H&M Recommender System

Introduction

H&M Group is a clothing business with 53 online markets and approximately 4,850 stores. They are concerned that customers might not quickly find what interests them or what they are looking for, and ultimately, they might not make a purchase. They want to enhance the shopping experience and help customers make the right choices. They think they can also reduce transportation emissions if they reduce customer returns. H&M want product recommendations based on data from previous transactions, as well as from customer and product meta data.

Submissions will be evaluated according to the Mean Average Precision @ 12 (MAP@12). They will make purchase predictions for all customer_id values provided, regardless of whether these customers made purchases in the training data. They explain that customers who did not make any purchase during test period are excluded from the scoring. There is no penalty for using the full 12 predictions for a customer that ordered fewer than 12 items. They encourage to make 12 predictions for each customer.

For each customer (customer_id), H&M want a prediction of up to 12 products (article_ids), which is the predicted items a customer will buy in the next 7-day period after the training time period.

H&M are expecting about 1,371,980 prediction rows, we will only have 1,362,281 because 9,699 customers have not purchased anything yet. We will have to make predictions on these customers without knowing their transaction history. We could look for customers with similar demographics but this may be computationally expensive and a big assumption, that similar demographics purchase similar products.

Importing the libraries

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
#used during data exploration and model evaluation
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
```

In [3]:
```
#working with datetime feature
from datetime import datetime
```

In [4]:
```
#handling missing values where not dropped
from sklearn.impute import SimpleImputer
from sklearn import preprocessing
```

In [5]:
```
#for evaluating our model
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score
```

In [6]:
```
#for dimension reduction
from sklearn.pipeline import Pipeline # to sequence training events
from sklearn.decomposition import TruncatedSVD
#from sklearn.decomposition import PCA
```

In [7]:
```
#model
from sklearn.neighbors import KNeighborsClassifier
```

In [8]:
```
#used to provide information to the user when running this notebook
from IPython.display import display, clear_output
```

Importing the dataset

In [12]:
```
#get transaction data
transactions_train_df = pd.read_csv("data/transactions_train.csv") # import the transactions dataset
```

In [13]:
```
#get product meta data
articles_df = pd.read_csv("data/articles.csv")
```

In [14]:
```
#get customer meta data
customers_df = pd.read_csv("data/customers.csv")
```

Exploratory Data Analysis & Dataset Preparation

In this section we first looked at what data was available, it's distribution, what was missing and what opportunities were available to reduce the number of features or dimensions in our dataset. Secondly we determined what models could work well with the data, finally we looked to fix any missing values or encoding categorical variables where needed.

An exploritory data analysis was conducted already by various other kaggle contestants such as (Karpov, 2022). Their analysis was reviewed as part of this workbook in order to reduce this EDA section and allow us to focus on model building, prediction and evaluation.

Data available consisted of images of every product, detailed metadata of every product, detailed metadata of every customer and purchase details for customers who bought products. These will be refered to as images,articles,customers and transactions respectively. Although it is assumed that images are an important part of how customers decide on the products they purchase, due to the data size and limited processing power, they will not be used here.

In [18]:
```
transactions_train_df.head(3)
```

Out[18]:

	t_dat	customer_id	article_id	price	sales_channel_id
0	2018-09-20	000058a12d5b43e67d225668fa1f8d618c13dc232df0ca...	663713001	0.050831	2
1	2018-09-20	000058a12d5b43e67d225668fa1f8d618c13dc232df0ca...	541518023	0.030492	2
2	2018-09-20	00007d2de826758b65a93dd24ce629ed66842531df6699...	505221004	0.015237	2

In [13]:
```
transactions_train_df.nunique()
```

Out[13]:
```
t_dat                 734
customer_id       1362281
article_id         104547
price                9857
sales_channel_id        2
dtype: int64
```

In [14]:
```
customers_df.head(3)
```

Out[14]:

	customer_id	FN	Active	club_member_status	fashion_news_frequency	age	
0	00000dbacae5abe5e23885899a1fa44253a17956c6d1c3...	NaN	NaN	ACTIVE	NONE	49.0	52043ee2162cf5aa7ee79974281641c6f
1	0000423b00ade91418cceaf3b26c6af3dd342b51fd051e...	NaN	NaN	ACTIVE	NONE	25.0	2973abc54daa8a5f8ccfe9362140c632
2	000058a12d5b43e67d225668fa1f8d618c13dc232df0ca...	NaN	NaN	ACTIVE	NONE	24.0	64f17e6a330a85798e4998f62d0930d14

In [15]:
```
customers_df.nunique()
```

Out[15]:
```
customer_id               1371980
FN                              1
Active                          1
club_member_status              3
fashion_news_frequency          4
age                            84
postal_code                352899
dtype: int64
```

In [16]:
```
articles_df.head(3)
```

Out[16]:

	article_id	product_code	prod_name	product_type_no	product_type_name	product_group_name	graphical_appearance_no	graphical_appearance_name
0	108775015	108775	Strap top	253	Vest top	Garment Upper body	1010016	Solid
1	108775044	108775	Strap top	253	Vest top	Garment Upper body	1010016	Solid
2	108775051	108775	Strap top (1)	253	Vest top	Garment Upper body	1010017	Stripe

3 rows × 25 columns

In [17]:
```
articles_df.nunique()
```

Out[17]:

