Homework 4 CSCE 421

December 1, 2019

1 Jose Garza 225008812

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn import svm, datasets
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
import random
import math
import graphviz

%matplotlib inline
```

[2]:
```
#load dataset
trainData = pd.read_csv('OnlineNewsPopularityTrain.csv',sep=',')
trainData = trainData.drop('url', axis = 1)



cols = trainData.columns[:-1]
trainDataX = pd.DataFrame(trainData, columns = cols)
trainDataY = pd.DataFrame(trainData, columns = [' shares'])
```

[3]:
```
testData = pd.read_csv('OnlineNewsPopularityTest.csv',sep=',')
testData = testData.drop('url', axis = 1)
```

```
cols = testData.columns[:-1]
testDataX = pd.DataFrame(testData, columns = cols)
testDataY = pd.DataFrame(testData, columns = [' shares'])
```

[4]:
```
# list(data.columns)
myList = []
for col in trainData.columns:
    myList.append(col)
```

2 Split Data (5-fold cross-validation)

[5]:
```
def SplitDataFiveFolds(myData, interval):
    #     [0....(1..intvl..)2.....3....4......]
    # takes padas dataframe

    folds = 5
    intervalSze = math.floor(len(myData)/folds)
    start = (interval*intervalSze)

    extra = len(myData)%folds-1 if interval==folds else 0
    end = start + intervalSze + extra


    # interval
    test = myData[start:end]

    trainOne = myData[:start]
    trainTwo = myData[end:]
    train = trainOne.append(trainTwo)


    cols = myData.columns[:-1]
    xTest = pd.DataFrame(test, columns = cols)
    yTest = pd.DataFrame(test, columns = [' shares'])
    xTrain = pd.DataFrame(train, columns = cols)
    yTrain = pd.DataFrame(train, columns = [' shares'])
    return xTest, yTest, xTrain, yTrain
```

2.1 Total Error

[6]:
```
def SQRSS(prediction,actual):
    totalSum = 0
    for i in range(len(actual)):
        totalSum += (prediction[i] - actual.iloc[i]) ** 2
```

```
    return math.sqrt(totalSum)
```

[7]:
```
def PercError(prediction,actual):
    total = 0
    for i in range(len(actual)):
        total += (abs(prediction[i]-actual.iloc[i])/actual.iloc[i])*100
    return total/len(actual)
```

[8]:
```
def Acc(pred, actual):
    ac = accuracy_score(pred, actual)
    return ac
```

[9]:
```
def CalcErr(prediction,actual):
    sqrss = SQRSS(prediction,actual)
    prcErr = PercError(prediction,actual)
    ac = Acc(prediction,actual)
    print('   SQRSS:          ' + str(sqrss)[:9])
#     print('   Percent Error: ' + str(prcErr)[:5] + '%')
#     print('   Accuracy:       ' + str(ac*100)[:4] + '%')
    return sqrss
```

3 (a) Decision Trees

[10]:
```
def DecisionTreeClassifier(depth,xTest, yTest, xTrain, yTrain):
    dtc = tree.DecisionTreeClassifier(max_depth=depth)
    dtc = dtc.fit(xTrain, yTrain)
    return dtc
```

[11]:
```
def DrawTree(dt):
    plt.figure()
    iris = load_iris()
    plt.rcParams["figure.figsize"] = (10 ,10)
    tree.plot_tree(dt.fit(iris.data, iris.target))
    plt.show()
```

[12]:
```
class Node:
    def __init__(self, value, depth, fold):
        self.value = value
        self.depth = depth
        self.fold = fold
        return
```

3.1 Run over all folds

[13]:
```
def DecisionTree(data, depth, fold, drawTree = False):
    x_test, y_test, x_train, y_train = SplitDataFiveFolds(data,foldInd)
    dtc = DecisionTreeClassifier(depth, x_test, y_test, x_train, y_train)

    # only draw tree once
    if drawTree :
        print('Tree: Depth = ' + str(depth))
        DrawTree(dtc)
        dtc = DecisionTreeClassifier(depth, x_test, y_test, x_train, y_train)

