ISYE6740-HW6

pkubsad

March 2021

1 **House price dataset.**

(a) (10 points) Fit the Ridge regression model to predict Price from all variable. You can use one-hot keying to expand the categorical variable Status. Use 5-fold cross validation to select the regularizer optimal parameter, and show the CV curve. Report the fitted model (i.e., the parameters), and the sum-of-squares residuals. You can use any package.

Answer: I have used sklearn's linear model to perform all the analysis for Q1. After using RidgeCV method withrange from range of alphas: [**0.005 - 5000000000.0**], the optimal alpha value determined by the library $\alpha = 0.10772173450159389$.

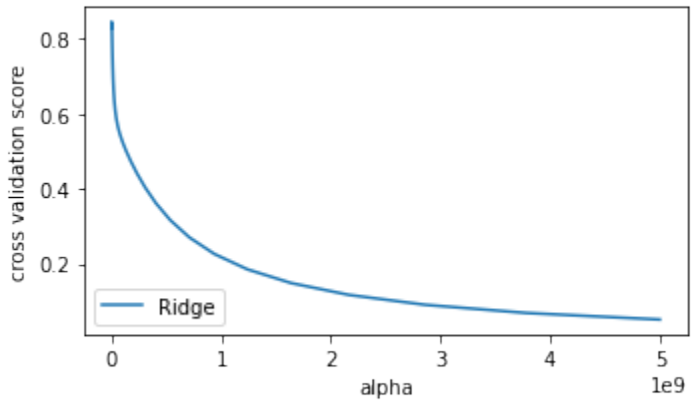


Figure 1: CV Curve

	0	1
0	50168.637286	Bathrooms
1	3632.595792	Bedrooms
2	-12274.491788	Foreclosure
3	1832.887507	Price/SQ.Ft
4	57635.404283	Regular
5	-19686.818918	Short Sale
6	207.593149	Size

Figure 2: Fitted Model

```
range of alphas: 0.005 5000000000.0
optimal alpha:  0.10772173450159389
mse for ridge regression:  21846054179.982845
r2 for ridge regression:  0.8204506806336742

array([[ 50168.63728641,   3632.59579198, -12274.49178788,
         1832.88750652,  57635.40428256, -19686.81891839,
          207.59314895]])
```

Figure 3: Residuals - Ridge

(b) (10 points) Use lasso to select variables. Use 5-fold cross validation to select the regularizer optimal parameter, and show the CV curve. Report the fitted model (i.e., the parameters selected and their coefficient). Show the Lasso solution path. You can use any package for this.

Answer: In the same lines as above, used sklearn's linear model library to perform this analysis. The output from LassoCV gives us the optimal regularizer parameter, $\alpha = 100.8230376891931$. Below are the illustrations of CV Curve, fitted model and solution path:

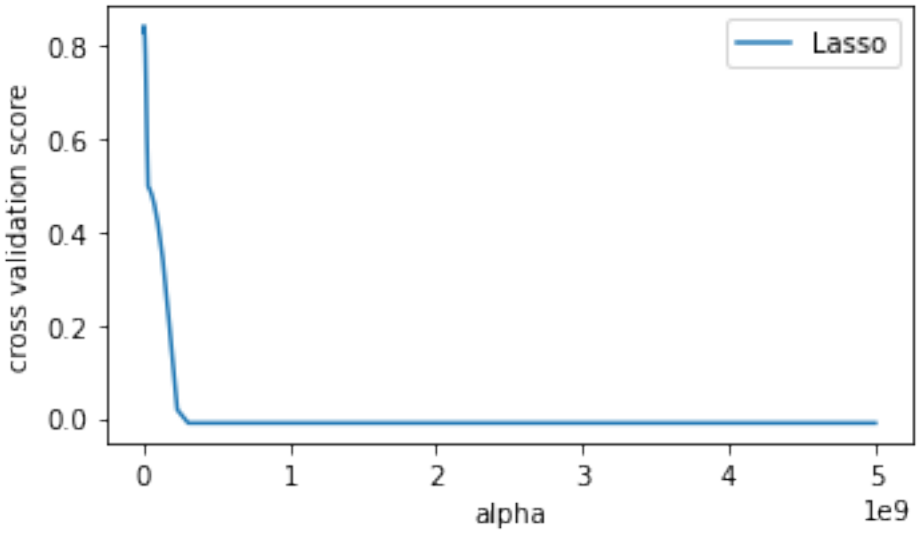


Figure 4: CV Curve - LASSO

	0	1
0	14743.630869	Bathrooms
1	-0.000000	Bedrooms
2	0.000000	Foreclosure
3	2018.141742	Price/SQ.Ft
4	42604.790481	Regular
5	-1962.037890	Short Sale
6	252.467580	Size

Figure 5: Lasso Model

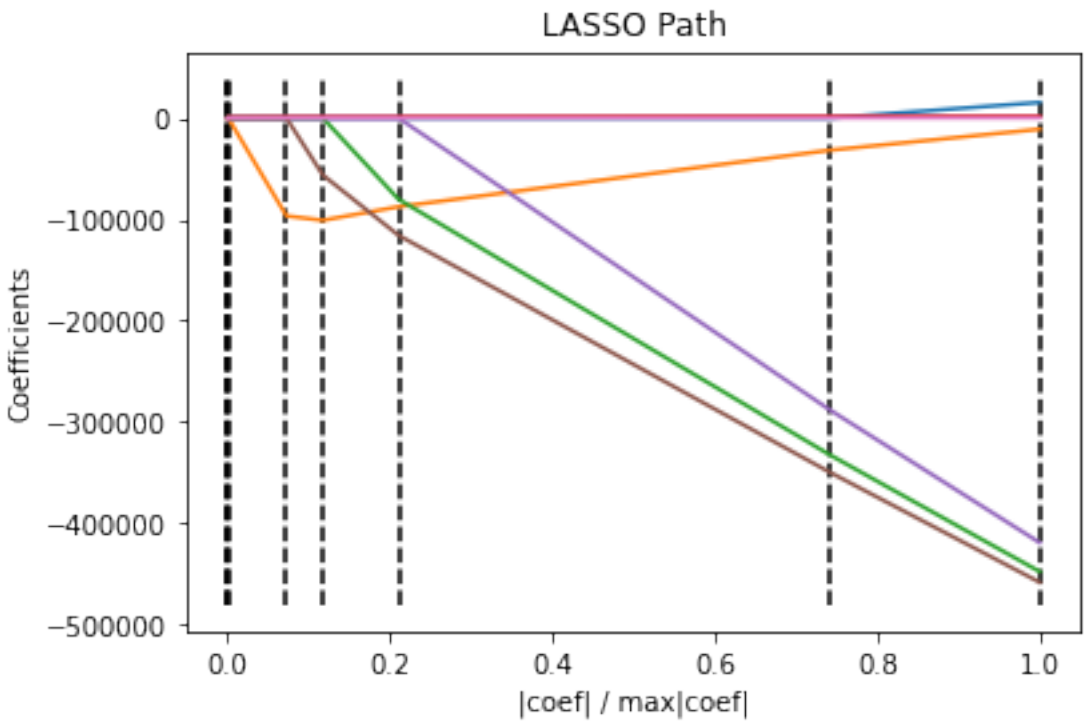


Figure 6: Lasso Path

(c) (5 points) Use elastic net to select variables. Report the fitted model (i.e., the parameters selected and their coefficient). Use 5-fold cross validation to select the regularizer optimal parameter. You can use any package for this.

