|  |  |
| --- | --- |
|  | |
| Machine Learning Assignment  TU060 : H&M Personalised Fashion Recommendations  Kaggle Competition | |
|  | |
| Ciaran Finnegan – Part Time – First Year 2021/2022  MSc in Computer Science (Data Science)  Student No : D21124026  8/5/2022 |  |
|  |  |

Table of Contents

1 Introduction 3

1.1 Purpose of Report 3

1.2 Project Strategy/Approach 3

1.3 Python Coding Good Practice 3

2 Data Preparation + Project Implementation 4

2.1 Data Import + Analysis 4

2.2 Data Preparation + Enrichment 4

2.3 Scaling Datasets 4

2.4 Creating Training and Test Datasets 4

2.5 Implementation of Nearest Neighbour Model 5

3 Model Evaluation Strategy 6

3.1 Find Predicted Products for Each Customer 6

3.2 Find Actual Products for Each Customer (Local Test Data) 6

3.3 Local Evaluation Metric 7

3.4 Local Evaluation Results 8

4 Observations + Conclusions 12

4.1 Establishing a ‘Baseline’ Set of Evaluations 12

4.2 Optimisation Steps 13

5 Kaggle Submission 15

6 References 16

# Introduction

## Purpose of Report

The purpose of this report is to explain the structure applied to a Jupyter Notebooks Python project that attempts to implement a Machine Learning model for the predictive H&M Fashion Kaggle project.

## Project Strategy/Approach

An established Machine Learning workflow defined the structure of this project to;

1. Load the H&M Kaggle datastores from CSV files.
2. Clean and augment this data, before splitting into Train and Test datasets
3. Building a KNN *NearestNeighbors* model using the Training datasets to generate predictions on future customer purchases.
4. Evaluate model effectiveness using the Test dataset and a bespoke derived metric on valid predictions.
5. Declare this model as a ‘Baseline’ and then iterate through a series of optimisation steps that attempt to improve model accuracy.
6. Apply this model to submit a file to the Kaggle competition.

## Python Coding Good Practice

Best practice techniques were applied in the writing of the Python code to implement this project.

* Use of a Jupyter Notebook plug-in for a Table of Contents (TOC). This clearly delineates the separate sections in the project and acts as a means of navigation between code blocks. The TOC also provides a *de facto* comments structure around specific code blocks to explain the purpose of the Python code.
* Use of Python functions to minimise the need for code re-use. Particularly useful in the part so the project that implement data preparation routines, visualisations, and the actual *NearestNeighbors* models.

# Data Preparation + Project Implementation

## Data Import + Analysis

The project begins by importing the ***transactions\_train.csv*** file and works from a random sample.

In order to avoid subsequently loading redundant data, only articles and customers appearing in the sampled transaction file are loaded into Panda dataframes by the project code.

## Data Preparation + Enrichment

Section 2 of the Jupyter Notebook merges the transaction data in step with the Article and Customer data, producing a single, merged dataset for modelling. (Additional basic Panda dataframe descriptions are generated in parallel).

To ***clean*** this merged dataframe, any duplicate rows are removed and any rows with missing data are dropped. Metrics in the Jupyter Notebook show that this is a small proportion of data and will not be a noteworthy influence on the ML modelling process.

## Scaling Datasets

Feature scaling transforms the values of numerical features so that they fall within a similar range to each other**[1]**.

Our *NearestNeighbors* algorithm, described below in Section 2.5, only works with numerical inputs because it involves calculating distances between datapoints**[2]**. This project scales the data to avoid high absolute value range features distorting the purchase predictions. The sklearn ***MinMaxScaler()*** function converts all numeric features to a value between 0 and 1.

Categorical features are encoded to numerical values.

## Creating Training and Test Datasets

A simple random 70:30 split of merged Kaggle data would not be appropriate for this ML prediction problem. The objective to make a list of predictions for customers in the 7-day period after the ***transactions\_train.csv*** time period ends.

