

# The H&M Customers preference prediction By Using Linear Support Vector Classification AND K-Nearest Neighbours

Mina Jamshidian D20124995

[Project Github Repository](#)

The purpose of this project is to predict what articles each customer will purchase in seven days after training data ends for the H & M store with a Machine Learning Algorithm. [Kaggle](#) provided data for this project, which consists of three files; a customers.csv file with information about each customer, an articles.csv file with information about each product for sale, and a transactions\_train.csv file with a summary of each customer's purchases. The articles.csv file contains 105,542 rows and 25 features consisting of information about each article's attributes, about half the features have the same value. In customers.csv, there are 1,371,980 rows, and 6 features containing information about customers. The transactions\_train.csv file contains 31,788,324 rows and 5 features that contain each transaction's information before training data end by customer\_id and articles\_id act like a foreign key for previous datasets.

For this project, the scikit-learn approach for choosing algorithms (Choosing the Right Estimator, 2022) for selecting algorithms was used in order to choose the most appropriate machine learning algorithm for predicting the article\_ids for customers. Based on the scikit-learn approach, two algorithms were selected for this project: Linear SVC and KNN.

In order to verify the model's performance, two approaches were considered for data partitioning:

1. The first one, a training data set and a test data set was prepared using 70% and 30% of the rows by using train\_test\_split function from model\_selection of sklearn.
2. For the second one, the model's performance is determined by dividing the final data by the last seven days, that means all seven days for test data, and all days before seven days for training data.

This project involved the import, exploration, preprocessing, and development of the features, and then the machine learning models were applied to solve the classification problem using the scikit-learn approach. To achieve the objective of this project, Linear Support Vector algorithms have been applied to build the models and to perform hyperparameter tuning using the Grid Search for K-Nearest Neighbours algorithm to determine the optimal parameters. A number of combinations of hyperparameters were tested based on random selections. A grid search was chosen because it is feasible to use (Malik, 2022) the parameters that are highly specified since overly large values of many hyperparameters are computationally expensive and require a great deal of time to build models.

Finally the accuracy of the models was determined by the best hyperparameters that were detected. After that the most accurate model would be tested by the final dataset for finding all preferences customer's articles\_id. After that, the predicted article\_ids for each customer should be combined by the top 12 article\_ids in the last transaction week for customers who did not predict article\_id, or for customers who predicted articles which were less than 12.

Following are the steps taken in developing an accurate model for predicting what articles all of H&M store's customers will buy after training data end:

# Data Preparation

In accordance with these guidelines, the data preparation was carried out for three datasets (Brownlee, 2020). It is necessary to import different packages which are useful for analysing, processing, and visualising the dataset which exist in the project Notebook in detail, and packages have been loaded. All datasets required for this project have been loaded (transaction\_train, customers and articles data). According to the first exploratory data analysis of transaction data, there were 734 days of transactions by 1,362,281 unique customers and 104,547 unique articles, which means that the target variable contains 104547 unique values.

## Articles Data:

There were several columns that had the same values, so all those columns were removed and this column was ultimately kept for articles data. Finally, I made sure there were no duplicate values or missing values in this dataset.

```
1 articles.nunique()
article_id          105542
product_type_name   131
product_group_name  19
graphical_appearance_name  30
colour_group_name   50
perceived_colour_value_name  8
perceived_colour_master_name  20
index_name          10
index_group_name     5
section_name        56
garment_group_name   21
dtype: int64
```

## Customers Data:

There are two columns in the customer data that were indicated as standing for once value. These two columns have been removed from customer data ('FN', 'Active'). The decision was made to delete the column "postal\_code" for this project since it was not necessary and appeared to be the same as the customer\_id column. After that three columns (club\_member\_status, fashion\_news\_frequency, age) were found that had a number of missing values. In the club\_member\_status column, "nan" values were replaced by "LEFT CLUB". In the fashion\_news\_frequency column, "None", "nan" and "NONE" values were replaced by "None". Due to a number of missing values in the age column, the median age was used to fill in the missing values. As a final step, I checked that this dataset did not contain duplicate values or missing values.

```
1 # Deleting all features have one value
2 # get number of unique values for each column
3 counts = customers.nunique()
4 # record columns to delete
5 to_del = [i for i,v in enumerate(counts) if v == 1]
6 colname = customers.columns[to_del]
7 print (colname)# drop useless columns
8 customers.drop(colname, axis=1, inplace=True)
9 print(customers.shape)
Index(['FN', 'Active'], dtype='object')
(1371980, 5)
```

```
1 #Checking the unique value of club_member_status feature
2 customers['club_member_status'].unique()
array(['ACTIVE', nan, 'PRE-CREATE', 'LEFT CLUB'], dtype=object)
```

```
1 customers.loc[~customers['club_member_status'].isin(['ACTIVE', 'PRE-CREATE']), 'club_member_status'] = 'LEFT CLUB'
```

```
1 customers.isnull().sum()
customer_id          0
club_member_status   6062
fashion_news_frequency  16009
age                  15861
postal_code          0
dtype: int64
```

```
1 # Checking the unique value of fashion_news_frequency feature
2 customers['fashion_news_frequency'].unique()
array(['NONE', 'Regularly', nan, 'Monthly', 'None'], dtype=object)
```

```
1 customers.loc[~customers['fashion_news_frequency'].isin(['Regularly', 'Monthly']), 'fashion_news_frequency'] = 'None'
```

```
customers['age'] = customers['age'].fillna(customers['age'].median())
```

