Building Recommendation Engine for Starbucks users - Capstone Proposal

I. Definition

Project Overview

Starbucks is one of the most well-known companies in the world. Starbucks Corporation is an American multinational chain of coffeehouses and roastery reserves headquartered in Seattle, Washington. As the largest coffeehouse in the world, Starbucks is seen to be the main representation of the United States' second wave of coffee culture. A chain with more than 30 thousand stores all over the world. It strives to provide the customers the best service and the best experience.
Starbucks offers his free app to make orders online, receive special offers and collect bonuses.

This project aims to optimize the customers' experience with improving the App by means of predicting and spreading the offers that will be definitely interesting for each particular customer.

Starbucks wants to find a way to give to each customer the right in-app special offer. There are 3 different kind of offers: Buy One Get One (BOGO), classic Discount or Informational (no real offer, it provides informations) on a product. Each customer is receiving offers via different channels and can ignore, view or respond to an offer.

The goal is to analyze historical data that represent user behavior, user details, user purchases and rewards gained via responding to the offers.
We provide analysis and build the model that can be able to give one-to-one offer recommendations, that should increase offer response rate by spreading more user customized offers.

The baseline, the main Recommendation Engine and alternative -Performance optimized models will be built and comparison between the models will be performed.

Problem Statement

The customer want to receive the offers that are the most appealing and customized. We will strive to improve the conversion of each offer by sending only the relevant offers to each customer.
This might help to improve retention and app usage rate by decreasing rate of ignoring offers being sent.

Metrics

The metrics used to evaluate the machine learning model is very important.
Choice of metrics influences how the performance of machine learning algorithms is measured and compared.
Let's next describe the metrics that are used for classification models in general and the ones used in our recommendation model in particular.

Confusion matrix

The Confusion matrix is one of the most intuitive and easiest metrics used for finding the correctness and accuracy of the model.
It is used for Classification problem where the output can be of two or more types of classes.

The Confusion matrix itself is not a performance measure as such, but almost all of the performance metrics are based on Confusion Matrix and the numbers inside it.

Terms associated with Confusion matrix:

1. *True Positives (TP)*: True positives are the cases when the actual class of the data point was 1(True) and the predicted is also 1(True)
   Ex: The case where a person is actually completed an offer and the model classifying his case as completed - counts under True positive.
2. *True Negatives (TN)*: True negatives are the cases when the actual class of the data point was 0(False) and the predicted is also 0(False
   Ex: The case where a person NOT responded and the model classifying his case as ignored - counts under True Negatives.
3. *False Positives (FP)*: False positives are the cases when the actual class of the data point was 0(False) and the predicted is 1(True). False is because the model has predicted incorrectly and positive because the class predicted was a positive one. (1)
   Ex: A person NOT responded to an offer (ignored) and the model classifying his case as offer completed - counts as False Positives.
4. *False Negatives (FN)*: False negatives are the cases when the actual class of the data point was 1(True) and the predicted is 0(False). False is because the model has predicted incorrectly and negative because the class predicted was a negative one. (0)

The ideal scenario that we all want is that the model should give 0 False Positives and 0 False Negatives. But that's not the case in real life as any model will NOT be 100% accurate most of the times.

We know that there will be some error associated with every model that we use for predicting the true class of the target variable. This will result in False Positives and False Negatives (i.e Model classifying things incorrectly as compared to the actual class).
There's no hard rule that says what should be minimized in all the situations. It purely depends on the business needs and the context of the problem you are trying to solve. Based on that, we might want to minimize either False Positives or False negatives.

Accuracy

Accuracy in classification problems is the number of correct predictions made by the model over all kinds predictions.
Accuracy is a good measure when the target variable classes when class data values are nearly balanced.

Precision

Precision is a measure that shows the amount of positive predictions that were correct. Calculation: True Positives / number of predicted positives.

Recall or Sensitivity:

Recall is a measure that shows the percentage of positive cases that were predicted overall.
Calculation: True Positives / number of actual positives

F1 Score

F1 - is a single score that kind of represents both Precision(P) and Recall(R).
One way to do that is simply taking their arithmetic mean. i.e (P + R) / 2 where P is Precision and R is Recall. But that's pretty bad in some situations.

