Text-based Machine Learning for E-Commerce Chatbot System

Abstract

Every online service or shopping experience now has some form of chat-bot that helps customers based on basic questions and then connects them to a customer care agent. The biggest advantage of a chat-bot is the fact that it is available to consumers 24/7. The lack of personalizing limits customer experience especially if they have to wait to be connected to an agent. By creating a more personalized chat-bot based on specific consumer input and customer history we can provide better recommendations.

Introduction

Online stores are able to provide customers an accelerated shopping experience, however, there still remains one major problem getting a user to a product: personalising product recommendations based on both products available and preferences of a customer. Although search engines are capable of indexing products with ease, including tools such as filtering to get to a final set of products, there still remains the issue of personalizing. We define personalizing for chat-bots as discovering product-to-product and user-to-product associations[1] in a table of products, such that it maintains variety, while keeping consistency similar to directly querying i.e search engine. The two key problems we try to solve are: variety of personalising and complexity of user-to-product solutions.

The idea of a personalised search engine lends itself to chat-bot systems that have been widely integrated into nearly all e-commerce services available. Chat-bots are used both as a part of customer service, but some systems integrate product recommendation systems based on user dialogue. However, this examines the two problems we intend to fix. First, the variety in which the chat-bots tend to recommend usually are the result of parsing key features in order to replicate ”searching” for a product, this tends to give the user great direct results, yet the results are, only as good as directly querying the keywords. Alternatively, many chat-bot systems may not only look at features parsed in a message, but as well use highly complex models involving series of language models and other expensive techniques such as Recurrent Neural Networks (RNN), factorisation machines, and Spatial Predictive Models (SPM)[4]. These complexities tend to find a good variety of results, however the trade-off is that these models are both costly to train and, in most scenarios to predict. Furthermore, in the case of adopting large language models into a site used by over thousands or millions of customers, it becomes infeasible to uphold due to the cost of maintenance, as well as the time it takes to return product recommendations. In our approach, we will look to maximize the advantages of the two architectures, through the use of simpler but robust machine learning methods, that preserve variety while giving strong predictions based on features parsed.

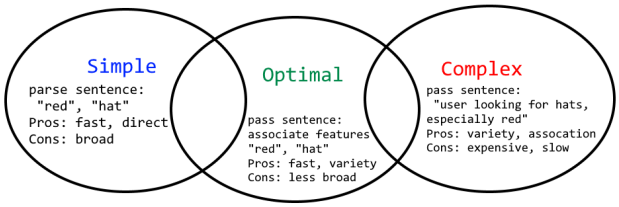

Figure 1: Comparison of different chat-bot recommendation architectures

In order to tackle this approach of developing a robust system for an e-commerce chat-bot, we plan to build three important concepts into our architecture: Feature Engineering, Association and Relevance. For any strong modelling of product recommendation, feature engineering is necessary to discover as many possible details a product may contain, such as brand name, information about textures, styles etc. All of these contain valuable data for making better predictions on products. Association further improves on feature engineering by finding whether products suggested are related due to random nature, or genuinely have key features that can make the products viable recommendations. Lastly, we look at relevance, given association is able to find products that are related, how strong of a recommendation is it based on the original problem? This allows us to reduce the final responses found by association, by enforcing stricter rules based on input. As a result, it should be expected that

using a high relevance metric will lead to results more inline with querying, since results are expected to be similar to key features. While low relevance, will give more variety, while good, must be regulated in order to enforce products anticipated by the user.

Methodology

In our system architecture (Figure 2.) we propose the following layout: given an initial personalised question-and-answer (Q&A), followed up by chat-bot responses, reduce the database of products to a list of products that maintain variety of relevant products. We will use two sub-models, Market Basket Analysis and KeyBert, in order to provide association and filtering techniques to our model.

Feature Extraction

In more detail, we first look to extract key features from a database[2], for a fashion store like Macy's or Nordstrom, these typically include: *Brand Names, Colors, Textures, Styles, Types*. These features allow use to use association rules in order to find relevant products based on each feature desired. During an instance of the chat-bot, we first ask the user for their preferences on features available, from those extracted, by using the Q&A. An example input to the Q&A may look like:

$$\{"BrandName" : "life",$$
$$"Colors" : "red",$$
$$"Textures" : "polyester",$$
$$"Styles" : "blend",$$
$$"Types" : "dress"\}$$

Finally, we encode the features as features to the original database, and create another database, containing information on whether a certain product contains each feature, as True/False values.