## Transaction Data:

In this dataset there are 31788324 transactions made by 1362281 customers who bought 104547 unique articles, but 2974905 transactions were repeated purchases and as they are real transactions for items that customer bought again in real word should not be deleted. Based on its foundation, this dataset has no missing value. Customers and articles that are the most popular should be identified by knowing the year, month, and the week that their transaction ended.

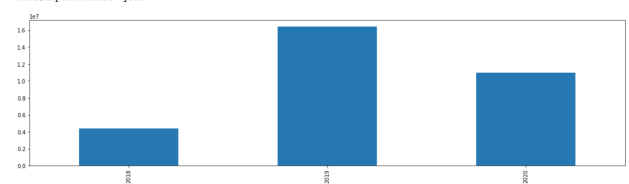
```
1 transactions['year'] = pd.DatetimeIndex(transactions['t_dat']).year
2 transactions['month'] = pd.DatetimeIndex(transactions['t_dat']).month
3 transactions.tail()
```

	t_dat	customer_id	article_id	price	sales_channel_id	year	month
31788319	2020-09-22	ff2282977442e327b45d8c89afde25617d00124d0f999...	929511001	0.059305	2	2020	9
31788320	2020-09-22	ff2282977442e327b45d8c89afde25617d00124d0f999...	891322004	0.042356	2	2020	9
31788321	2020-09-22	ff380805474b287b05cb2a7507b9a013482f7dd0b0c0e...	918325001	0.043203	1	2020	9
31788322	2020-09-22	ff4d3a8b1f3b60af93e78c30a7cb4c75eda2590d3e5...	833459002	0.006763	1	2020	9
31788323	2020-09-22	ffe3b6b73545df065b521e19f64b6f6e93b6d450a2d0...	898573003	0.033881	2	2020	9

```
1 ## Find weeks since each transaction train ended
```

```
1 transactions["t_dat"] = pd.to_datetime(transactions["t_dat"])
2 transactions["week_since"] = (transactions["t_dat"].max() - transactions["t_dat"]).dt.days // 7
3 transactions["week_since"].value_counts()
```

```
1 transactions[['year', 'customer_id']].groupby('year').count().plot(kind='bar', legend=None)
<AxesSubplot: xlabel='year'>
```



## Frequently customers:

The above diagram indicates that customers have more transactions in 2019 and 2020, then it would be necessary to specify which months in two years, customers have more transactions. In the first nine months of 2019 and 2020, customers had more transactions than the last three months.

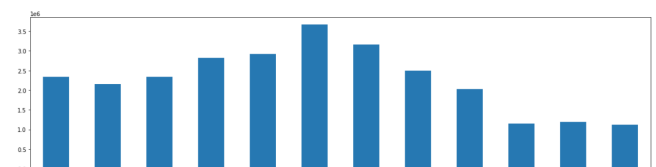
Based on the analysis of the number of transactions each customer had, it was discovered that each customer had transactions 20 times on average, and 75% of customers have transactions below 24 times, while just 25% have transactions above 25 times. The transaction data were limited for customers who have transactions more than 25 times in the more frequently occurring months in 2019 and 2020. Due to a large number of customers in the transaction data, the data was capped at customers who were discovered in the previous step, as It was explained.

```
1 # Limiting cust_Count above 25 count.
2 popularity_threshold = 25
3 fr_cust = freq_customer_count.query('count_cust >= @popularity_threshold')
```

```
1 freq_cust_trans = transactions[transactions['customer_id'].isin(fr_cust['customer_id'])]

1 last_two_year_transactions=transactions[(transactions.year >= 2019)]

1 last_two_year_transactions[(['month', 'customer_id']).groupby('month').count().plot(kind='bar', legend=None)
<AxesSubplot: xlabel='month'>
```



```
1 freq_customer_count['count_cust'].describe().round()
```

```
count      1180621.0
mean        20.0
std         32.0
min          1.0
25%          3.0
50%          9.0
75%         24.0
max        1431.0
Name: count_cust, dtype: float64
```

## Frequently Articles:

The purpose of this project is to predict article\_id for each customer for seven days after training data was finished, and the fashion industry is a very trendy industry and people tend to purchase the latest products, so it was considered to find the most frequently purchased item in the last month in transaction data.

The study found that the average of the articles sold was around 36 times and 75% of articles sold below 24 times so I use `quantile(np.arange(.9, 1, .01))` to identify the total transactions for each article from 90% to 99%. According to my findings, only 5% of articles sold more than 150 times. Because of the large number of articles in the transactions data, it was limited to articles that sold over 150 times in the last month, so it gives me the data I need to predict articles for customers after training data ends. In order to complete the last part of the project, for customers who do not have transactions, I obtained the top 12 articles in the last week before training data ended.

```
1 last_month_article_count['count_art'].describe()
count      28297.000000
mean        36.872813
std        108.455142
min          1.000000
25%          2.000000
50%          6.000000
75%         24.000000
max        2703.000000
Name: count_art, dtype: float64

1 # The median of articles that were bought just two
2 # Let's look at from 90% to 99% to identify the total article_id that was chosen by customer
3
4 print(last_month_article_count['count_art'].quantile(np.arange(.9, 1, .01)))
0.90    83.00
0.91    93.00
0.92   105.00
0.93   120.00
0.94   141.00
0.95   168.00
0.96   204.00
0.97   261.00
0.98   355.00
0.99   522.00
Name: count_art, dtype: float64
```

```
1 freq_art_cust_trans = freq_cust_trans[freq_cust_trans['article_id'].isin(freq_art['article_id'])]
2 freq_art_cust_trans
```

## Merging Three Datasets:

Articles and customers data are limited by more frequent transactions that were discovered, then have been merged by transactions.

```
1 freq_customers=customers[customers['customer_id'].isin(freq_art_cust_trans['customer_id'])]
2 freq_articles=articles[articles['article_id'].isin(freq_art_cust_trans['article_id'])]

1 freq_info = freq_art_cust_trans.merge(freq_customers,on='customer_id').merge(freq_articles,on='article_id')
2 freq_info
```

## Feature Engineering for Final Dataset

- After merging the data, all instances with NA or duplicate rows were dropped.

```
def get_age(age):
    if 16 <= age < 25:
        return "16-24"
    elif 25 <= age < 35:
        return "25-34"
    elif 35 <= age < 45:
        return "35-44"
    elif 45 <= age < 55:
        return "45-54"
    elif 55 <= age < 65:
        return "55-64"
    else:
        return "64+"
```

- The “age” column was binned into six categories and added as a new column to customer data by the name of “age\_range” and then the “age” column was removed from final data. The main motivation of binning was to make the model more robust and prevent overfitting (Team, 2021).
- To extract more details about each article and its price, we added summary statistics for each article\_id and price.
- To extract more details about customers and articles that they bought, the mean of them were calculated.
- In order to prevent losing any information, all categorical features in final datasets were converted to binary numerical by one-hot encoding method due to their non-ordinal nature and inability to be understood by algorithms (Wikipedia contributors, 2021).
- A function called Minmaxscaler() was used to scale some of the features in the final data since some features had different scales. It was used as this function was found to be more accurate (Henderi et al., 2020) for KNN (Brownlee, 2020b).

```
1 freq_info = freq_info.drop(["age"], axis=1)

1 freq_info_price = (freq_info.groupby(["article_id"]).agg({"price": ["min", "max", "mean", "std"]})).reset_index().rename({"price": "art_mean_count_cust"}, axis=1)
2
3 freq_info_price.columns = freq_info_price.columns.map(lambda x: "_".join(x))
4
5 freq_info_price = freq_info_price.fillna(0.0)
```

```
1 freq_info_mean_cus_art = (freq_info.groupby(["customer_id", "article_id"]).size().groupby(["article_id"]).mean()).reset_index().rename({"size": "art_mean_count_cust"}, axis=1)
2
3 freq_info_mean_cus_art
```

article_id	art_mean_count_cust
0	111586001
1	111586001
2	123173001
3	148033001
4	156231001
...	...
1562	937252001
1563	938804001
1564	942187001
1565	944306001
1566	945015001

```
1 #Encoding of categorical columns
2 #Checking unique values and converting them to int using pd.get_dummies()
3 categorical_columns = ['club_member_status', 'fashion_news_frequency', 'age_range', 'product_type_name',
4 'product_group_name', 'graphical_appearance_name', 'colour_group_name',
5 'perceived_colour_value_name', 'perceived_colour_master_name', 'index_name',
6 'index_group_name', 'section_name', 'garment_group_name']
7
8 #Transforming the categorical columns-label encoding
9 final_data_dummy = pd.get_dummies(final_data, columns=categorical_columns)
```

## Data Sampling

As the final dataset contains more than 1 million rows, by using the sample() function (Majumder, 2021) only 10% of the final dataset will be randomly used for the next sections of the project.

```
final_data_dummy_scale_random = final_data_dummy_scale.sample(frac=.1)
final_data_dummy_scale_random
```

```
1 final_data_dummy_scale_random.shape
(180823, 222)
```

## Data Partitioning

Following data sampling, training and test data are extracted from the dataset to develop the predictive model. Due to being uncertain whether test data at that time would be accurate or not, the first step was to split data by skit\_learn function, and then in the next step data was splitted by conditional as train and test data.

