In [28]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import random
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor
import gensim
import warnings
warnings.filterwarnings('ignore')
```

### Preprocess Data

In [7]:
```
data = pd.read_csv('OnlineNewsPopularityTrain.csv', delimiter = ',')
data.columns = [c.strip() for c in data.columns]

columns = []
for c in data.columns:
    if c!='url' and c!='shares' and c!='topic' and c!='topic_numeric':
        columns.append(c)
X = data.loc[:, columns]
y = data.loc[:, 'shares']
testData = pd.read_csv('OnlineNewsPopularityTest.csv', delimiter = ',')
testData.columns = [c.strip() for c in testData.columns]
X_test = testData.loc[:, columns]
y_test = testData.loc[:, 'shares']
# assert (len(X.columns) == len(X_test.columns))
```

In [8]:
```
len(X)
```

Out[8]: 38422

In [9]:
```
def cross_val(model, xData, yData, num_fold):
    fold = list(range(num_fold))
    np.random.seed(100)
    xData['fold'] = xData.apply(lambda x: random.choice(fold), axis=1)
    rss=[]
    for i in fold:
        # model = model(max_depth=depth)

        X_testData = xData.loc[(xData['fold']==i)]
        X_trainData = xData.loc[(xData['fold']!=i)]
        y_testData = yData.loc[X_testData.index]
        y_trainData = yData.loc[X_trainData.index]
        model.fit(X_trainData, y_trainData)
        y_pred = model.predict(X_testData)
        rss.append(np.sqrt(mean_squared_error(y_testData, y_pred)*len(y_
testData)))
    xData.pop('fold')
    return sum(rss)/len(rss)
```

In [10]:
```
def GetSmallKey(thisDict):

    smallkey = list(thisDict.keys())[0]
    small = thisDict[smallkey]
    for key, value in thisDict.items():
        if thisDict[key] < small:
            small = thisDict[key]
            smallkey = key
    return smallkey
```

## 1(a)

In [11]:
```
thisDict = {}
for max_depth in range(1,10):
    model = DecisionTreeRegressor(max_depth=max_depth)
    score = cross_val(model, X, y, 5)
    thisDict[max_depth]=score
optimalDepth = GetSmallKey(thisDict)
optimalDepth
```

Out[11]:
```
2
```

In [12]:
```
model = DecisionTreeRegressor(max_depth=optimalDepth)
model.fit(X,y)
```

Out[12]:
```
DecisionTreeRegressor(criterion='mse', max_depth=2, max_features=None,
                      max_leaf_nodes=None, 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='bes
t')
```

## 1(c)

In [13]:
```
print(model.feature_importances_)
```

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.        ]
```

In [14]:
```
featureList = []
for i, j in zip(columns, model.feature_importances_):
    if j > 0:
      featureList.append(i)
```

In [15]:
```
featureList
```

Out[15]:
```
['kw_avg_avg', 'self_reference_min_shares', 'self_reference_avg_shares
s']
```

In [16]:
```
y_pred = model.predict(X_test)
rss = np.sqrt(mean_squared_error(y_test, y_pred)*len(y_test))
print(rss)
```

```
290019.59784158063
```

## 1(b)

In [29]:
```
class RandomForest:
    def __init__(self, num_tree = 10, feature_fraction = 0.9, sample_fra
ction = 0.9, max_depth = 10):
        self.num_tree = num_tree
        self.max_depth = max_depth
        self.feature_fraction = feature_fraction
        self.sample_fraction = sample_fraction
        self.models = []
        self.columns = []

    def fit(self, x, y):
        for _ in range(self.num_tree):
            m = DecisionTreeRegressor(max_depth = self.max_depth)
            new_x = x.sample(frac=self.sample_fraction)
            new_y = y.loc[new_x.index]
            s = pd.Series(new_x.columns).sample(frac=self.feature_fracti
on)
            new_x = new_x.loc[:, s]
            # print(len(new_x.columns))
            m.fit(new_x, new_y)
            self.models.append(m)
            self.columns.append(s)
        return self

    def predict(self, x):
        result=np.zeros(len(x))
        for columns, model in zip(self.columns, self.models):
            result+=np.array(model.predict(x.loc[:, columns]))
        return result/self.num_tree
```

In [30]:
```
thisDict ={}
thisMatrix = np.zeros([4,4])
for max_depth in range(1,5):
    for num_tree in range(1,5):

        modelRandomForest = RandomForest(num_tree = num_tree, max_depth=
max_depth)
        score = cross_val(modelRandomForest, X, y, 5)
        thisDict[num_tree, max_depth]=score
        thisMatrix[num_tree-1, max_depth-1] = score
```