Answer. Same lines as above, regularizer optimal parameter , $\alpha = 466.30167344161$

	0	1
0	15.764964	Bathrooms
1	-3.574742	Bedrooms
2	-1.649758	Foreclosure
3	2045.073923	Price/SQ.Ft
4	20.706935	Regular
5	-18.056825	Short Sale
6	267.275949	Size

Figure 7: Elastic Net Model

2 **AdaBoost.** (25 points)

(a) (15 points) For each iteration $t = 1, 2, 3$, compute ϵ_t, α_t, Z_t, D_t by hand (i.e., show the calculation steps) and draw the decision stumps on the figure (you can draw this by hand).

Answer. Dataset: $X, Y, Z = \{-1, 0, +1\}, \{-0.5, 0.5, +1\}, \{0, 1, -1\}, \{0.5, 1, -1\}, \{1, 0, +1\}, \{1, -1, +1\}, \{0, -1, -1$
Iteration 1: Classifier: If $X < -0.25$ then $+1$ and $X > -0.25$ then -1

Result of Classification: = [1,1,-1,-1,-1,-1,-1,-1]

$D_1 = 1 * 1/m = 1 * 1/8 = [0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125]$

$\epsilon_t = \frac{misclassifiedpoints}{totalpoints} = \frac{2}{8} = 0.25$

$\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t}) = \frac{1}{2} ln(\frac{1-0.25}{0.25}) = 0.54930614433$

$Z_t = \sum_i^m D_t(i) e^{-\alpha_t y^i h_t(x^i)} = 6 * 0.125 * e^{-\alpha_t} + 2 * 0.125 e^{\alpha_t} = 0.8660225$

$D_2 = \frac{D_1}{Z_1} e^{-\alpha_t y^i h_1(x^i)} = [0.08333333, 0.08333333, 0.08333333, 0.08333333, 0.25, 0.25, 0.08333333, 0.08333333]$

Calculating in the same lines:
Iteration 2: Classifier: If $X > +0.75$ then $+1$ and $X < 0.75$ then -1
Iteration 3: Classifier: If $Y > -0.25$ then $+1$ and $Y < -0.25$ then -1

	ϵ_t	α_t	Z_t	$D_t(1)$	$D_t(2)$	$D_t(3)$	$D_t(4)$	$D_t(5)$	$D_t(6)$	$D_t(7)$	$D_t(8)$
1	0.25	0.549306	0.866025	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
2	0.166667	0.804719	0.745356	0.083333	0.083333	0.083333	0.083333	0.25	0.25	0.083333	0.083333
3	0.1	1.098612	0.6	0.25	0.25	0.05	0.05	0.15	0.15	0.05	0.05

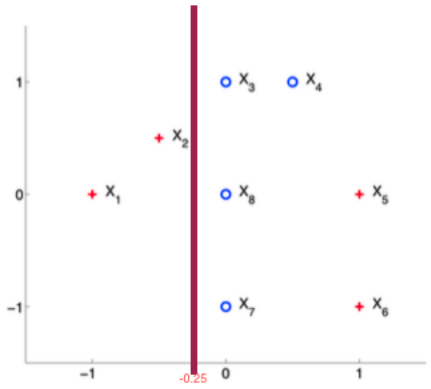


Figure 8: Iteration 1

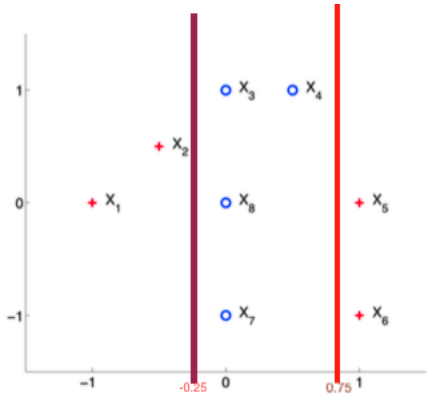


Figure 9: Iteration 2

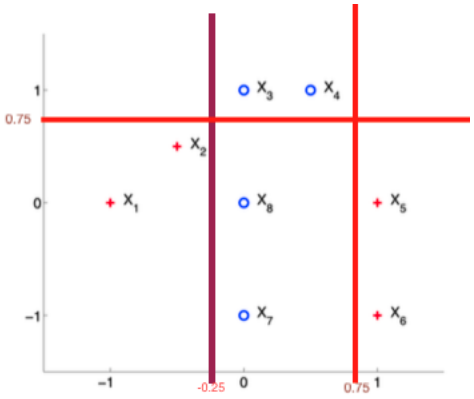


Figure 10: Iteration 3

(b) (10 points) What is the training error of this AdaBoost? Give a short explanation for why AdaBoost outperforms a single decision stump.

Answer.

The training error of AdaBoost is 0. All the points are classified correctly. Adaboost algorithm classifies at multiple iterations. At each iteration, the classification is focussed on particular part of the dataset. With wieghted focus in each iteration, this can create better classifier than other classification models.

3 **Random forest and one-class SVM for email spam classifier** (25 points)

(a) (5 points) Build a CART model and visualize the fitted classification tree.

Answer.