1. The final dataset is divided into 70% for training data and 30% for testing data for model evaluation. It was done because the number of training data has to be sufficient to create a reliable model. As a result of this approach, the training data consisted of 126576 rows with 218 and testing data contained 54247 rows. The model was tested with testing data after building it by using training data, and its prediction accuracy was compared with the valid target values. Partitioning the datasets was carried out using random samples, hence the rows indices in the training and test datasets were also random.
2. The model's performance is evaluated by dividing the final seven day which means data was divided by the date after “2020-09-16” for the test data and by the dates before that for the training data.

```
data_train.shape: (126576, 218)
data_test.shape: (54247, 218)
target_train.shape: (126576,)
target_test.shape: (54247, 218)
```

# Model Implementation

Based on the Scikit-Learn approach, two Machine Learning classifiers have been developed for this classification problem. Below is a description of the process of developing, implementing, and validating the classification models by considering of two approaches of data partitioning:

## Linear Support Vector Classifier

The stratified sampling was performed on the training data to transform it to a balanced dataset. A 10-fold stratified cross validation was used to build the linear SVC model and fit to the training data. Following this, the model was tested with test data. Balanced accuracy (*Sklearn.Metrics.Balanced\_accuracy\_score*, 2022) and Matthews correlation coefficient (*Sklearn.Metrics.Matthews\_corrcoef*, 2022) were used to assess the model's performance (Grandini et al., 2020).

### For Train and Test Split (70/30)

The cross-validation technique yielded an average accuracy score of 85.98% after evaluation of the model on the test folds and 4641.19 seconds of build time.

Applying the model to test data showed that the Linear SVC model had the balanced accuracy of 0.74 and the Matthews corrcoref coefficient of 0.85.

```
1 # Performing Stratified 10-fold Cross-Validation
2 # dataset is imbalanced
3 from sklearn.model_selection import KFold, StratifiedKFold
4 import gc
5 from sklearn.svm import LinearSVC
6
7 skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=60)
8 measured = np.zeros((data_train.shape[0]))
9 score = 0
10 # LinearSVC Model Evaluation by Stratified 10-fold Cross-Validation
11 start=time.time()
12 for times, (trn_idx, test_idx) in enumerate(skf.split(data_train.values, target_train.values)):
13     svc_model = LinearSVC(random_state=0)
14     svc_model.fit(data_train.iloc[trn_idx], target_train.iloc[trn_idx])
15     measured[test_idx] = svc_model.predict(data_train.iloc[test_idx])
16     score += svc_model.score(data_train.iloc[test_idx], target_train.iloc[test_idx]) # returns mean accuracy
17     print("Fold: {} score: {}".format(times, svc_model.score(data_train.iloc[test_idx],
18                                                             target_train.iloc[test_idx])))
19     gc.collect()
20
21 end=time.time()
22 print("Stratified 10-fold Cross-Validation for LinearSVM Model in seconds:",(end-start))
23
24 Fold: 0 score: 0.8617475114552062
25 Fold: 1 score: 0.8610364986569758
26 Fold: 2 score: 0.858666455996208
27 Fold: 3 score: 0.857876441775952
28 Fold: 4 score: 0.861505071891294
29 Fold: 5 score: 0.8613525043450783
30 Fold: 6 score: 0.8591293355455479
31 Fold: 7 score: 0.861420557794106
32 Fold: 8 score: 0.8596823891917516
33 Fold: 9 score: 0.8622106344315399
34 Stratified 10-fold Cross-Validation for LinearSVM Model in seconds: 4641.195543050766
```

```
1 # Average Score for model
2 print('Average score', score / skf.n_splits)
3
4 Average score 0.8604632836381494
```

```
1 print("accuracy_score",accuracy_score(target_test,scv_pred, normalise=True)*100)
2 print("balanced_accuracy_score",balanced_accuracy_score(target_test,scv_pred))
3 print("matthews_corrcoref",matthews_corrcoref(target_test,scv_pred))
```

```
accuracy_score 85.98447840433573
balanced_accuracy_score 0.7495195747001503
matthews_corrcoref 0.8595640210504135
```