Therefore, the guidelines to generate Train and Test splits are as follows;

* Transaction data runs up to September 20th, 2020. (Begins on 20/9/2018).
* Predictions should be specific to the late September period. Possibly reflecting purchasing trends in the Autumn period or other such time-of-year factors.
* Test Data is filtered out from the transactions training file based on the ‘last week’ of data: Sept 15th – 22nd of each year. This Test data acts as a ‘proxy’ for the unseen test data for the last week of September 2020 used by Kaggle for submission evaluation.
* Training data is filtered out based on the Aug 23rd to Sept 14th period for each year in the Kaggle transaction data file.
* Thus, Training and Test dataset are specific subsets of the overall transaction data because the objective is to model customer purchase behaviour in late September.

## Implementation of Nearest Neighbour Model

The underlying logic in this project’s Machine Learning process can be summarised as follows;

* Train a ‘*NearestNeighbors’* model against the Training data described in Section 2.3. Types of transactions will be grouped according to their common characteristics.
* Customer-by-customer, pick their last transaction in the Test data and generate the ***NN*** (parameter) nearest neighbours (other customers) and select their top ***N*** (another parameter) similar purchases from the trained model.
* Return the ***N*** predicted purchases from the model output and compare against what the customer actually bought in the Test data (as Section 2.3 explains, the Test data is intended to represent the ‘future’ 7-day purchases).

This project implements a series of Python functions to set up and deploy the ‘*NearestNeighbors’* model.

Graphical user interface, text

Description automatically generated

Figure – n – Call function to train NN model

Text

Description automatically generated

Figure – n – Fit NN algorithm against training data

The choice for initial ‘baseline’ parameters for the *NearestNeighbors* model are based on a combination of Scikit Learn documentation**[3]** and a *Towards Data Science* article by Kevin Liao(2018)**[4]**.

Timeline

Description automatically generatedThe baseline Training dataset uses thirty-nine numerical features (see figure on left).

General recommendation are that with feature sets greater than fifteen, a choice of ‘*brute’* force algorithm may be preferred. *Cosine Similarity* may often also be a better measure of distance for nearest neighbour recommender searches. However, these settings will be tuned for performance re-evaluation in Section 4.2.

# Model Evaluation Strategy

## Find Predicted Products for Each Customer

A programme loop is set up based on the number of unique customers in the Test dataset (a count based on last customer transaction in Test data).

The latest customer transaction in the Test data is used as the basis for the input into the project *NearestNeighbors* model. The model is wrapped within a function that finds the **22** ‘nearest neighbour’ customers and extracts the **12** most popular articles bought by the ‘neighbourhood’.

A screenshot of a computer

Description automatically generated

Figure – n – Generate list of predicted purchases

In the figure above, the twelve articles returned by ***get\_ArticlePredictions()*** into the ***df\_TestCust\_Preds*** dataset are the predictions used in the evaluation against actual customer purchases in the Test set. This generates the value of **MP**, described in Section 3.3 below.

## Find Actual Products for Each Customer (Local Test Data)

As described in Section 3.1 the project code is set to loop through each Customer in the Test dataset and generate predictions through the *NearestNeighbors* model. The same loop then calls a project function for each customer to generate a list of actual purchases recorded in the Test data. This generates the value of **AP**, described in Section 3.3 below.

Text, letter

Description automatically generated

Figure – n – Generate list of actual purchases

See the project Jupyter Workbook for the implementation of all user-defined functions used in the project.

## Local Evaluation Metric

The H&M Personalized Fashion Recommendation competition in Kaggle uses a Mean Average Precision@12 calculation**[5]** to evaluate and rank submissions.

To simplify the process for this project, the following derivation of the Kaggle metric is applied to the output of the purchase prediction process to generate an accuracy percentage metric :

**U** : Total number of unique customers in the Test set.

**AP** : Actual Purchases made by customer in Test set.

**MP** : Matching Predictions from the deployed model.



In the project Jupyter Notebook, Sections ***6.2.3*** and ***6.2.4*** set up the *NearestNeighbors* prediction loop and present the evaluation

Refer to the Notebook for specifics on the Python function calls and code structure.

