Decision Trees

In [1]:

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

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 [2]:

```
biden = pd.read_csv('nes2008.csv')
biden.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1807 entries, 0 to 1806
Data columns (total 6 columns):
biden     1807 non-null int64
female    1807 non-null int64
age       1807 non-null int64
educ      1807 non-null int64
dem       1807 non-null int64
rep       1807 non-null int64
dtypes: int64(6)
memory usage: 84.8 KB
```

2. Create a training set consisting of 75% of the observations, and a test set with all remaining obs.

3. Create empty objects to store training and testing MSE, and then 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, λ. Then, plot the training set and test set MSE across shrinkage values.

In [5]:

```
X = biden.drop(columns = ['biden'])
y = biden[['biden']]
y = np.ravel(y)
X.head()
```

Out[5]:

	female	age	educ	dem	rep
0	0	19	12	1	0
1	1	51	14	1	0
2	0	27	14	0	0
3	1	43	14	1	0
4	1	38	14	0	1

In [7]:

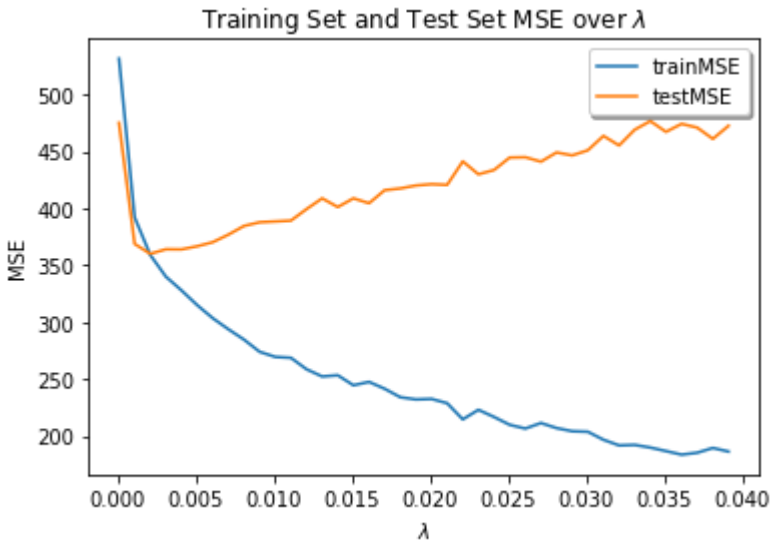
```
X = biden.drop(columns = ['biden'])
y = np.ravel(biden[['biden']])

trainMSE = []
testMSE = []

for lambda_ in np.arange(0.0001, 0.04, 0.001):
    params = {'loss': 'ls', 'n_estimators': 1000, 'max_depth': 5, 'min_samples_split
    clf = GradientBoostingRegressor(**params)
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_s
    result = clf.fit(X_train, y_train)
    y_predTrain = result.predict(X_train)
    y_predTest = result.predict(X_test)
    trainMSE.append(metrics.mean_squared_error(y_train, y_predTrain))
    testMSE.append(metrics.mean_squared_error(y_test, y_predTest))
```

In [21]:

```
plt.plot(np.arange(0.0001, 0.04, 0.001), trainMSE)
plt.plot(np.arange(0.0001, 0.04, 0.001), testMSE)
plt.legend(('trainMSE', 'testMSE'), loc='upper right', shadow=True)
plt.xlabel("$\lambda$")
plt.ylabel("MSE")
plt.title('Training Set and Test Set MSE over $\lambda$')
plt.show()
```



4. The test MSE values are insensitive to some precise value of λ as long as its small enough. Update the boosting procedure by setting λ equal to 0.01 (but still over 1000 trees). Report the test MSE and discuss the results. How do they compare?

In [22]:

```
params = {'loss': 'ls', 'n_estimators': 1000, 'max_depth': 5, 'min_samples_split': 2
clf = GradientBoostingRegressor(**params)
result = clf.fit(X_train, y_train)
y_predTest = result.predict(X_test)
print(metrics.mean_squared_error(y_test,y_predTest))
```

```
386.46145299359534
```

The test MSE is 386.46 when λ equals to 0.01, which is pretty small comparing to when λ is a little bit larger. Based on the plot, the smallest MSE can be reached when λ is around 0.004. Therefore, in that range, the test MSE is pretty insensitive over minor changing in λ.

5. Now apply bagging to the training set. What is the test set MSE for this approach?

In [26]:

```
bagging = BaggingRegressor()
result = bagging.fit(X_train, y_train)
y_pred = result.predict(X_test)
print("The test MSE for bagging is", metrics.mean_squared_error(y_test,y_pred))
```

```
The test MSE for bagging is 452.24445978834945
```

6. Now apply random forest to the training set. What is the test set MSE for this approach?

In [27]:

```
rf = RandomForestRegressor()
result = rf.fit(X_train, y_train)
y_pred = result.predict(X_test)
print("The test MSE for random forest is", metrics.mean_squared_error(y_test,y_pred)
```

```
The test MSE for random forest is 464.34629543093723

/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:
245: FutureWarning: The default value of n_estimators will change from
10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

7. Now apply linear regression to the training set. What is the test set MSE for this approach?

In [28]:

```
lr = LinearRegression()
result = lr.fit(X_train, y_train)
y_pred = result.predict(X_test)
print("The test MSE for linear regression is", (metrics.mean_squared_error(y_test,y_
```

