Identifying Users With Similar Buying Habits and Preferences

1. Data Preprocessing

Load data

In [ ]:
```
import pandas as pd

aisles_df = pd.read_csv('aisles.csv')
dept_df = pd.read_csv('departments.csv')
prodorder_prior_df = pd.read_csv('order_products__prior.csv')
productorder_train_df = pd.read_csv('order_products__train.csv')
order_df = pd.read_csv('orders.csv')
product_df = pd.read_csv('products.csv')
```

Merge into one dataframe

Only keep the users that exist in both 'prior' table and train eval set of 'orders' table.

In [ ]:
```
prodname_order_prior = pd.merge(prodorder_prior_df,
                                product_df, how='left', on='product
_id')

userorder_prior_prod = pd.merge(prodname_order_prior,
                                order_df, how='left', on='order_id'
)
```

In [ ]:
```
userorder_prior_prod_inner = pd.merge(userorder_prior_prod,
                                      order_df[order_df['eval_set']
=='train'][['user_id','eval_set']],
                                      how='inner', on='user_id')
```

In [ ]:
```
userorder_prior_prod_inner.head()
```

2. Feature Extraction

The features that will be extracted directly are:

- Mean of order_dow (order placed day of week)
- Mean of order_hour_of_day
- Mean of days_since_prior_order
- Total number of orders made
- Total number of products bought

Then we need another vectorized feature of product name: combine all the products name into one row per user, for word2Vector analysis.

In [ ]:
```
habits_user = userorder_prior_prod_inner[['user_id','order_id',
                                          'product_name','order_dow',
                                          'order_hour_of_day','days_since_
prior_order']]
```

In [ ]:
```
import numpy as np

user_average = habits_user.groupby('user_id')['order_dow',
                                    'order_hour_of_day',
                                    'days_since_prior_order'].agg(np.nanmea
n)

user_order = habits_user.groupby('user_id').order_id.nunique()
user_average['num_of_orders'] = user_order
```

```
prod_num = habits_user.groupby('user_id')['order_id'].agg('count')
user_average['num_of_products'] = prod_num
```

In [ ]:
```
list_of_names = []
for p_name in habits_user.groupby('user_id')['product_name']:
        list_of_names.append(' '.join(p_name[1]))

user_average['product_name'] = list_of_names
```

In [ ]:
```
user_average.head()
```

Extract Vectorized Text Feature: Use PySpark word2Vector

In [ ]:
```
from pyspark.sql import SparkSession
from pyspark.ml.feature import word2Vector

spark = SparkSession.builder.appName("User Habit").getOrCreate()

prodname_df = pd.DataFrame(user_average['product_name'])
```

In [ ]:
```
prodname_df.head()
```

In [ ]:
```
fraction_sample = 0.2
productname_sample_df  = prodname_df.sample(frac = fraction_sample, ran
dom_state=321)
userid_sample  = productname_sample_df.index
print(userid_sample)
```

In [ ]:
```
df_list = []
for row in productname_sample_df['product_name']:
    tuple = (row.split(' '),)
    df_list.append(tuple)

print(len(df_list))
```

In [ ]:
```
N = len(df_list)//100
mod = len(df_list) % 100
doc_df = spark.createDataFrame(df_list[0:100], ["product_name"])

for i in range(1,N):
    doc_df_sub = spark.createDataFrame(df_list[100*i:100*(i+1)], ["prod
uct_name"])
    doc_df = doc_df.union(doc_df_sub)

doc_df_sub = spark.createDataFrame(df_list[100*N:len(df_list)], ["produ
ct_name"])
doc_df = doc_df.union(doc_df_sub)
```

In [ ]:
```
word2Vec = Word2Vec(vectorSize=5, minCount=0, inputCol="product_name",
outputCol="res")
mdl = word2Vec.fit(doc_df)

res = mdl.transform(doc_df)
```

In [ ]:
```
features_vectored = [ ]
for row in res.collect():
    text, vector = row
    features_vectored.append(vector)
```

