the test set MSE for this approach?
###################################################################################################
##
linear = LinearRegression()
result = linear.fit(X_train, y_train)
y_pred = result.predict(X_test)
print(r"The test MSE for linear regression is {}".format(metrics.mean_squared_error(y_test,y_pred)))
```

```
The test MSE for linear regression is 354.2515074673563
```

8. Compare test errors across all fits. Discuss which approach generally fits best and how you concluded this.

Across all fits, MSE has the lowest value with Linear Regression. From bootstrapping it is clear that lambda of ~0.0025 has the lowest test MSE. Admittedly this conclusion is conditonal on the split situation.

Support Vector Machines

In [2]:

```
ojdata=pd.read_csv("oj.csv", index_col=0)
ojdata['response']=np.where(ojdata.Purchase=='MM',1,0)
ojdata.drop(columns=['Purchase'], inplace=True)
ojdata['Store']=np.where(ojdata.Store7=='Yes',1,0)
ojdata.drop(columns=['Store7'], inplace=True)
```

In [3]:

```
ojdata.head()
```

Out[3]:

	WeekofPurchase	StoreID	PriceCH	PriceMM	DiscCH	DiscMM	SpecialCH	SpecialMM	LoyalCH	SalePriceMM	SalePriceCH	PriceD
1	237	1	1.75	1.99	0.00	0.0	0	0	0.500000	1.99	1.75	0.
2	239	1	1.75	1.99	0.00	0.3	0	1	0.600000	1.69	1.75	-0.
3	245	1	1.86	2.09	0.17	0.0	0	0	0.680000	2.09	1.69	0.
4	227	1	1.69	1.69	0.00	0.0	0	0	0.400000	1.69	1.69	0.
5	228	7	1.69	1.69	0.00	0.0	0	0	0.956535	1.69	1.69	0.

In [4]:

```
##########################################################################################
##
# 1. Create a training set with a random sample of size 800, and a test set containing the remaining ob
servations
##########################################################################################
##
# scikit-learn bootstrap
from sklearn.utils import resample
# data sample
# prepare bootstrap sample
boot = resample(ojdata, replace=False, n_samples=800, random_state=0)
# out of bag observations
oob=pd.concat([ojdata, boot]).drop_duplicates(keep=False)
```

In [7]:

```
##########################################################################################
##
# 2. Fit a support vector classifier to the training data with cost = 0.01,
# with Purchase as the response and all other features as predictors. Discuss the results.
##########################################################################################
##
from sklearn import svm
X_train = boot.drop(columns=['response'])
y_train = boot[['response']]
X_test = oob.drop(columns=['response'])
y_test = oob[['response']]
clf = svm.SVC(C=100, kernel='linear')
result = clf.fit(X_train, np.ravel(y_train))
y_pred = result.predict(X_test)
```

```
y_pred = result.predict(X_test)
print(r"The accurate rate of training set for support vector classifier with cost = 0.01 is {}".format(result.score(X_train,y_train)))
print(r"The accurate rate of training set for support vector classifier with cost = 0.01 is {}".format(result.score(X_test,y_test)))
```

```
The accurate rate of training set for support vector classifier with cost = 0.01 is 0.8475
The accurate rate of training set for support vector classifier with cost = 0.01 is 0.7961538461538461
```

In [8]:

```
#####################################################################################################
# 3. Display the confusion matrix for the classification solution, and also report both the training and test set error rates.
#####################################################################################################
from sklearn.metrics import confusion_matrix
print(r"The confusion matrix is as follows:")
pd.DataFrame(confusion_matrix(y_test, y_pred))
```

```
The confusion matrix is as follows:
```

Out[8]:

	0	1
0	130	29
1	24	77

In [31]:

```
#####################################################################################################
# 4. Find an optimal cost in the range of 0.01 to 1000 (specific range values can vary; there is no set vector of range values you must use).
#####################################################################################################
from sklearn.model_selection import GridSearchCV
parameters = {'C':[0.001, 0.01, 0.1, 0.5, 1, 5, 10]}
svc = svm.SVC(kernel='linear')
clf = GridSearchCV(svc, parameters)
result = clf.fit(X_train, np.ravel(y_train))
y_pred = result.predict(X_test)
sorted(clf.cv_results_.keys())
print(r"The confusion matrix is as follows:")
pd.DataFrame(confusion_matrix(y_test, y_pred))
```

```
The confusion matrix is as follows:
```

Out[31]:

	0	1
0	130	29
1	23	78

In [32]:

```
print(r"The optimization result is as follows:")
optimization = pd.DataFrame(clf.cv_results_)
optimization.sort_values('rank_test_score', inplace=True)
optimization[['rank_test_score','param_C', 'mean_test_score']].reset_index(drop=True)
```

```
The optimization result is as follows:
```

Out[32]:

	rank_test_score	param_C	mean_test_score
0	1	1	0.84500
1	1	10	0.84500
2	3	5	0.84125
3	4	0.5	0.83875
4	5	0.1	0.82750
5	6	0.01	0.71750
6	7	0.001	0.61125

In [9]:

```
for cost in [0.001, 0.01, 0.1, 1, 10]:
    clf = svm.SVC(C=cost, kernel='linear')
    result = clf.fit(X_train, np.ravel(y_train))
    y_pred = result.predict(X_test)
    print(r"The accurate rate of training set for support vector classifier with cost = {} is {}".format
(cost, metrics.accuracy_score(y_test, y_pred)))
```

```
The accurate rate of training set for support vector classifier with cost = 0.001 is 0.6115384615384616
The accurate rate of training set for support vector classifier with cost = 0.01 is 0.7384615384615385
The accurate rate of training set for support vector classifier with cost = 0.1 is 0.7846153846153846
The accurate rate of training set for support vector classifier with cost = 1 is 0.8
The accurate rate of training set for support vector classifier with cost = 10 is 0.7961538461538461
```

In [33]:

```
####################################################################################################
##
# 5. Compute the optimal training and test error rates using this new value for cost. Display the confu
sion matrix for the classification solution,
# and also report both the training and test set error rates. How do the error rates compare?
####################################################################################################
##
print(r"The optimal confusion matrix for test set is as follows:")
pd.DataFrame(confusion_matrix(y_test, y_pred))
```

```
The optimal confusion matrix for test set is as follows:
```

Out[33]:

	0	1
0	130	29
1	23	78

In [36]:

```
print(r"The optimal confusion matrix for train set is as follows:")
pd.DataFrame(confusion_matrix(y_train, result.predict(X_train)))
```

```
The optimal confusion matrix for test set is as follows:
```

Out[36]:

	0	1
0	441	48
1	73	238

In [38]:

```
print(r"The accurate rate of training set The optimal confusion matrix with cost = 10 is {}".format(met
rics.accuracy_score(y_train, result.predict(X_train))))
print(r"The accurate rate of test set The optimal confusion matrix with cost = 10 is {}".format(metrics
.accuracy_score(y_test, y_pred)))
```

```
The accurate rate of training set The optimal confusion matrix with cost = 10 is 0.84875
The accurate rate of test set The optimal confusion matrix with cost = 10 is 0.8
```

In [40]:

```
# Convert to pdf
# https://stackoverflow.com/questions/15998491/how-to-convert-ipython-notebooks-to-pdf-and-html
```