```
The test MSE for linear regression is 354.2515074673564
```

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

Based on the test MSE we got from all models, linear regression has the smallest MSE over all the other methods. Therefore, it's fair to say that linear regression is the best model in this case.

Support Vector Machines

1. Create a training set with a random sample of size 800, and a test set containing the remaining observations.

In [119]:

```
oj = pd.read_csv("OJdata.csv", index_col=0).reset_index()
oj.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1070 entries, 0 to 1069
Data columns (total 18 columns):
Purchase          1070 non-null object
WeekofPurchase    1070 non-null int64
StoreID           1070 non-null int64
PriceCH           1070 non-null float64
PriceMM           1070 non-null float64
DiscCH            1070 non-null float64
DiscMM            1070 non-null float64
SpecialCH         1070 non-null int64
SpecialMM         1070 non-null int64
LoyalCH           1070 non-null float64
SalePriceMM       1070 non-null float64
SalePriceCH       1070 non-null float64
PriceDiff         1070 non-null float64
Store7            1070 non-null object
PctDiscMM         1070 non-null float64
PctDiscCH         1070 non-null float64
ListPriceDiff     1070 non-null float64
STORE             1070 non-null int64
dtypes: float64(11), int64(5), object(2)
memory usage: 150.6+ KB
```

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.

3. Display the confusion matrix for the classification solution, and also report both the training and test set error rates.

In [120]:

```
oj['y'] = np.where(oj.Purchase=='MM', 1, 0)
oj['Store'] = np.where(oj.Store7 == 'Yes', 1, 0)
oj.drop(columns=['Store7', 'Purchase'], inplace=True)

train = oj.sample(frac = 800/1070, replace=False)
test = pd.concat([oj, train]).drop_duplicates(keep=False)

X_train = train.drop(columns=['y'])
y_train = train[['y']]
X_test = test.drop(columns=['y'])
y_test = test[['y']]
clf = svm.SVC(C=100)
result = clf.fit(X_train, y_train)
y_pred = result.predict(X_test)
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.p
y:724: DataConversionWarning: A column-vector y was passed when a 1d a
rray was expected. Please change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:193: Fu
tureWarning: The default value of gamma will change from 'auto' to 'sc
ale' in version 0.22 to account better for unscaled features. Set gamm
a explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

In [121]:

```
print("The error rate of training set is ", 1 - result.score(X_train,y_train))
print("The error rate of test set is ", 1 - result.score(X_test,y_test))
print('Total number of support vectors are', len(clf.support_vectors_))
print('The confusion matrix is:')
pd.DataFrame(confusion_matrix(y_test, y_pred))
```

```
The error rate of training set is  0.12624999999999997
The error rate of test set is  0.16091954022988508
Total number of support vectors are 353
The confusion matrix is:
```

Out[121]:

	0	1
0	153	26
1	16	66

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).

In [ ]:

```
parameters = {'C':[0.001, 0.1, 1, 10, 100]}
svc = svm.SVC(kernel='linear')
result = GridSearchCV(svc, parameters).fit(X_train, y_train)
y_pred = result.predict(X_test)
sorted(clf.cv_results_.keys())
```

In [113]:

```
print("Confusion Matrix:")
pd.DataFrame(confusion_matrix(y_test, y_pred))
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_sp
lit.py:1978: FutureWarning: The default value of cv will change from 3
to 5 in version 0.22. Specify it explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)

Confusion Matrix:
```

Out[113]:

	0	1
0	135	25
1	27	64

In [114]:

```
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
```

Out[114]:

	rank_test_score	param_C	mean_test_score
0	1	1	0.83500
1	2	10	0.83125
2	2	100	0.83125
3	4	0.1	0.82125
4	5	0.001	0.60500

Based on the optimization table, we can tell the best result is when cost equals to 1 (The parameter C is the inverse number of cost).

5. Compute the optimal training and test error rates using this new value for cost. Display the confusion matrix for the classification solution, and also report both the training and test set error rates. How do the error rates compare? Discuss the results in substantive terms (e.g., how well did your optimally tuned classifer perform? etc.)

In [122]:

```
clf = svm.SVC(C=1, kernel='linear')
result = clf.fit(X_train, y_train)
y_pred = result.predict(X_test)
print("Confusion matrix with optimal cost:")
pd.DataFrame(confusion_matrix(y_test, y_pred))
```

```
Confusion matrix with optimal cost:

/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.p
y:724: DataConversionWarning: A column-vector y was passed when a 1d a
rray was expected. Please change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[122]:

	0	1
0	155	24
1	14	68

In [123]:

```
print("The error rate of training set with cost = 1 is", 1 - metrics.accuracy_score(
print("The error rate of test set with cost = 1 is", 1 - metrics.accuracy_score(y_te
```

```
The error rate of training set with cost = 1 is 0.17374999999999996
The error rate of test set with cost = 1 is 0.14559386973180077
```

Based on the results, the error rate of training set and test set is 0.174 and 0.146 when cost = 1, comparing with 0.126 and 0.161 when cost = 0.01. After changing to the optimal cost, the error rate of training set increases and the error rate of test set decreases. Therefore, it is pretty hard to say that how well we tune the classifier, which is dependent on how we evaluate the result.

In [ ]:

```

```