    # Make Prediction
    pred_dtc = dtc.predict(x_test)
    err = SQRSS(pred_dtc,y_test)
    return err
```

3.1.1 Get Best Depth from Training Data

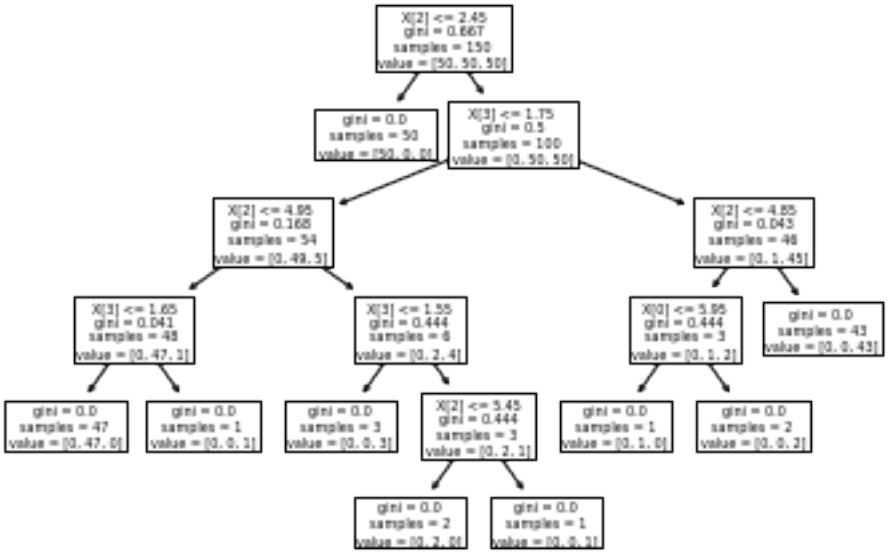
[33]:
```
# Decisiom Tree over all folds for depth 1 --> depth
avgErrTrainData = []
folds = 5
mini = Node(math.inf,0,0)
maxDepth = 12

for d in range(1,maxDepth+1):
    depErrs = []
    for foldInd in range(folds):
        # Make DTC
        err = DecisionTree(trainData, d, foldInd, False)
        depErrs.append(err)

        # Check if best
        if(err<mini.value):
            mini = Node(err,d, foldInd)

    avgErrTrainData.append(sum(depErrs)/len(depErrs) )
```

[36]:
```
print('Best Depth: ' + str(mini.depth))
print('Fold iteration: ' + str(mini.fold))
print('Best SQRSS: ' + str(mini.value))
print()
print('Average Errors at: ')
for i in range(len(avgErrTrainData)):
    print('Average Error for Depth (' + str(i+1)  + '): ' +␣
 ↪str(avgErrTrainData[i])[:10])
```

```
Best Depth: 11
Fold iteration: 3
Best SQRSS: 634005.8133873852

Average Errors at:
Average Error for Depth (1): 1004311.08
Average Error for Depth (2): 1004083.93
Average Error for Depth (3): 1003068.14
Average Error for Depth (4): 1002698.63
Average Error for Depth (5): 1002800.00
Average Error for Depth (6): 1002513.95
Average Error for Depth (7): 1001970.55
Average Error for Depth (8): 1001623.11
Average Error for Depth (9): 1002152.52
Average Error for Depth (10): 1002227.27
Average Error for Depth (11): 1001287.32
Average Error for Depth (12): 1001723.98
```

3.1.2 Use Testing data to make prediction using best depth

[16]:
```
bestDTC = DecisionTreeClassifier(mini.depth, testDataX, testDataY, trainDataX,␣
↪trainDataY)

DrawTree(bestDTC)
bestDTC = DecisionTreeClassifier(mini.depth, testDataX, testDataY, trainDataX,␣
↪trainDataY)
# Make Prediction
pred_dtc = bestDTC.predict(testDataX)
err = SQRSS(pred_dtc,testDataY)
```



[43]:
```
dot_data = tree.export_graphviz(bestDTC, out_file=None,
                                feature_names=testData.columns[:-1],
                                filled=True, rounded=True, leaves_parallel=True,
                                max_depth = 2)
graph = graphviz.Source(dot_data)
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

