Recommender systems are increasingly popular for consumer services such as Amazon, Netflix and Spotify. One can hardly think of a widely adopted product that is not driven by recommendations. Yet, no popular consumer service that enables consumers to order food incorporates a recommendation engine in a major way.

Background

The initial goal of this project was to use a small dataset of 10,000 users and their food preferences obtained from a food ordering app, to build a recommender system that recommends different food items that the user has not tried. After reading the recommender engine for small datasets literature, and implementing two different models, it become clear that content based recommendation would provide the best quality of recommendations. To work well, content recommenders require detailed tags. In this particular case, the dataset did not contain tags about the food, say its portion size, spice level or temperature. While this information could have been collected, it would have been onerously time consuming and outside the scope of this 490 project.
As a result, instead of recommending items that a user has not tried before, the recommender system developed and deployed focuses on food items that pair well with one another or items that are frequently bought together. This feature is akin to one that Amazon has adopted quite successfully. Across their shopping cart, emails and product pages, Amazon employs recommendations presented to the user as “frequently bought together” or “customers who bought this item also bought”. Similarly, the recommender system developed provides the user with recommendations of food items that users frequently purchased together.

Overview

An initial goal of this project was to develop a recommender system that was deployed to real world users as well. To that end, I was able to use existing menus and photographs, as well as starter front-end code, to then develop a full functional app to process in person orders at a restaurant. This app had the menu of a given restaurant, to which the ability to process credit card payments and thereby accept orders was added. Now that users are able to place orders, the recommender service is able to provide recommendations of items that are frequently bought together to someone ordering in person at the store. In addition, the existing UI can be simplified now that the recommender system provides items frequently bought together.
after the user clicks the first item. Upon selecting the first item, a user can easily choose from among the other options recommended to them, rather than going back and re-searching for other items they may like.

Importance & Real World Use Cases

Furthermore, this project taps into and furthers a growing trend where physical retailers are increasingly looking to enhance the customer experience using data collected through other means. Here, while the bulk of the data is collected through the primary app, it is still used to build recommender engines that improve the in store experience through a separate app. Consequently, the powerful consumer trend of personalization & recommendation can be brought to the physical, “logged-out” experience of ordering at a restaurant. In addition, small chains and mom & pop restaurants are nowhere close to offering such features and experiences to their customers. To compete with larger restaurant chains like Dominos and McDonalds, they are quite receptive and eager to adopt apps that allow them to extend their current capabilities. Dominos for example, re-engineered their entire kitchen to be API-friendly to facilitate a faster and more transparent delivery experience through their customer app. McDonalds on the other hand has begun reducing labor costs by adopting self-serve checkout experiences, where customers can place orders and pay for them as well without the need for employees to interact with them. While these features
help larger chains become even more efficient, they leave smaller restaurants behind and less consumer friendly. With the recommender system and app developed through this 490 project, which from the start has been focused on real world adoption, the hope is smaller restaurants are better equipped to compete with the technology that has so far been out of their reach.

**User Experience**

To understand the recommendation service and the user experience it facilitates, it is easier to first understand the larger app that it is housed in. The app is delivered through an iPad located in a restaurant. Before a user begins using the app, it beckons them with a large tap here to order sign. As they then tap anywhere on the screen, the app displays an entire menu of the restaurant. The menu has photographs of items, as well as a clear distinction of which categories they fall under. A user can search using these categories or they can search for specific items using a search bar. By combining search and categories with a clear list of photographs and bold names they can explore, the app is conducive to both the customer experience of ordering a specific item, as well as browsing and comparing several different options.

As a user selects the first item, they begin to see recommendations for similar items that were purchased frequently by other customers. Often, these recommendations provide a 1-click way for the user to add an item they
would have otherwise taken 3-4 clicks to add. Other times, these items recommended provide options they might consider in this purchase or in the future.

Once the customer finalizes their entire order, they can checkout using a credit or debit card associated with the PayPal Here reader attached to the iPad. Currently, the checkout experience would need some more work to be more robust to the numerous failure modes of a card transaction failing or being processed incorrectly. Only with a higher level of robustness would the app function by itself in a live restaurant. But once a successful credit card transaction is processed, the order is pushed to an existing API. That API and related server processes the transaction and pushes it to an iPad in the restaurant’s kitchen where it will be serviced for the customer. The ultimate user experience provided by this self serve app powered by recommendations is similar to that of the Amazon bookstore — they’re able to use their rich online dataset to better curate an in-person experience for their customers. Similarly, the recommender system is the first step to use data collected from the primary app to design a better user experience that extends far beyond the primary consumer app.
Constraints on the Recommender System

Recommender systems are seldom adopted in a physical experience, especially where the customer is logged out and there is limited information about their prior tastes and preferences. So along with the design choices made for speed, scalability, memory and quality of recommendations, it is important to keep in mind the information constrained environment the recommender system deals with.