```
article_id                    105542
product_code                   47224
prod_name                      45875
product_type_no                  132
product_type_name                131
product_group_name                19
graphical_appearance_no           30
graphical_appearance_name         30
colour_group_code                 50
colour_group_name                 50
perceived_colour_value_id          8
perceived_colour_value_name        8
perceived_colour_master_id        20
perceived_colour_master_name      20
department_no                    299
department_name                  250
index_code                        10
index_name                        10
index_group_no                     5
index_group_name                   5
section_no                        57
section_name                      56
garment_group_no                  21
garment_group_name                21
detail_desc                    43404
dtype: int64
```

There are over 31.9 million transactions and over 3gb in size. With our limited space and processing power, this made working with the dataset slow and unweidly. Instead we were only able to sample this dataset.

For the customer's dataset we have 1,37 million customers from 352,899 locations, it is assumed that FN stands for whether h&m have the customers is signed up for fashion news. Several other features are also available such as post code. We will have to convert NaNs into zero values, we will do the same for Active. For Fashion news frequency we will have to encode these orginal categories. This might also determine customer quality for recommendation.

105,542 products are in the articles dataset. Regarding columns in this dataset, every item had a unique identifier called the article id, but it also had a product code and a product name. The identifier and the product code were not the same, it was assumed that this was due to size differences or colour variations in H&M's clothing (e.g a v-neck polo shirt could be in a small, medium and large, as well as having two colours, black and white).

The product name could probably be dropped later as the product code and name seem to match. This seems to be true for the product type name, colour group name and graphical appearance name. We could drop the names and keep the numbers. We will however keep product group name as there doesn't seem to be a corisponding type_no. We will have to encode this ourselves.

Making Recommendations

In a perfect scenario for a recommendation system we would have a table of m users by n items, with each product given a rating r_ij by each user. However We could have hundreds of users and thousands of products. A user may not have tried every product so our table would have missing values. To solve this issue we would need to predict the value for missing cells (rhat_ij). A good prediction would mean a good recommendation to the user. Another way to make recommendations to users would be to rank the top k products for each user. This would be based on information we have available on products and users. In essence the prediction problem boils down to how we rate products.

In order for us to rate products we would first need some metric to rate them by. Secondly, we would need to decide the prerequisites that a product must meet to recieve said rating. Thirdly we would then need to calculate the score for each item that satisfies the prerequisites and finally we would output a list of items in decreasing order. Unfortunately, H&M have not provided any labelling for us. W+e must make our own. This makes the problem of recommendation more difficult.

In our Transaction dataset we have customers who bought products at a particular price and time. We started here, with these features to try create a simple collaborative model recommendation system. The model tried to learn from a customer's historical purchases and make predictions about their future purchases.

Since we don't actual have a customer item ratings like a 1 to 5 rating per item, we will assume qty of purchase indicates customer interest in products. If we have outliers they may bias our data. In this case we can assume that 68% of customer transaction will lie within 1 standard deviation from the mean so we could take anything 3 standard deviations from the mean as rare events (1%) and remove them.

We will pick a random customer with a few recent transactions and try to predict their future buying habits based on past data about them. We will configure a dataset of purchase made by the customer in the past. We will then use the meta data of the customer and the products to form a model we can use to predict if the customer will by a certain product or not.

Prepare the Transaction Data

In [18]:
```
#First we will convert our date text into a panda date type.
transactions_train_df["t_dat"] = pd.to_datetime(transactions_train_df["t_dat"])
```

In [19]:
```
#we convert articles to string instead of default int.
transactions_train_df['article_id'] = transactions_train_df['article_id'].values.astype(str)
```

In [20]:
```
# we want to see distributions and std dev
transactions_train_df.describe()
```

Out[20]:

	price	sales_channel_id
count	3.178832e+07	3.178832e+07
mean	2.782927e-02	1.704028e+00
std	1.918113e-02	4.564786e-01
min	1.694915e-05	1.000000e+00
25%	1.581356e-02	1.000000e+00
50%	2.540678e-02	2.000000e+00
75%	3.388136e-02	2.000000e+00
max	5.915254e-01	2.000000e+00

We will use a 3 week date range between 2020-09-8 and 2020-09-22 to reduce our dataset size.

In [21]:
```
mask = (transactions_train_df['t_dat'] >= '2020-09-01') & (transactions_train_df['t_dat'] <= '2020-09-22')
```

In [22]:
```
features_df = transactions_train_df.loc[mask]
features_df['customer_id'].size
```

Out[22]:
```
798269
```

In [23]:
```
features_df = features_df[['article_id','customer_id', 't_dat', 'price', 'sales_channel_id']]
```

Prepare Product Data

In [24]:
```
#we convert articles to string instead of default int.
articles_df['article_id'] = articles_df['article_id'].values.astype(str)
```

In [25]:
```
#merge product meta data with transactions
features_df = features_df.merge(articles_df, left_on='article_id', right_on='article_id')
```

In [26]:
```
features_df.columns
```

Out[26]:
```
Index(['article_id', 'customer_id', 't_dat', 'price', 'sales_channel_id',
       'product_code', 'prod_name', 'product_type_no', 'product_type_name',
       'product_group_name', 'graphical_appearance_no',
       'graphical_appearance_name', 'colour_group_code', 'colour_group_name',
       'perceived_colour_value_id', 'perceived_colour_value_name',
       'perceived_colour_master_id', 'perceived_colour_master_name',
       'department_no', 'department_name', 'index_code', 'index_name',
       'index_group_no', 'index_group_name', 'section_no', 'section_name',
       'garment_group_no', 'garment_group_name', 'detail_desc'],
      dtype='object')
```

Regarding columns in the articles data set, every item had a unique identifier called the article id, but it also had a product code and a product name. The identifier and the product code were not the same, it was assumed that this was due to size differences or colour variations in H&M's clothing (e.g a v-neck polo shirt could be in a small, medium and large, as well as having two colours, black and white).

The product name could probably be dropped as the product code and name seem to match. This seems to be true for the product type name, colour group name and graphical appearance name. We could drop the names and keep the numbers. We will however keep product_group_name as there doesn't seem to be a corisponding type_no. We will have to encode this ourselves.

There was no missing data so we did not need to do anything like imputing missing data with sklearn SimpleImputer

In [27]:
```
features_df.drop(['prod_name',
                  'product_type_name',
                  'graphical_appearance_name',
                  'colour_group_name',
                  'perceived_colour_value_name',
                  'perceived_colour_master_name',
                  'department_name',
                  'index_name',
                  'index_group_name',
                  'section_name',
                  'garment_group_name',
                  'detail_desc'], axis=1)
```

Out[27]:

	article_id	customer_id	t_dat	price	sales_channel_id	product_code	product_type_no	product_group_
0	777148006	0001d44dbe7f6c4b35200abdb052c77a87596fe1bdcc37...	2020-09-01	0.013542	1	777148	252	Garment Upper
1	777148006	5ac5e1825104ed5fe3333e75b9337eebc4b45ad761056b...	2020-09-03	0.013542	1	777148	252	Garment Upper
2	777148006	0dcf3023ea1992a78a1fcc769b6befc956f7308186496d...	2020-09-06	0.013542	1	777148	252	Garment Upper
3	777148006	28b30893bbe946358103760387e3dcd09fdb7b077a942f...	2020-09-06	0.042356	2	777148	252	Garment Upper
4	777148006	278f23c7fac720c2b96b25455d640860bdfa8bb3c867cf...	2020-09-10	0.013542	1	777148	252	Garment Upper
...	...	...	...	...	...	...	...	
798264	737994021	f71529889de7a28df0015fad0a043941ecc98883286ef0...	2020-09-22	0.030492	1	737994	273	Garment Lower
798265	533261032	f79e372e21c1359dfebc7da0bf7f321d55e47b3275c351...	2020-09-22	0.033881	2	533261	256	Garment Upper
798266	865792012	f82c91decd5f9abd0a7a72eae0d4911b00ed4f5b4f04f9...	2020-09-22	0.008458	2	865792	273	Garment Lower
798267	772659001	f96661e9e56449885d4c4b90d3227e4abea5e0d2382e2d...	2020-09-22	0.016932	1	772659	274	Garment Lower
798268	807775001	fd5ce8716faf00f6a83616f609e0403ac516727d4ca4aa...	2020-09-22	0.033881	2	807775	272	Garment Lower

798269 rows × 17 columns

Prepare Customer Data

In [28]:
```
#merge customer meta data with transactions
features_df = features_df.merge(customers_df, left_on='customer_id', right_on='customer_id')
```

In [29]:
```
#drop post code as it is similar to customer_id
```

