APSSDC

Andhra Pradesh State Skill Development Corporation

Day08 Machine Learning Using Python

Day08 Objectives

- Random Forest Classifier
- Unsupervised Learning
- Clustering
- Types of Clustering
- KMeans Clustering

Random Forest

Random forest (or random forests) is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees.

- The term came from random decision forests that was first proposed by **Tin Kam Ho** of Bell Labs in 1995.
- The method combines **Breiman's** "bagging" idea and the random selection of features.

Features and Advantages

The advantages of random forest are:

- It is one of the most accurate learning algorithms available. For many data sets, it produces a highly accurate classifier.
- It runs efficiently on large databases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.

[Telecom Churn (https://raw.githubusercontent.com/AP-State-Skill-Development-Corporation/Datasets/master/Classification/Orange_Telecom_Churn_Data.csv)](https://raw.githubusercontent.com/AP-State-Skill-Development-Corporation/Datasets/master/Classification/Orange_Telecom_Churn_Data.csv)

In [2]:

```
import sklearn

sklearn.__version__
```

Out[2]:

```
'0.23.1'
```

In [13]:

```
import pandas as pd
import numpy as np

url = 'https://raw.githubusercontent.com/AP-State-Skill-Development-Corporation/Datasets/ma
data = pd.read_csv(url)
```

In [8]:

```
data.shape
```

Out[8]:

```
(5000, 21)
```

In [7]:

```
data.head().T
```

Out[7]:

	0	1	2	3	4
state	KS	OH	NJ	OH	OK
account_length	128	107	137	84	75
area_code	415	415	415	408	415
phone_number	382-4657	371-7191	358-1921	375-9999	330-6626
intl_plan	no	no	no	yes	yes
voice_mail_plan	yes	yes	no	no	no
number_vmail_messages	25	26	0	0	0
total_day_minutes	265.1	161.6	243.4	299.4	166.7
total_day_calls	110	123	114	71	113
total_day_charge	45.07	27.47	41.38	50.9	28.34
total_eve_minutes	197.4	195.5	121.2	61.9	148.3
total_eve_calls	99	103	110	88	122
total_eve_charge	16.78	16.62	10.3	5.26	12.61
total_night_minutes	244.7	254.4	162.6	196.9	186.9
total_night_calls	91	103	104	89	121
total_night_charge	11.01	11.45	7.32	8.86	8.41
total_intl_minutes	10	13.7	12.2	6.6	10.1
total_intl_calls	3	3	5	7	3
total_intl_charge	2.7	3.7	3.29	1.78	2.73
number_customer_service_calls	1	1	0	2	3
churned	False	False	False	False	False

In [5]:

```
data.churned.value_counts()
```

Out[5]:

```
False    4293
True      707
Name: churned, dtype: int64
```

In [10]:

```
data.dtypes
```

Out[10]:

```
state                           object
account_length                   int64
area_code                        int64
phone_number                    object
intl_plan                       object
voice_mail_plan                 object
number_vmail_messages            int64
total_day_minutes              float64
total_day_calls                  int64
total_day_charge               float64
total_eve_minutes              float64
total_eve_calls                  int64
total_eve_charge               float64
total_night_minutes            float64
total_night_calls                int64
total_night_charge             float64
total_intl_minutes             float64
total_intl_calls                 int64
total_intl_charge              float64
number_customer_service_calls    int64
churned                           bool
dtype: object
```

In [11]:

```
data.columns
```

Out[11]:

```
Index(['state', 'account_length', 'area_code', 'phone_number', 'intl_plan',
       'voice_mail_plan', 'number_vmail_messages', 'total_day_minutes',
       'total_day_calls', 'total_day_charge', 'total_eve_minutes',
       'total_eve_calls', 'total_eve_charge', 'total_night_minutes',
       'total_night_calls', 'total_night_charge', 'total_intl_minutes',
       'total_intl_calls', 'total_intl_charge',
       'number_customer_service_calls', 'churned'],
      dtype='object')
```

In [14]:

```
data.drop(['state', 'area_code', 'phone_number'], inplace = True, axis = 'columns')

data.head()
```

Out[14]:

	account_length	intl_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total
0	128	no	yes	25	265.1	
1	107	no	yes	26	161.6	
2	137	no	no	0	243.4	
3	84	yes	no	0	299.4	
4	75	yes	no	0	166.7	

del data[['state', 'area_code', 'phone_number']]

In [18]:

```
for i in ['intl_plan','voice_mail_plan']:
    data[i] = data[i].replace({'yes':True, 'no': False}).astype(bool)

data.head()
```

Out[18]:

	account_length	intl_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total
0	128	False	True	25	265.1	
1	107	False	True	26	161.6	
2	137	False	False	0	243.4	
3	84	True	False	0	299.4	
4	75	True	False	0	166.7	

In [19]:

```
X = data.drop('churned', axis = 'columns')
Y = data['churned']
```

In [20]:

```
from sklearn.model_selection import train_test_split


x_tr, x_tt, y_tr, y_tt = train_test_split(X,Y, test_size = 0.3, random_state = 42)
```