II. Analysis

Data Exploration

The data is contained in three files:

- portfolio.json - containing offer ids and meta data about each offer (duration, type, etc.)
- profile.json - demographic data for each customer
- transcript.json - records for transactions, offers received, offers viewed, and offers completed

Here is the schema and explanation of each variable in the files:

portfolio.json

- id (string) - offer id
- offer_type (string) - type of offer ie BOGO, discount, informational
- difficulty (int) - minimum required spend to complete an offer
- reward (int) - reward given for completing an offer
- duration (int) - time for offer to be open, in days
- channels (list of strings)

profile.json

- age (int) - age of the customer
- became_member_on (int) - date when customer created an app account
- gender (str) - gender of the customer (note some entries contain 'O' for other rather than M or F)
- id (str) - customer id
- income (float) - customer's income

transcript.json

- event (str) - record description (ie transaction, offer received, offer viewed, etc.)
- person (str) - customer id
- time (int) - time in hours since start of test. The data begins at time t=0
- value - (dict of strings) - either an offer id or transaction amount depending on the record

Building recommendation matrix - Userdata

To understand a customer behaviour, we would need to extract customer actions given any particular offer.

We are going to build customer recommendation matrix by joining all given datasets, using pre-processed data using the method described below.

We extract the `offer_id` value from *transcript* dataset and join by this value with *portfolio* dataset, which contains all necessary data on each particular offer.
Next we join the same data with the rows present in *profile* dataset on person's `id` field.
This wall give us the full representations about how many offer have been received by each particular customer and details via which exact channels.
Also it would contain the data about how many rewards offer costs, the duration and type of an offer.

Next, we are going to filter the data by the latest action performed by a customer. Filtering details are described in the Feature Engineering section.

Feature Engineering

To be able to train our model we would need to encode categorical and continuous data in the form that is understandable by the machine learning model.

First of all, the `channel` field will be extracted into set of separate binary encoded values.
For example, for `mobile` channel: `0` will define that mobile channel has not been uses for an offer, while `1` will define that this channel was used during advertising campaign.

Additionally we will also extract how many days each customer has been a member as a value `member_days` that could help us to make correct prediction.

To increase the performance for our models we would also perform binning of the values in `income` field.

Values from the `gender` proved to be important feature and values will be mapped to categorical values, as the following:

Gender - X	Gender - 0	Gender - M	Gender - F
0	1	2	3

Gender `X` would represent the data that have been left blank in customer profile. It almost alway comes hand-in-hand with blank value in `income` field.

Usually such data can be marked as missing data and dropped from the dataset, however we threat this data as category and will try to extract insight from such data as well. Especially considering the fact, that some customers prefer not to specify their gender or income, however we would like anyway to provide such customers with a valid offers and hopefully increase the retention of such customers and turn them into real ones. As the data exploration clearly indicates that such customers almost always view an offer and sometimes even respond to it.

Next, we would process `event` field, which provide us with insight on how exactly customer has responded to an offer.
The field contains the following values:

- offer received if the offer was received by the customer,
- offer viewed if the customer has, at least viewed an offer,
- offer completed if customer has responded to an offer and made a purchase in store or via an application.

To be able to build customer recommendation matrix we would need to extract the latest action made by the customer, for each and given offer, either it was ignored viewed or responded. We would like to keep only the latest action performed by the customer.

Algorithms and Techniques

Everyday we deal with online platforms that use recommendation systems. There are different approaches to implement such systems and it depends on the product, the available data and more. Recommendations usually based on the history of the user, similarities with other users or other signals collected about the user behavior. In practice, different signals and algorithms are usually merged to get better results. But let's focus on one approach; item-based collaborative filtering. Using this approach, we learn about the similarities between the offers and response to offers by particular users from the outcomes we already have in the data.

Ideally, with a good model, users/offers close to each other in the space should have similar characteristics.

We will use this idea and tabular data extracted as described in `Building recommendation matrix – Userdata` section as tabular data method where we use the embedding matrices as lookup tables to get the vectors corresponding to each user/offers. These vectors will be considered as features and we can add other fully connected layers, or even try something else other than neural networks.

Instead of the traditional ways, we can use embedding layers and learn about the latent factors while training our neural network using the given ratings.

As a result we will have:

- Users embedding (mXk)
- Offers embedding (nXk)

In this study we will be building multilayer Neural Network (NN) with Embedding layers that would strive to represent user/offer matrix with additional parameters that would be beneficial for NN Model to extract insights from the data.

We also have been testing with various values for batch size, dropout with several optimizers and learning rates. Among which SGD (Stochastic Gradient Descent) optimizer with momentum proved to achieve lowest loss across train and validation data.

For example Recommendation Engine with the same topology and batch size achieved accuracy of *68,3%* with SGD and only *38,3%* with Adam optimizer (which is a standard optimizer choice for most use-cases).

Benchmark

As the benchmark we will use recommendation engine that just learned on offer and user id and as the recommendation parameter an `event` values will be used.
During the final evaluation step, we perform hyperparameter tuning over less complex tree-based models and will compare the obtained results.