In [ ]:
```
features_vectored_array=[]
for vectors in features_vectored:
    features_vectored_array.append(vectors.values)
```

In [ ]:
```
column_names = []
for i in range(1,6):
    name = "vectorized_feature_" + str(i)
    column_names.append(name)

features_vectored_df = pd.DataFrame(np.array(features_vectored_array).r
eshape(len(df_list),5),
                    columns = column_names)

features_vectored_df['user_id'] = userid_sample
```

In [ ]:
```
features_vectored_df.head()
```

Combine All Features: Concatenate word2Vector feature with other features into one dataframe

In [ ]:
```
sample_useravg = user_average[user_average.index.isin(userid_sample)]

sample_useravg.reset_index(level=0, inplace=True)

userfeatures_habits = pd.merge(sample_useravg, features_vectored_df, how='inner', on="user_id")

userfeatures_habits.drop('product_name', axis=1, inplace=True)
```

In [ ]:
```
userfeatures_habits.head()
```

3. Cluster Users: PySpark K-Means

PCA: Reduce features to 2-dimensional

In [ ]:
```
from sklearn.decomposition import PCA
userfeatures_habits_only = pd.read_csv('userfeatures_habits_only.csv')
userfeatures_habits = pd.read_csv('userfeatures_habits.csv')
pca = PCA(n_components=2).fit(userfeatures_habits_only)
pca_2d = pca.transform(userfeatures_habits_only)
```

In [ ]:
```
pca_dataframe = pd.DataFrame(pca_2d)

pca_dataframe['user_id'] = userfeatures_habits['user_id']
```

Find the optimal K

Find the optimal number of clusters by calculating the within set sum of squared error (WSSSE). As the number of cluster increases, WSSSE will decrease. The best choice is at the elbow of WSSSE graph.

In [ ]:
```
datapca = sc.textFile("pca_feature_df.txt")
parseddatapca = datapca.map(lambda line: array([float(x) for x in line.
split('\t')]))


def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))
```

In [ ]:
```
WSSSE_listpca = []

K_range = range(5,185,5)
for K in K_range:

    clusters = KMeans.train(parseddatapca, K, maxIterations=10, initial
izationMode="random")

    WSSSE_pca = parseddatapca.map(lambda point: error(point)).reduce(la
mbda x, y: x + y)
    print(" k:"+str(K)+" -- Within Set Sum of Squared Error = " + str(W
SSSE_pca) + "=======")
    WSSSE_listpca.append(WSSSE_pca)
```

In [ ]:
```
WSSSE_datapca = {'K':K_range, "WSSSE": WSSSE_listpca}
WSSSE_pca_dataframe = pd.DataFrame(WSSSE_datapca)
```

In [ ]:
```
import matplotlib.pyplot as plt
fig = plt.figure()
WSSSE_pca_dataframe.plot(x='K', y='WSSSE')
plt.axvline(40,
            color='darkorange', linestyle='dashed', linewidth=2)
plt.xlabel('Clusters')
```

```
plt.title('Within Set Sum of Squared Error of K-Means')
plt.show()
```

The optimal k is usually one where there is an “elbow” in the WSSSE graph. So choose k = 40.

Run K-Means mdl with optimal K=40

In [ ]:
```
k_optimal = 40
clusters = KMeans.train(parseddatapca, k_optimal, maxIterations=10, ini
tializationMode="random")
```

Get the cluster labels

In [ ]:
```
predict_clusters = clusters.predict(parseddatapca)

cluster_res = [ ]
for row in predict_clusters.collect():
    cluster_res.append(row)
```

Get the centers for each user

In [ ]:
```
def GetCenter(point):
    center = clusters.centers[clusters.predict(point)]
    return center

RDDCenter = parseddatapca.map(lambda point: GetCenter(point))

ress_center = [ ]
for row in RDDCenter.collect():
    ress_center.append(row)

ress_center = pd.DataFrame(ress_center,columns=['x','y'])
```