In [21]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [38]:

```
model = RandomForestClassifier(n_estimators=150, n_jobs = -1, random_state = 42)
```

In [40]:

```
model.fit(x_tr, y_tr)
```

Out[40]:

```
RandomForestClassifier(n_estimators=150, n_jobs=-1, random_state=42)
```

In [41]:

```
pred_tt = model.predict(x_tt)
pred_tr = model.predict(x_tr)
```

In [42]:

```
from sklearn.metrics import confusion_matrix, accuracy_score

confusion_matrix(pred_tr, y_tr)
```

Out[42]:

```
array([[3000,    0],
       [   0,  500]], dtype=int64)
```

In [43]:

```
confusion_matrix(pred_tt, y_tt)
```

Out[43]:

```
array([[1284,   48],
       [   9,  159]], dtype=int64)
```

In [44]:

```
y_tt.shape
```

Out[44]:

```
(1500,)
```

In [45]:

```
accuracy_score(pred_tt, y_tt)
```

Out[45]:

```
0.962
```

In [46]:

```
imp = model.feature_importances_
```

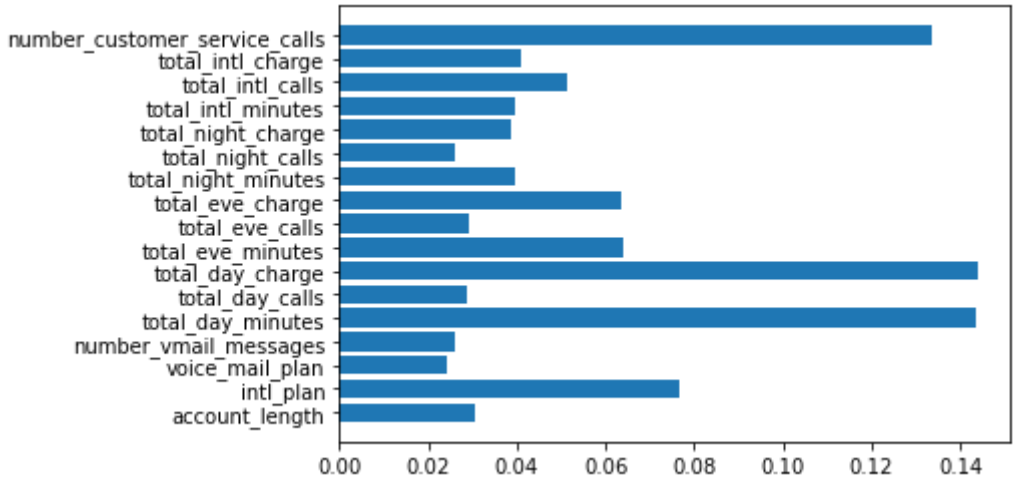
In [48]:

```
import matplotlib.pyplot as plt


plt.barh(X.columns, imp)
```

Out[48]:

```
<BarContainer object of 17 artists>
```



Introduction

There are many models for **clustering** out there. In this notebook, we will be presenting the model that is considered one of the simplest models amongst them. Despite its simplicity, the **K-means** is vastly used for clustering in many data science applications, especially useful if you need to quickly discover insights from **unlabeled data**. In this notebook, you will learn how to use k-Means for customer segmentation.

Some real-world applications of k-means:

- Customer segmentation
- Understand what the visitors of a website are trying to accomplish
- Pattern recognition
- Machine learning
- Data compression

Using k-means for customer segmentation

KMeans_Clustering Visualization (https://www.naftaliharris.com/blog/visualizing-k-means-clustering/)

Customer Dataset (https://raw.githubusercontent.com/AP-State-Skill-Development-Corporation/Datasets/master/Clustering/Cust_Segmentation.csv)

In [69]:

```
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/AP-State-Skill-Development-Corporation/

df.head()
```

Out[69]:

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	Address	DebtIncomeRati
0	1	41	2	6	19	0.124	1.073	0.0	NBA001	6.
1	2	47	1	26	100	4.582	8.218	0.0	NBA021	12.
2	3	33	2	10	57	6.111	5.802	1.0	NBA013	20.
3	4	29	2	4	19	0.681	0.516	0.0	NBA009	6.
4	5	47	1	31	253	9.308	8.908	0.0	NBA008	7.

