Machine Learning Engineer Nanodegree

Capstone Project - Building Recommendation Engine for Starbucks users

Oleh Bodunov
April 22, 2020

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.

Which one to use

It is clear that recall gives us information about a classifier's performance with respect to false negatives (how many did we miss), while precision gives us information about its performance with respect to false positives(how many did we caught).
Precision is about being precise. So even if we managed to capture only one cancer case, and we captured it correctly, then we

are 100% precise.
Recall is not so much about capturing cases correctly but more about capturing all cases that have “cancer” with the answer as “cancer”. So if we simply always say every case as “cancer”, we have 100% recall.
So basically if we want to focus more on minimizing False Negatives, we would want our Recall to be as close to 100% as possible without precision being too bad and if we want to focus on minimizing False positives, then our focus should be to make Precision as close to 100% as possible.

Both precision and recall output a value between 0 and 1. What we would like is a high precision and a high recall, but this is very rarely the case.
By testing a range of models and plotting their precision and recall values, their curves will give you an idea of the ideal tradeoff you should be aiming for.
One way to tell if our algorithm is biased towards a positive class is if we have a very low precision, but a very high recall.

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.

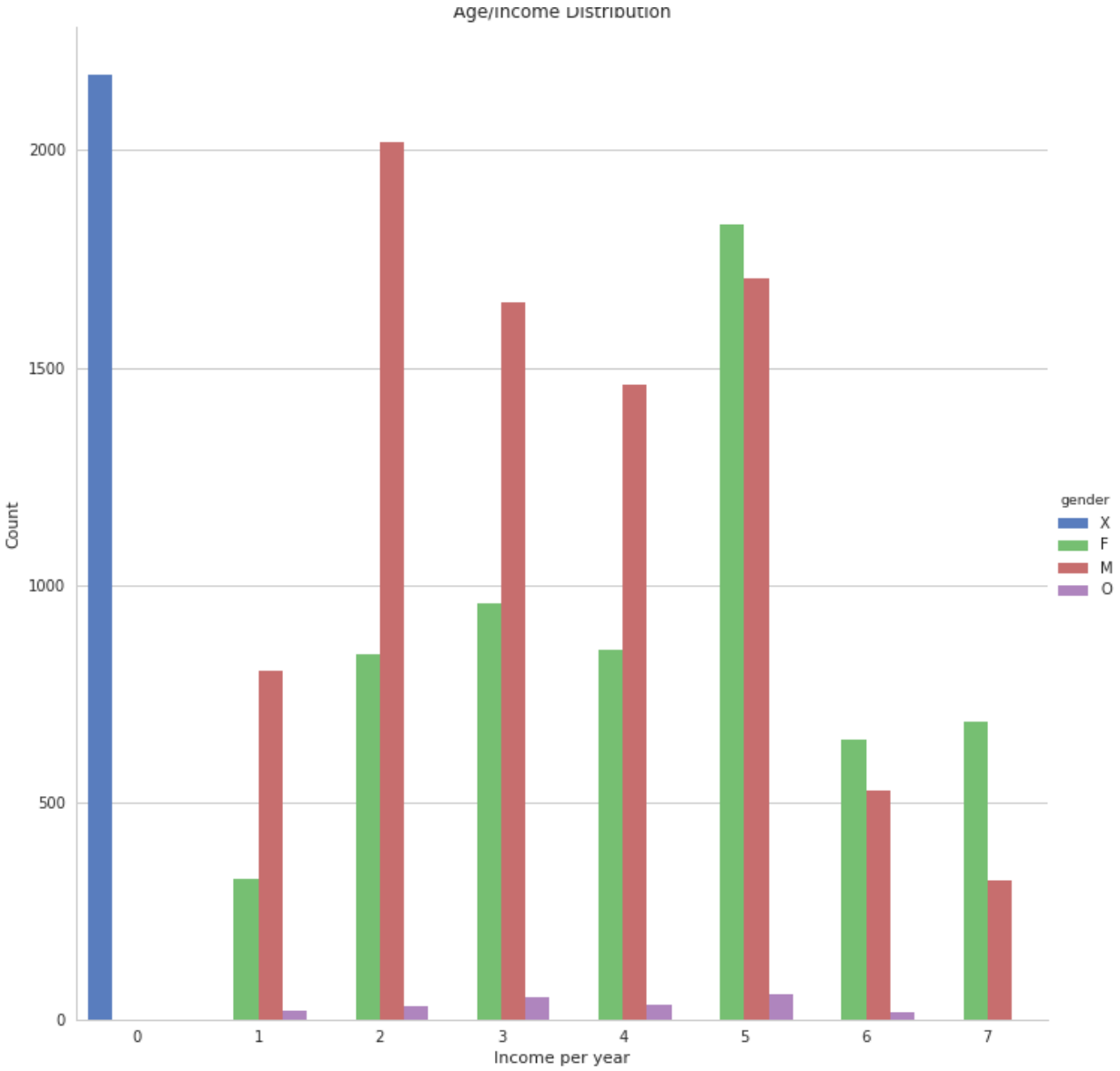
Next, we would process `event` field, which provide us with insight on how exactly customer has responded to an offer.
The field contains the following values:

- 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.

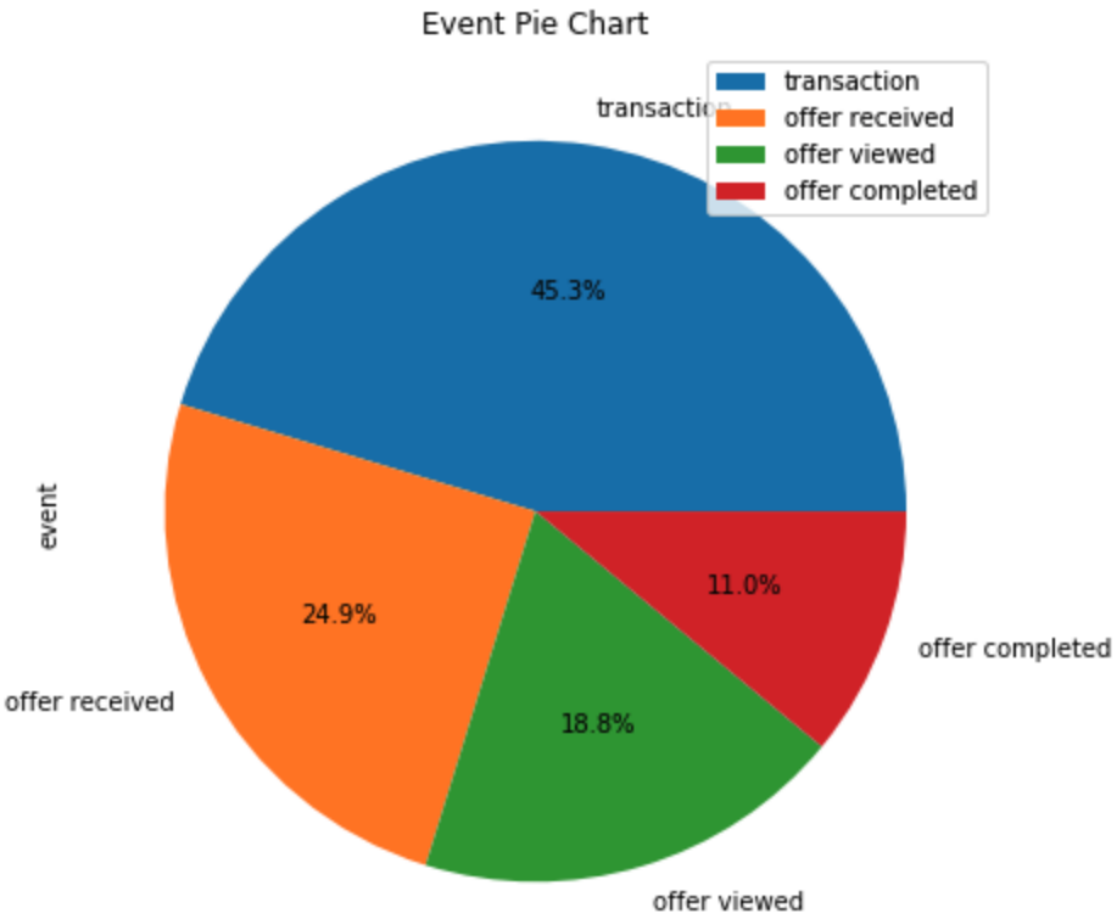
To be able to build customer recommendation matrix we would need to extract the latest action made by the customer, for each and given offer, either it was ignored viewed or responded. We would like to keep only the latest action performed by the customer.