```
features_df = features_df.drop(['postal_code'], axis=1)
```

In [30]:
```
#we reorganise columns
features_df = features_df[['customer_id',#the customer
                           'FN',#customer meta data
                           'Active',
                           'club_member_status',
                           'fashion_news_frequency',
                           'age',
                           'product_code',#product meta data
                           'product_type_no',
                           'product_group_name',
                           'graphical_appearance_no',
                           'colour_group_code',
                           'perceived_colour_value_id',
                           'perceived_colour_master_id',
                           'department_no',
                           'index_code',
                           'index_group_no',
                           'section_no',
                           'garment_group_no',
                           't_dat',#transaction meta data
                           'price',
                           'sales_channel_id',
                           'article_id']]#the product
```

In [31]:
```
#convert from objects and floats to categories and ints
features_df['club_member_status'] = features_df['club_member_status'].astype('category')
features_df['fashion_news_frequency'] = features_df['fashion_news_frequency'].astype('category')
```

In [32]:
```
#check for missing values
def find_missing(df):
    missing = df.isnull().sum() # ref: https://stackoverflow.com/questions/59694988/python-pandas-dataframe-find-missing-values
    print(df.shape)
    print(missing)
```

In [33]:
```
find_missing(features_df)
```

```
(798269, 22)
customer_id                     0
FN                         443296
Active                     448465
club_member_status           1358
fashion_news_frequency       1752
age                          2931
product_code                    0
product_type_no                 0
product_group_name              0
graphical_appearance_no         0
colour_group_code               0
perceived_colour_value_id       0
perceived_colour_master_id      0
department_no                   0
index_code                      0
index_group_no                  0
section_no                      0
garment_group_no                0
t_dat                           0
price                           0
sales_channel_id                0
article_id                      0
dtype: int64
```

In [34]:
```
features_df.iloc[:, 8:-13].values
```

Out[34]:
```
array([['Garment Upper body'],
       ['Garment Upper body'],
       ['Garment Lower body'],
       ...,
       ['Accessories'],
       ['Garment Full body'],
       ['Garment Upper body']], dtype=object)
```

In [35]:
```
features_df.head()
```

Out[35]:

	customer_id	FN	Active	club_member_status	fashion_news_frequency	age	product_code	product_type_no	pro
0	0001d44dbe7f6c4b35200abdb052c77a87596fe1bdcc37...	1.0	1.0	ACTIVE	Regularly	44.0	777148	252	Ga
1	0001d44dbe7f6c4b35200abdb052c77a87596fe1bdcc37...	1.0	1.0	ACTIVE	Regularly	44.0	835801	252	Ga
2	0001d44dbe7f6c4b35200abdb052c77a87596fe1bdcc37...	1.0	1.0	ACTIVE	Regularly	44.0	923134	272	Ga
3	0001d44dbe7f6c4b35200abdb052c77a87596fe1bdcc37...	1.0	1.0	ACTIVE	Regularly	44.0	865929	254	Ga
4	0001d44dbe7f6c4b35200abdb052c77a87596fe1bdcc37...	1.0	1.0	ACTIVE	Regularly	44.0	935858	252	Ga

5 rows × 22 columns

In [36]:

```
features_df['FN'] = features_df['FN'].fillna(0)
features_df['Active'] = features_df['Active'].fillna(0)

club_member_status = features_df.iloc[:, 3:-18].values
fashion_news_frequency = features_df.iloc[:, 4:-17].values
age = features_df.iloc[:, 5:-16].values

#ref: https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html
imputer_med = SimpleImputer(missing_values=np.nan, strategy='median')
imputer_mf = SimpleImputer(missing_values=np.nan, strategy='most_frequent')

#we replace missing values with the most frequent
imputer_mf.fit(club_member_status)
club_member_status = imputer_mf.transform(club_member_status)

imputer_mf.fit(fashion_news_frequency)
fashion_news_frequency = imputer_mf.transform(fashion_news_frequency)

#we replace any missing age values with the median age
imputer_med.fit(age)
age = imputer_med.transform(age)

#now add corrected columns back into our main customer dataframe
```

```
features_df.iloc[:, 3:-18] = club_member_status
features_df.iloc[:, 4:-17] = fashion_news_frequency
features_df.iloc[:, 5:-16] = age

#replace minus sign in text and check result of dataset after imputing missing values
features_df.columns = features_df.columns.str.replace('-', '')