4 (b) Random Forest Regression

[18]:
```
def RandForest(data):
    # random forest consist of 8 random features
    randInd= random.sample(range(0, len(trainData.columns)-1), 8)

    randFeatures = []
    for i in randInd:
        randFeatures.append(data.columns[i])

    randFeatures.append(' shares')

    randForest = pd.DataFrame(data, columns = list(randFeatures))
    return randForest
```

[19]:
```
def RandForestNDepths(maxDepth, errList):

    folds = 5
    depErrs = []
    avgErrDepth = []
    err = 0
    for d in range(1,maxDepth+1):
        err = 0
        #gets a combination of random features
        rf = RandForest(trainData)
        print('Random Forest Tree: Features (')

        for col in rf.columns:
            print(col)
        print(')')
        for foldInd in range(folds):
            # Make a Decision Tree with the data from the Random Forest
            err += DecisionTree(rf, d, foldInd)

        avg = err / folds
        avgErrDepth.append(avg)

    errList.append(avgErrDepth)
```

[20]:
```
class ErrListNode:
    def __init__(self, err, depth, tree):
        self.err = err
        self.depth = depth
        self.tree = tree
        return
```

[21]:
```
def RandForestDepthX(depth, curNode, drawTree):

    folds = 5
    depErrs = []
    avgErrDepth = []
    err = 0
    err = 0
    #gets a combination of random features
    rf = RandForest(trainData)
    for foldInd in range(folds):
        # Make a Decision Tree with the data from the Random Forest
        err += DecisionTree(rf, depth, foldInd, drawTree)

    avg = err / folds
    curNode.err = avg
```

[22]:
```
# Computes The Average Error for N Trees with Depths of 0 --> maxDepth
avgErrNTrees = []
avgErr = []
maxDepth = 8
totalTrees = 6
# 1 trees at depth 1
# 2 trees at depth 1
# 3 trees at depth 1

# 1 trees at depth 2
# 2 trees at depth 2
# 3 trees at depth 2


errAtDepth = []

for d in range(1,maxDepth+1):
    for t in range(1,totalTrees+1 ):
        cur = ErrListNode(0,d,t)
        RandForestDepthX(d, cur, False)
        errAtDepth.append(cur)
```