Exploratory Visualization

First we visualize how income is aligned with gender and offer outcome.
The binned income amounts will be used, where 0 - is all user that have `None` as income, and 7 - is the income above 100.000.

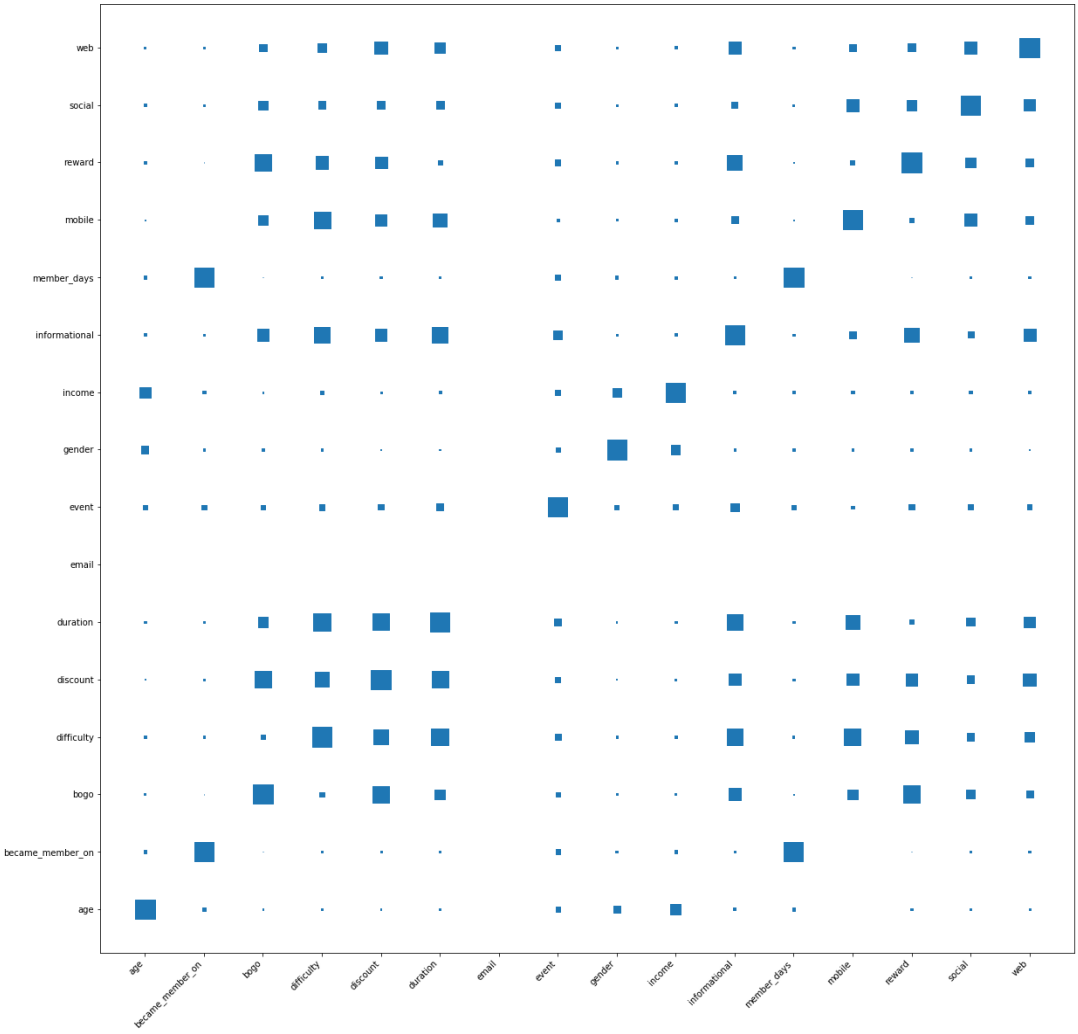


Interesting to note that all users that decided to to specify age are also left income per year unspecified. And the amount is quite high as is shown on the above bar plot.
Men also tend to earn more than women, while women tend to respond to an offer more often.