### For Conditional Train and Test Split by Last Seven Days (2020-09-16)

The cross-validation technique yielded an average accuracy score of 77.37% after evaluation of the model on the test folds and 8520.89 seconds of build time. Applying the model to test data showed that the Linear SVC model had the balanced accuracy of 0.78 and the Matthews corrcoref coefficient of 0.77.

```
1 skf_m = StratifiedKFold(n_splits=10, shuffle=True, random_state=60)
2 measured= np.zeros((X_train_m.shape[0]))
3 score_m = 0
4
5 start=time.time()
6 for times, (trn_idx_m, test_idx_m) in enumerate(skf_m.split(X_train_m.values, y_train_m.values)):
7     svc_model_m = LinearSVC(random_state=0)
8     svc_model_m.fit(X_train_m.iloc[trn_idx_m], y_train_m.iloc[trn_idx_m])
9     measured[test_idx_m] = svc_model_m.predict(X_train_m.iloc[test_idx_m])
10    score_m += svc_model_m.score(X_train_m.iloc[test_idx_m], y_train_m.iloc[test_idx_m])
11    print("Fold: {} score_m: {}".format(times,svc_model_m.score(X_train_m.iloc[test_idx_m],
12                                                                y_train_m.iloc[test_idx_m])))
13    gc.collect()
14
15 end=time.time()
16 print("Stratified 10-fold Cross-Validation for LinearSVM Model in seconds:",(end-start))
17
18 Fold: 0 score: 0.8811748998664887
19 Fold: 1 score: 0.8767801513128616
20 Fold: 2 score: 0.8773920783266578
21 Fold: 3 score: 0.8818424566088118
22 Fold: 4 score: 0.8775589675122385
23 Fold: 5 score: 0.8793947485536271
24 Fold: 6 score: 0.8791722296395194
25 Fold: 7 score: 0.8788940809968847
26 Fold: 8 score: 0.8803377969401948
27 Fold: 9 score: 0.8803377969401948
28 time taken for performing LinearSVM Model Stratified 10-fold Cross-Validation in seconds: 8520.895906209946
```

```
1 print('Average score', score_m / skf_m.n_splits)
```

```
Average score 0.8792877206697479
```

```
1 scv_pred_m=svc_model.predict(X_test_m)
2 scv_pred_m
```

```
array([894668005, 889870001, 754238023, ..., 921226004, 930380001,
       859125001])
```

```
1 print("accuracy_score",accuracy_score(y_test_m,scv_pred_m, normalize=True)*100)
2 print("balanced_accuracy_score",balanced_accuracy_score(y_test_m,scv_pred_m))
3 print("matthews_corrcoref",matthews_corrcoref(y_test_m,scv_pred_m))
```

```
accuracy_score 77.37089201877934
balanced_accuracy_score 0.7815012722646311
matthews_corrcoref 0.7737056774626357
```

## K-Nearest Neighbour Classifier

A hyper-parameter tuning was conducted in order to determine the optimal value of K, which refers to neighbours number, for building a KNN model from the dataset. K values were selected at random from a range of possible values, and then the best value was chosen for the model. This was followed by generating a parameter grid for the range of values, which generated a grid of those values. Grid Search was applied to all values in stratified 10-fold cross validation to determine which hyper-parameter is most suitable for the KNN model.

### For Train and Test Split (70/30)

Grid search operations took 3390.839 seconds to complete. The grid search identified that 1, for k, was the optimal hyperparameter value for the KNN model on the training data in this project and by this hyper-parameter, the highest accuracy score achieved during cross validation was 97.88%. Applying the model to test data showed that the KNN model had the balanced accuracy of 0.93 and the Matthews corrcoeff coefficient of 0.97.

```
1 from sklearn.model_selection import GridSearchCV
2 from sklearn.neighbors import KNeighborsClassifier
3 # Implementing K-Nearest Neighbour
4 # Implementing Grid-Search to determine Hyperparameter k
5 # defining the parameter values that would be searched
6 # Max limit=10 was chosen since the large computation time
7 k_range = list(range(1, 10))
8 # creating a parameter grid: mapping the parameter names to the values that should be searched
9 # By a python dictionary
10 # key: parameter name
11 # value: list of values that should be searched for that parameter
12 # single key-value pair for param_grid
13 param_grid = dict(n_neighbors=k_range)
14 # Creating a knn model
15 knn = KNeighborsClassifier()
16 # using 10-fold cross validation
17 # using gridsearch for testing all values for n_neighbors
18 # For integer and None inputs,
19 # if the estimator is a classifier 'y' is binary or multiclass, stratifiedKFold would be used.
20 # For other types, KFold is used.
21 start=time.time()
22 # Instantiating the grid
23 grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy')
24 end=time.time()
25 print('Gridsearch for knn model in seconds:',(end-start))

time taken for gridsearch for knn model in seconds: 0.0001181831359863281

1 # fitting the grid by data
2 start=time.time()
3 grid.fit(data_train, target_train)
4 end=time.time()
5 print('Fitting knn model with the best parameters in seconds:',(end-start))

Fitting knn model with best parameters in seconds: 3390.839021921158

1 print(grid.best_params_)
({'n_neighbors': 1})

1 # the mean score for the best performing value for "n_neighbors" would be checked
2 # Single best score that was achieved beyond all params
3 print(grid.best_score_)

0.9762751177178612

1 print(grid.best_estimator_)
KNeighborsClassifier(n_neighbors=1)

1 # training the model by using all data and the best parameters
2 # Instantiating the model by best parameter that was found
3 knn_model = KNeighborsClassifier(n_neighbors=1)

1 start=time.time()
2 knn_model.fit(data_train, target_train)
3 end=time.time()
4 print("time taken for fitting knn model in seconds:",(end-start))

time taken for fitting knn model in seconds: 0.046576738357543945

1 # Prediction on the test data
2 pred_knn= knn_model.predict(data_test)

1 print("accuracy_score",accuracy_score(target_test,pred_knn, normalize=True)*100)
2 print("balanced_accuracy_score",balanced_accuracy_score(target_test,pred_knn))
3 print("matthews_corrcoeff",matthews_corrcoeff(target_test,pred_knn))

accuracy_score 97.88559736022269
balanced_accuracy_score 0.9328864959031814
matthews_corrcoeff 0.9788022404637375
```