Market Basket Analysis

By association we use those feature responses from the Q&A to find all relevant products, by comparing all combinations of the features, and keep a list of products that are likely associated, and dropping those that may be random. Market Basket Analysis (MBA) is capable of finding these association using the $lift$ metric, which evaluates assocation based on the frequency of both products existing together, divided by how likely each one exists in a database. The frequency of a product/s is called the $support$ given as a probability.

$$\text{Support}(A) = P(\text{Frequency of } A \text{ in Database } D)$$

$$\text{Lift}(X_1 \cdots X_n) = \frac{P(\text{Frequency of features } X_1 \cdots X_n \text{ in Database } D)}{\prod_{X_i} P(\text{Frequency of } X_i \text{ in Database } D)}$$

A higher lift means a higher likelihood of a set of products to not be associated by random, the usual rate of association in lift is > 1.0. Therefore, we can look at all 2-combinations of each feature from the Q&A and calculate the lift of each, saving them when they meet a certain lift criteria.

Cosine Similarity Measure Although MBA returns a variety of results, as our intended proposed goal, one difficulty is that MBA may find too many results, especially those which may not be relevant enough compared to the original Q&A response. In order to keep relevant results while maintaining a regulated amount of variety, we use the cosine similarity measure based on the Q&A response. By one-hot-encoding the features in the Q&A and comparing it to each product found by MBA, we can measure how close a product is, to the original set of questions.

$$\text{Q\&A} = \{"\text{life}", "\text{red}", "\text{dress}", "\text{blend}"\}$$
$$\text{One-hot(Q\&A)} = (1, 1, 1, 1)$$

$$\text{Product} = \{"\text{life}, "\text{orange}", "\text{dress}", "\text{blend}"\}$$
$$\text{One-hot(Product)} = (1, 0, 1, 1)$$

$$\text{Cosine-Similarity}(A, B) = \frac{A \cdot B}{\|A\|_2 \|B\|_2}$$
$$\text{Cosine-Similarity}(A, B) = \frac{3}{2\sqrt{3}} = 0.8$$

In order to keep enough related values, the similarity measure is based of how many inputs are used in the $Q\&A$, for 4 inputs, it may be wise to adjust the similarity threshold to 0.5, to allow for two incorrect product features.

KeyBert and Content-based Filtering

After passing the MBA phase, we allow the user to input chat-bot messages, now instead of directly having features, we need a method of extracting them, furthermore, handling clutter that exists in a sentence. For our model, we use the language model KeyBert, a subset of BERT for extracting key phrases. The original BERT model uses the idea of computing phrase embedding and document embedding based on whole sentences, this generates a vector space based on all input documents, then ranking candidate phrases using similarity measures between the vector space created by the model[6]. In our case, we are only interested in finding features relevant to our database, therefore we use two hyperparameters to adjust features returned. That are we only keep the top-N words found in a given sentence, and drop any values that score low relevancy. For example, if we filter the keep the top-3 results, and drop values with a score < 0.5:

$$\text{Message: "I am looking for a blue pair of jeans, any brand is fine but it needs to be fit"}$$
$$\text{KeyBert(Message)} = \{("jeans", 0.89), ("blue", .73),$$
$$("pair", .41), ("brand", 0.36), ("fit", .53)\}$$

$$\text{Filtered(KeyBert(Message))} = (\text{jeans, blue, fit})$$

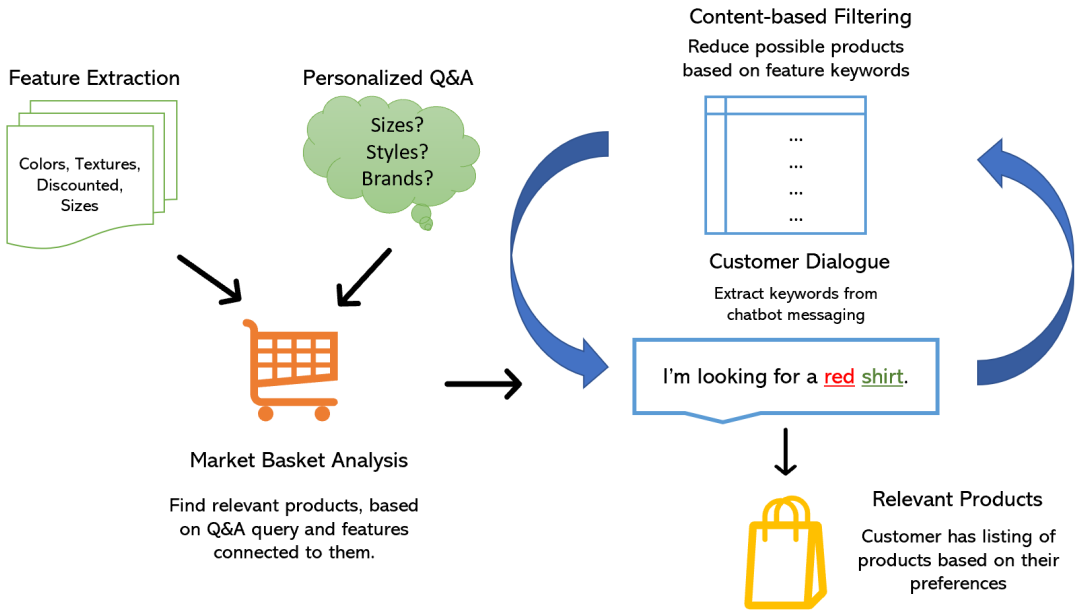

Figure 2: Proposed system architecture

Lastly, we use the filtered features, and use those to directly query products found in the database after MBA. In order to not completely drop all the products, we add another hyper-parameter that adjust the rate in which queried values are dropped from the MBA database. Finally, after running through each chat-bot message, we have the final set of products to recommend to the user.

Experiments

E-commerce Dataset

To test our system architecture, we the the following e-commerce fashion datatset (`https://www.kaggle.com/datasets/mukuldeshantri/ecommerce-fashion-dataset`), which includes nearly $30,000$ women's fashion products. These include clothing, jewellery, perfumes, and other relevant products.

Pre-processing In the e-commerce dataset, the features we are interested in are $BrandName$ (string-format of brand name for each product) and $Details$ (string-format of description of product). The primary issue we face is extracting features, since there is no clear order between elements such as textures or styles in the $Details$ column. We handle this by using simple natural language processing techniques to separate each word, then use techniques such as dropping irrelevant words, numbers, empty strings etc. What we are left with is a space-separated string for each product, making it straightforward to apply a matching criteria for each substring, in order to create more detailed feature columns. In our case, we generate four new columns, $Colors$ (red, blue), $Textures$ (cotton, polyester), $Styles$ (festive, sports), $Types$ (shirt, jeans). After using the top-100 features to fill in the new columns, we are left with $1/3$ of the database containing at least one new feature for each column, while less than 1% of the products do not contain any new features. We may be able to address this with more optimal methods of feature extraction, however, our system architecture does not rely on all features being available for a product to be chosen.

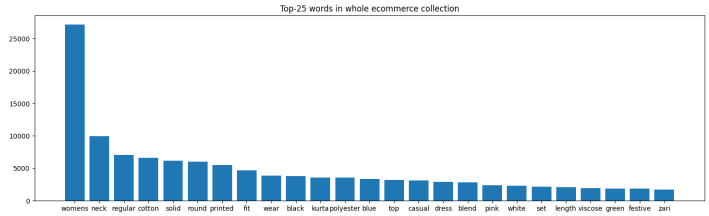

Figure 3: Top-25 features found in Details

BrandName	Details	Sizes	MRP	SellPrice	Discount	Category
life	solid cotton blend collar neck womens a-line dress - indigo	Size:Large,Medium,Small,X-Large,X-Small	Rs\n1699	849	50% off	Westernwear-Women
only	polyester peter pan collar womens blouson dress - yellow	Size:34,36,38,40	Rs\n3499	2449	30% off	Westernwear-Women
fratini	solid polyester blend wide neck womens regular top - off white	Size:Large,X-Large,XX-Large	Rs\n1199	599	50% off	Westernwear-Women
zink london	stripes polyester sweetheart neck womens dress - black	Size:Large,Medium,Small,X-Large	Rs\n2299	1379	40% off	Westernwear-Women
life	regular fit regular length denim womens jeans - stone	Size:26,28,30,32,34,36	Rs\n1699	849	50% off	Westernwear-Women

Figure 4: Original Dataset

Evaluation Metrics

We use three evaluation criterias that measure relevancy and capacity to produce better results than direct querying. These metrics are run on the final selection of products generated by the input Q&A and list of chat-bot messages.

Score and Score Capacity The first criteria we use is the Score s metric, this uses the same method as $Cosine - Similarity$, however it takes the mean of all similarity measurements based on the direct prompt given by Q&A. When $s = 1$, all selection of products have the same features as the Q&A prompt, while a lower score is how much further it is from the prompt. However, the score metric on it's own is expected to be similar to the cosine-similarity found during MBA, since they both use the same computation. To remedy this, we divide the score by the proportion of total found features divided by how many features are found when directly querying the Q&A responses, this gives us the Score Capacity σ metric:

$$s = \frac{\Sigma_{product_i}\text{Cosine-Similarity}(product_i)}{N}, \text{for } i \cdots N$$

$$\sigma = \frac{s}{\frac{\text{total \# of products found via model}}{\text{total \# of products found via querying}}}$$

$\sigma = s$: Results good as querying

$\sigma < s$: Results find more than querying

$\sigma > s$: Results find less than querying

The greater than score capacity, the more potential it has to find a subset of the database that catches both variety and relevance, while narrowing down results.

Query Distance We use one last metric, Query Distance ω which is a direct computation of the difference between total number of products found in model and querying, this gives us a measure of the variability with respect to the queried results:

$\omega = 0$: Results good as querying

$\omega > 0$: Results find more than querying

$\omega < 0$: Results find less than querying

Results

To test our results, we use five sample points, each containing information a filled Q&A section, each containing specific features found in a randomly selected product. As well as two chat-bot messages for each sample, containing sentences that include features to extract via KeyBert.

Hyper-parameters

There are three hyper-parameters we look at: lift $\lambda = \{0, \infty\}$, similarity threshold $\alpha = \{0, 1\}$ and the KeyBert+Content-based filter rate $\beta = \{0, 1\}$. Lift is used to adjust the lift value acceptable during MBA, similarity threshold is the drop rate for comparison of MBA products and the Q&A query, and lastly, the KeyBert filter rate, is the rate of dropping out specific values found in the filtering process, a higher filter rate corresponds to more values being dropped each message.

	score	score_capacity	query_dist
0	0.730395	0.391537	0.302030
1	0.713458	1.265813	-0.279070
2	0.774597	53.253521	-0.971326
3	0.718046	4.936565	-0.746032
4	0.774597	9.261482	-0.845638

Figure 5: Regular ($\lambda = 1.0, \alpha = .7, \beta = .1$)

	score	score_capacity	query_dist
0	0.605958	0.063724	0.809689
1	0.599482	0.130943	0.641460
2	0.637357	1.510976	-0.406650
3	0.609079	0.886226	-0.185345
4	0.634567	0.162030	0.593195

Figure 6: Less Strict ($\lambda = .8, \alpha = .6, \beta = .1$)

	score	score_capacity	query_dist
0	0.738383	0.562480	0.135220
1	0.713020	1.324868	-0.300236
2	0.774597	53.253521	-0.971326
3	0.712717	5.025565	-0.751592
4	0.774597	30.430583	-0.950355

Figure 7: More Strict ($\lambda = 2.0, \alpha = .7, \beta = .1$)

	score	score_capacity	query_dist
0	0.759720	0.288568	0.449449
1	0.713714	0.658629	0.040140
2	0.774597	53.253521	-0.971326
3	0.727364	2.469446	-0.544944
4	0.774597	1.314902	-0.258581

Figure 8: High Filtering ($\lambda = 1.0, \alpha = .7, \beta = .9$)

Impact of Hyper-parameters

As seen in the figure below (Figure 8), there are a few key observations we can take from the evaluation of this model.

- Given a well-adjusted similarity threshold α, drops the most values from the original dataset
- Otherwise, a low α drops too many values, scoring similarly, to querying
- It is proven that the score $s \approx \alpha$, for high values of s, otherwise defaults to a greater value (Less Strict).
- Filtering more values β makes the total number of recommended products, close to that for querying.
- A high score capacity is proportional to a negative query distance $\sigma \propto -\omega$

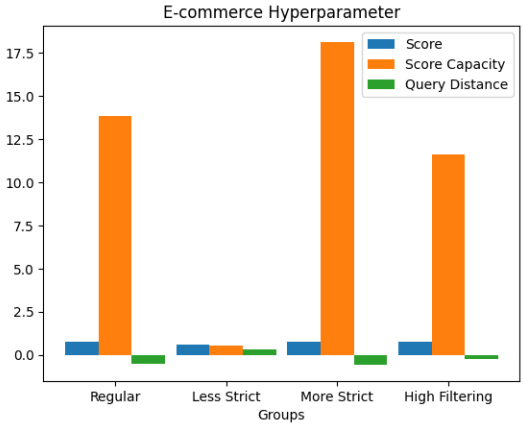

Figure 9: Average evaluation of each set of hyper-parameters

Related Works

Lastly, in the modern day, there are already thousands of techniques for building recommender systems, therefore it is necessary to do a comparison between our work and others that draw ideas for tuning chat-bots. One approach is Inductive Conformal Recommender Systems, the goal of this algorithm is to calculate uncertainty measures in each product recommendation[3]. This lends to the idea of significance and an error bound, in terms of whether a product would be realistically chosen. This is a very similar approach to our cosine-similarity, give that both measurements try to find a level of "error" between a set of inputs and their recommendations. The second approach we look at are Abductive-Deductive Chatbots, that uses the method of First-Order logic in order to infer new knowledge from a database[5]. These use highly strict logic rules to maneuver through a knowledge-base containing many loosely connected clauses. Although we do not use a knowledge-base dataset, a connection between our model is notably that clauses attempt to infer new data, similarly to how MBA looks to find associations in data, we use the 2-combinations in MBA in order to find as many possible connections, while still maintaining a level of relevance in products selected.

Conclusion

In this paper, we proposed a system architecture that is capable of taking the advantages of direct querying and the complexity from machine learning methods. We address methods in build an "optimal" model, and how they find results that improve upon the worst and best case scenarios. Furthermore, we examine different a variety of hyper-parameters that allow us to adjust the final results, in order to find a better structured architecture. Based on the results, we believe that the "More Strict" set of hyper-parameters, allow us to find a variety of relevant products, while also cutting down on the randomness factor involved when doing association. The current model we propose shows that it is reasonable to reduce the complexity of a model, while still finding results that maintain user personalizing. Further work, may include other measurements for association and sub-models in order to find a compact subset of products, language models for extracting features from the original database, as well as different scoring methods for evaluating relativity to direct querying.

References

1. *Academic Paper*
   Amila Silva and Ling Luo and Shanika Karunasekera and Christopher Leckie eds. 2020. *OMBA: User-Guided Product Representations for Online Market Basket Analysis.*

2. *Online Article*
   Harshil Patel eds. 2021. *What is Feature Engineering — Importance, Tools and Techniques for Machine Learning.*

3. *Academic Paper*
   Venkateswara Rao Kagita and Arun K Pujari and Vineet Padmanabhan and Vikas Kumar eds. 2021. *Inductive Conformal Recommender System.*

4. *Academic Paper*
   Sahar Moradizeyveh eds. 2022. *Intent Recognition in Conversational Recommender Systems.*

5. *Academic Paper*
   Carmelo Fabio Longo1 and Corrado Santoro eds. 2022. *A Framework to build Abductive-Deductive Chatbots, based on Natural Language Processing and First-Order Logic.*

6. *Academic Paper*
   Xinnian Liang and Shuangzhi Wu and Mu Li and Zhoujun Li eds. 2021. *Unsupervised Keyphrase Extraction by Jointly Modeling Local and Global Context.*