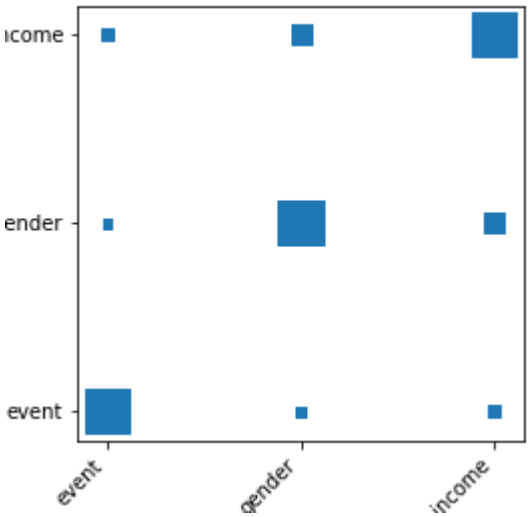


Correlations

Correlation is used to find which values are closely related with each other. A simplified view on correlations is shown below:



Correlation between features seems to be quite weak. However it can be noted that `bogo` is strongly related to `discount` and `reward` fields, while `mobile` channel is correlated with `difficulty` field. Which is quite expected.



If we look closely how event outcome is related to gender or income we can notice that correlation is quite weak, so other additional parameters should be definitely be taken into account.

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.

III. Methodology

Data Preprocessing

To be able any ML model to extract features from the data - values should be in proper format. Usually this is integer or float variables that represents actual or encoded values from the dataset.
As discussed in `Feature Engineering` section several fields will be encoded as binary values, values such as `income` can be distributed over some mean value with high variability for each particular row (particular user income). Such data can be `binned` - i.e categorized by the bins, where each particular income value will fall into (i.e low, high).

User and offer ids represent actual items that recommendations systems will be based upon, thus as discussed in `Algorithms and Techniques` section, this fields will be encoded as categorical values (integers) and will be used to build embedding matrix.

For the final performance accelerated models we will use less preprocessed data for benchmarking decision tree-based models which are very tolerant to unnormalized, continuous, or multidimensional data.

Implementation

Our recommendation engine model will have the following architecture:

```
Model(
  (bn_cont_u): BatchNorm1d(5, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn_cont_o): BatchNorm1d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (u_emb): Embedding(16994, 20)
  (m_emb): Embedding(10, 20)
  (layers): Sequential(
    (0): Linear(in_features=55, out_features=1024, bias=True)
    (1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Dropout(p=0.4, inplace=False)
    (3): ReLU()
    (4): Linear(in_features=1024, out_features=1024, bias=True)
    (5): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Dropout(p=0.4, inplace=False)
    (7): ReLU()
    (8): Linear(in_features=1024, out_features=512, bias=True)
    (9): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): Dropout(p=0.4, inplace=False)
    (11): ReLU()
    (12): Linear(in_features=512, out_features=256, bias=True)
    (13): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): Dropout(p=0.4, inplace=False)
    (15): ReLU()
    (16): Linear(in_features=256, out_features=128, bias=True)
    (17): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (18): Dropout(p=0.4, inplace=False)
    (19): ReLU()
    (20): Linear(in_features=128, out_features=3, bias=True)
  )
)
```

We are going to to use Batch Normalization and Dropout to be able to deal with overfitting of the model toward previously seen data.
These techniques proved to be useful in almost every complex ML model.

The following loss function will be used:

```
batch_size = 16
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.08, momentum=0.9)
```

Refinement

Initial model described in BaselineModel notebook achieved only 50% of accuracy, by further adding features that were additionally engineered the model had been gradually improved by 15-18%.
A refinement of the Recommendation Engine model has been performed by training the model on dataset without outliers (customers that have decided to anonymize income and gender fields). However doing this we achieved only 1-2% improvement.

Building performance optimized models

Also, an alternative approach was described using decision tree-based algorithms such as Random Forest and XGBoost which proved to be comparable and sometimes even better as our initial recommendation engine model. however, the more complex NN model can prove to perform and scale much better when number of users/offers will grow (millions with thousands offers). Starting with model that are 70-74% accurate an alternative model were defined.
The alternative approach aims to track offers that are mostly ignored by certain (or a group of users) and to make positive predictions if the offer has proven to be interesting for a customer. An alternative model achieved accuracy of 98,2 %.
This approach can be used as an ad hoc recommendation method or provide fine granular prediction on set of customers (big city or country with not many opened stores).