### For Conditional Train and Test Split by Last Seven Days (2020-09-16)

Grid search operations took 7742.91 seconds to complete. The grid search identified that 1, for k, was the optimal hyperparameter value for the KNN model on the training data in this project and by this hyper-parameter, the highest accuracy score achieved during cross validation was 94.55%. Applying the model to test data showed that the KNN model had the balanced accuracy of 0.93 and the Matthews corrcoeff coefficient of 0.94.

```
1 k_range = list(range(1, 10))
2 param_grid_m = dict(n_neighbors=k_range)
3 knn_m = KNeighborsClassifier()
4 grid_m = GridSearchCV(knn_m, param_grid_m, cv=10, scoring='accuracy')
5
6 start=time.time()
7 grid_m.fit(X_train_m, y_train_m)
8 end=time.time()
9 print("time taken for fitting knn model in seconds:",(end-start))

time taken for fitting knn model in seconds: 7742.913081169128

1 print(grid_m.best_params_)
({'n_neighbors': 1})

1 print(grid_m.best_score_)

0.9840396643832946

1 knn_model_m = KNeighborsClassifier(n_neighbors=1)

1 start=time.time()
2 knn_model_m.fit(X_train_m, y_train_m)
3 end=time.time()
4 print("time taken for fitting knn model in seconds:",(end-start))

time taken for fitting knn model in seconds: 0.0721590518951416

1 predicted_knn_m= knn_model_m.predict(X_test_m)
2 predicted_knn_m[0:12]

array([[894668005, 889870001, 754238023, 852584001, 898918002, 562245106,
        898573003, 903926002, 911214003, 881111001, 768912001, 751471039]])

1 print("accuracy_score",accuracy_score(y_test_m,predicted_knn_m, normalize=True)*100)
2 print("balanced_accuracy_score",balanced_accuracy_score(y_test_m,predicted_knn_m))
3 print("matthews_corrcoeff",matthews_corrcoeff(y_test_m,predicted_knn_m))

accuracy_score 94.55399061032864
balanced_accuracy_score 0.9387786259541986
matthews_corrcoeff 0.9454808746630516
```

## Models Comparison

As part of the solution for this project, two different models were built to predict which items would be purchased by customers in the H&M store seven day after training data ended. Following are the results:



Classifier	Time of Build Model (second)	Accuracy Prediction on Test data
Linear Support Vector Classifier (70/30)	4641.19	85.98 %
Linear Support Vector Classifier (Last 7 days)	8520.89	77.37%
K-Nearest Neighbour Classifier (70/30)	3390.83	97.88%
K-Nearest Neighbour Classifier (Last 7 days)	7742.91	94.55%

For evaluating the performance of the models, time costs were taken into account, as well as the accuracy of the prediction, since this parameter can be used to evaluate a multiclass classification algorithm (Grandini et al., 2020).

It was chosen to represent accuracy despite the imbalance of the dataset as the Stratified 10-fold cross validation was performed, it automatically took into account equal numbers of records per class.

After performing Grid Search, results obtained by the KNN model with k=1 produced the most accurate results in two types of data partitioning based on time of model implantation and accuracy. KNN was the faster and more accurate model builder in comparison to Linear SVC. KNN appears to be the most accurate way of predicting which articles customers will select.

## Model Evaluation After Feature Selection

It was found that KNN represented the best model for predicting the article\_id for customers based on all features in the dataset.

Due to the redundant features of this project problem, the best predictors could explain the majority of the data variance. In turn, this may lead to significant improvements in prediction and model fitting. Feature selection was performed, KNN models were constructed using the most effective features, and then the results of both models were compared. Results were completely different from the first assumption that the time of building model and average accuracy dropped.