## Local Evaluation Results

### Small Sample – 1% of Transaction Data

The volume of data in the Kaggle transaction csv file is significant (31.8M rows). In order to iteratively build up the project code a 1% sample was used for the majority of the development phase (317K rows).

In the example below, which is consistent with other development phase 1% test runs, evaluating the *NearestNeighbors* model performance on such a sample size produced a Train and Test set of ***6387*** and ***17083*** rows, respectively.

Section 2.3 of this report shows how the Train/Test datasets are a deliberate subset of transaction data that attempts to mimic the required Kaggle prediction criteria.

The figures below are interactive Plotly Bar Charts**[6]** that show the proportion of Train/Test datasets when compared against the full sample of transaction data.

Chart, bar chart

Description automatically generated

Chart, bar chart

Description automatically generated

Figure - n - Bar Chart breakdown of Test v Train v Full Trxn dataset sizes (1% Sample)

Using the evaluation metric described in Section 3.3, the 1% sample result produces the following output of **35%** accuracy in purchase prediction.

Graphical user interface, text

Description automatically generated

Figure – n - *NearestNeighbors* Prediction Accuracy on 1% Sample of H&M Kaggle Transactions

### Low Medium Sample – 10% of Transaction Data

Extracting a 10% sample of the Kaggle transaction data (3.18M rows) took a considerable amount of time for model building and evaluation and was only executed twice. The figures and metrics below are from the final test run.

Chart, bar chart

Description automatically generated

Chart, bar chart

Description automatically generated

Figure – n - Bar Chart breakdown of Test v Train v Full Trxn dataset sizes (10% Sample)

Again, using the evaluation metric described in Section 3.3, a sample 10x times larger than in Section 3.4.1 produced an output of **56%** accuracy in purchase prediction.

Text

Description automatically generated

Figure – n - *NearestNeighbors* Prediction Accuracy on 10% Sample of H&M Kaggle Transactions

### Higher Volume Sample – 99% of Transaction Data

Extracting Training and Test datasets from the (almost) full 31.8 M transactions in the source Kaggle files, with the subsequent modelling and evaluation, was also an extremely processor intensive activity. (Processing took 2+ days).

It was necessary to export the Jupyter Notebook *.ipynb* file into a standard *.py* Python file and run in a PowerShell Window on a local PC. Hence, the output looks different to the other local evaluation screenshots, and the code had not yet been refined to include Train and Test dataset sizes.

Text

Description automatically generated

Figure – n – CMD Output - *NearestNeighbors* Prediction Accuracy on 99% Sample of Transactions

One can infer the probable size of the Train and Test datasets from the previous 1% and 10% runs. The accuracy of **40%** shows a degradation in performance from the 10% run.

The fact that the number of Predicted Purchases is lower than Actual Purchases (from the Test Data) is discussed in Section 4.2.1 of this report and is affected by the value of the ‘Neighbour’ parameter when building the KNN Model.

# Observations + Conclusions

## Establishing a ‘Baseline’ Set of Evaluations

Section 3.4 of this report provides a set of evaluation results for the *NearestNeighbors* model built to generate H&M purchase predictions, with samples of varying sizes. The code to build and train this model was considered a ‘baseline’ from which additional optimisations to improve prediction accuracy can be applied.

Observations from this ‘Baseline’ set of evaluation results can be summarised as follows;