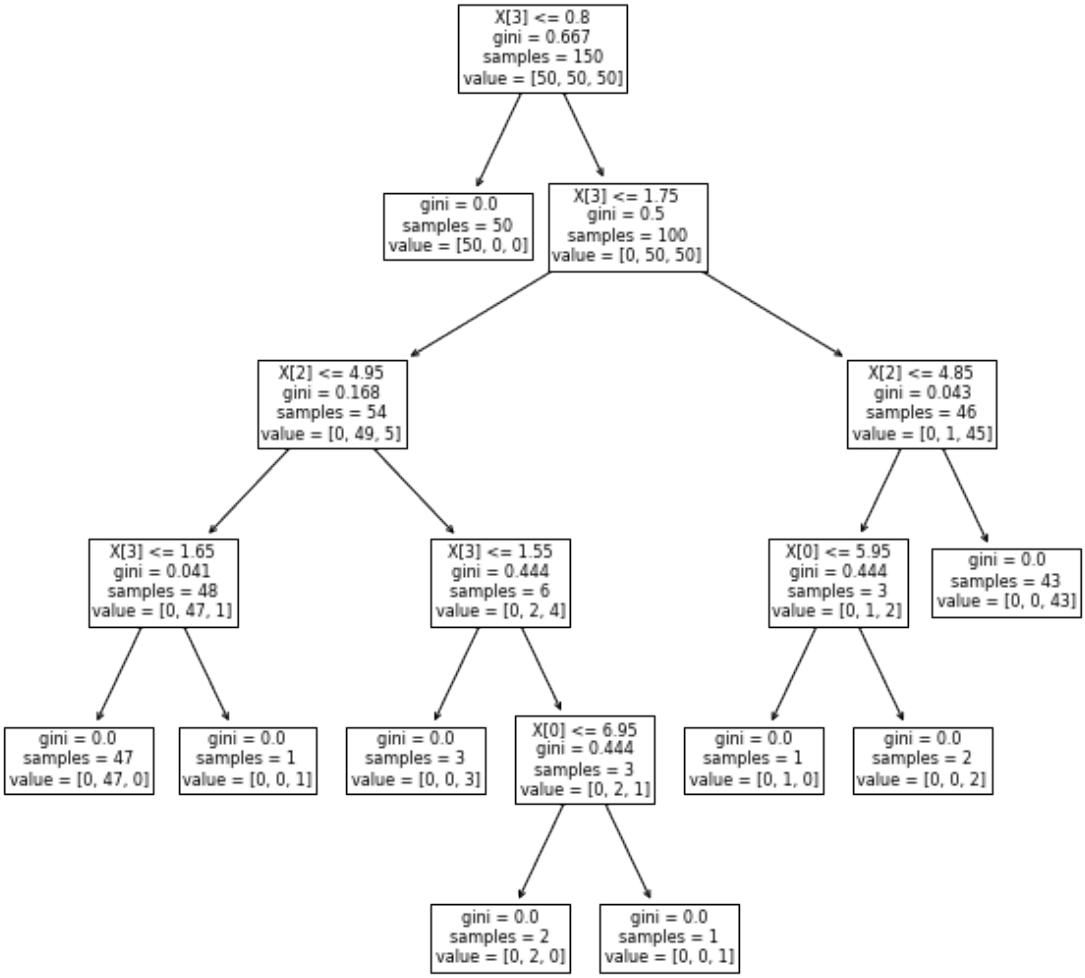
[23]:
```
# Get Best SQRSS at best total Trees and best depth
best = math.inf
bestDepth = math.inf
bestTotalTrees = math.inf

for i in range(0,maxDepth*totalTrees):
    currVal = errAtDepth[i]
    if currVal.err < best:
```

```
        best = currVal.err
        bestDepth = currVal.depth
        bestTotalTrees = currVal.tree

print('Optimal trees: ' + str(bestTotalTrees)  + " (SQRSS)-" + str(best)[:9])
print('At Depth: ' + str(bestDepth) )
```