#lower case columns
features_df.columns = map(str.lower, features_df.columns)

find_missing(features_df)
```

```
(798269, 22)
customer_id                   0
fn                            0
active                        0
club_member_status            0
fashion_news_frequency        0
age                           0
product_code                  0
product_type_no               0
product_group_name            0
graphical_appearance_no       0
colour_group_code             0
perceived_colour_value_id     0
perceived_colour_master_id    0
department_no                 0
index_code                    0
index_group_no                0
section_no                    0
garment_group_no              0
t_dat                         0
price                         0
sales_channel_id              0
article_id                    0
dtype: int64
```

In [37]:
```
#We will encode and scale after we have split our data
features_df.tail()
```

Out[37]:

	customer_id	fn	active	club_member_status	fashion_news_frequency	age	product_code	product_type_no
798264	d5013b57392ac330a87fdf6c04d439594bfb8776afe035...	0.0	0.0	ACTIVE	NONE	34.0	828321	265
798265	d98a24c79ecfe9e1e52f9ff5d0ffc4f84740c317591530...	1.0	1.0	ACTIVE	Regularly	75.0	818890	76
798266	d98a24c79ecfe9e1e52f9ff5d0ffc4f84740c317591530...	1.0	1.0	ACTIVE	Regularly	75.0	818890	76
798267	e991c3fcb6730496d8ba1c521121d55fb1dfd0ab98b748...	1.0	1.0	ACTIVE	Regularly	21.0	930405	265
798268	f1b9cf466441305d09034354ccbb6f18faf9deaa99b85b...	0.0	0.0	ACTIVE	NONE	28.0	790006	262

5 rows × 22 columns

Split Data Into Train & Test Set

We take the past 2 weeks as training data and 1 week in the future as test data

In [38]:
```
train_mask = (features_df['t_dat'] >= '2020-09-01') & (features_df['t_dat'] <= '2020-09-14')
train_df = features_df.loc[train_mask]
train_df['customer_id'].size
```

Out[38]:
```
531905
```

In [39]:
```
test_mask = (features_df['t_dat'] >= '2020-09-15') & (features_df['t_dat'] <= '2020-09-22')
test_df = features_df.loc[test_mask]
test_df['customer_id'].size
```

Out[39]:
```
266364
```

Encode Data (After Timesplit)

In [40]:
```
#we now encode any categorical variables in our training and testing data
le = preprocessing.LabelEncoder()
```

```
train_df.iloc[:,3] = le.fit_transform(train_df.iloc[:,3])#club_member_status
train_df.iloc[:,4] = le.fit_transform(train_df.iloc[:,4])#fashion_news_frequency
train_df.iloc[:,8] = le.fit_transform(train_df.iloc[:,8])#product_group_name
train_df.iloc[:,14] = le.fit_transform(train_df.iloc[:,14])#index_code

test_df.iloc[:,3] = le.fit_transform(test_df.iloc[:,3])#club_member_status
test_df.iloc[:,4] = le.fit_transform(test_df.iloc[:,4])#fashion_news_frequency
test_df.iloc[:,8] = le.fit_transform(test_df.iloc[:,8])#product_group_name
test_df.iloc[:,14] = le.fit_transform(test_df.iloc[:,14])#index_code
```

```
C:\Users\newlo\AppData\Local\Temp\ipykernel_3704\2122401799.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
s-a-copy
  train_df.iloc[:,3] = le.fit_transform(train_df.iloc[:,3])#club_member_status
C:\Users\newlo\AppData\Local\Temp\ipykernel_3704\2122401799.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
s-a-copy
  train_df.iloc[:,4] = le.fit_transform(train_df.iloc[:,4])#fashion_news_frequency
C:\Users\newlo\AppData\Local\Temp\ipykernel_3704\2122401799.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
s-a-copy
  train_df.iloc[:,8] = le.fit_transform(train_df.iloc[:,8])#product_group_name
C:\Users\newlo\AppData\Local\Temp\ipykernel_3704\2122401799.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
s-a-copy
  train_df.iloc[:,14] = le.fit_transform(train_df.iloc[:,14])#index_code
C:\Users\newlo\AppData\Local\Temp\ipykernel_3704\2122401799.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
s-a-copy
  test_df.iloc[:,3] = le.fit_transform(test_df.iloc[:,3])#club_member_status
C:\Users\newlo\AppData\Local\Temp\ipykernel_3704\2122401799.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
s-a-copy
  test_df.iloc[:,4] = le.fit_transform(test_df.iloc[:,4])#fashion_news_frequency
C:\Users\newlo\AppData\Local\Temp\ipykernel_3704\2122401799.py:10: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df.iloc[:,8] = le.fit_transform(test_df.iloc[:,8])#product_group_name
C:\Users\newlo\AppData\Local\Temp\ipykernel_3704\2122401799.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df.iloc[:,14] = le.fit_transform(test_df.iloc[:,14])#index_code
```

In [41]:
```
#encode date as ordinal after we split our training and test data
train_df['t_dat'] = train_df['t_dat'].apply(lambda x: x.toordinal())
#reverse encoding. convert from ordinal to date
#features_df['t_dat'] = features_df['t_dat'].apply(lambda x: datetime.fromordinal(x))
```

```
C:\Users\newlo\AppData\Local\Temp\ipykernel_3704\592797208.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df['t_dat'] = train_df['t_dat'].apply(lambda x: x.toordinal())
```

In [42]:
```
#encode date as ordinal after we split our training and test data
test_df['t_dat'] = test_df['t_dat'].apply(lambda x: x.toordinal())
```