* **Actual purchases are less than 12 per customer**. Our Train and Test days is deliberately limited to a specific time window. Thus, even the ‘full’ 99% sample data does not contain enough Test data transactions for each customer to generate 12 ‘actual’ purchases. This seems logical as presumably very few customers are likely to buy twelve products in the one-week window for the Kaggle competition. Thus, this project has assumed that the predictions will generally be a *superset* of ‘actual’ transactions. Accuracy is based only on the number of actual transactions predicted.
* **Setting an initial value for the number of nearest neighbours of ‘22’**. The project code always attempts to generate 12 predictions. Experimenting with the Kaggle submission, as part of the Baseline, shows that the parameter value used for number of ‘nearest neighbours’ in the model will influence the number of predictions. An initial value of ‘22’ was selected because this was found to be the lowest number that consistently generated 12 predictions for the Kaggle submission. Any lower would fail to consistently create 12 predictions based on the purchases of the nearest neighbour for each customer (higher values add computational overhead). This is an optimisation step we revisit in Section 4.2.1 of this report.
* **Limit customer numbers to those who might actually buy a product in the prediction window.** Kaggle shows that the number of customers in the competition csv file is 1.38M. Our final 99% sample only involves just over 158K unique customers. This project has deliberately limited focus to that subset of customers likely to make any actual purchase in the late September one-week competition window. Every customer in the transaction train period must have a prediction made for the Kaggle submission. However, customers who did not make a purchase in that 7-day period are excluded from scoring.
* **Accuracy initially increased with data volume.** The 10% sample run was more accurate that the 1% sample (by over 20%). However, this performance dropped back significantly when a model was trained with the full Transaction Kaggle datafile. There may be data outside of the necessary time window in the ‘full’ transaction file that is distorting the prediction model. Future refinement may be to also sample Kaggle transactions on a time basis and not by just by random selection. (Unfortunately, this was not done in this project due to time constraints).

Section 4.2 below describes the steps taken to attempt to improve on the accuracy of this project’s *NearestNeighbors* prediction model/process.

## Optimisation Steps

### Refine the ‘Nearest Neighbour’ Value for the KNN Model

A 2020 article by Analytics Vidhya, looking at simple movie recommender systems, advised that care is required when choosing the number of ‘neighbours’ to select**[7]**. In general, too small a value can introduce noise in the result and, more specifically for the Kaggle competition, the code may fail to return enough predictions.

Too high a value for the *kneighbors* functions would have a computational overhead.

Text

Description automatically generated

Running trials on creating the Kaggle submission in this project proved that a ‘*n\_neighbors’* value of **22** was the lowest value that consistently. Increasing this value into ranges 30 -75 slowed the processing and did not improve any of the 1% or 10% accuracy results.

Therefore, it was assumed that there was no benefit in increasing the ‘*n\_neighbors’* value beyond **22**.

### PCA on Merged Kaggle H&M Data

Following on from Section 2.5 of this report, Scikit Learn documentation recommends that the use of a ‘Brute’ force algorithm should be avoided on larger datasets, such as those found in this Kaggle competition. However, the numbers of features should also ideally be reduced to less than 15 before using a tree-based algorithm.

Part 8 of the project Jupyter Notebook conducts a Principal Component Analysis to determine the benefit of dimensionality reduction on the merged H&M Transaction/Customer/Articles dataset.

Chart, bar chart

Description automatically generatedThe PCA code produced the following graph;

Figure – n – PCA on project H7M dataset

This was unlike the lecture example of penguin size analysis conducted by Horst and Gorman**[8]**. In the project data set we do not see an obvious grouping of most data variance in the PC1 – PC3 dimensions. Even training a model on only the first five (or ten) Principal Components would still exclude a considerable proportion of valuable information.

Therefore, dimensionality reduction was **not** conducted in this project in order to improve accuracy.

A more arbitrary reduction of features (eliminating encoded categorical columns) was used instead in parallel with model parameter changes in Section 4.2.3 below.

### Alter Default Parameter Values for the NearestNeighbors Model

‘*Euclidean’* distance is the most popular method for distance metrics in KNN and Gokte(2022) notes that ‘*Cosine’* is most often used to calculate similarity between two vectors**[9]**, which explains the tendency to use this approach in Recommender Systems.

The Scikit learn documentation advises that brute force computation of distances becomes computationally expensive as the number of samples grows**[3]**. Even with the date filtering of the Train and Test dataset from the Kaggle transaction file, the processing overhead on the baseline approach is significant.

A *tree-based approach* may reduce the number of distance calculations but would need to be done in parallel with the feature reduction steps described at the end of Section 4.2.2 above.