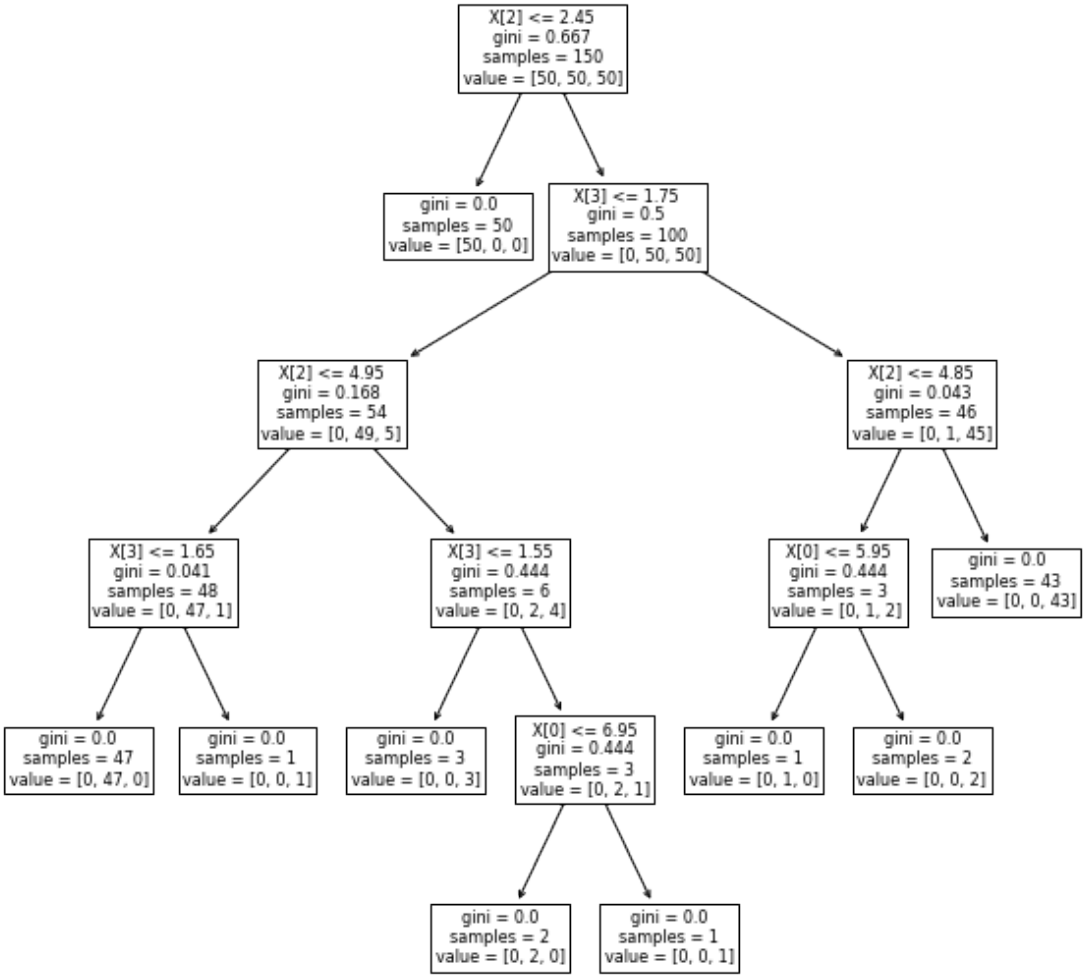
```
Optimal trees: 3 (SQRSS)-707349.11
At Depth: 5
```

[24]:
```
avgErrTest = []
for t in range(1,bestTotalTrees+1):
    rf = RandForest(testData)
    err = 0
    #gets a combination of random features
    rf = RandForest(testData)

    # Make a Decision Tree with the data from the Random Forest
    dtc = DecisionTreeClassifier(bestDepth, testDataX, testDataY, trainDataX,␣
 ↪trainDataY)
    print('Tree: ' + str(t))
    DrawTree(dtc)
    dtc = DecisionTreeClassifier(bestDepth, testDataX, testDataY, trainDataX,␣
 ↪trainDataY)


    # Make Prediction
    pred_dtc = dtc.predict(trainDataX)
    err = SQRSS(pred_dtc,trainDataY)


    # Make Prediction
    pred_dtc = dtc.predict(testDataX)
    err = SQRSS(pred_dtc,testDataY)
    avgErrTest.append(err)

avgTest = sum(avgErrTest) / len(avgErrTest)
print('Optimal trees: ' + str(bestTotalTrees)  + " (SQRSS)-" + str(avgTest)[:9])
print('At Depth: ' + str(bestDepth) )
```