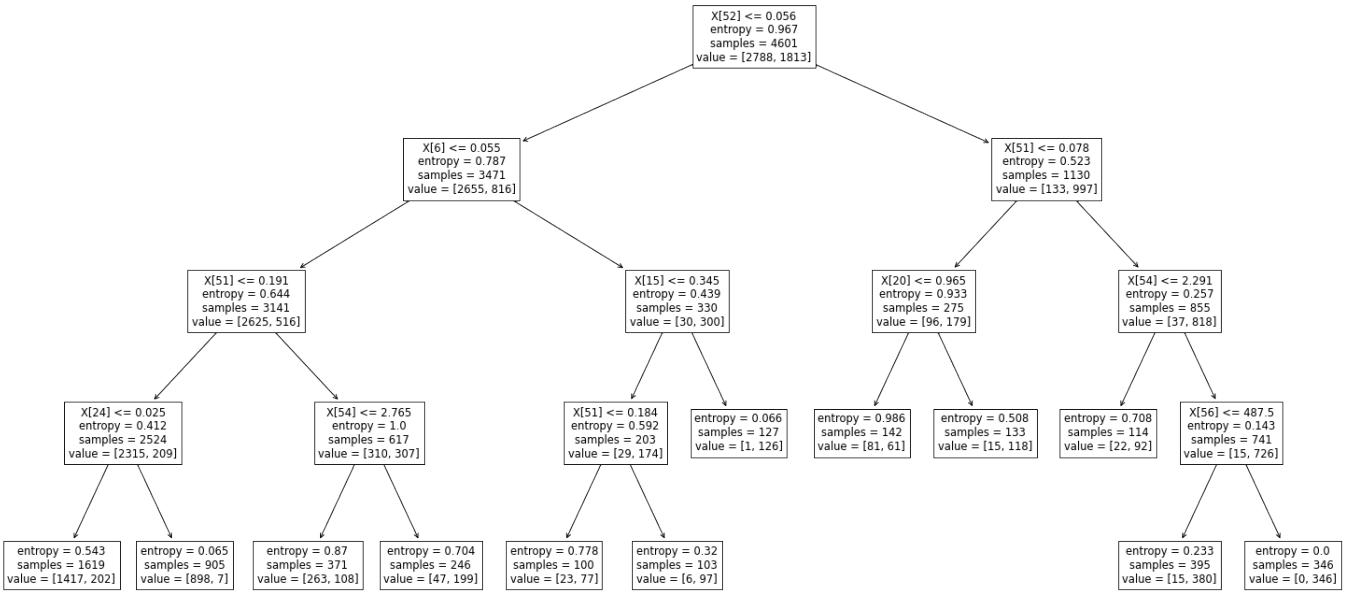


Figure 11: CART model tree classification

(b) (10 points) Now also build a random forest model. Partition the data to use the first 80% for training and the remaining 20% for testing. Compare and report the test error for your classification tree and random forest models on testing data. Plot the curve of test error (total misclassification error rate) versus the number of trees for the random forest, and plot the test error for the CART model (which should be a constant with respect to the number of trees).

Answer. I have used sklearn's Random Forest classifier and looping through nTrees =[1,1000]. For each iteration of nTree, I have calculated the misclassification rate as

$$miscalculation\_rate = \frac{FalsePositives + FalseNegatives}{TotalNumberofdatapoints}$$

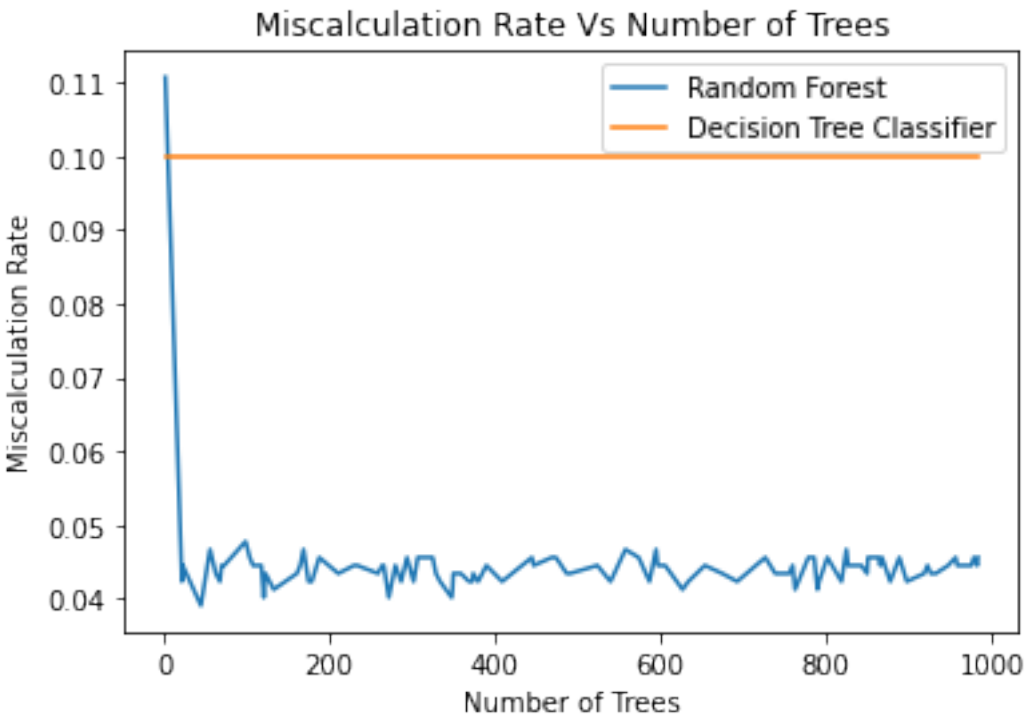


Figure 12: Miscalculation Rate vs Number of Trees

Used sklearn's GridSearchCV method to calculate the optimal Decision Tree Classifier and ran the same classification with Random Forest and Decision Tree. Decision tree is single tree classifier and hence the misclassfication rate stays the same irrespective of the number of trees used for classification. The results of the classification are as follows (Confusion matrix, miscalculation rate):

```
confusion matrix for random forest:
[[520  11]
 [ 31 359]]
miscalc rate for random forest:
0.04560260586319218
================
confusion matrix for decision tree:
[[513  18]
 [ 74 316]]
miscalc rate for decision tree:
0.0998914223669924
```

Figure 13: Miscalculation Rate of Random Forest Classifier vs Decision Tree Classifier

(c) (10 points) Now we will use a one-class SVM approach for spam filtering. Partition the data to use the first 80% for training and the remaining 20% for testing. Extract all *non-spam* emails from the training block (80% of data you have selected) to build the one-class kernel SVM using RBF kernel (you can turn the kernel bandwidth to achieve good performance). Then apply it on the 20% of data reserved for testing (thus this is a novelty detection situation), and report the total misclassification error rate on these testing data.

Answer. I have split the data using sklearn's train_test_split method to split the data 80,20. In the train set, filtered it for col 57 == 0 (Non-spam emails). Used sklearn's OneClassSVM on the filtered data set to build the model. Observations from the result:
Number of data point in training set = **3680**
Number of non-spam emails in training set=**2257**
Number of spam email in training set = **1423**
Number of data point in testing set = **921**
Number of non-spam emails in testing set=**531**
Number of spam email in testing set = **390**

Number of spam email reported by model's prediction in training set = **1581**
Number of spam email reported by model's prediction in testing set = **527**
Miscalculation rate = **0.2975**

Snapshot from programme's output:

```
(3680, 58)
(921, 58)
counts in train data:
0    2257
1    1423
Name: 57, dtype: int64
counts in test data:
0    531
1    390
Name: 57, dtype: int64
No of spams detected by model in training set:
1581
No of spams detected by model in test set:
527
No of non-spams detected by model in test set:
394
miscalculation rate:
0.2975027144408252
```

Figure 14: Programme output for one class svm

4.0 **Locally weighted linear regression and bias-variance tradeoff.** (25 points)

a (5 points) Show that the ridge regression which introduces a squared ℓ_2 norm penalty on the parameter in the maximum likelihood estimate of β can be written as follows

$$\hat{\beta}(\lambda) = \arg\min_{\beta} \left\{ (X\beta - y)^T W (X\beta - y) + \lambda \|\beta\|_2^2 \right\}$$

for property defined diagonal matrix W, matrix X and vector y.