```
C:\Users\newlo\AppData\Local\Temp\ipykernel_3704\1173125632.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['t_dat'] = test_df['t_dat'].apply(lambda x: x.toordinal())
```

In [43]:
```
train_df.head(1)
```

Out[43]:

	customer_id	fn	active	club_member_status	fashion_news_frequency	age	product_code	product_type_no	proc
0	0001d44dbe7f6c4b35200abdb052c77a87596fe1bdcc37...	1.0	1.0	0	2	44.0	777148	252	

1 rows × 22 columns

In [44]:
```
test_df.head(1)
```

Out[44]:

	customer_id	fn	active	club_member_status	fashion_news_frequency	age	product_code	product_type_no	proc
35	4078d35f7b2ae7a56cfdaec8c42959bf28f1f7ed742ab6...	0.0	0.0	0	1	25.0	835801	252	

1 rows × 22 columns

Create Feature & Target Matrix

In [45]:
```
#These are the attributes of our customers
X_train = train_df.iloc[:, 1: 20].values
X_train.shape
```

Out[45]:
```
(531905, 19)
```

In [46]:
```
# this is the product or in our case the class
y_train = train_df.iloc[:, 21].values
y_train.shape
```

Out[46]:
```
(531905,)
```

In [47]:
```
# these are future customers
X_test = test_df.iloc[:, 1: 20].values
X_test.shape
```

Out[47]:
```
(266364, 19)
```

In [48]:
```
# these are future products
y_test = test_df.iloc[:, 21].values
```

```
y_test.shape
```

Out[48]:
```
(266364,)
```

Feature Scaling: MinMax - range (0,1)

In [49]:
```
min_max_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
```

In [50]:
```
X_train_scaled = min_max_scaler.fit_transform(X_train)
X_train_scaled
```

Out[50]:
```
array([[1.        , 1.        , 0.        , ..., 0.08333333, 0.        ,
        0.02590795],
       [1.        , 1.        , 0.        , ..., 0.08333333, 0.        ,
        0.03594979],
       [1.        , 1.        , 0.        , ..., 0.16666667, 0.        ,
        0.02423431],
       ...,
       [0.        , 0.        , 0.        , ..., 0.16666667, 1.        ,
        0.07062762],
       [0.        , 0.        , 0.        , ..., 0.41666667, 1.        ,
        0.04933891],
       [0.        , 0.        , 0.        , ..., 0.79166667, 1.        ,
        0.06607531]])
```

In [ ]:
```
np.savetxt("data/X_train_scaled_sample.csv", X_train_scaled, delimiter=",")
```

In [ ]:
```
np.savetxt("data/y_test_scaled_sample.csv", y_train, delimiter=",")
```

Model Building

Training the K-NN model on our Training set

In [97]:
```
#steps = [('svd', TruncatedSVD(n_components=15)), ('knn', hm_clf)]
# n_neighbors: number of neighbors. Default is 5
# metric="minkowski", p=2: will calculate distance as eucledian distance formula

#steps = [('pca', PCA(n_components = 0.95)), ('knn', KNeighborsClassifier(n_neighbors=5, metric="minkowski", p=2))]
```

```
#steps = [('svd', TruncatedSVD(n_components=15)), ('knn', KNeighborsClassifier(n_neighbors=5, metric="minkowski", p=2))]
#steps = [('svd', TruncatedSVD(n_components=15)), ('knn', KNeighborsClassifier(n_neighbors=4, metric="minkowski", p=2))]
steps = [('svd', TruncatedSVD(n_components=15)), ('knn', KNeighborsClassifier(n_neighbors=3, metric="minkowski", p=2))]
model = Pipeline(steps=steps)
```

In [98]:
```
model.fit(X_train_scaled, y_train)
```

Out[98]:
```
Pipeline(steps=[('svd', TruncatedSVD(n_components=15)),
                ('knn', KNeighborsClassifier(n_neighbors=3))])
```

Predicting a new result

In [85]:
```
#print(classifier.predict(sc.transform([[0001d44dbe7f6c4b35200abdb052c77a87596fe1bdcc37, 30,2020-09-08]])))
```

Predicting the Test set results

In [86]:
```
y_pred = model.predict(X_test_scaled)
```

In [80]:
```
y_pred # here we see the model predict products for the test data.
```

Out[80]:
```
array(['811235004', '897817004', '711053003', ..., '904734001',
       '919273004', '775825001'], dtype=object)
```

In [99]:
```
len(y_pred)
```

Out[99]:
```
266364
```

Model Evaluation

In [87]:
```
print("Accuracy of H&M KNN Classifier:", accuracy_score(y_test, y_pred))
```

```
Accuracy of H&M KNN Classifier: 0.4259922512051178
```

In [88]:
```
print("Precision Score for H&M KNN Classifier:", precision_score(y_test, y_pred, average='macro'))
```

```
Precision Score for H&M KNN Classifier: 0.17221084804834588
```

```
d:\projects\tudublin\tu060\machine learning\venv\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarnin
g: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control thi
s behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Results & Discussion

Results

TEST 1: H&M KNN Classifier Model Accuracy (k=5) was 0.00497 and Precision Score was 0.001, both less than 1% accurate.

- We used the last 3 weeks in September 2020 from our transaction dataset to be used later for training and testing data.
- We used Customer attributes, the full date range (YYYY-MM-DD) and transaction attributes as features to predict products.
- We filled in missing data in our customer attributes using SimpleImputer to impute most frequent categories and median was used to impute missing ages.
- We split our data up by time instead of random assignment. This meant 2 weeks (in the past) were used for training and 1 week (in the future) was used for testing testing
- We encoded our categorical data using the LabelEncoder and date feature by converting the date into an ordinal number. This was after the training/test split.
- We created our training and testing datasets and scaled them using MinMax range (0 to 1)
- To build the KNN Classifier model, we used k=5 and our metric was minkowski p2, This meant we calculated the distance using eucledian distance formula.
- To evaluate the model, we used accuracy as a measuring score. Which compared our predicted products to what was in the test dataset.

TEST 2: H&M KNN Classifier Model (k=5) Accuracy increased to 0.432 and Precision Score increased to 0.161:

- We included product attributes in with Customer attributes, the full date range (YYYY-MM-DD) and transaction attributes as features to predict products. This can lead to curse of dimensionality. The KNN Classifier can perform poorly with too many features. In our case the results were better but the time it took to process the data won't scale to the full transaction file.
- For the next test we should reduce the amount of features we have. To do this we must use dimension reduction techniques such as principle component analysis (when we have lots of features) or singular value decomposition (SVD) when we have sparse data.

