|  |  |
| --- | --- |
|  | |
| 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

4.3 Project Observations/Future Suggestions 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 is applied in 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 model using the Train 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 to improve model accuracy.
6. Apply this model to submit a file to the Kaggle competition.

## Python Coding Good Practice

A number of 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 delimitate 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 KNN 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. (Some basic Panda dataframe descriptions are generated in parallel).

To clear this merged dataframe, any duplicate rows are removed and any rows with missing data are dropped. Metrics in the Jupyter Notebook shows that this is a very small proportion of data and will not be a significant 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[x].

Our *NearestNeighbors* algorithm, described below in Section 2.5, only works with numerical inputs because it involves calculating distances between datapoints[x]. 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 merge project 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 very 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 projects 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).

Including features used, prediction algorithm(s) used, parameters, etc.;

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[x] and Towards Data Science article by Kevin Liao(2018)[x].

Timeline

Description automatically generatedThe baseline Training dataset uses 39 numerical features (see figure on left).

General recommendation are that with feature sets greater than 15, a choice of ‘*brute’* force algorithm may be preferred. *Cosine Similarity* may often also be a better measure of distance for nearest neighbour search. 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 transaction in Test data).

The latest customer transaction in the Test data is used as the basis for the input into the project KNN model. The model is wrapped within a function that finds the 15 ‘nearest neighbour’ customers and extracts the 12 most popular articles bought by the ‘neighbourhood’.

A screenshot of a computer

Description automatically generated with medium confidence

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 KNN 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[x] 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.

## 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 most of the development phase (317K rows).

In the example below, which is consistent with other development phase 1% test runs, evaluating the KNN 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[x] 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 - KNN 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 time larger than in Section 3.4.1 produced an output of **56%** accuracy in purchase prediction.

Text

Description automatically generated

Figure – n - KNN 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 – Command Line Output - KNN 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 KNN 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 almost always be a superset of ‘actual’ transactions. Accuracy is based only on how many actual transactions are predicted.
* **Setting an initial value for the number of nearest neighbours of ‘15’**. 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 KNN model will influence the number of predictions. An initial value of ‘15’ 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.

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

## Optimisation Steps

<Diagram>

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

Run Kaggle submission section…

### Feature Reduction

Find process to assess feature importance for NearestNeighbour…

### Alter Default Parameter Values for the NearestNeighbor 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[x], which explains some of the tendency to use this approach in Recommender Systems.

The Scikit learn documentation advises that brute force computation of distances becomes computationally very expensive as the number of samples grows[]. 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 in Section 4.2.2 above.

A new model was retrained with “*algorithm = ‘kd\_tree’*” and “*metric = ‘l2’*” (distance) produced the following results on a 3% sample.

<>

Figure – n – Tree Based Neighbour Search Evaluation Results

### Implement Cross Validation When Training KNN Model

Find example of NN CrossValidation…

## Project Observations/Future Suggestions

<Diagram>

Including features used, prediction algorithm(s) used, parameters, etc.;

.

# Kaggle Submission

One could ipso lorem…

Partially following suggested Kaggle submission process from Harururu and Ryotaru[x]…

# References

[1] Polkovnikov, I. (2016). Unified Control and Data Flow Diagrams Applied to Software Engineering and other Systems. Retrieved 15 April 2022, from https://doi.org/10.48550/arXiv.1610.02374

[2]

[3]