Answer.: Given linear regression model

$$y_i = \beta^{*^T} . x_i + \epsilon_i$$

We have to determine the likelihood function $L(\beta, (x_i, y_i))$ and maximum likelihood function will be deteremined by doing partial derivative wrt β. $\frac{\partial L(\beta,(x_i,y_i))}{\partial \beta} = 0$. Recall that $y_i \tilde{} N(x_i \beta^{*^T}, \sigma^2)$, we can establish the Ridge rigression's penalty term as :

$$\hat{\beta}(\lambda) = argmin_{\beta} \{ -\frac{1}{2} \sum (y_i - x_i\beta)^2 + \lambda ||\beta||_2^2 \}$$

Reduced to matrix form, we can write $\sum (Y - X\beta)^2 = ||Y - X\beta||_2^2 = (Y - X\beta)^T (Y - X\beta)$. Replacing in the equation

$$\hat{\beta}(\lambda) = argmin_{\beta} \{ -\frac{1}{2} (Y - X\beta)^T (Y - X\beta) + \lambda ||\beta||_2^2 \}$$

Adjusting for constant and sign,as it is a derivative equal to 0

$$\hat{\beta}(\lambda) = argmin_{\beta} \{ (X\beta - Y)^T (X\beta - Y) + \lambda ||\beta||_2^2 \}$$

Introducing the wieghtage matrix (W) representing the local weights, we can rewrite the equation as

$$\hat{\beta}(\lambda) = argmin_{\beta} \{ (X\beta - Y)^T . W (X\beta - Y) + \lambda ||\beta||_2^2 \}$$

W is a nXn dimensional diagonal matrix with $W_{ii} \varepsilon [0, 1]$ representing the weight of the i^{th} observation.

b (5 points) Find the close-form solution for $\widehat{\beta}(\lambda)$ and its distribution conditioning on $\{x_i\}$.

Answer. Source: https://arxiv.org/pdf/1509.09169.pdf, page 50
In order to find the closed form solution of the $\widehat{\beta}$, we take the derivative of $\hat{\beta}(\lambda)$ with respect to β equal to zero. Then we solve for $\hat{\beta}$:

$$\frac{\partial \hat{\beta}(\lambda)}{\beta} = 0$$

$$2X^T WY - 2X^T WX\beta - 2\Delta\beta + 2\Delta\beta_0 = 0. where \Delta = \lambda I$$

This is solved by:

$$\widehat{\beta(\lambda)} = (X^T WX + \Delta)^{-1} (X^T WY)$$

$$\widehat{\beta(\lambda)} = (X^T WX + \lambda I)^{-1} (X^T WY)$$

The Expectation function is given by

$$E(\hat{\beta}) = (X^T WX + \lambda I)^{-1} (X^T WX\beta)$$

The variance of β is given by

$$Var[\hat{\beta}] = \sigma^2(X^TWX + \lambda I)^{-1}X^TW^2X(X^TWX + \lambda I)^{-1}$$

The error term is gaussian, y is guassian, $\hat{\beta}$ is also gaussian $\hat{\beta} \sim \mathcal{N}(\mathbb{E}[\hat{\beta}], Var[\hat{\beta}])$

c (5 points) Derive the bias as a function of λ and some fixed test point x.

Answer. As mentioned in the hw walk through we have to find Bias term as,

$$Bias = \mathbb{E}[\hat{\beta}(\lambda)^T x] - \beta^* x$$

$$Bias = \mathbb{E}[((X^TWX + \lambda I)^{-1}(X^TWY))^T x] - \beta^* x$$

Based on conditional probability, we can use the $E(\gamma y) = \gamma E(y) = \gamma X\beta^*$, for any given random varaible y.

$$Bias = x\mathbb{E}[((X^TWX + \lambda I)^{-1}(X^TWY))^T] - \beta^* x$$

$$Bias = x\mathbb{E}[((X^TWX + \lambda I)^{-1}(X^TWY))^T] - \beta^* x$$

$$Bias = x((X^TWX + \lambda I)^{-1}(X^TWX\beta^*))^T - \beta^* x$$

d (5 points) Derive the variance term as a function of λ.
Answer. We can calculate the variance term as follows:

$$VAR(\hat{\beta}(\lambda)^T x) = x.VAR(\hat{\beta}(\lambda)^T$$

substituting the equation from q 4.ii for $VAR[\hat{\beta}(\lambda)]$

$$= x(\sigma^2(X^TWX + \lambda I)^{-1}X^TW^2X(X^TWX + \lambda I)^{-1})^T$$

e (5 points) Now assuming the data are one-dimensional, the training dataset consists of two samples $x_1 = 1.5$ and $x_2 = 1$, and the test sample $x = 0.5$. The true parameter $\beta_0^* = 1$, $\beta_1^* = 0.5$, the noise variance is given by $\sigma_1^2 = 2$, $\sigma_2^2 = 1$. Plot the MSE (Bias square plus variance) as a function of the regularization parameter λ.

Answer.

$$MSE = Bias^2 + Variance$$

$$MSE = [x((X^TWX+\lambda I)^{-1}(X^TWX\beta^*))^T - \beta^* x]^2 + [x(\sigma^2(X^TWX+\lambda I)^{-1}X^TW^2X(X^TWX+\lambda I)^{-1})^T]$$

Q1

April 22, 2021

[5]:
```
import pandas as pd
from pandas import DataFrame
import numpy as np
from sklearn.linear_model import␣
 ↪RidgeCV,Ridge,Lasso,LassoCV,lasso_path,lars_path,ElasticNet,ElasticNetCV
from sklearn.model_selection import␣
 ↪train_test_split,validation_curve,cross_val_score
from sklearn.metrics import mean_squared_error,r2_score
import matplotlib.pyplot as plt
from sklearn.metrics import  make_scorer
from itertools import cycle
import warnings
warnings.filterwarnings('ignore')
```

[6]:
```
# source: http://www.science.smith.edu/~jcrouser/SDS293/labs/lab10-py.html
data=pd.read_csv('./data/RealEstate.csv')

alphas = 10**np.linspace(10,-2,100)*0.5
# alphas = np.logspace(-3, -1, 30)
print("range of alphas:",min(alphas),max(alphas))
data.drop(columns=["MLS","Location"],inplace=True)
dummies=pd.get_dummies(data.Status)
data=pd.concat([data,dummies],axis=1).drop(columns=["Status"])
# display(data)

X=data[data.columns.difference(['Price'])]
y=data[["Price"]]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
 ↪random_state=42)
ridgeCV=RidgeCV(alphas=alphas, normalize = True,cv=5)
ridgeCV.fit(X_train,y_train)
optimal_alpha=ridgeCV.alpha_
print("optimal alpha: ",optimal_alpha)
ridge= Ridge(alpha=optimal_alpha,normalize=True)
model=ridge.fit(X,y)
mse=mean_squared_error(y,ridge.predict(X))
r2=ridge.score(X,y)
```