TEST 3: H&M KNN Classifier Model (k=5) Accuracy decreased to 0.432 and Precision Score increased to 0.161:

- we tried PCA with 95% variance kept. It didn't improve our score or our speed. We will try SVD in the next text

TEST 4: H&M KNN Classifier Model (k=5) Accuracy decreased to 0.422 and Precision Score increased to 0.151:

- SVD reduced our features down to 15 and also reduced our accuracy but sped up processing, we will try PCA to compare

TEST 5: H&M KNN Classifier Model (k=5) Accuracy decreased to 0.038 and Precision Score increased to 0.0144:

- We removed customer and tried to predict customer age with products as a test. Speed increased dramatically. The idea behind this would be to use age to select customer_id per product.

TEST 6: H&M KNN Classifier Model (k=4) Accuracy decreased to 0.415 and Precision Score increased to 0.1601:

- We tried k=4

TEST 7: H&M KNN Classifier Model (k=3) Accuracy decreased to 0.426 and Precision Score increased to 0.1722:

- We tried k=3

Discussion

Overall Test 2 was the best. We tried in Test 3 to introduce PCA but it did not improve our result. We did however use SVD to reduce our number of features to 15. This reduced our score but significantly increased our processing time. This trade off was deemed acceptable to progress to a full train and test with Kaggle.

References

- https://www.kaggle.com/code/martandsay/knn-multi-classification-animal-classification/notebook
- https://towardsdatascience.com/multiclass-classification-using-k-nearest-neighbours-ca5281a9ef76
- https://analyticsindiamag.com/singular-value-decomposition-svd-application-recommender-system/
- https://machinelearningmastery.com/singular-value-decomposition-for-dimensionality-reduction-in-python/
- https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60
- https://www.kaggle.com/code/lichtlab/h-m-data-deep-dive-chap-1-understand-article
- https://www.kaggle.com/code/vanguarde/h-m-eda-first-look

- https://www.kaggle.com/code/debarshichanda/understanding-mean-average-precision
- https://www.kaggle.com/code/paweljankiewicz/hm-create-dataset-samples
- https://www.kaggle.com/code/souamesannis/tips-to-work-efficiently-with-the-dataset

Generate Kaggle Predictions File

In [1]:
```
#working with files and memory management
import gc

#working with data
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

#working with datetime feature
from datetime import datetime

#handling missing values where not dropped
from sklearn.impute import SimpleImputer
from sklearn import preprocessing

#for evaluating our model
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score

#for dimension reduction
from sklearn.pipeline import Pipeline # to sequence training events
from sklearn.decomposition import TruncatedSVD

#model
from sklearn.neighbors import KNeighborsClassifier

#used to provide information to the user when running this notebook
from IPython.display import display, clear_output
```

In [2]:
```
#H&M Collaborative KNN Model Based Recommendation System
def HmRecSys_data_prep():

    #GET DATA
    #output message for user
    clear_output(wait=True)
```

```
display('Importing Data. Please wait...')

#get transaction data
transactions_train_df = pd.read_csv("data/transactions_train.csv",
                                    dtype={"article_id": "str"}) # import the transactions dataset

#get product meta data
articles_df = pd.read_csv("data/articles.csv", dtype={"article_id": "str"})

#get customer meta data
customers_df = pd.read_csv("data/customers.csv")

#output message for user
clear_output(wait=True)
display('Preparing Data. Please wait...')

#PREPARE DATA
#prepare transactions dataset
features_df = transactions_train_df[['article_id',
                                     'customer_id',
                                     't_dat',
                                     'price',
                                     'sales_channel_id']]

del transactions_train_df
gc.collect()

clear_output(wait=True)
display('imported transactions and arranged columns...')

#First we will convert our date text into a panda date type.
features_df["t_dat"] = pd.to_datetime(features_df["t_dat"])

clear_output(wait=True)
display('converted date into datetime object...')

clear_output(wait=True)
display('converted article_ids to strings...')

#merge product meta data with transactions
features_df = features_df.merge(articles_df, left_on='article_id', right_on='article_id')

del articles_df
gc.collect()
```

```
clear_output(wait=True)
display('merged articles with transactions...')

#we drop cols we don't need from products dataset
features_df.drop(['prod_name',
                  'product_type_name',
                  'graphical_appearance_name',
                  'colour_group_name',
                  'perceived_colour_value_name',
                  'perceived_colour_master_name',
                  'department_name',
                  'index_name',
                  'index_group_name',
                  'section_name',
                  'garment_group_name',
                  'detail_desc'], axis=1)

clear_output(wait=True)
display('rearranged columns of features dataset for merger...')

clear_output(wait=True)
display('merging customers with features dataset...')

#merge customer meta data with transactions
features_df = features_df.merge(customers_df, left_on='customer_id', right_on='customer_id')

del customers_df
gc.collect()

clear_output(wait=True)
display('rearranging customers and articles with feature dataset...')

#we reorganise columns
features_df = features_df[['customer_id',#the customer
                           'FN',#customer meta data
                           'Active',
                           'club_member_status',
                           'fashion_news_frequency',
                           'age',
                           'product_code',#product meta data
                           'product_type_no',
                           'product_group_name',
                           'graphical_appearance_no',
```

```
                            'colour_group_code',
                            'perceived_colour_value_id',
                            'perceived_colour_master_id',
                            'department_no',
                            'index_code',
                            'index_group_no',
                            'section_no',
                            'garment_group_no',
                            't_dat',#transaction meta data
                            'price',
                            'sales_channel_id',
                            'article_id']]#the product
clear_output(wait=True)
display('rearranged columns of features dataset...')

#FIX MISSING DATA
#convert from objects and floats to categories and ints
features_df['club_member_status'] = features_df['club_member_status'].astype('category')
features_df['fashion_news_frequency'] = features_df['fashion_news_frequency'].astype('category')

features_df['FN'] = features_df['FN'].fillna(0)
features_df['Active'] = features_df['Active'].fillna(0)

club_member_status = features_df.iloc[:, 3:-18].values
fashion_news_frequency = features_df.iloc[:, 4:-17].values
age = features_df.iloc[:, 5:-16].values

#ref: https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html
imputer_med = SimpleImputer(missing_values=np.nan, strategy='median')
imputer_mf = SimpleImputer(missing_values=np.nan, strategy='most_frequent')

#we replace missing values with the most frequent
imputer_mf.fit(club_member_status)
club_member_status = imputer_mf.transform(club_member_status)

imputer_mf.fit(fashion_news_frequency)
fashion_news_frequency = imputer_mf.transform(fashion_news_frequency)

#we replace any missing age values with the median age
imputer_med.fit(age)
age = imputer_med.transform(age)

#now add corrected columns back into our main customer dataframe
features_df.iloc[:, 3:-18] = club_member_status
```

```
features_df.iloc[:, 4:-17] = fashion_news_frequency
features_df.iloc[:, 5:-16] = age

#replace minus sign in text and check result of dataset after imputing missing values
features_df.columns = features_df.columns.str.replace('-', '')

#lower case columns
features_df.columns = map(str.lower, features_df.columns)

clear_output(wait=True)
display('filled in missing values and fixed column names...')

# ENCODE DATA
#encode our categorical variables
le = preprocessing.LabelEncoder()
features_df.iloc[:,3] = le.fit_transform(features_df.iloc[:,3])#club_member_status
features_df.iloc[:,4] = le.fit_transform(features_df.iloc[:,4])#fashion_news_frequency
features_df.iloc[:,8] = le.fit_transform(features_df.iloc[:,8])#product_group_name
features_df.iloc[:,14] = le.fit_transform(features_df.iloc[:,14])#index_code

#encode date as ordinal after we split our training and test data
features_df['t_dat'] = features_df['t_dat'].apply(lambda x: x.toordinal())

clear_output(wait=True)
display('encoded features...')

# SPLIT DATA X FEATURES Y PREDICTOR
#These are the attributes of our customers
X_train = features_df.iloc[:, 1: 20].values

# this is the product or in our case the class
y_train = features_df.iloc[:, 21].values

del features_df
gc.collect()

clear_output(wait=True)
display('split data into X features and y predictor...')

# SCALE FEATURES: MinMax - range (0,1)
min_max_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))

X_train_scaled = min_max_scaler.fit_transform(X_train)
```

```
    clear_output(wait=True)
    display('features scaling complete...')

    clear_output(wait=True)
    display('exporting X_train_scaled np array to text file...')

    np.savetxt("data/X_train_scaled.csv", X_train_scaled, delimiter=",")

    clear_output(wait=True)
    display('exporting y_train np array to text file...')
    np.savetxt("data/y_train.csv", y_train, delimiter=",", fmt='%s')

    del X_train
    del y_train
    del X_train_scaled
    gc.collect()

    clear_output(wait=True)
    display('Data preparation complete.')
```

