KNN Classification Model for H&M Product Recommendation

Joshua Kobuskie

Abstract

The problem that has been attempted is to predict articles of clothing from the H&M catalog that are likely to be bought by customers of H&M in the next 7 days. Data about the customers, articles available for sale, and previous transactions have been provided. To solve the problem, a K-Nearest Neighbor Classification model approach was utilized that was trained on one year's worth of previous transactional data. This data was a composite of transaction information, customer information, and article information. A prediction in the form of an article_id from the H&M catalog has been made for all customers, even those without any prior transactions. Customers without prior transaction history were assigned based on the most popular articles recommended, rather than by the K-Nearest Neighbor model. The result produced was a CSV file containing a prediction of an article_id to be purchased for each customer uniquely identified by their cusomter_id. The results produced during testing were accurate approximately 70% of the time after modifying the hyperparameters to optimize for both speed and performance. Higher accuracy can be achieved, but at a greater computational and time cost. In testing with smaller datasets, accuracy above 90% was achieved, but the execution time of such models became impractical on a larger scale, including the dataset provided.

Introduction

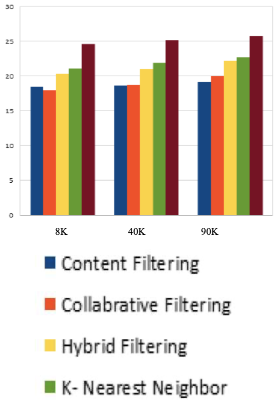
The problem I have been working on is the H&M Personalized Fashion Recommendations challenge. In this competition, I have been tasked with predicting what articles of clothing from the H&M store a given customer is going to purchase in the next 7 days. The articles are limited to those listed in the articles.csv file, and all articles are represented by a 10-digit "article_id". Every customer in the dataset must have a predicted article, even if they have no transactional history. If a customer did not make a transaction in the 7 days, they are discarded from scoring, as per the competition's instructions. The methods used to achieve these results are open to the participant to choose. I have chosen to implement a K-Nearest Neighbor approach.

This problem is important because it has real world applications. By being able to accurately predict what articles customers are likely to buy, it can allow H&M to better advertise to their customers and increase sales. With a functional and effective model, H&M can curate their website to better entice customers, and allow them to stay competitive in the marketplace. This model will likely grow a returning customer base, since the customers who do make purchases will typically be offered articles that they are more likely to be satisfied with, leaving a positive impression of the brand. This will also reduce returns, since customers will be less likely to return articles that they are satisfied with, and this will further boost the profit margins of H&M.

My results have been collected in a CSV file that is ready for submission, as per the competitions rules. This CSV file has been stored in the variable "submit". The data within the CSV file was generated from a data frame containing two columns, "customer_id" and "prediction", which was generated by my model. The first column, "customer_id", contains the list of all customers for the competition data set. In total, the data set contains 1,371,980 customers. The second column, "prediction", contains the predicted 10-digit "article_id" value generated by the model for each customer. During testing, my trained model achieved approximately 70% accuracy over multiple trials on subsets of the data.

Related Work

To gain insight and direction into how to approach creating a machine learning model for product recommendations, I read THE USE OF MACHINE LEARNING ALGORITHMS IN RECOMMENDER SYSTEMS: A SYSTEMATIC REVIEW. In this academic paper, Professor Parinita J. Chate conducts a system review of different machine learning methods currently used for product recommender systems. Almost all of the models studied were predictive models. Of these, there were four general categories that the systems fell into. Systems could be seen as content-based, content-filtering, collaborative-filtering, or hybrid-filtering which was a combination of the previous two filtering methods. My approach is similar to those presented in the content-based models inspected by Chate. I have chosen to implement a K-Nearest Neighbor algorithm to solve my recommender challenge. This approach is studied in the paper in detail, and utilizes predictive analytics and content-based models to predict a product to each customer. However, this approach suffers from a similar issue to that discussed in the paper, cold starts. When customers have little or no transactional data, it is hard for the machine to predict what product to recommend to a customer. To overcome this issue, I chose to make my machine different from a traditional K-Nearest Neighbor model. If the customer had no data to predict on, my guess would be very likely to be incorrect. Therefore, in my machine, customers with no data were excluded from the model and assigned the most common suggestion from the remaining dataset. This guess was correct more often in testing than the random results produced by the model, and slightly improved performance, while also limiting the dataset size to decrease execution time. I chose to implement my K-Nearest Neighbor approach based on the reasonable size of the transactional history provided and the strength displayed by the algorithm as compared to other methods tested and reviewed by Chate. As displayed in the bar graph taken from Chate's systematic review, the K-Nearest Neighbor algorithm consistently outperformed other recommender methods, regardless of dataset size.