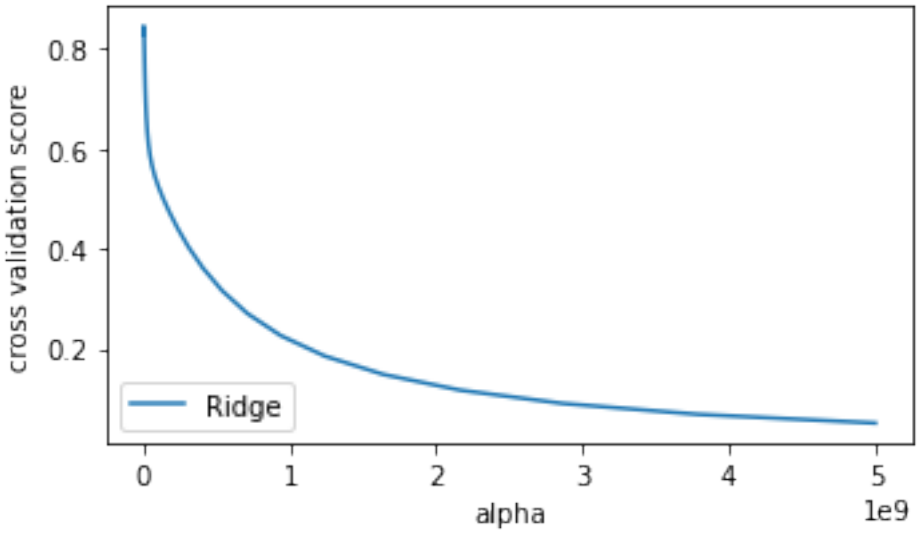
```
print("mse for ridge regression: ",mse)
print("r2 for ridge regression: ",r2)


ridge.fit(X_train,y_train)
# coeff=pd.Series(ridge.coef_, index = X.columns)
display(ridge.coef_)
display(DataFrame(list(zip(ridge.coef_[0], X_train.columns))))
```

```
range of alphas: 0.005 5000000000.0
optimal alpha:  0.10772173450159389
mse for ridge regression:  21846054179.982834
r2 for ridge regression:  0.8204506806336742
```

```
array([[ 50168.63728641,   3632.59579198, -12274.49178788,
          1832.88750652,  57635.40428256, -19686.81891839,
           207.59314895]])
```

	0	1
0	50168.637286	Bathrooms
1	3632.595792	Bedrooms
2	-12274.491788	Foreclosure
3	1832.887507	Price/SQ.Ft
4	57635.404283	Regular
5	-19686.818918	Short Sale
6	207.593149	Size

[4]:
```
plt.figure(figsize=(5, 3))

for Model in [Ridge]:
    scores = [cross_val_score(Model(alpha),X_train,y_train,cv=5).mean()
            for alpha in alphas]
    plt.plot(alphas, scores, label=Model.__name__)

plt.legend(loc='lower left')
plt.xlabel('alpha')
plt.ylabel('cross validation score')
plt.tight_layout()
plt.show()
```

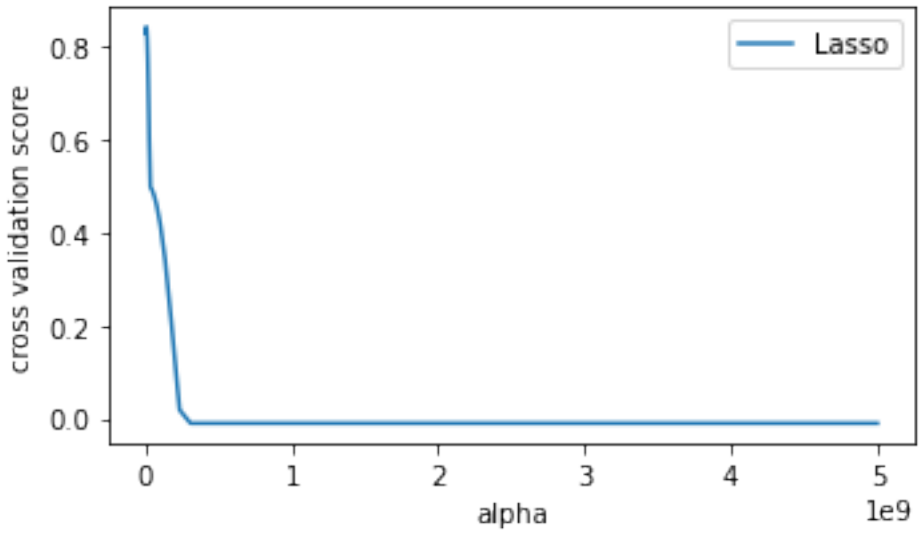


[7]:
```
lassocv = LassoCV(alphas = None, cv = 5, max_iter = 100000, normalize = True)
lassocv.fit(X_train, y_train.values.flatten())
optimal_alpha=lassocv.alpha_
print("optimal alpha: ",optimal_alpha)
lasso = Lasso(max_iter = 10000, normalize = True,)
lasso.set_params(alpha=lassocv.alpha_)
lasso.fit(X_train, y_train)
mse=mean_squared_error(y_test.values.flatten(), lasso.predict(X_test))
print("mean squared error",mean_squared_error)
alphas_lasso, coefs_lasso, _ = lasso_path(X_train, y_train, alpha=lassocv.
 ↪alpha_)
print("coeff by lasso:",lasso.coef_)
display(DataFrame(list(zip(lasso.coef_, X_train.columns))))

plt.figure(figsize=(5, 3))

for Model in [Lasso]:
    scores = [cross_val_score(Model(alpha),X_train,y_train,cv=5).mean()
            for alpha in alphas]
    plt.plot(alphas, scores, label=Model.__name__)

plt.legend(loc='upper right')
plt.xlabel('alpha')
plt.ylabel('cross validation score')
plt.tight_layout()
plt.show()
```

```
optimal alpha:  100.8230376891931
mean squared error <function mean_squared_error at 0x11e647e50>
```

```
coeff by lasso: [14743.63086893    -0.             0.          2018.14174203
 42604.79048105 -1962.03789044   252.46758038]
```