Text

Description automatically generatedFigure – n – Tree Based Neighbour Search Evaluation Results.

A new model was retrained with “*algorithm = ‘kd\_tree’*” and “*metric = ‘l2’*” (Euclidean distance) produced the following results on a **5%** sample.

The processing time was marginally faster than with the ‘baseline‘ model, but the 51% score for the ‘optimised’ model was almost identical to a 5% sample ‘baseline’ run, and slightly less accurate than the 10% sample ‘baseline’ figure 56% .

These changes in the *NearestNeighbors* model parameters are implemented in the final project Jupyter notebook, but more for slight improvements in computational performance than for increases in accuracy (which were not really evident).

The project Jupyter Notebook uses parameters ‘*local’* and ‘*optim’* to create separate *NearestNeighbors* models, and ‘*kaggle’* for the competition prediction submission.

### Observations on ‘Optimisations’

None of the approaches in Section 4.2 of this report made any marked improvement on the ‘baseline model’, with the exception of a slight computational efficiency recorded in sub-section 4.2.3 above.

It is likely that refinements to the Train/Test data split, and more elaborate time-sequence analysis of the source Kaggle transaction train data, could generate better prediction results if this project were attempted in the future.

# Kaggle Submission

Partially following suggested Kaggle submission process from Harururu and Ryotaru**[10]** , Part 7 of the project Jupyter notebook builds up a competition submission file.

A partial view of the output generated by the project notebook can be seen in the image below.

Text

Description automatically generated with medium confidence

Figure – n – Project Kaggle Submission file

The project *ipynb* file was uploaded to a Kaggle Notebook and executed.

At time of drafting this report, and submission of the assignment, the Kaggle session was still in progress. It is therefore no possible to confirm a score in this Kaggle competition.

Graphical user interface, text, application, email

Description automatically generated

Figure – n- Kaggle Submission

# References

[1] Kumar, A. (2020). MinMaxScaler vs StandardScaler – Python Examples. Retrieved 7 May 2022, from <https://vitalflux.com/minmaxscaler-standardscaler-python-examples>

[2] k-Nearest Neighbor: An Introductory Example - Numerical Features Required. (2022). Retrieved 7 May 2022, from <https://quantdev.ssri.psu.edu/sites/qdev/files/kNN_tutorial.html>

[3] 1.6. Nearest Neighbors. (2022). Retrieved 6 May 2022, from <https://scikit-learn.org/stable/modules/neighbors.html#unsupervised-neighbors>

[4] Liao, K. (2018). Prototyping a Recommender System Step by Step Part 1: KNN Item-Based Collaborative Filtering. Retrieved 6 May 2022, from <https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea>

[5] H&M Personalized Fashion Recommendations | Kaggle. (2022). Retrieved 6 May 2022, from <https://www.kaggle.com/competitions/h-and-m-personalized-fashion-recommendations/overview/evaluation>

[6] Bar chart using Plotly in Python - GeeksforGeeks. (2021). Retrieved 6 May 2022, from <https://www.geeksforgeeks.org/bar-chart-using-plotly-in-python/>

[7] Dedhia, H. (2020). Recommendation System using K-Nearest Neighbors |Use Case in Python. Retrieved 7 May 2022, from https://www.analyticsvidhya.com/blog/2020/08/recommendation-system-k-nearest-neighbors/#h2\_14

[8] Horst, A., & Gorman, K. (2020). PCA with penguins and recipes. Retrieved 7 May 2022, from https://allisonhorst.github.io/palmerpenguins/articles/pca.html

[9] Anil Gokte, S. (2022). Most Popular Distance Metrics Used in KNN and When to Use Them - KDnuggets. Retrieved 6 May 2022, from <https://www.kdnuggets.com/2020/11/most-popular-distance-metrics-knn.html>

[10] Harururu, H., & Ryotaru, U. (2022). H&M EDA & Customer Clustering by KMeans. Retrieved 6 May 2022, from https://www.kaggle.com/code/hirotakanogami/h-m-eda-customer-clustering-by-kmeans