IV. Results

Model Evaluation and Validation

An evaluation of multiple models with several parameters had been performed. It was practically established that using different optimizers, as well as tuning network depth and batch size can improve model accuracy.
The most accurate models have the following results:

Accuracy
Best accuracy tested on randomly selected sample of data with 1000 rows.

Algorithm	Accuracy
Recommendation Engine 1	**68.36%**
Random Forest Model	**67.7%**
XGBoost Model	**70.7%**

With increasing the size of test data - accuracy grows by 3-5% for each model.

Accuracy of performance optimized models
Alternative approach that predict positive reaction or ignore action (negative) for an offer:

Algorithm	Accuracy	Precision
XGB Model	**98.0%**	0.96
Random Forest	**97.6%**	0.96

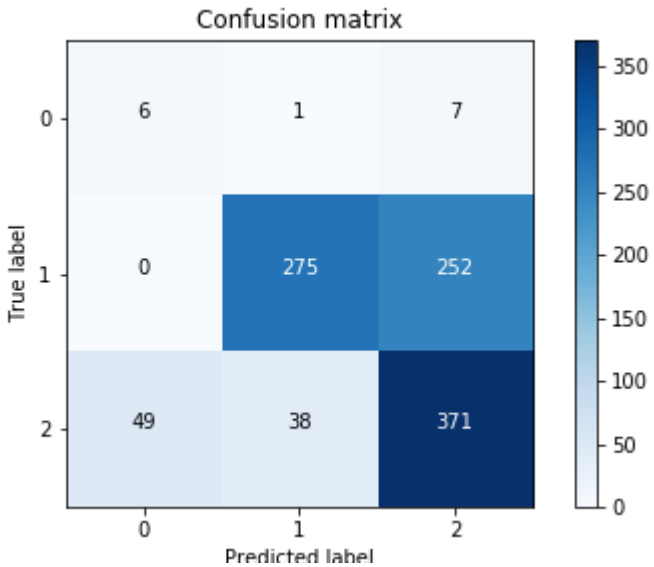
Metrics
Other metrics are the following.

Algorithm	F1 Score	Recall	Precision
Recommendation Engine	0.66	0.67	0.73
Random Forest Model	0.674	0.67	0.67
Random Forest Model	0.695	0.7	0.69

Confusion Matrix

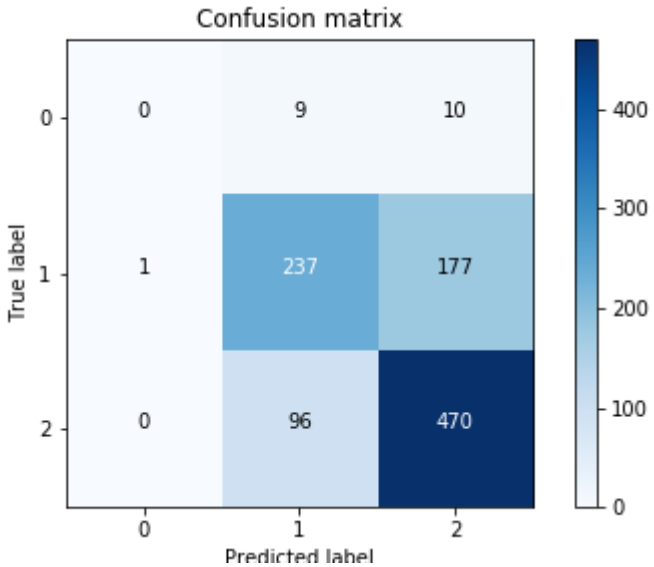
Recommendation Engine

For the classes [0,1,2] confusion matrix looks like the following:



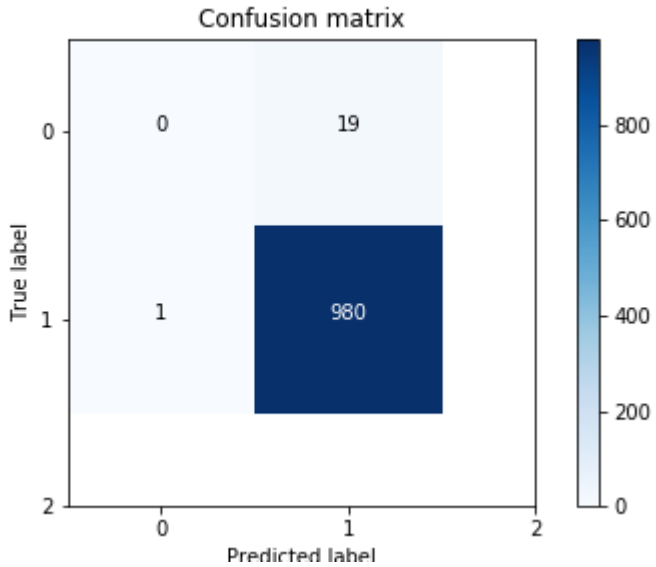
Where 0 - offer ignored, 1 - offer viewed, 2 - offer completed.

XGB Model



An alternative model can easily identify response/ignore outcome when binary value is used.
The model definitely can be used and can perform quick general prediction on large subsets of data.

Alternative Performance Optimized XGB Model



Justification

Compared to baseline model every model proved to gain much better performance and accuracy when trained using more additional features.

Fine-grained classification (if user will only view but not respond to an offer, or vise versa) is a challenging tasks due to very uncorellated data.

However if coarse-grained classification should be performed (yes/no question) it is possible to achieve 98% accuracy with approach described in this repository, by using performance optimized model.

V. Conclusion

Reflection

Let's summarize all the steps followed in this process.

Three different datasets provided by Starbucks containing information about offers in Starbucks app were used.
Datasets contain many information that can be extracted using feature engineering.
To be able to build recommendation system based on the provided datasets the `userdata` users/offers matrix was build. This matrix further was used to build and learn embedding matrix that represent relations between each user and each offer. Embedding layers had become part of NN Model and were trained along the neural network weight itself. The final model can predict the best event outcome, if received, for each user. This can be used to construct, test and then sent the most appropriate offers to the users.

This `userdata` dataset were later used to build Recommendation Engine Model that proved to predict with higher accuracy whether user will ignore or respond to an offer. Model proved even leveraging that fact if user will only view the received offer but actually will not perform any action later. As for example particular customer may view an offer and later spread the information to his/her friends and actually implicitly provide offer conversion. This particular feature can be later used by Sales or Marketing departments to decide the most appropriate channels.

Training such complex models is very challenging tasks. A lot of tuning and reflection on model performance should be made during and after the model is trained.
Also, selection of different parameters, optimizers as well as number of neurons in each layer always change the performance of the model and one proves himself/herself to be a real Machine Learning Engineer if this part can be leveraged with passion to explore.
Several Recommendation engines with topology that differ in size and depth were tested but the one that is described in `3–RecommendationEngine` notebook proved to be the most accurate one.
Using Batch Normalization, Dropout and smaller batch size for loading data for training also proved to be beneficial for this model.

Also it should be noted that for this particular case - representation of collaborative filtering as user/offer embeddings proved to simplify building and improving relations between users and offers that each particular user may like.

An additional coarse-grained model were introduced and tested which also proved to have high accuracy, achieving even 98% accuracy on sample data.

In summary, the model that were introduced in this study can be used in production as-is or with further hyperparameter tuning. However, all the models that we shown prove to be very accurate and easily can differentiate if an offer will be ignored or an action will be performed on the offer. Which is the main goal of any recommendation System.

Improvement

First of all, more hyperparameter tuning can be done on Recommendation Engine model. Especially one can test different Embedding sizes (the ones were studied in this work are 20, 50, 100). The construction and learning the weights of an embedding matrix is an ongoing research field by itself.

Next, an XGBoost on AWS Sagemaker Service can be leveraged to improve fine-grained XGB model (multiclass predictor, currently around 70% accuracy). By automating hyperparameter search the aforementioned model can be improved even more.

Additionally, further feature engineering can be done. For example leveraging number of transactions, average amount of transaction, time between user viewed and completed and offer can be extracted. Number of discounts and how long user is an active customer, as well as most active moths by a customer - these are some of the new features that can be extracted to further improve the model.
All these features are aiming to provide better insights on how much time particular user spend in the application and what drives him/her to respond to each particular offer.

Related works

Matrix Factorization
Collaborative Similarity Embedding for Recommender Systems
Regularizing Matrix Factorization with User and Item Embeddings for Recommendation