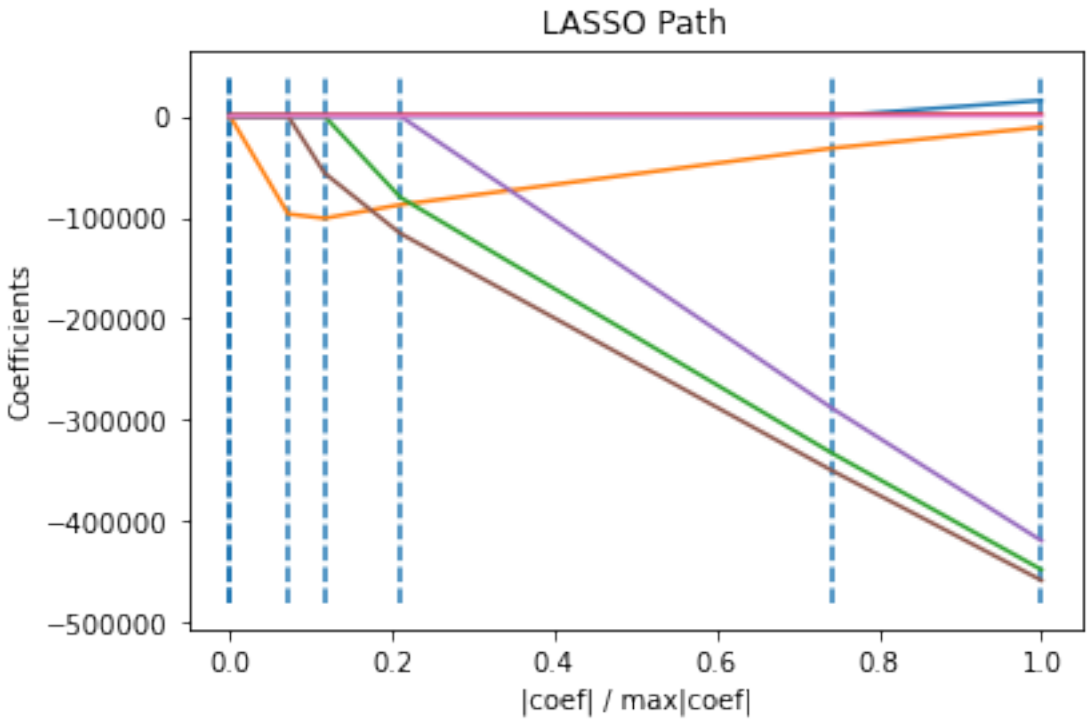
	0	1
0	14743.630869	Bathrooms
1	-0.000000	Bedrooms
2	0.000000	Foreclosure
3	2018.141742	Price/SQ.Ft
4	42604.790481	Regular
5	-1962.037890	Short Sale
6	252.467580	Size



[8]:
```
# source:
X=X.to_numpy()
y=y.values.flatten()
print(type(X))
print(type(y))
print(X.shape)
print(y.shape)
_, _, coefs = lars_path(X, y, method='lasso', verbose=True)

xx = np.sum(np.abs(coefs.T), axis=1)
xx /= xx[-1]

plt.plot(xx, coefs.T)
ymin, ymax = plt.ylim()
plt.vlines(xx, ymin, ymax, linestyle='dashed')
plt.xlabel('|coef| / max|coef|')
plt.ylabel('Coefficients')
```

```
plt.title('LASSO Path')
plt.axis('tight')
plt.show()
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
(781, 7)
(781,)
.
```



[9]:
```
# source: https://www.datatechnotes.com/2019/08/
 ↪elasticnet-regression-example-in-python.html
regr=ElasticNetCV(alphas=alphas,cv=5,l1_ratio=0.5)
regr.fit(X_train, y_train)
print("optimal alpha:",regr.alpha_)
en=ElasticNet(alpha=regr.alpha_,l1_ratio=0.5)
en.fit(X_train,y_train)
print(en.coef_)
display(DataFrame(list(zip(en.coef_, X_train.columns))))
```

```
optimal alpha: 466.30167344161
[ 1.57649644e+01 -3.57474231e+00 -1.64975771e+00  2.04507392e+03
  2.07069346e+01 -1.80568254e+01  2.67275949e+02]
```

	0	1
0	15.764964	Bathrooms
1	-3.574742	Bedrooms
2	-1.649758	Foreclosure
3	2045.073923	Price/SQ.Ft
4	20.706935	Regular
5	-18.056825	Short Sale
6	267.275949	Size

Q3

April 18, 2021

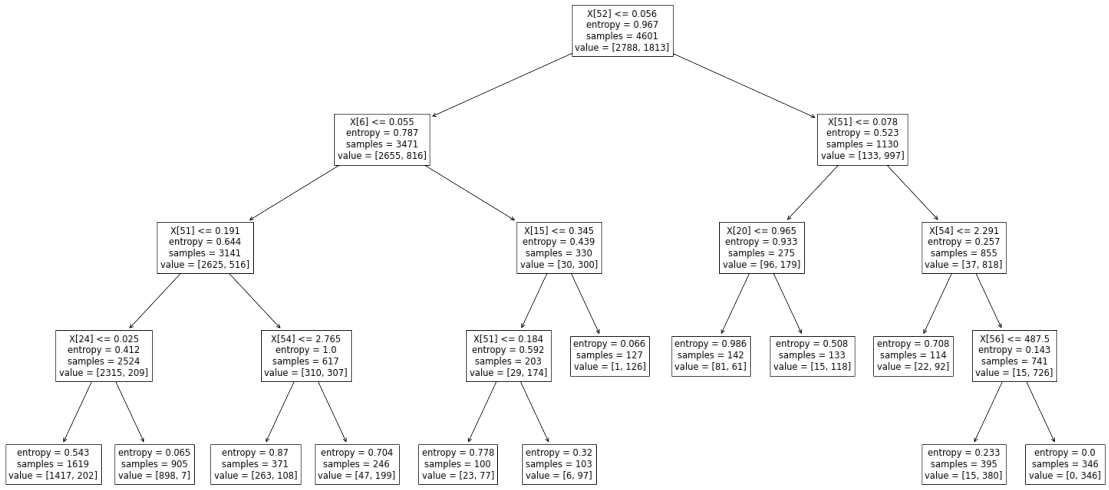
[1]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn.metrics import f1_score,classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from sklearn import svm
import random
```

[160]:
```
data = pd.read_csv('./data/spambase.data', header=None)
display(data)
output=pd.value_counts(data.iloc[:,-1])
display(output)

X = data.iloc[:,:-1]
y = data.iloc[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
decisionTree = DecisionTreeClassifier(criterion='entropy',
                              max_depth=4,
                              min_samples_leaf=100)
decisionTree.fit(X, y)
plt.figure(figsize=(26,13))
plot_tree(decisionTree);
```

	0	1	2	3	4	5	6	7	8	9	…	48	\
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00	…	0.000	
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	…	0.000	
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	…	0.010	
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	…	0.000	
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	…	0.000	
…	…	…	…	…	…	…	…	…	…	…	…	…	
4596	0.31	0.00	0.62	0.0	0.00	0.31	0.00	0.00	0.00	0.00	…	0.000	
4597	0.00	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00	0.00	…	0.000	
4598	0.30	0.00	0.30	0.0	0.00	0.00	0.00	0.00	0.00	0.00	…	0.102	
4599	0.96	0.00	0.00	0.0	0.32	0.00	0.00	0.00	0.00	0.00	…	0.000	

```
4600  0.00  0.00  0.65  0.0  0.00  0.00  0.00  0.00  0.00  0.00  …  0.000

          49   50     51     52     53     54   55     56  57
0      0.000  0.0  0.778  0.000  0.000  3.756   61    278   1
1      0.132  0.0  0.372  0.180  0.048  5.114  101   1028   1
2      0.143  0.0  0.276  0.184  0.010  9.821  485   2259   1
3      0.137  0.0  0.137  0.000  0.000  3.537   40    191   1
4      0.135  0.0  0.135  0.000  0.000  3.537   40    191   1
…      …   …    …      …      …      …   …    ..
4596   0.232  0.0  0.000  0.000  0.000  1.142    3     88   0
4597   0.000  0.0  0.353  0.000  0.000  1.555    4     14   0
4598   0.718  0.0  0.000  0.000  0.000  1.404    6    118   0
4599   0.057  0.0  0.000  0.000  0.000  1.147    5     78   0
4600   0.000  0.0  0.125  0.000  0.000  1.250    5     40   0