In [3]:

```
def HmRecSys_model_train():

    #output message for user
    clear_output(wait=True)
    display('Importing X_train np array from text file. Please wait...')

    X_train_scaled = np.loadtxt('data/X_train_scaled.csv', delimiter=',')

    #output message for user
    clear_output(wait=True)
    display('Importing y_train df from text file. Please wait...')

    #get product meta data
    y_train = np.loadtxt('data/y_train.csv', delimiter=',')

    # CREATE PIPELINE
    #create model pipeline
    clear_output(wait=True)
    display('Creating pipeline...')

    steps = [('svd', TruncatedSVD(n_components=15)),
             ('knn', KNeighborsClassifier(n_neighbors=3, metric="minkowski", p=2))]
```

```
    model = Pipeline(steps=steps)

    #output message for user
    clear_output(wait=True)
    display('training model please wait...')

    #TRAIN MODEL
    model.fit(X_train_scaled, y_train)

    #output message for user
    clear_output(wait=True)
    display('model trained...')
```

In [4]:

```
def HmRecSys_model_predict():

    #GET DATA
    #output message for user
    clear_output(wait=True)
    display('Importing np array from text file. Please wait...')

    X_train_scaled = np.loadtxt('data/X_test_scaled.csv', delimiter=',')


    #output message for user
    clear_output(wait=True)
    display('Importing Data. Please wait...')

    #get product meta data
    articles_df = pd.read_csv("data/articles.csv", dtype={"article_id": "str"})

    #get customer meta data
    customers_df = pd.read_csv("data/customers.csv")

    #get transaction data
    transactions_train_df = pd.read_csv("data/transactions_train.csv",
                                        dtype={"article_id": "str"}) # import the transactions dataset

    clear_output(wait=True)
    display('Getting price mode')

    #get popular price to pay
    p = t_df['price'].mode()
```

```
    clear_output(wait=True)
    display('Getting sales mode')

    #get popular sales channel to buy from
    s = t_df['sales_channel_id'].mode()

    del t_df
    gc.collect()

    clear_output(wait=True)
    display('Getting dates for next 7 days')

    #predict in next 7 days (2020-09-29)
    date = {'date': ['2020-09-29']}
    d_df = pd.DataFrame(date)
    d_df['date'] = pd.to_datetime(df['date'], format='%Y-%m%-d')
    d_df['date'] = d_df['date'].apply(lambda x: x.toordinal())
    d = d_df['date'].iloc[:].values

    del d_df
    gc.collect()

    #output message for user
    clear_output(wait=True)
    display('opening CSV file to start writing...')

    write_file = "ros_predictions4.csv"
    with open(write_file, "wt", encoding="utf-8") as output:
        #add headers first
        output.write("customer_id,prediction" + '\n')

        #now we loop through each row and write predictions to csv file
        for index_i, cus in customers_df.iterrows():

            #we keep trying different products until we make a hit then we add it to the list
            for index_j, art in articles_df.iterrows():
                #get their meta data
                features_df = [cus['FN'],
                               cus['Active'],
                               cus['club_member_status'],
                               cus['fashion_news_frequency'],
                               cus['age'],
                               art['product_code'],
                               art['product_type_no'],
```

```
                                art['product_group_name'],
                                art['graphical_appearance_no'],
                                art['colour_group_code'],
                                art['perceived_colour_value_id'],
                                art['perceived_colour_master_id'],
                                art['department_no'],
                                art['index_code'],
                                art['index_group_no'],
                                art['section_no'],
                                art['garment_group_no'],
                                d,
                                p,
                                s]

                    #normalise data to query customer
                    q_cus_scaled = min_max_scaler.fit_transform(features_df)

                    #free up memory
                    del features_df
                    gc.collect()

                    #make a prediction
                    y_pred = model.predict(q_cus_scaled)
                    result.append(y_pred)

                    #create prediction csv file
                    r = []
                    r.append(cus.customer_id + ",")
                    for n in result:
                        p = names.iloc[n]
                        r.append("0" + str(p))
                        prediction =  ' '.join(r)
                    #write predictions to csv file
                    output.write(prediction + '\n')
                    clear_output(wait=True)
                    display('Predicting: ' + str(index_i) + ", " + str(index_j))
```