In [31]:
```
optimal = GetSmallKey(thisDict)
```

In [32]:
```
thisDict
```

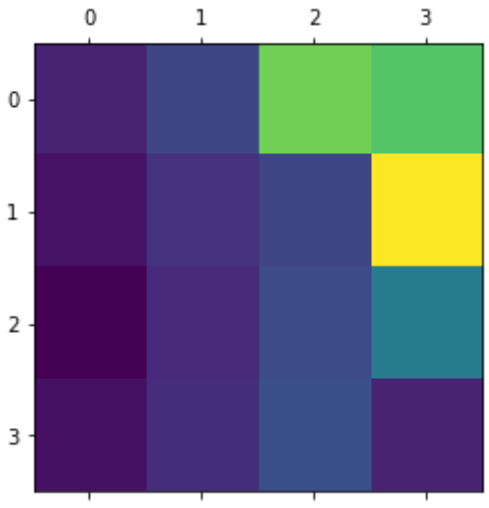
Out[32]:
```
{(1, 1): 1252714.126365335,
 (2, 1): 1235078.8161193202,
 (3, 1): 1217213.3767764056,
 (4, 1): 1234405.4080761534,
 (1, 2): 1294032.5805795307,
 (2, 2): 1271490.5769890002,
 (3, 2): 1261013.816688199,
 (4, 2): 1265479.5815478512,
 (1, 3): 1507239.726901819,
 (2, 3): 1293968.7571174733,
 (3, 3): 1301704.9207236224,
 (4, 3): 1306492.7489997814,
 (1, 4): 1486981.7486795057,
 (2, 4): 1585803.6695444856,
 (3, 4): 1372217.1797709402,
 (4, 4): 1254299.2295549854}
```

In [33]:
```
thisMatrix
```

Out[33]:
```
array([[1252714.12636533, 1294032.58057953, 1507239.72690182,
        1486981.74867951],
       [1235078.81611932, 1271490.576989  , 1293968.75711747,
        1585803.66954449],
       [1217213.37677641, 1261013.8166882 , 1301704.92072362,
        1372217.17977094],
       [1234405.40807615, 1265479.58154785, 1306492.74899978,
        1254299.22955499]])
```

In [34]:
```
plt.matshow(thisMatrix)
```

Out[34]:
```
<matplotlib.image.AxesImage at 0x1a1b894630>
```



In [35]:
```
modelRandomForest = RandomForest(num_tree = optimal[0], max_depth=optima
l[1])
modelRandomForest.fit(X,y)
y_pred = modelRandomForest.predict(X_test)
rss = np.sqrt(mean_squared_error(y_test, y_pred)*len(y_test))
print(rss)
```

```
290801.60687465017
```

## 1(d)

In [36]:
```
thisDict ={}
hidden_layer_sizes = [(10, 5, 5),(20,10)]
for h in hidden_layer_sizes:
    for learning_rate in [0.1, 0.01]:
        for activation_func in ['logistic', 'relu']:
            modelMLP = MLPRegressor(hidden_layer_sizes=h, learning_rate_
init = learning_rate, activation = activation_func)
            score = cross_val(modelMLP, X, y, 5)
            thisDict[h,learning_rate,activation_func]=score
```

In [39]:
```
opt = GetSmallKey(thisDict)
modelMLP = MLPRegressor(hidden_layer_sizes=opt[0], learning_rate_init =
opt[1], activation = opt[2])
modelMLP.fit(X,y)
y_pred = modelMLP.predict(X_test)
rss = np.sqrt(mean_squared_error(y_test, y_pred)*len(y_test))
print(rss)
```

```
295702.212887975
```

In [42]:
```
optFeatureData = X.loc[:, featureList]
optFeatureDataTest = X_test.loc[:,featureList]
modelMLP.fit(optFeatureData,y)
y_pred = modelMLP.predict(optFeatureDataTest)
rss = np.sqrt(mean_squared_error(y_test, y_pred)*len(y_test))
print(rss)
```

```
295628.10712484404
```

### Extra credit

In [60]:
```
data['topic']= data['url'].str.split('/').apply(lambda x: x[-2] if x[-1]
=='' else x[-1]).str.split('-')
testData['topic']= testData['url'].str.split('/').apply(lambda x: x[-2]
if x[-1]=='' else x[-1]).str.split('-')

word2vec = gensim.models.Word2Vec(list(data['topic']), size=100, window=
5, min_count=1, workers=4)
data['topic_numeric'] = data['topic'].apply(lambda x: np.mean(word2vec[x
],axis=0))
testData['topic'] = testData['topic'].apply(lambda x: [w for w in x if w
in word2vec.wv.vocab])


testData['topic_numeric'] = testData['topic'].apply(lambda x: np.mean(wo
rd2vec[x],axis=0) if len(x)>0 else np.zeros(100))

X = pd.concat([data.loc[:, columns], pd.DataFrame(np.stack(data['topic_n
umeric'].values), index=X.index)], axis=1)
X_test= pd.concat([X_test.loc[:, columns], pd.DataFrame(np.stack(testDat
a['topic_numeric'].values), index=X_test.index)],
                  axis=1)


thisDict = {}
for max_depth in range(1,10):
    model = DecisionTreeRegressor(max_depth=max_depth)
    score = cross_val(model, X, y, 5)
    thisDict[max_depth]=score
optimalDepth = GetSmallKey(thisDict)
optimalDepth
```

Out[60]:
```
1
```

In [61]:
```
model = DecisionTreeRegressor(max_depth=optimalDepth)
model.fit(X,y)
y_pred = model.predict(X_test)
rss = np.sqrt(mean_squared_error(y_test, y_pred)*len(y_test))
print(rss)
```

```
290349.02367745846
```

In [57]:
```
optFeatureData = X.loc[:, featureList]
optFeatureDataTest = X_test.loc[:,featureList]
model.fit(optFeatureData,y)
y_pred = model.predict(optFeatureDataTest)
rss = np.sqrt(mean_squared_error(y_test, y_pred)*len(y_test))
print(rss)
```

```
290349.02367745846
```

In [ ]: