MIS 637 Data Analytics & Machine Learning

Efficient clustering of customers based on the features of the target customers

Professor: M. Daneshmand

Student: Puneeth Panuganti

K-means Clustering

In [ ]:
```
import pandas as pd import
numpy as np import tensorflow
as tf import matplotlib.pyplot
as plt import matplotlib.cm as
cm import pickle import
datetime
import plotly.express as px import
seaborn as sns
import plotly.graph_objects as go
```

In [ ]:
```
my_drive_path = '/content/gdrive/MyDrive/'
dataset_path = my_drive_path + '/lpetrocelli-retail-banking-demo-data/' onlyfiles =
[f for f in listdir(dataset_path) if isfile(join(dataset_path, f))] onlyfiles
```

In [ ]:
```
df_clients = df_clients_origin.join(df_positions_origin.set_index('client_id'), on='cli
df_clients
```

In [ ]:
```
df_clients[['account_id','sex','age','city']]
```

In [ ]:
```
df_clients_onehot = pd.DataFrame()
df_clients_onehot['account_id']=df_clients['account_id']
```

In [ ]:
```
sex_dict = {'Male':1.0,'Female':0.0}
df_clients_onehot['sex'] = df_clients['sex'].apply(lambda x: sex_dict.get(x))
df_clients_onehot["age"] = df_clients['age']
```

In [ ]:
```
df_clients_onehot["latitude"] = df_clients["latitude"] df_clients_onehot["longitude"] =
df_clients["longitude"]
```

In [ ]:
```
df_clients_onehot
```

In [ ]:
```
df = df_clients_onehot df = df
[['sex','latitude','longitude','age']]
df_clients_normalized=(df-df.mean())/df.std()
df_clients_normalized['account_id']=df_clients_onehot['account_id']
df_clients_normalized
```

In [ ]:
```
# Convert date to python datetime df_transactions
= df_transactions_origin
df_transactions['fulldate'] = pd.to_datetime(df_transactions['fulldate'],format="%Y-
%mdf_transactions['timestamp'] = df_transactions.fulldate.values.astype(np.int64) // 10 *
df_transactions
```

In [ ]:
```
# Grouping by account_id
df_rfm =
df_transactions.groupby(['account_id']).agg({
'fulldate': lambda x: (snapshot_date -
x.max()).days,
        'trans_id': 'count',         'amount':
'sum'})
```

In [ ]:
```
# Rename the columns
df_rfm.rename(columns={'fulldate': 'Recency','trans_id':
'Frequency','amount': 'Monetar df_rfm
```

In [ ]:
```
# Plot RFM distributions
plt.figure(figsize=(12,10))# Plot distribution of R
plt.subplot(3, 1, 1);
sns.histplot(df_rfm['Recency'], kde=True,)# Plot
distribution of plt.subplot(3, 1, 2);
sns.histplot(df_rfm['Frequency'], kde=True)# Plot
distribution of plt.subplot(3, 1, 3);
sns.histplot(df_rfm['Monetary'], kde=True)# Show the
plot plt.show()
```

In [ ]:
```
RFM_norm1 = pd.DataFrame(X_rfm)
RFM_norm1.columns = ['Frequency','Amount','Recency'] RFM_norm1.head()
```

In [ ]:
```
transaction_type_dict = {'Debit':1.0,'Credit':0.0}
df_transactions['type'] = df_transactions['type'].apply(lambda x:
transaction_type_dict df_transactions
```

In [ ]:
```
average_loss = np.average(training_history.history['loss'])
      print("Average test loss: ", round(average_loss , 4))
      plot_model(model, show_shapes=True, show_layer_names=True)
```

In [ ]:

```
# Importing pandas and seaborn libraries for data manipulation and charting
import pandas as pd
```

K-nearest Neighbour

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# Function to clean column names
 def column_name_remove_space(df):
for x in df.columns:         if "
" in x:
            df = df.rename(columns={x:x.replace(" ","_").replace("(","")
                                   .replace(")","").replace(",","_").replace("/","_")}
return df
```

In [ ]:

```
# Check for null values
data.isna().any(axis=0).any()
```

In [ ]:

```
# Checking number of unique values in each column
data.nunique()
```

In [ ]:

```
# Check all the unique values for all the columns having less than 100 unique values
# Avoiding the value prints for those columns which have large number of unique values
 for col in data.columns:
if data[col].nunique()<100:
        print(col, sorted(data[col].unique()), '\n')
```

In [ ]:

```
# Fix all columns data =
column_name_remove_space(data)
data.drop(columns=to_drop,inplace=True)
```

In [ ]:

```
# Check data by printing first few rows data.head()
```

In [ ]:

```
# Check data by printing last few rows data.tail()
```

In [ ]:

```
from sklearn.preprocessing import MinMaxScaler                                                    i
 def get_dummy_data_with_output(dummy_variable_columns, data):
    global dummy_data_file_index
    dummy_data = pd.get_dummies(data, prefix=dummy_variable_columns, columns=dummy_var
dummy_data = column_name_remove_space(dummy_data)
    dummy_data.to_csv('dummy_var_data'+str(dummy_data_file_index)+'.csv', index=False)
y = dummy_data['Churn_Yes']
    dummy_data.drop(columns=['Churn_Yes'], inplace=True)
sc = MinMaxScaler()
    dummy_data.loc[:,to_transform] = sc.fit_transform(dummy_data.loc[:,to_transform])
dummy_data_file_index += 1     return y, dummy_data
```

```
y, X = get_dummy_data_with_output(dummy_variable_columns, data)
```

In [ ]:

```
# Starting with imports

from sklearn.model_selection import train_test_split from sklearn.linear_model import
LogisticRegression from sklearn.metrics import confusion_matrix, accuracy_score,
classification_report from sklearn.neighbors import KNeighborsClassifier from
sklearn.metrics import roc_curve from sklearn.metrics import roc_auc_score
```

In [ ]:

```
# Split the training and test set 7:3
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=.3, random_state=0)
```

In [ ]:

```
# Creating reusable functions to create prediction models and print their performance m

# Function to plot confusion matrix def
confusion_matrix_plot(y_test, y_pred):
    plt.matshow(confusion_matrix(y_test, y_pred))
plt.title('Confusion matrix')     plt.colorbar()
plt.ylabel('True label')     plt.xlabel('Predicted
label')     plt.grid(b=None)     plt.show()

# Function to create KNN model with default parameters
def get_knn_model(y_train,X_train, X_test):     model =
KNeighborsClassifier(n_jobs=-1)     model.fit(X_train,
y_train)     print(model.get_params())
    y_pred = list(map(round, model.predict(X_test)))
arr = np.c_[y_pred, y_test]
    print('\nPrinting predicted and actual values:\n',arr)
print('Confusion Matrix \n',confusion_matrix(y_test, y_pred))
print('Accuracy Score: ',accuracy_score(y_test, y_pred))
confusion_matrix_plot(y_test, y_pred)     return model, arr



# Function to create AUC chart and print AUC score. def
roc(model, X_test, y_test):
    probs = model.predict_proba(X_test)     fpr,
tpr, _ = roc_curve(y_test, probs[:,1])
plt.plot(fpr, tpr, marker='.')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')     plt.show()
    print('AUC: %.3f' % roc_auc_score(y_test, probs[:,1]))
```

```
roc(model_knn, X_test, y_test);
print(classification_report(y_test, y_pred))
```

In [ ]:

In [ ]:

```
# Run knn model
model_knn, arr_knn = get_knn_model(y_train,X_train, X_test);
```