The initial recommender system was aimed at recommending items across the full range of restaurants. Towards that goal, design considerations centered around speed and memory, given a large search space. Ultimately, collaborative filtering was not an effective strategy. Given the atypical sparsity from the small user base, the quality of recommendations generated was of a low quality. Instead, the new recommender system focuses on generating recommendations at a given restaurant. In addition, since each user is not logged in, there is little history to leverage and no easy way to identify users without reducing the quality of the core ordering experience. As a result, rather than focusing on a collaborative filtering model that compares one user to other similar users, the recommender system adopted instead compares items that are similar to one another.
Architecture

The seminal paper published by Amazon in 1998, which introduced the idea of Item to Item collaborative filtering and was aptly titled and patented as, "Collaborative recommendations using item-to-item similarity mappings" [1], forms the basis of this recommender engine as well. For a given restaurant, there are usually no more than a 100 different items with their add-on variants considered as well. In addition, there are usually at least 1,000 users for a given restaurant. Consequently, for most restaurant order histories examined, there were at least a few users that ordered each item variant. This allows the recommender system to consider all item variants in the menu while making recommendations.

The recommender system is modeled off of the Item to Item Amazon paper [1]. While the paper describes one large n*n matrix constructed for item recommendations, this recommender engine’s key difference is in the design of the matrix. While Amazon would want users to purchase items from across the entire store, across all categories, the analogous would not make sense in this setting. Instead, one n*n matrix is constructed for each restaurant. Thereby, the recommender system can serve similar items from the bounded set of one restaurant.
Implementation

As the paper goes on to describe, the implemented training algorithm is presented above. At the moment, these matrices are trained using data from the primary app under the assumption that user preferences are largely the same. Nonetheless, there may be subtle differences between preferences, and as more data is collected from the in restaurant app, the matrices will be re-trained using data directly acquired as a result of the recommender system. At the moment, all training is performed offline, once a day to keep the recommendations fresh and relevant. The first time the matrices are generated, the entire database is iterated over to increment co-occurrence counts of 2 items being purchased by the same user. This is a time intensive operation, taking up to $O(n^2 \times m)$, where $n$ is the number of users and $m$ is the number of item variants in a restaurant. Furthermore, this computation occurs in effect everyday — updates to cooccurrence counts are easy and
quick, however the similarity computation using cosine similarity is still the bottleneck, where there isn’t a straightforward algorithm to use the previous days matrix.

**Design Tradeoffs**

Overall, this design for the recommender system works well. While setting up the data pipeline and associated cron system for daily updates isn’t the most ideal, it is a perfectly good tradeoff. The recommender system can take between a few seconds and a few minutes to train offline currently, and it may take even more computer power in the future. But the positive takeaway achieved from that tradeoff is an $O(1)$ recommendation service time. Given an item and restaurant posted to a REST API, the recommender system looks up top items in the row represented by their chosen item. While the memory used by this system is $O(r*n^2)$, where $r$ is the number of restaurants and $n$ is the number of item-variants on average, the $O(1)$ serving time makes the slightly higher memory footprint well worth it. In any case, it is likely that the recommender service will be overhauled with more user information in the future, well before the memory footprint is a limiting factor.

**Next Steps**

There are numerous improvements to be made to the recommender system in the future, to improve the quality of recommendations and provide...
a better user experience. As mentioned, the matrices are currently built with
data from a secondary app instead of the primary in-store purchases app.
There may be a boost in quality of recommendations when there is enough in
person data collected, albeit this is expected to be minor. The main source of
in app information that is likely to make the biggest difference would be the
recommended items that are not acted upon. For example, if 3 items are
recommended to a user and only one is added to their cart, that particular
should receive a small boost in its weighting and other two items should
receive a small dip in their weighting conditioned on the initially chosen item.
As a result, the recommender system may become dynamic rather than the
rigid weighting scheme it uses which is based solely on cooccurrence counts.
Furthermore, tracking metrics such as click through rate on a recommended
item and conversion rate on a recommended item will help tune “hyper-
paramenters” such as the number of recommended items to show.

With a built out app and recommender system, I’m looking forward to
polishing the interface of the app, making it more robust, and then releasing it
at a few locations to gain valuable user data & feedback to iterate on.