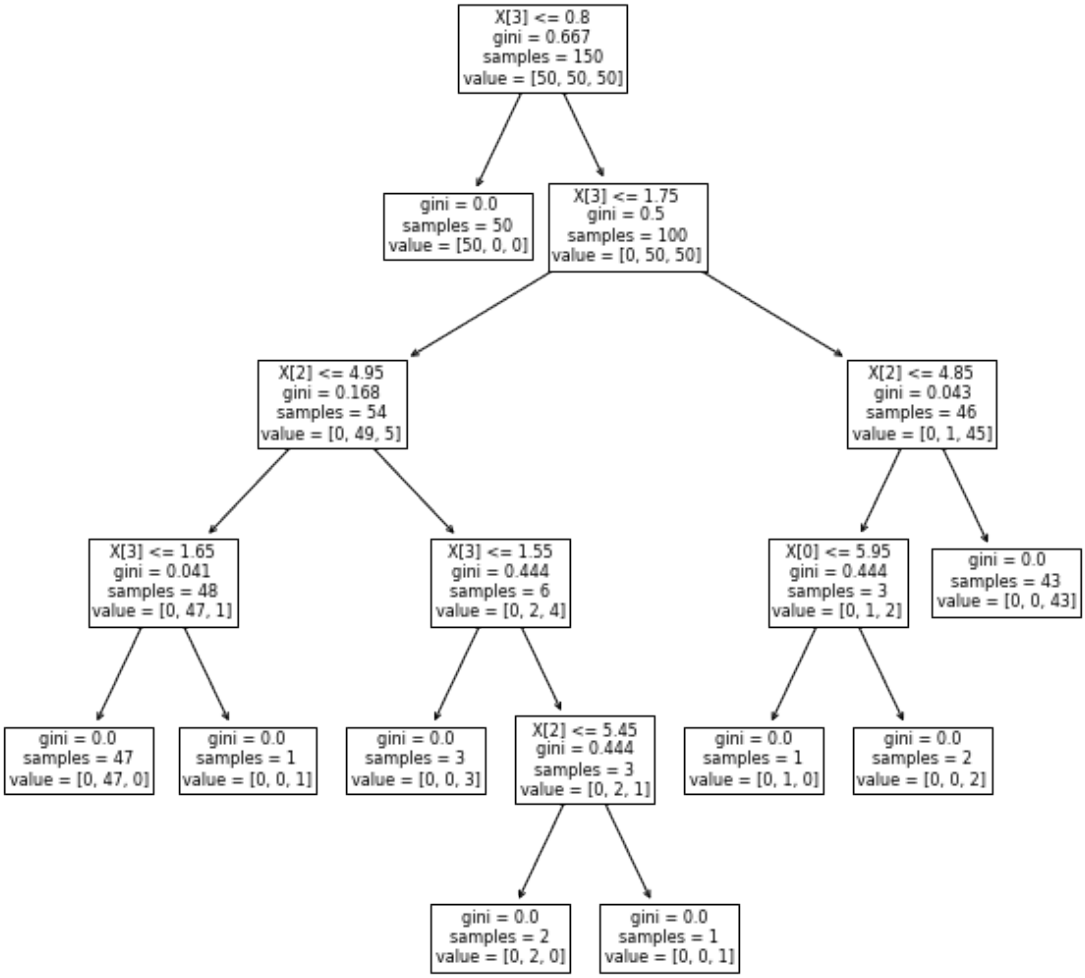
```
Tree: 1
```


X[3] <= 0.8
gini = 0.667
samples = 150
value = [50, 50, 50]
gini = 0.0
samples = 50
value = [50, 0, 0]
X[3] <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
X[2] <= 4.95
gini = 0.168
samples = 54
value = [0, 49, 5]
X[2] <= 4.85
gini = 0.043
samples = 46
value = [0, 1, 45]
X[3] <= 1.65
gini = 0.041
samples = 48
value = [0, 47, 1]
X[3] <= 1.55
gini = 0.444
samples = 6
value = [0, 2, 4]
X[0] <= 5.95
gini = 0.444
samples = 3
value = [0, 1, 2]
gini = 0.0
samples = 43
value = [0, 0, 43]
gini = 0.0
samples = 47
value = [0, 47, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]
gini = 0.0
samples = 3
value = [0, 0, 3]
X[0] <= 6.95
gini = 0.444
samples = 3
value = [0, 2, 1]
gini = 0.0
samples = 1
value = [0, 1, 0]
gini = 0.0
samples = 2
value = [0, 0, 2]
gini = 0.0
samples = 2
value = [0, 2, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]