In [5]:
```
HmRecSys_data_prep()
```

```
'features scaling complete...'
```

```
[[0.         0.         0.         ... 0.66666667 0.         0.08590504]
 [0.         0.         0.         ... 0.66666667 0.00545703 0.08590504]
 [0.         0.         0.         ... 0.66666667 0.         0.0515201 ]
 ...
 [0.         0.         0.         ... 0.375      1.         0.1002321 ]
 [0.         0.         0.         ... 0.83333333 1.         0.01283704]
 [1.         1.         0.         ... 0.25       1.         0.10309751]]
```

In [6]:
```
HmRecSys_model_train()
```

```
'training model please wait...'
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
Input In [6], in <cell line: 1>()
----> 1 HmRecSys_model_train()

Input In [3], in HmRecSys_model_train()
     26 display('training model please wait...')
     28 #TRAIN MODEL
---> 29 model.fit(X_train_scaled, y_train)
     31 #output message for user
     32 clear_output(wait=True)

File d:\projects\tudublin\tu060\machine learning\venv\lib\site-packages\sklearn\pipeline.py:390, in Pipeline.fit(self, X, y, **fit
_params)
    364 """Fit the model.
    365
    366 Fit all the transformers one after the other and transform the
   (...)
    387     Pipeline with fitted steps.
    388 """
    389 fit_params_steps = self._check_fit_params(**fit_params)
--> 390 Xt = self._fit(X, y, **fit_params_steps)
    391 with _print_elapsed_time("Pipeline", self._log_message(len(self.steps) - 1)):
    392     if self._final_estimator != "passthrough":

File d:\projects\tudublin\tu060\machine learning\venv\lib\site-packages\sklearn\pipeline.py:348, in Pipeline._fit(self, X, y, **fi
t_params_steps)
    346     cloned_transformer = clone(transformer)
    347 # Fit or load from cache the current transformer
--> 348 X, fitted_transformer = fit_transform_one_cached(
    349     cloned_transformer,
    350     X,
    351     y,
    352     None,
    353     message_clsname="Pipeline",
    354     message=self._log_message(step_idx),
    355     **fit_params_steps[name],
    356 )
    357 # Replace the transformer of the step with the fitted
    358 # transformer. This is necessary when loading the transformer
    359 # from the cache.
    360 self.steps[step_idx] = (name, fitted_transformer)
```

```
File d:\projects\tudublin\tu060\machine learning\venv\lib\site-packages\joblib\memory.py:349, in NotMemorizedFunc.__call__(self, *
args, **kwargs)
    348 def __call__ (self, *args, **kwargs):
--> 349     return self.func(*args, **kwargs)

File d:\projects\tudublin\tu060\machine learning\venv\lib\site-packages\sklearn\pipeline.py:893, in _fit_transform_one(transforme
r, X, y, weight, message_clsname, message, **fit_params)
    891 with _print_elapsed_time(message_clsname, message):
    892     if hasattr(transformer, "fit_transform"):
--> 893         res = transformer.fit_transform(X, y, **fit_params)
    894     else:
    895         res = transformer.fit(X, y, **fit_params).transform(X)

File d:\projects\tudublin\tu060\machine learning\venv\lib\site-packages\sklearn\decomposition\_truncated_svd.py:210, in TruncatedS
VD.fit_transform(self, X, y)
    206     if k >= n_features:
    207         raise ValueError(
    208             "n_components must be < n_features; got %d >= %d" % (k, n_features)
    209         )
--> 210     U, Sigma, VT = randomized_svd(
    211         X, self.n_components, n_iter=self.n_iter, random_state=random_state
    212     )
    213 else:
    214     raise ValueError("unknown algorithm %r" % self.algorithm)

File d:\projects\tudublin\tu060\machine learning\venv\lib\site-packages\sklearn\utils\extmath.py:395, in randomized_svd(M, n_compo
nents, n_oversamples, n_iter, power_iteration_normalizer, transpose, flip_sign, random_state)
    391 if transpose:
    392     # this implementation is a bit faster with smaller shape[1]
    393     M = M.T
--> 395 Q = randomized_range_finder(
    396     M,
    397     size=n_random,
    398     n_iter=n_iter,
    399     power_iteration_normalizer=power_iteration_normalizer,
    400     random_state=random_state,
    401 )
    403 # project M to the (k + p) dimensional space using the basis vectors
    404 B = safe_sparse_dot(Q.T, M)

File d:\projects\tudublin\tu060\machine learning\venv\lib\site-packages\sklearn\utils\extmath.py:237, in randomized_range_finder
(A, size, n_iter, power_iteration_normalizer, random_state)
    235     Q = safe_sparse_dot(A.T, Q)
    236 elif power_iteration_normalizer == "LU":
```

```
--> 237     Q, _ = linalg.lu(safe_sparse_dot(A, Q), permute_l=True)
    238     Q, _ = linalg.lu(safe_sparse_dot(A.T, Q), permute_l=True)
    239 elif power_iteration_normalizer == "QR":

File d:\projects\tudublin\tu060\machine learning\venv\lib\site-packages\scipy\linalg\_decomp_lu.py:216, in lu(a, permute_l, overwrite_a, check_finite)
    214 overwrite_a = overwrite_a or (_datacopied(a1, a))
    215 flu, = get_flinalg_funcs(('lu',), (a1,))
--> 216 p, l, u, info = flu(a1, permute_l=permute_l, overwrite_a=overwrite_a)
    217 if info < 0:
    218     raise ValueError('illegal value in %dth argument of '
    219                      'internal lu.getrf' % -info)

MemoryError: Unable to allocate 5.92 GiB for an array with shape (31788324, 25) and data type float64
```