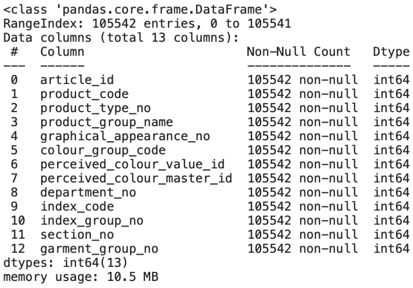


Bar chart 1: Precision Comparison

Data

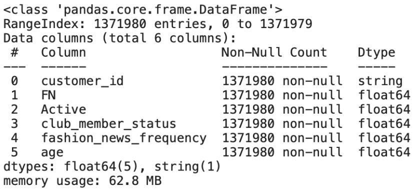
The data I am working with for my project consists of three files, articles.csv, customers.csv, and transactions_train.csv. All data was read in and preprocessed using the pandas library. All csv files were converted to dataframes for processing.

The articles.csv file contains 105,542 different article entries. Each article entry contains the article_id as well as 24 other data points that describe the article. These data points ranged from the department number that the article belongs to to the color or graphical appearance of the item. This data consisted of mostly integer index numbers used for each group, and string objects used to describe the groups. In most cases, these strings were redundant, and only the integer values were used. However, the product_group_name and index_code were useful string parameters that I wanted to preserve, so I utilized the labelEncoder.fit_transform() function to create integer indexes linked to each unique category in these two fields. I then trimmed the data to only include these numerical data points, leaving me with 13 usable columns of data for each article in the articles.csv data.



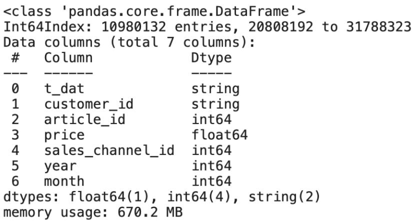
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 105542 entries, 0 to 105541
Data columns (total 13 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   article_id                  105542 non-null  int64
 1   product_code                105542 non-null  int64
 2   product_type_no             105542 non-null  int64
 3   product_group_name          105542 non-null  int64
 4   graphical_appearance_no     105542 non-null  int64
 5   colour_group_code           105542 non-null  int64
 6   perceived_colour_value_id   105542 non-null  int64
 7   perceived_colour_master_id  105542 non-null  int64
 8   department_no               105542 non-null  int64
 9   index_code                  105542 non-null  int64
 10  index_group_no              105542 non-null  int64
 11  section_no                  105542 non-null  int64
 12  garment_group_no            105542 non-null  int64
dtypes: int64(13)
memory usage: 10.5 MB
```

The customers.csv file contains 1,371,980 entries consisting of 7 columns worth of data for each customer. Each customer is identified by a unique customer_id string, and has personal information associated with them including postal_code, club_member_status, and fashion_news_frequency. The postal codes were very difficult to convert to usable data, and thus were dropped. The other values were represented as strings, but were processed to be usable in the form of floats, since they were either not involved, somewhat involved, or frequently involved with the corresponding categories. Each level of engagement was represented as a 0.0, 0.5 or 1.0 respectively. Finally, age data was also provided in the form of an integer. However, there were many null values within this data set, where customers declined to answer certain questions such as their age. For null ages, the average customer age from the known dataset was used to fill in this gap in the data. After verifying all the data points were non-null and numerical, this document was ready for the model with 6 usable columns of data for each customer.



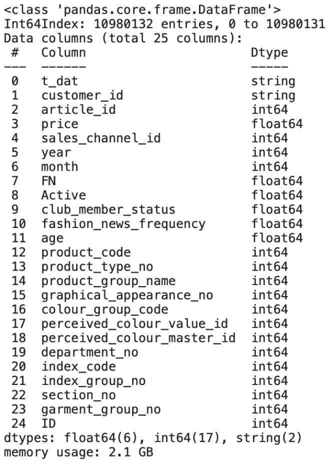
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1371980 entries, 0 to 1371979
Data columns (total 6 columns):
 #   Column                  Non-Null Count    Dtype
---  ------                  --------------    -----
 0   customer_id             1371980 non-null  string
 1   FN                      1371980 non-null  float64
 2   Active                  1371980 non-null  float64
 3   club_member_status      1371980 non-null  float64
 4   fashion_news_frequency  1371980 non-null  float64
 5   age                     1371980 non-null  float64
dtypes: float64(5), string(1)
memory usage: 62.8 MB
```

The transactions_train.csv file contained 31,788,324 transaction entries. Each transaction had 5 attributes associated with it, including the transaction date t_dat, customer_id, article_id, price, and sales_channel_id. From this data, I extracted the month and year of the transaction separately into different columns. From there, I selected only recent transactions from 2020 to train my model with. This decision was made as it gives more weight to more recent trends in predicting future purchases, as well as narrowing the data set size. One major issue I encountered while trying to import the data was the size of the files. It was not possible to import all of the datasets and keep everything until the end of the execution. Therefore, I had to choose when to delete data and utilize the garbage collection library to clear out space in memory whenever possible to keep the program from requesting too much memory and restarting. Finally, I converted the customer_id and t_dat to strings and prepared to merge the data.

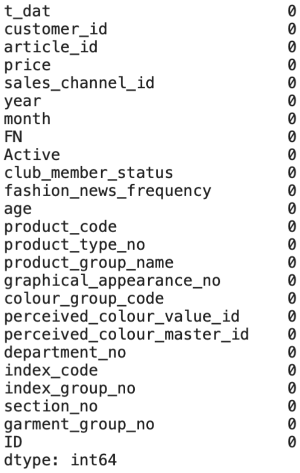


```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10980132 entries, 20808192 to 31788323
Data columns (total 7 columns):
 #   Column             Dtype
---  ------             -----
 0   t_dat              string
 1   customer_id        string
 2   article_id         int64
 3   price              float64
 4   sales_channel_id  int64
 5   year               int64
 6   month              int64
dtypes: float64(1), int64(4), string(2)
memory usage: 670.2 MB
```

At this point, I had read in all three files provided to me that gave me information on the list of customers that I had to predict purchases for, their transaction history from the last 24 months, and the articles available for sale, as well as details relating to each of these three categories. I now had to merge my data together to make it usable in my model. To do this, I merged the transactions table with the customers table, and then again merged that table with the articles table. This created a table containing all of the information about a customer and an article and connected them in a transaction. This also meant that each transaction now had 24 data points associated with it describing the transaction, customer, and article. This would be very useful in training my model, but the model can only analyze numerical values and this table still contains string data including the customer_id. This data needs to be preserved and be accessible later, but cannot be used for the model and offers no real value. Therefore, I created an ID column to uniquely associate each transaction with each customer_id that could be used later to join the data back together. To save space, I delete the transactions and articles data, since it is all now used in my new merged table, and do one final check to make sure that there are no null values left in my data set. I can finally create a seperate table containing the customer_id and ID indexes and begin processing the data.



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10980132 entries, 0 to 10980131
Data columns (total 25 columns):
 #   Column                        Dtype
---  ------                        -----
 0   t_dat                         string
 1   customer_id                   string
 2   article_id                    int64
 3   price                         float64
 4   sales_channel_id              int64
 5   year                          int64
 6   month                         int64
 7   FN                            float64
 8   Active                        float64
 9   club_member_status            float64
 10  fashion_news_frequency        float64
 11  age                           float64
 12  product_code                  int64
 13  product_type_no               int64
 14  product_group_name            int64
 15  graphical_appearance_no       int64
 16  colour_group_code             int64
 17  perceived_colour_value_id     int64
 18  perceived_colour_master_id    int64
 19  department_no                 int64
 20  index_code                    int64
 21  index_group_no                int64
 22  section_no                    int64
 23  garment_group_no              int64
 24  ID                            int64
dtypes: float64(6), int64(17), string(2)
memory usage: 2.1 GB
```



```
t_dat                         0
customer_id                   0
article_id                    0
price                         0
sales_channel_id              0
year                          0
month                         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
ID                            0
dtype: int64
```

Methods

The method I used to solve the problem of predicting the articles likely to be purchased by a given customer for the H&M product recommendation challenge was a K-Nearest Neighbor Classification Model. I chose this approach based on its demonstrated effectiveness of the model at the task and the large database of prior transactional history available to train it with. This large collection of content to train the learner allowed me to avoid cold-start problems and make accurate predictions.

My approach was the right choice because of its ability to be tailored to meet the needs of the client and its effectiveness. By increasing or decreasing the K value, it is possible to increase and decrease both the accuracy and runtime of the model. If the client is willing to compromise on the time required to execute the model per customer, the accuracy can be improved. For a large dataset, the K can be limited to produce somewhat less accurate results in a much quicker timeframe. This allows for the client to adjust the function of the model to prioritize either accuracy or speed for advertising to a potential customer, and thus allows for the likelihood of a customer being satisfied with a specific article and the success of advertising to be controlled. IN some situations, the client may want a somewhat useful prediction generated rapidly for instant advertising, but in other situations the client may have the ability to run the model in advance and store the predicted articles for each customer in a database that can be accessed at a later date for advertising purposes. The second option would allow for the full strength of the model to be used, with little concern over the speed of its execution. This method was an effective solution to the problem presented, as shown by its ability to outperform other types of models in the systemic review done by Chate.

Originally, I had considered another approach to this problem using multi-classification and probability metrics. Using this approach, I would have been able to calculate the probability of a customer being satisfied with an article, and only suggest the highest probability article or articles to that customer. However, upon further inspection of this method, it would have required me to use each of the 105,542 different articles as separate classes. This would have led to a very wide probability distribution, and likely would have taken a very long time to train. For these methods, the gradient would have to be calculated after every pass and this would be very computationally intensive for so many classes. For these reasons, I disregarded the idea of probabilistic multi-classification and instead pursued the proven K-Nearest Neighbor method.

Experiments

While implementing my K-Nearest Neighbor method, I utilized knowledge that I had accumulated throughout the course including my knowledge of sampling, hyperparameters, and classification methods. I utilized my knowledge of sampling to ensure that my model would not be overfit to the data. By adjusting the size of my training set, I was able to increase the accuracy of the model. However, as the sample size grew, so too did the likelihood of overfitting. As I increased the size of the training set, it began to take up more and more of my data, and produced better and better results. However, once I began testing the model on the dataset as a whole, my accuracy decreased significantly, showing that the model was failing to accurately predict for all customers, and was overfit to the data. By adjusting the sample size, I was able to reasonably optimize the accuracy of my model and while also limiting runtime for the best results. With my current sample size of 25,000 customers, I am able to accurately predict customer articles approximately 70% of the time. This is a reasonable level of success, without overloading the resources provided to me by the Kaggle environment. The Kaggle environment was quickly overwhelmed with the larger and more complicated training sizes, and execution times grew rapidly. Below is a small table displaying execution times for a few different sample sizes.

Sample Size	2,500	25,000	125,000
Execution Time	44m	1hr 2m	2+ hr
Accuracy	39.24%	69.12%	N/A

Additionally, I was able to utilize my knowledge from the course to control the hyperparameter K in the model. By limiting the K, I was limiting the number of neighbors that were explored for each data point, and this directly impacted the performance of my model. By increasing the number of neighbors explored, I was able to increase accuracy, but at a very heavy cost. The model was able to generate accurate test values above 90%, but the model could not complete execution on the whole dataset in a reasonable amount of time. In all trials with K greater than 1, accuracy was high, but execution time was drastically higher. None of the models completed execution in less than two hours, and thus the idea to increase K was severely limited due to the intense computational burden it placed on the model with such a large data set.

Conclusion

By the end of my implementation of the K-Nearest Neighbor Classification model, I had produced a predicted article given the information about a customer and their previous transactional history. At this point, I was able to merge the customer_id back onto the newly generated predicted values based on their ID index generated previously. If a customer_id had no transactional history, it was not included in the prediction column, but a suggestion was still required. In these cases, the most popular and thus most commonly suggested article was added as the predicted article for that specific customer. I then padded the predicted article_id with leading 0s to be a 10-digit integer, and created a dataframe containing only customer_id and predicted for each unique customer_id provided. This dataframe was then converted to a CSV file and is ready for submission.

After completing the challenge, I have gained a much stronger understanding of machine learning and classification methods. Through my exploration of different techniques, I was able to gain a broad understanding of the strengths and weaknesses of different classifiers and the use cases for each. I have also learned about hyperparameter optimization, and the resource constraints on models. With both time and memory constraints, it was nearly impossible to implement the best possible model that could be conceived of using this method. Instead, I had to make compromises to retain accuracy while decreasing execution time and being careful to not overload the memory limitations of Kaggle.

Possible extensions of my K-Nearest Neighbor approach include creating a database to store the predictions for a customer. This would allow for the model to be run for long periods of time per person, and increase its accuracy to recommend only the best selection of articles to customers. By utilizing a database, the model can run for hours or days in advance on a very large dataset, and produce predictions for all customers that will be highly effective. These articles can then be pushed as advertisements to customers at any point in the future with no additional computational cost. By implementing this technique, it is possible to limit the number of compromises that need to be made during optimization. Additionally, it may be a valuable idea in the future to limit the number of data points retained for the model. The current model utilizes 25 different data points to make a prediction about a customer's article preference. However, some of these values may be redundant or not heavily weighted by the model. Therefore, these values are adding significant computational cost without significant benefits in accuracy. By limiting the number of data points, it may be possible to increase speed without decreasing the accuracy of the model.