Following are the steps to select features and apply them to a KNN model:

- In SelectKBest, the k highest scoring attributes were identified and f\_classif was used to determine the F-value between the feature labels and features for classification. This function is the most appropriate to use since it determines the F-value between the feature labels and features.
- To build the model, the 174 best features were selected from 218 available features, which equates to the best 80%.
- Then, as part of the data preparation process for building the model, the non-selected features were removed from the target data.
- The KNN model was then built in a similar manner to that used previously. The 53.25% accuracy score was obtained from stratified cross validation of training data with best hyperparameters.

```
1 print("accuracy_score", accuracy_score(target_test, pred_knn_feature, normalize=True)*100)
2 print("balanced_accuracy_score", balanced_accuracy_score(target_test, pred_knn_feature))
3 print("matthews_corrcoeff", matthews_corrcoeff(target_test, pred_knn_feature))

accuracy_score 58.016111490036316
balanced_accuracy_score 0.5707964601769911
matthews_corrcoeff 0.5805452820977463
```

## Predicting All Articles for each Customer By KNN

The KNN model was found to be more accurate than Linear SVC in predicting articles, so it was selected to predict all articles for each customer as a result of the fact that the model was tested after a sampling. Hence, for this part of the analysis, the dataset before the data sampling was used to predict which products each customer would buy. Next, all articles that were found were merged into the final dataset, and finally,

the resultant data was grouped by customer\_id with the head of 12 and then all articles for each customer were gathered.

```
1 final_data = final_data_dummy_scale.copy()
2
3 final_model = final_data.loc[:, ~final_data.columns.isin(['t_dat', 'customer_id',
4 'article_id', 'week_since'])]
5
6 result_freq = knn_model.predict(final_model)
7
8 result_freq
9 array([673677002, 673677002, 673677002, ..., 903824003, 903824003,
10 903824003])
11
12 len(result_freq)
13 1808228
14
15 final_data['article'] = result_freq.tolist()
16
17 final_data.head()
```

```
1 cust_l2=final_data.groupby('customer_id').head(12)
2
3 # cust_l2['customer_id'].value_counts()
4
5 cust_l2_clean=cust_l2.drop_duplicates(subset = ['customer_id','article'])
6
7 cust_l2_clean_articles=cust_l2_clean[['customer_id','article']]
8
9 cust_l2_clean_articles['article'] = ' 0'+cust_l2_clean['article'].astype('str')
10
11 result = cust_l2_clean_articles.groupby('customer_id').sum().reset_index()
12 result
```

	customer_id	article
0	0000423b00ade91418ccea3b26c6af3dd342b51fd051e...	0673677002
1	00007d5de826758b65a93dd34c6e29ca66842531d8699...	0372860001 0160442007 0160442010 0730983001
2	00009a94e9ec3e45add5ba56d5210ea898de4b46c685...	0706016006 0610776002 0706016001 0554450027 0...
3	0000b2f1829a23b244eac422ef13df3cccaadac85368e6...	0706016015 0751471001 0572988001 0917606004
4	00001c71aaf65983c3d195c273f7b4d50bbf17761cd91...	0809238001 0841383002 0749699001 0678942032 0...

Lastly, the sample\_submission.csv file was added to the Notebook for collecting the final customer\_id, and the predicted article\_ids which were predicted in the previous step were merged with this data. In the case of customers who did not predict article\_id, or for customers who predicted articles which were less than 12, articles were added to the top 12 articles last week.

```
1 submission = pd.read_csv("../input/sample_submission.csv", on_bad_lines="skip")
2
3 submission = submission[['customer_id']]
4
5 submission = submission.merge(result, on='customer_id', how='left').fillna('')
6
7 submission.head()
```

	customer_id	prediction
0	00000dbacae5abe5e23885899a1fa44253a17956c6d1c3...	
1	0000423b00ade91418ccea3b26c6af3dd342b51fd051e...	0673677002
2	000058a12d5b43ae67d225668fa1f8d618c13dc232df0ca...	
3	00005ca1c9ed5f5146b52ac8639a40ca9d57aef4d1bd2...	
4	00006413d8573cd20ed7128e53b7b13819fe5cfc2d801f...	

```
1 submission['prediction'] = submission['prediction'] + top12
2
3 submission['prediction'] = submission['prediction'].str.strip()
4
5 submission['prediction'] = submission['prediction'].str[:131]
6 # 10*12 = 120 plus 11 spaces is 131, it was done to keeping 12 predictions for each customer
```

	customer_id	prediction
0	00000dbacae5abe5e23885899a1fa44253a17956c6d1c3...	0924243001 0924243002 0918522001 0923758001 08...
1	0000423b00ade91418ccea3b26c6af3dd342b51fd051e...	0673677002 0924243001 0924243002 0918522001 09...
2	000058a12d5b43ae67d225668fa1f8d618c13dc232df0ca...	0924243001 0924243002 0918522001 0923758001 08...
3	00005ca1c9ed5f5146b52ac8639a40ca9d57aef4d1bd2...	0924243001 0924243002 0918522001 0923758001 08...
4	00006413d8573cd20ed7128e53b7b13819fe5cfc2d801f...	0924243001 0924243002 0918522001 0923758001 08...