KMeans ress Summary

In [ ]:
```
summary_kmeans = ress_center
summary_kmeans['clusters'] = cluster_res
summary_kmeans['user_id'] = userfeatures_habits['user_id']

summary_kmeans = pd.merge(pca_dataframe, summary_kmeans ,how='inner', o
n='user_id')


summary_kmeans.head()
```

Visualization of Kmeans ress

In [ ]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

fig = plt.figure()


labels_color = summary_kmeans['clusters'].unique()


rgb_values = sns.color_palette("Set2", 40)


map_color = dict(zip(labels_color, rgb_values))


plt.scatter(summary_kmeans['x'], summary_kmeans['y'], c=summary_kmeans[
'clusters'].map(map_color))
plt.title("Centers for K-Means Clusters")
plt.show()
```

In [ ]:
```
import numpy as np
import seaborn as sns
```

```
import matplotlib.pyplot as plt

fig = plt.figure()

labels_color = summary_kmeans['clusters'].unique()

rgb_values = sns.color_palette("Set2", 40)

map_color = dict(zip(labels_color, rgb_values))


plt.scatter(summary_kmeans[0], summary_kmeans[1], c=summary_kmeans['clu
sters'].map(map_color), s = 0.5)
plt.title("K-Means Clusters")
plt.show()
```

Most Popular Products in Each User Cluster

In [ ]:
```
import pandas as pd
summary_kmeans = pd.read_csv("../output/summary_kmeans.csv")

cluster_order_info = pd.merge(summary_kmeans, userorder_prior_prod_inne
r, how='left', on='user_id')
prod_cluster = cluster_order_info[['user_id','clusters','product_name'
]]
```

In [ ]:
```
count_cluster = prod_cluster.groupby(['clusters','product_name']).agg(
'count')

top_prods = count_cluster['user_id'].groupby(level=0, group_keys=False)
.nlargest(10).reset_index()
```

In [ ]:
```
import matplotlib.pyplot as plt

top_prods[top_prods['clusters'] == 0][['product_name','user_id']]
```

```
In [ ]: widetop_products =top_prods.pivot(index='clusters', columns='product_na
me', values='user_id').fillna(0)
widetop_products_percent = widetop_products.div(widetop_products.sum(ax
is=0), axis=1)
longtop_products = widetop_products_percent.unstack().reset_index()
longtop_products.columns.values[2]='count'
```

```
In [ ]: fig, (ax1, ax2, ax3, ax4) = plt.subplots(4,figsize=(20,40),sharey=True)


grp1 = []
for i in range(10):
    grp1.append(i)
ax1.plot(widetop_products_percent.loc[grp1].transpose())
ax1.legend(widetop_products_percent.transpose().columns[0:10],title="Cl
uster ID",loc='upper right',prop={'size': 12})
ax1.set_title('Percent of Products in Each Cluster',size=20)


grp2 = []
for i in range(10,20):
    grp2.append(i)
ax2.plot(widetop_products_percent.loc[grp2].transpose())
ax2.legend(widetop_products_percent.transpose().columns[10:20], title=
"Cluster ID",loc='upper right',prop={'size': 12})


grp3 = []
for i in range(20,30):
    grp3.append(i)
ax3.plot(widetop_products_percent.loc[grp3].transpose())
ax3.legend(widetop_products_percent.transpose().columns[20:30],title="C
luster ID",loc='upper right',prop={'size': 12})


grp4 = []
for i in range(30,40):
    grp4.append(i)
ax4.plot(widetop_products_percent.loc[grp4].transpose())
```

```
ax4.legend(widetop_products_percent.transpose().columns[30:40],title="C
luster ID",loc='upper right',prop={'size': 12})



for ax in fig.axes:
    plt.sca(ax)
    plt.xticks(rotation=90, size=12)

plt.subplots_adjust(wspace=0, hspace=0.7)

plt.show()
fig.set_dpi(300)
fig.savefig('prod_cluster_frequency.png')
```