In [70]:

```
df.drop(['Customer Id', 'Address', 'Defaulted'], axis = 'columns', inplace = True)
```

In [71]:

```
df.head()
```

Out[71]:

	Age	Edu	Years Employed	Income	Card Debt	Other Debt	DebtIncomeRatio
0	41	2	6	19	0.124	1.073	6.3
1	47	1	26	100	4.582	8.218	12.8
2	33	2	10	57	6.111	5.802	20.9
3	29	2	4	19	0.681	0.516	6.3
4	47	1	31	253	9.308	8.908	7.2

In [67]:

```
df.isnull().sum()
```

Out[67]:

```
Age                  0
Edu                  0
Years Employed       0
Income               0
Card Debt            0
Other Debt           0
Defaulted          150
DebtIncomeRatio      0
dtype: int64
```

In [ ]:

```

```

In [72]:

```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()

scaled = ss.fit_transform(df)

scaled
```

Out[72]:

```
array([[ 0.74291541,  0.31212243, -0.37878978, ..., -0.68381116,
        -0.59048916, -0.57652509],
       [ 1.48949049, -0.76634938,  2.5737211 , ...,  1.41447366,
         1.51296181,  0.39138677],
       [-0.25251804,  0.31212243,  0.2117124 , ...,  2.13414111,
         0.80170393,  1.59755385],
       ...,
       [-1.24795149,  2.46906604, -1.26454304, ...,  0.5766659 ,
         0.03863257,  3.45892281],
       [-0.37694723, -0.76634938,  0.50696349, ..., -0.68757659,
        -0.70147601, -1.08281745],
       [ 2.1116364 , -0.76634938,  1.09746566, ...,  0.13611081,
         0.16463355, -0.2340332 ]])
```

In [73]:

```
scaled.mean()
```

Out[73]:

```
-5.642545254565501e-17
```

In [74]:

```
from sklearn.cluster import KMeans
```

In [75]:

```
model = KMeans(n_clusters=3)
```

In [76]:

```
model.fit(scaled)
```

Out[76]:

```
KMeans(n_clusters=3)
```

In [78]:

```
clusters = model.predict(scaled)

clusters
```

```
array([2, 0, 2, 1, 0, 2, 0, 1, 0, 0, 0, 0, 0, 1, 2, 2, 0, 0, 0, 2, 2, 0,
       0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 1, 0, 1, 0, 0, 2, 0, 0, 2, 2,
       2, 1, 0, 2, 2, 2, 0, 2, 0, 2, 2, 2, 2, 2, 1, 2, 0, 0, 0, 0, 1, 0,
       2, 0, 0, 2, 1, 0, 0, 0, 1, 0, 0, 0, 2, 0, 2, 2, 1, 2, 0, 0, 2, 2,
       0, 0, 0, 2, 0, 0, 0, 0, 0, 2, 1, 2, 2, 0, 0, 0, 0, 1, 2, 1, 0, 0,
       0, 2, 0, 0, 0, 0, 0, 0, 0, 1, 2, 2, 0, 2, 0, 2, 0, 0, 0, 2, 1, 0,
       1, 0, 2, 1, 0, 0, 1, 2, 2, 2, 0, 0, 1, 2, 0, 0, 1, 0, 0, 0, 1, 2,
       2, 2, 0, 1, 2, 0, 2, 2, 2, 2, 1, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 1,
       0, 0, 0, 0, 2, 0, 2, 1, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 2, 0,
       0, 0, 2, 1, 0, 0, 0, 1, 2, 2, 2, 0, 0, 1, 2, 0, 0, 0, 1, 2, 0, 0,
       2, 0, 2, 0, 0, 0, 0, 1, 2, 2, 0, 0, 0, 1, 1, 0, 0, 0, 2, 0, 0, 2,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 2, 2, 2, 0, 2, 2, 2, 2, 0, 2, 0,
       1, 1, 0, 2, 0, 2, 1, 2, 0, 0, 2, 2, 0, 2, 0, 2, 2, 0, 1, 0, 0, 1,
       2, 0, 2, 0, 0, 0, 0, 2, 1, 0, 2, 2, 2, 1, 0, 0, 0, 2, 0, 1, 2, 2,
       0, 0, 0, 0, 2, 2, 2, 2, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       2, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 1, 2, 2, 2, 2, 1, 0, 2, 1,
       0, 2, 0, 0, 1, 0, 2, 1, 2, 0, 2, 0, 0, 0, 1, 2, 0, 0, 2, 1, 2, 0,
       0, 0, 2, 2, 0, 2, 2, 0, 0, 0, 2, 2, 1, 0, 0, 2, 0, 2, 0, 0, 2, 0,
       0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 0, 2, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0,
       0, 2, 0, 0, 1, 0, 1, 0, 1, 1, 2, 0, 0, 2, 2, 1, 0, 0, 2, 0, 0, 1,
```

In [79]:

```
df['cluster'] = clusters
```

In [80]:

```
df.groupby('cluster').mean()
```

Out[80]:

cluster	Age	Edu	Years Employed	Income	Card Debt	Other Debt	DebtIncomeRatio
0	29.826552	1.715203	4.351178	28.027837	0.908141	1.810308	9.893790
1	41.000000	2.118812	15.603960	105.643564	5.456960	9.838455	17.574257
2	41.507092	1.556738	13.024823	56.436170	1.294475	2.758365	7.980496

In [94]:

```
import matplotlib.pyplot as plt

plt.scatter(scaled[:,0], scaled[:,3], c = clusters.astype(float))
plt.xlabel('cluster0')
plt.ylabel('cluster2')
plt.title('cluster0 vs cluster2')
```

Out[94]:

```
Text(0.5, 1.0, 'cluster0 vs cluster2')
```