[4601 rows x 58 columns]
```

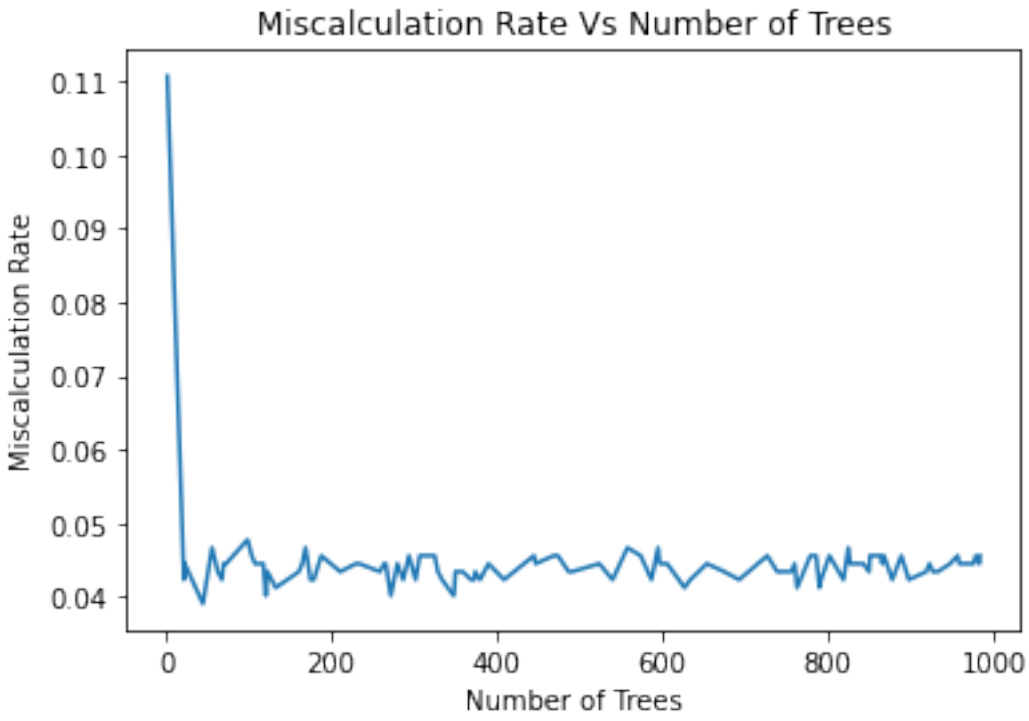
```
0    2788
1    1813
Name: 57, dtype: int64
```



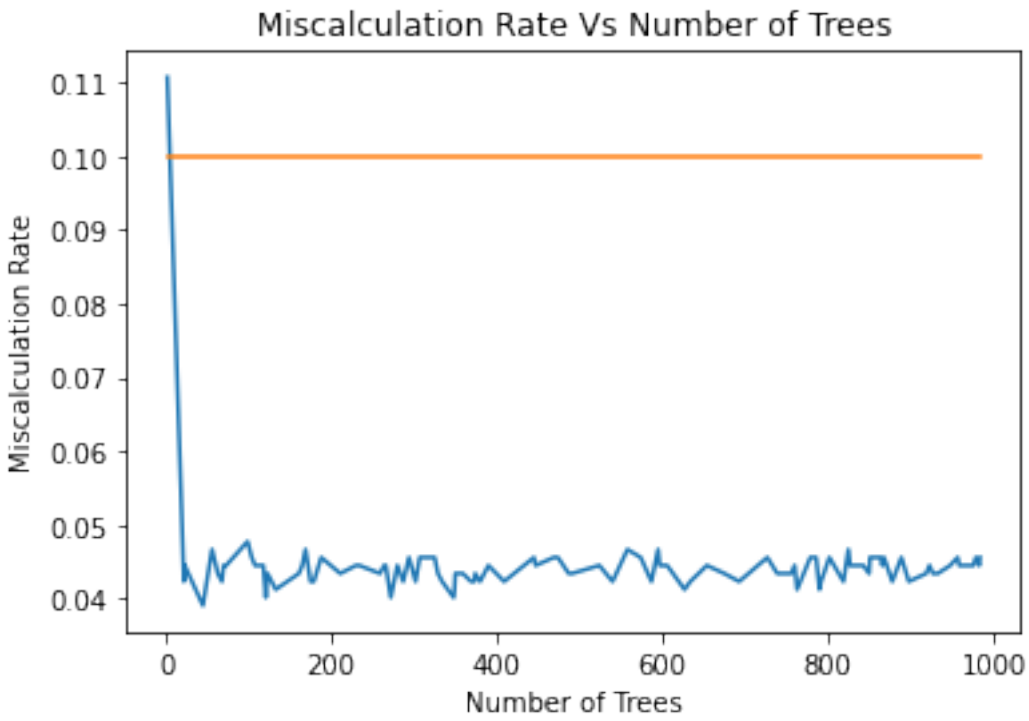
[66]:
```
totalNumberOfDataSets=X_test.shape[0]
print(totalNumberOfDataSets)
miscalcs=[]
nTrees=[]
ran=random.sample(range(0, 1000), 100)
ran.sort()
print(ran)
for nTree in ran:
```

```
    rf=RandomForestClassifier(n_estimators=nTree)
    model=rf.fit(X_train,y_train)
    modelPredict=model.predict(X_test)
    cm=confusion_matrix(y_test, modelPredict)
    fp=cm[0,1]
    fn=cm[1,0]
    miscalc=(fp+fn)/totalNumberOfDataSets
    miscalcs.append(miscalc)
    nTrees.append(nTree)
```