## Mean Average Precision

Using the real final transaction file for last week as a validation dataset, the Mean Average Precision for this project prediction and valid articles was calculated by following this kaggle address, and the final result for this project submission was 0.1007.


```
1 mapk(
2     sub['valid_true'].map(lambda x: x.split()),
3     sub['prediction'].map(lambda x: x.split()),
4     k=12
5 )
```

0.10074669646376394

A final version of my file was uploaded to the Kaggle competition and I received the above score.

2160

Mina Jm



0.0100


2

15h

Submission and Description

[submission.csv](#)  
15 hours ago by [Mina Jm](#)  
add submission details

Public Score  
0.0100

 Your Best Entry!  
Your most recent submission scored 0.0100, which is an improvement of your previous score of 0.0076. Great job!

Tweet this

## Reflecting on Results

The project proposed only Machine Learning algorithms for this multiclass classification problem. After Feature Selection, the accuracy scores achieved by the KNN model using Cross Validation Strategy were 58.01%, indicating that the accuracy had declined compared to the accuracy scored by the KNN model before feature selection. Results indicate that selecting features for classification cannot improve the accuracy of a model always, as was demonstrated in this article (Chen et al., 2020).

Accordingly, there is no definitive way to understand which Machine Learning models will be effective for which types of problems. As is evident from the discussion above, to solve a machine learning problem, the problem and data must be understood first. The most typical machine learning algorithms, or the algorithms




mentioned in the `skit_learn` algorithm approach, must then be selected that are most appropriate for solving such problems, and the models should be developed using the given training data. Therefore, based on the models that have been built and compared, different results may be obtained by different projects, depending upon their understanding of the given problem. In the majority of cases, the best results are dependent upon understanding the data and the preprocessing of the data.

Due to the low MAP 12 for this project, by repeating this project by clustering, it will be possible to improve the results.

#### **Last conclusion about the project (6th May):**

Following completion of this project, it has come to my attention that I had a major issue understanding the data. I had to split transaction data by the last seven days as the validation data, and the rest of data should be used as training data. In this case, data partitioning must be based on training data, not validation data. Each part of the project must be repeated after considering this concept.

## References

1. *Choosing the right estimator.* (2022). Scikit-Learn. [https://scikit-learn.org/stable/tutorial/machine\\_learning\\_map/index.html](https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)
2. Malik, F. (2022, March 8). *What Is Grid Search? - FinTechExplained.* Medium. <https://medium.com/fintechexplained/what-is-grid-search-c01fe886ef0a>
3. Majumder, P. (2021, December 30). *Probability Sampling : Implementation in Python - Analytics Vidhya.* Medium. <https://medium.com/analytics-vidhya/probabability-sampling-implementation-in-python-b7d3749eedb>
4. Grandini, M., Bagli, E., & Visani, G. (2020). *Metrics for Multi-Class Classification: An Overview.* <https://arxiv.org/abs/2008.05756>
5. *sklearn.metrics.balanced\_accuracy\_score.* (2022). Scikit-Learn. [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced\\_accuracy\\_score.html#sklearn.metrics.balanced\\_accuracy\\_score](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html#sklearn.metrics.balanced_accuracy_score)
6. *sklearn.metrics.matthews\_corrcoef.* (2022). Scikit-Learn. [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.matthews\\_corrcoef.html#sklearn.metrics.matthews\\_corrcoef](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.matthews_corrcoef.html#sklearn.metrics.matthews_corrcoef)
7. Chen, R. C., Dewi, C., Huang, S. W., & Caraka, R. E. (2020). Selecting critical features for data classification based on machine learning methods. *Journal of Big Data*, 7(1). <https://doi.org/10.1186/s40537-020-00327-4>
8. Team, D. S. (2021, April 20). *Feature Engineering.* DATA SCIENCE. <https://datascience.eu/machine-learning/feature-engineering/>
9. Wikipedia contributors. (2021, December 10). *One-hot.* Wikipedia. <https://en.wikipedia.org/wiki/One-hot>
10. Brownlee, J. (2020, June 30). *Tour of Data Preparation Techniques for Machine Learning.* Machine Learning Mastery. <https://machinelearningmastery.com/data-preparation-techniques-for-machine-learning/>
11. Henderi, H., Wahyuningsih, T., & Rahwanto, E. (2021). Comparison of Min-Max normalization and Z-Score Normalization in the K-nearest neighbor (kNN) Algorithm to Test the Accuracy of Types of Breast Cancer. *International Journal of Informatics and Information Systems*, 4(1), 13-20.
12. Brownlee, J. (2020b, August 28). *How to Use StandardScaler and MinMaxScaler Transforms in Python.* Machine Learning Mastery. <https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/>
13. H. (2022, March 1).  *H&M: Calculate MAP@12 from Public Notebooks* . Kaggle. <https://www.kaggle.com/code/hervind/h-m-calculate-map-12-from-public-notebooks>