Tree: 2


X[2] <= 2.45
gini = 0.667
samples = 150
value = [50, 50, 50]
gini = 0.0
samples = 50
value = [50, 0, 0]
X[3] <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
X[2] <= 4.95
gini = 0.168
samples = 54
value = [0, 49, 5]
X[2] <= 4.85
gini = 0.043
samples = 46
value = [0, 1, 45]
X[3] <= 1.65
gini = 0.041
samples = 48
value = [0, 47, 1]
X[3] <= 1.55
gini = 0.444
samples = 6
value = [0, 2, 4]
X[0] <= 5.95
gini = 0.444
samples = 3
value = [0, 1, 2]
gini = 0.0
samples = 43
value = [0, 0, 43]
gini = 0.0
samples = 47
value = [0, 47, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]
gini = 0.0
samples = 3
value = [0, 0, 3]
X[0] <= 6.95
gini = 0.444
samples = 3
value = [0, 2, 1]
gini = 0.0
samples = 1
value = [0, 1, 0]
gini = 0.0
samples = 2
value = [0, 0, 2]
gini = 0.0
samples = 2
value = [0, 2, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]

Tree: 3




```
Optimal trees: 3 (SQRSS)-306066.95
At Depth: 5
```

5 (c) Feature exploration

If we observe the top features that determine the share popularity of models, we see: 'Avg. keyword', 'data_channel_is_entertainment', 'is_weekend', and 'timedelta'.

The intution behind why the Decision Tree has these as the main factors to guide our samples is can be derived by recognizing that these features are key values to how popular an article is.

For instance, if a channel is entertaining, odds are, people would enjoy the conent and share it with their peers. Also, if the article includes average keywords, it would give the users a more user-friendly tone behind the content. Lastly, if it is the weekend or if the article is short, the

readers would most likely have more engaging expereince since they wouldn't be discouraged to read it.

On another note, if we observe the 'gini' attribute, this measures the inequality behind features. We can see that the the higher this value, the higher this feature is in the tree. This also gives us an intuition since we can more confidently label certain samples early on.

6 (d) Feedforward neural network

6.1 MLPClassifier

[25]:
```
# different number of layers, node per layer, activation functions, dropout,␣
 ↪and learning rate.

# The ith element represents the number of neurons in the ith hidden layer.
#  input layer -> 50 neurons -> 50 neurons -> output layer
layers = (50 , 50, )

# activation : {'identity', 'logistic', 'tanh', 'relu'}, default 'relu'
activationFnc = ['relu', 'identity', 'logistic', 'tanh']

# learning_rate : {'constant', 'invscaling', 'adaptive'}, default 'constant'
learningRate = ['constant', 'invscaling', 'adaptive']


ffnn = MLPClassifier(alpha=1,
                     max_iter=6000,
                     hidden_layer_sizes = layers,
                     activation = activationFnc[0],
                     learning_rate = learningRate[0]
                    )

x_test, y_test, x_train, y_train = SplitDataFiveFolds(trainData,0)

ffnn.fit(x_train, y_train.values.ravel())
score = ffnn.score(x_test, y_test.values.ravel())
```

6.2 Hyper-Parameter Tuning

[26]:
```
iris = datasets.load_iris()
parameters = {'activation':activationFnc, 'learning_rate': learningRate}

clf = GridSearchCV(estimator = ffnn, param_grid = parameters, cv=5)
clf.fit(iris.data, iris.target)
```

[26]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=MLPClassifier(activation='relu', alpha=1,
                                     batch_size='auto', beta_1=0.9,
                                     beta_2=0.999, early_stopping=False,
                                     epsilon=1e-08, hidden_layer_sizes=(50, 50),
                                     learning_rate='constant',
                                     learning_rate_init=0.001, max_iter=6000,
                                     momentum=0.9, n_iter_no_change=10,
                                     nesterovs_momentum=True, power_t=0.5,
                                     random_state=None, shuffle=True,
                                     solver='adam', tol=0.0001,
                                     validation_fraction=0.1, verbose=False,
                                     warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'activation': ['relu', 'identity', 'logistic', 'tanh'],
                         'learning_rate': ['constant', 'invscaling',
                                           'adaptive']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

[27]:
```
print('Best Score: ' + str(clf.best_score_)[:6])
```

```
Best Score: 0.98
```