```
921
[2, 22, 23, 45, 56, 63, 68, 70, 73, 99, 104, 108, 118, 121, 123, 133, 161, 165,
169, 175, 179, 188, 211, 230, 234, 259, 264, 266, 272, 280, 287, 294, 302, 307,
325, 329, 334, 348, 349, 350, 359, 369, 373, 374, 380, 390, 409, 445, 447, 469,
474, 487, 491, 524, 540, 558, 574, 587, 595, 597, 606, 627, 634, 654, 675, 693,
727, 739, 752, 756, 760, 763, 779, 786, 790, 792, 802, 819, 825, 827, 843, 850,
851, 864, 866, 868, 878, 889, 898, 919, 923, 927, 933, 948, 957, 959, 976, 979,
983, 984]
```



[164]:

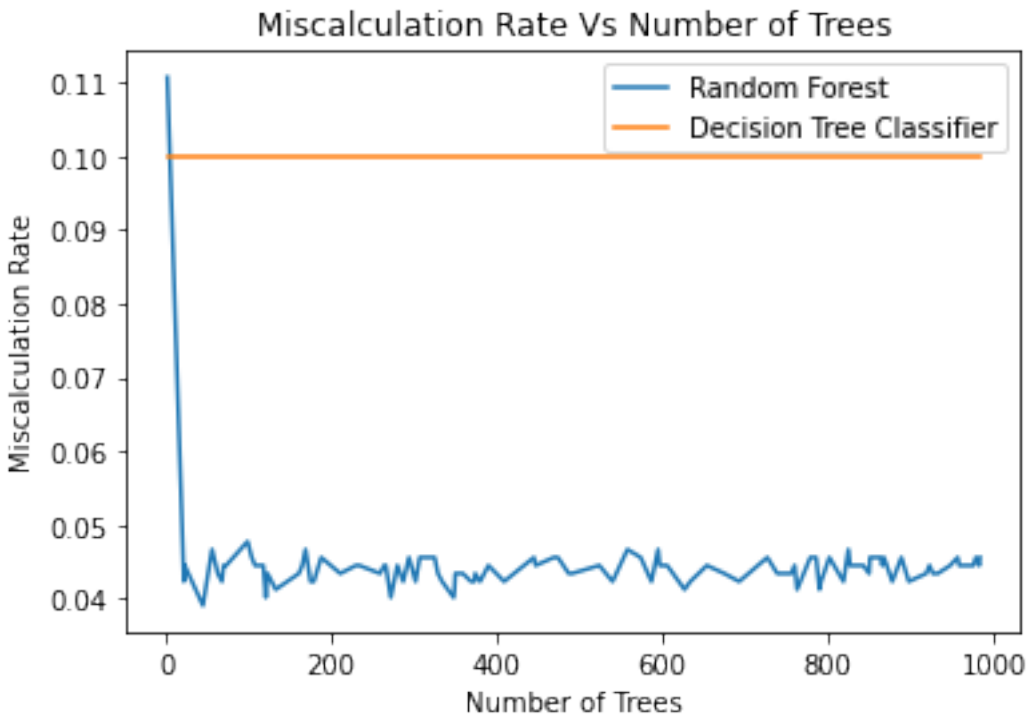


[161]:
```
optimalTrees= nTrees[miscalcs.index(min(miscalcs))]
print(optimalTrees)

rfModel=RandomForestClassifier(n_estimators=optimalTrees)
fit=rfModel.fit(X_train,y_train)
modelPredict=model.predict(X_test)
cm=confusion_matrix(y_test, modelPredict)
fp=cm[0,1]
fn=cm[1,0]
miscalc=(fp+fn)/totalNumberOfDataSets
print("confusion matrix for random forest:")
print(cm)
print("miscalc rate for random forest: ")
print(miscalc)


gs_dt = GridSearchCV(estimator=DecisionTreeClassifier(random_state=20),
                     param_grid={'max_depth': np.arange(1,31)},
                     scoring='roc_auc',
                     cv=5)
gs_dt.fit(X_train, y_train)
dtModel=gs_dt.best_estimator_
print("================")
```

```
predictions=dtModel.predict(X_test)
cm_dt=confusion_matrix(y_test,predictions)
fp_dt=cm_dt[0,1]
fn_dt=cm_dt[1,0]
miscalc_dt=(fp_dt+fn_dt)/totalNumberOfDataSets
print("confusion matrix for decision tree:")
print(cm_dt)
print("miscalc rate for decision tree: ")
print(miscalc_dt)
```

```
45
confusion matrix for random forest:
[[520  11]
 [ 31 359]]
miscalc rate for random forest:
0.04560260586319218
================
confusion matrix for decision tree:
[[513  18]
 [ 74 316]]
miscalc rate for decision tree:
0.0998914223669924
```

[168]:
```
a=np.empty(len(nTrees));
a.fill(miscalc_dt)

plt.plot(nTrees,miscalcs,label='Random Forest')
plt.plot(nTrees,a,label='Decision Tree Classifier')
plt.title('Miscalculation Rate Vs Number of Trees')
plt.xlabel('Number of Trees')
plt.ylabel('Miscalculation Rate')
plt.legend()
plt.show()
```



[176]:
```
# non_spam_emails=X_train[X_train[57]==0]
# clf = svm.OneClassSVM(nu=0.1, kernel="rbf", gamma=0.1)
# clf.fit(X_train)
# display(y_train[y_train[0]==0])
# X = data.iloc[:,:-1]
# y = data.iloc[:,-1]

train,test=train_test_split(data,test_size=0.2, random_state=42)
print(train.shape)
print(test.shape)
print("counts in train data:")
print(train[57].value_counts())
print("counts in test data:")
print(test[57].value_counts())

noofSpam_test=test[57].value_counts()[0]
noofNonSpam_test=test[57].value_counts()[1]
noofData_test = test.shape[0]

X_train=train[train[57]==0]
X_train=X_train.iloc[:,:-1]

# display(train)
```

```
# display(X_train)

clf = svm.OneClassSVM(nu=0.1, kernel="rbf", gamma="auto")
clf.fit(X_train)
y_pred_train = clf.predict(train.iloc[:,:-1])
n_error_train = y_pred_train[y_pred_train == -1].size
print("No of spams detected by model in training set:")
print(n_error_train)

y_pred_test = clf.predict(test.iloc[:,:-1])
y_pred_non_spam = y_pred_test[y_pred_test == -1].size
y_pred_spam = y_pred_test[y_pred_test == 1].size
print("No of spams detected by model in test set:")
print(y_pred_non_spam)
print("No of non-spams detected by model in test set:")
print(y_pred_spam)

tot_miscalc = abs(y_pred_non_spam - noofNonSpam_test) + abs(y_pred_spam -␣
 ↪noofSpam_test)
tot_miscalc_rate = tot_miscalc / noofData_test
print("miscalculation rate:")
print(tot_miscalc_rate)
```

```
(3680, 58)
(921, 58)
counts in train data:
0    2257
1    1423
Name: 57, dtype: int64
counts in test data:
0    531
1    390
Name: 57, dtype: int64
No of spams detected by model in training set:
1581
No of spams detected by model in test set:
527
No of non-spams detected by model in test set:
394
miscalculation rate:
0.2975027144408252
```

[59]:
```
X = np.array([1.5, 1]).T
x=0.5
beta = np.array([1, 0.5]).T
I = np.array([1,1])
np.logspace(2.0, 3.0, num=4)
```

```
# lambdas = [0,0.25,0.5,0.75,1.00,1.25,1.50,1.75,2.00]
lambdas = [0.25]
bias_squared=[]
for l in lambdas:
    b2= (x*((np.dot(X.T,X) + (np.dot(l,I)))**-1)*((np.dot(X.T,X))*beta).T) - np.
 ↪dot(beta,x)
    t1=((np.dot(X.T, X)+np.dot(l,I))**-1)
    t2=((X.T * X)*beta).T
    t3=np.dot(beta,X.T)

    bias_squared.append(b2)
print(bias_squared)
```

```
[array([-0.03571429, -0.01785714])]
```

[ ]: