Using Starbucks App User Data To Predict Effective Offers

Machine Learning Engineer Nanodegree

Capstone Project Report by Vishad Bhalodia

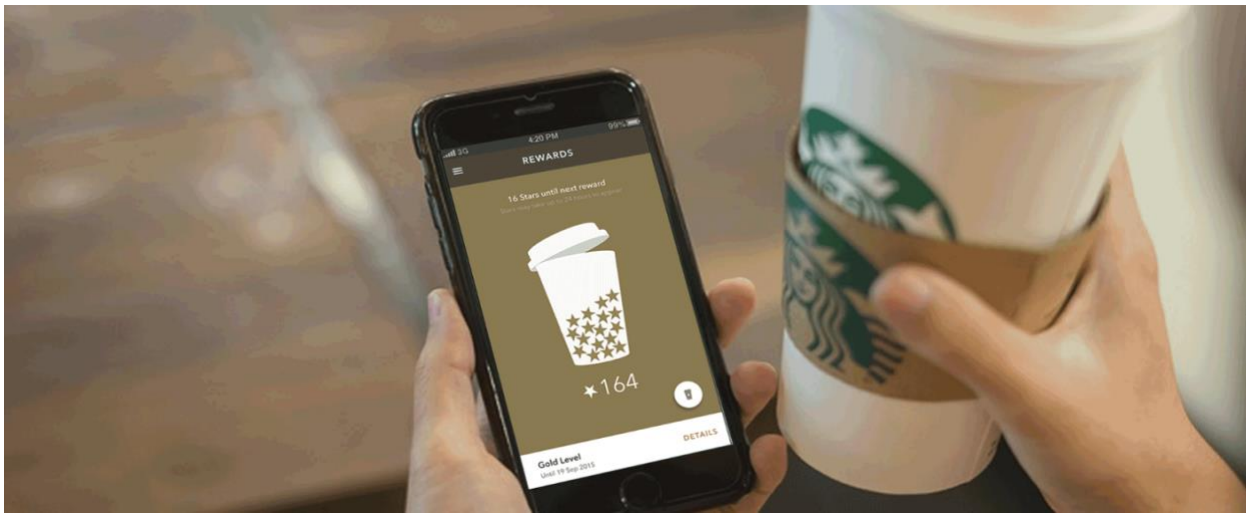


TABLE OF CONTENTS

1. Definition

1.1 Project Overview

Starbucks: Winning on rewards, loyalty, and data

This was the headline of an article[1] written by a Harvard Business School student which aptly summarizes Starbucks' marketing strategy that has reaped huge benefits in terms of market share and revenues.

Starbucks Rewards Program generates customer loyalty, increased revenue, and data for the company to create meaningful 1:1 relationships and personalized marketing efforts. Starbucks has attributed their rewards program for most of the increase in their revenues over the past 2 years.

There are three types of offers that can be sent: buy-one-get-one (BOGO), discount, and informational. In a BOGO offer, a user needs to spend a certain amount to get a reward equal to that threshold amount. In a discount, a user gains a reward equal to a fraction of the amount spent. In an informational offer, there is no reward, but neither is there a requisite amount that the user is expected to spend. The offers can be delivered via multiple channels: e-mail, social media, on the web, or via the Starbucks's app.

The data set used in this project is provided by Udacity and Starbucks as part of the Machine Learning Engineer Nanodegree program. It contains simulated data that mimics customer behavior on the Starbucks rewards mobile app. The program used to create the data simulates how people make purchasing decisions and how those decisions are influenced by promotional offers. Each person in the simulation has some hidden traits that influence their purchasing patterns and are associated with their observable traits. People produce various events, including receiving offers, opening offers, and making purchases. As a simplification, there are no explicit products to track. Only the amounts of each transaction or offer are recorded.

---

1 https://digital.hbs.edu/platform-digit/submission/starbucks-winning-on-rewards-loyalty-and-data/

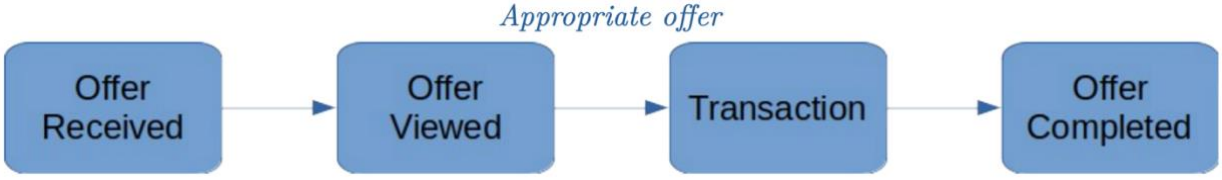
1.2 Problem Statement

In this project, as outlined in my project proposal, we set out to answer two major questions –

1. Use the data to identify which groups of people are most responsive to each type of offer and how it relates to their characteristics.
2. Build a model that predicts whether a customer will respond to an offer or not.

The business motivation behind these two questions are twofold: firstly, from the business perspective, Starbucks can better target their offers towards customers with higher propensity to take up the offers. In doing this, they have the potential to maximize on revenues obtained from offer take up to the right customers, plus they can save on marketing and promotional costs by sending more relevant offers.

Secondly, from the customer point of view, the customer gets a better experience with Starbucks app, as they will be receiving personalized, relevant offers.
In the context of this project, an effective offer is that one where the customer sees the offer received and buys products under its influence, completing the offer lifecycle.



There are many "paths" which lead to ineffective offers (detailed in Section 3 Observations of the companion Jupyter notebook) – here, we will only concentrate on the effective offers since we want to predict whether a customer will respond to an offer or not.

We will follow the following workflow for our solution:

1. **Data loading and exploration:**
   Load files and present some data visualization in order to understand the distribution and characteristics of the data, and possibly identify inconsistencies like missing values, data skewness and categorical features with too many categories.

2. **Data cleaning and pre-processing:**
   Having analyzed the data, we now apply several standard techniques to fix possible issues found. These include (but not limited to) missing data imputation, categories encoding, and data standardization.

3. **Feature engineering and data transformation:**
   Prepare the data to correspond to the problem stated and feed the machine learning algorithms. The transcript dataset must be structured and labelled with an effective offer label for supervised learning.
4. **Preparing different models and transform inputs for models:**
   After that, we develop the different models that we want to check our predictions on. We create different machine learning models using different algorithms such as Gradient Boosting, Logistic Regression, Support Vector Machine, K-Nearest Neighbors, and Random Forest.
5. **Model Prediction and Tuning for Improvement:**
   We then evaluate the models on accuracy and F1 score and analyzing the confusion matrix.
6. **Conclusion and future improvements:**
   Finally, we present the resulting predictions, along with justification on model performance and future improvements.

1.3 Metrics

The result that needs to be evaluated is whether we successfully predicted that the user responds to the offer or not. We will use different statistical measures like accuracy score, F1 score to evaluate the model predictions. We will also look at confusion matrix to analyze false positives and false negatives.

We will use a naive model that assumes all offers were successful as a baseline for evaluating the performance of models that will be used. Since the accuracy of the naive model is ~50%, we can target to achieve an accuracy of 70% for the classification problem.

2. Analysis

2.1 Data Exploration and Exploratory Visualization

This section presents some data analysis along with visualization (charts) in order to understand the distribution and characteristics of the data, and possibly identify inconsistencies. Data exploration is one of the most important parts of the machine learning workflow because it allows you to notice any initial patterns in data distribution and features that may inform how to proceed with modelling and clustering the data.

The data for this project is contained in three files:

1. **profile.json**

   Rewards program users (17000 users x 5 fields)

   - gender: (categorical) M, F, O, or null
   - age: (numeric) missing value encoded as 118
   - id: (string/hash)
   - became_member_on: (date) format YYYYMMDD
   - income: (numeric)

	gender	age	id	became_member_on	income
0	None	118	68be06ca386d4c31939f3a4f0e3dd783	20170212	NaN
1	F	55	0610b486422d4921ae7d2bf64640c50b	20170715	112000.0
2	None	118	38fe809add3b4fcf9315a9694bb96ff5	20180712	NaN
3	F	75	78afa995795e4d85b5d9ceeca43f5fef	20170509	100000.0
4	None	118	a03223e636434f42ac4c3df47e8bac43	20170804	NaN

This dataset presents gender as a categorical feature which should be mapped to columns (one hot encoding) – Also see that some users do not have a gender mapped (null), so we should remove those entries.

First step is to analyze the missing values in the dataset –

	Null count
gender	2175
age	0
id	0
became_member_on	0
income	2175

Checking the above table, we can observe null values in columns income and gender, as well as missing value encoded as 118 in the column age.

By analyzing dataset, it is possible to conclude that all the missing values occur in the same rows –

```
profile.query("age == 118")
```

	gender	age	id	became_member_on	income
0	None	118	68be06ca386d4c31939f3a4f0e3dd783	20170212	NaN
2	None	118	38fe809add3b4fcf9315a9694bb96ff5	20180712	NaN
4	None	118	a03223e636434f42ac4c3df47e8bac43	20170804	NaN
6	None	118	8ec6ce2a7e7949b1bf142def7d0e0586	20170925	NaN
7	None	118	68617ca6246f4fbc85e91a2a49552598	20171002	NaN
...	...	...	...	...	...
16980	None	118	5c686d09ca4d475a8f750f2ba07e0440	20160901	NaN
16982	None	118	d9ca82f550ac4ee58b6299cf1e5c824a	20160415	NaN
16989	None	118	ca45ee1883624304bac1e4c8a114f045	20180305	NaN
16991	None	118	a9a20fa8b5504360beb4e7c8712f8306	20160116	NaN
16994	None	118	c02b10e8752c4d8e9b73f918558531f7	20151211	NaN

We should remove them since we cannot fairly “estimate” these values – any estimation required expert knowledge. Thus, we’ll remove all nulls in the data cleaning step later.

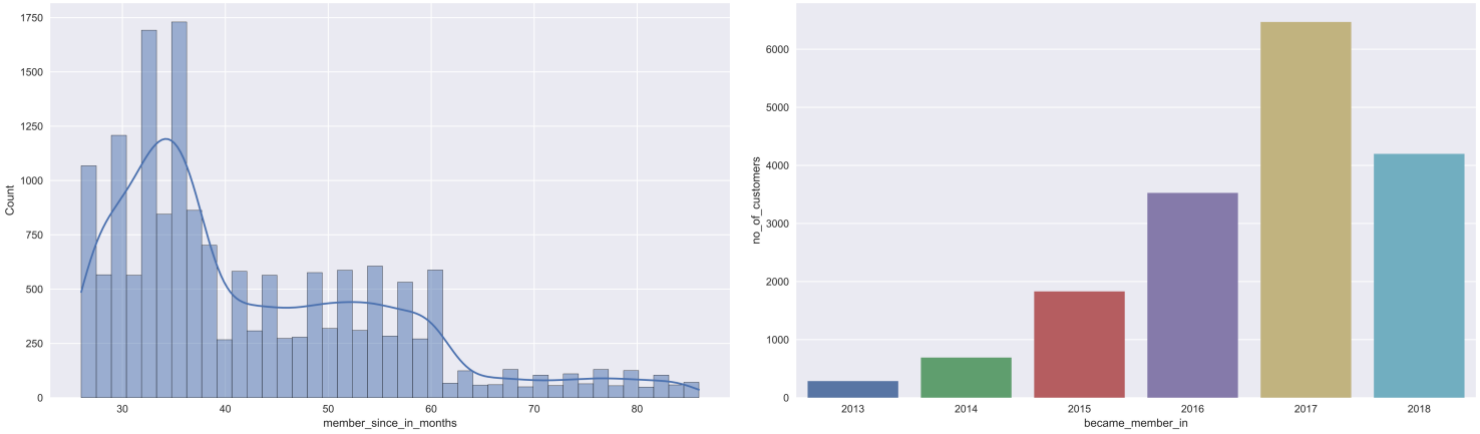
Now let’s check gender and income distributions. First up, income distribution –

This plot shows that income is not a well distributed feature – this means it could be a good feature for the model. The density plot is bimodal, including some other local peaks. Also, the plot shows that the distributions for both men and women are right skewed.

Checking the income by gender, we note that the number of men showing income below $80,000 is considerably higher than women.

Let’s check the gender – age distribution now –

Age distribution plot depicts that the median age of a customer is 60, in each gender, and most of the customers belong to the age range between 40 to 70. This also shows that the distribution seems bimodal there are two peaks – one around 20-30 age and one around 50-60 age. Since age is a discreet and not a continuous variable, we should add one-hot encodings for various age ranges as features (using pandas get_dummies method).



The above two plots show membership distributions – First, we convert the date of membership into months and years of membership wr.t an anchor date, which we took as 30-Sep-2020 for reference. Here, we see that the feature became_member_in (year, discrete) presents a left-skewed distribution, and has better explanatory power than the membership in months. There is a peak around 2017 and most of the users seem to have joined after 2016 only – presumably due to a successful marketing campaign. Since the year of membership is a discrete variable, we will convert it to a one-hot encoding similar to the one done for age.

2. **portfolio.json**

Next dataset that we would analyze is the portfolio dataset.

Offers sent during 30-day test period (10 offers x 6 fields)

- reward: (numeric) money awarded for the amount spent
- channels: (list) web, email, mobile, social
- difficulty: (numeric) money required to be spent to receive reward
- duration: (numeric) time for offer to be open, in days
- offer_type: (string) bogo, discount, informational
- id: (string/hash) – offer_id

	reward	channels	difficulty	duration	offer_type	id
0	10	[email, mobile, social]	10	7	bogo	ae264e3637204a6fb9bb56bc8210ddfd
1	10	[web, email, mobile, social]	10	5	bogo	4d5c57ea9a6940dd891ad53e9dbe8da0
2	0	[web, email, mobile]	0	4	informational	3f207df678b143eea3cee63160fa8bed
3	5	[web, email, mobile]	5	7	bogo	9b98b8c7a33c4b65b9aebfe6a799e6d9
4	5	[web, email]	20	10	discount	0b1e1539f2cc45b7b9fa7c272da2e1d7

There are three types of offers that can be sent: BOGO, discount, and informational:

- In a BOGO offer, a user needs to spend a certain amount to get a reward equal to that threshold amount.
- In a discount, a user gains a reward equal to a fraction of the amount spent.
- In an informational offer, there is no reward, but neither is there a prerequisite amount that the user is expected to spend.

Offers can be delivered via multiple channels: email, social media, on the web, and mobile (via the Starbucks app). Every offer has a validity period (duration) before the offer expires. We see that informational offers have a validity period even though these ads are merely providing information about a product. Here, the duration is the assumed period in which the customer is feeling the influence of the offer after receiving the advertisement.

On the whole, the dataset has no null values and is very clean – we can see that there are a total of 10 offers.

offer_type	offer_id
bogo	4
discount	4
informational	2
total	10

As we can see, offer_type and channels are presented as a categorical values, which must be converted to columns (one hot encoding) before passed to a learning algorithm.
Reward, difficulty, and duration have values in different ranges similar to the income feature in the profile dataset. Since the size of values can increase bias an algorithm, we will scale these feature columns between 0 and 1 by standardizing them.

3. transcript.json

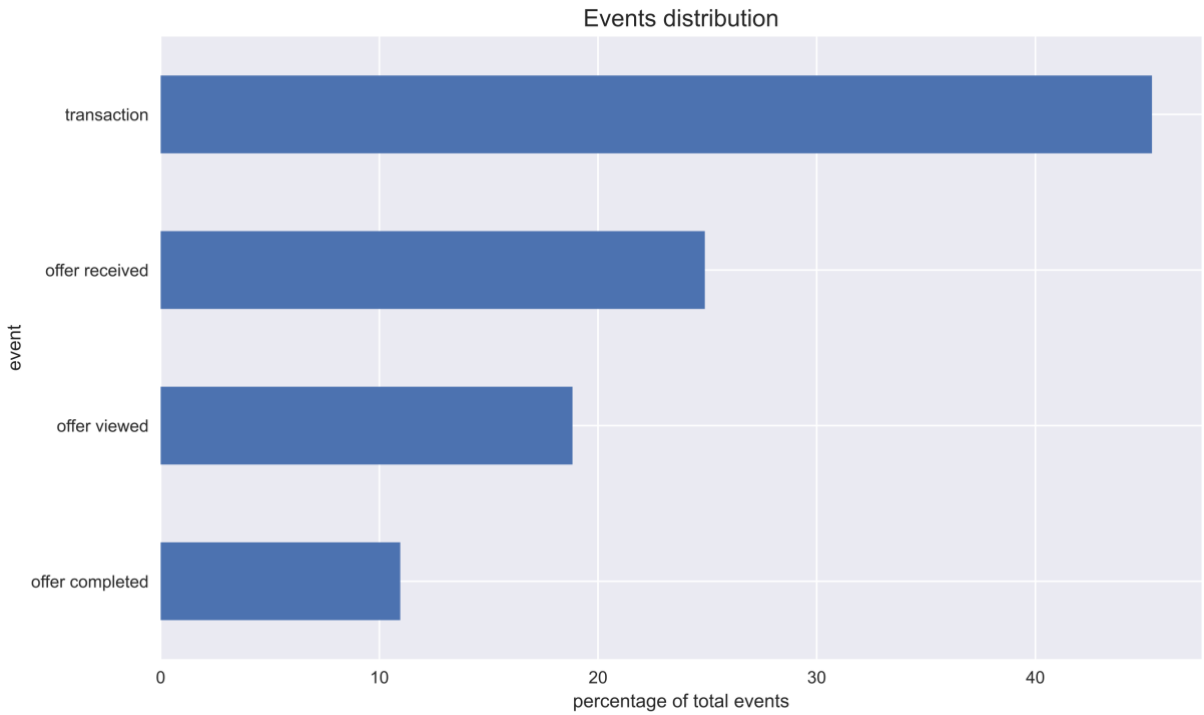
Event log (306648 events x 4 fields)

- person: (string/hash)
- event: (string) offer received, offer viewed, transaction, offer completed
- value: (dictionary) different values depending on event type
- offer id: (string/hash) not associated with any "transaction"
- amount: (numeric) money spent in "transaction"
- reward: (numeric) money gained from "offer completed"
- time: (numeric) hours after start of test

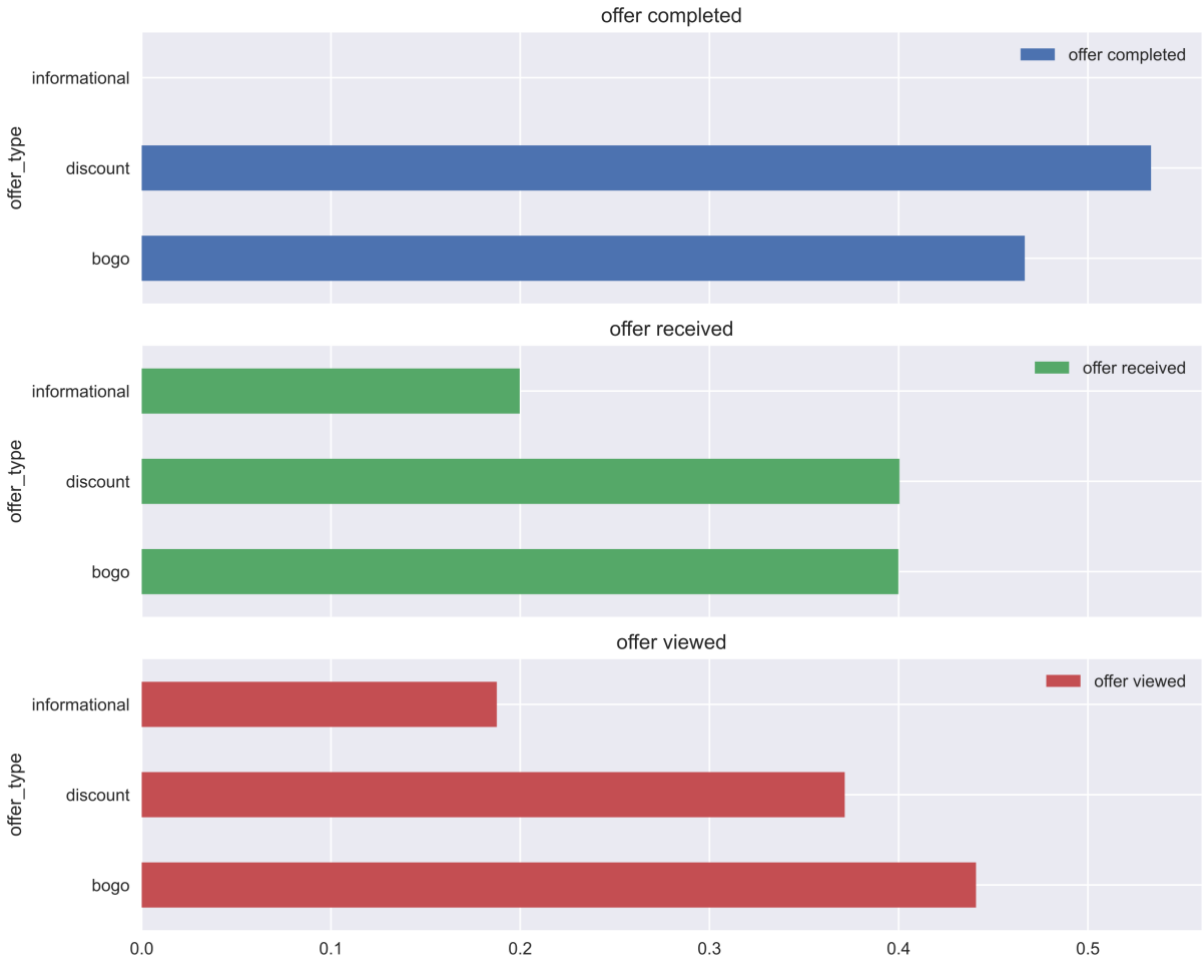
This is the trickiest dataset – let's analyze it. The dataset is clean in the sense that there are no null values. However, one of the columns has dictionaries as values.

	person	event	value	time
0	78afa995795e4d85b5d9ceeca43f5fef	offer received	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	0
1	a03223e636434f42ac4c3df47e8bac43	offer received	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}	0
2	e2127556f4f64592b11af22de27a7932	offer received	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}	0
3	8ec6ce2a7e7949b1bf142def7d0e0586	offer received	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}	0
4	68617ca6246f4fbc85e91a2a49552598	offer received	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}	0

Comparing the customers with the profile dataset, we see that the number of people (i.e. number of unique IDs) in transcript are the same as the number of people in the profile dataset, so that is good news.
Expanding the dictionary keys to different separate columns, we see that there are some duplicated offer id and offer_id columns – we will merge them by pandas fillna method and retain only the offer_id column.



From the above plot, it's clear that that even though 40% of the events had a transactions, only 10% of the events had completed offers.



From the above plot, it seems that both BOGO and discount offers have been sent equally – no preference over them. However, even though more BOGO offers are viewed, more discount offers are completed – thus it might convey that people try optimize their spending for the maximum value which is understandable.

2.2 Algorithms and Techniques

As we see in the section above, Data Exploration and Exploratory Visualization section, it is necessary to handle some features before using them to train machine learning models.

For the profile dataset, we'll first remove the nulls using dropna method of the gender and income columns as subset. Subsequently, we'll one-hot encode age ranges, gender and became_member_in (year) columns. We will also scale income column later using the MinMaxScalar method from sklearn package in order to standardize it between 0 and 1.

For the portfolio dataset, we'll one-hot encode channels and offer_type columns using the pandas get_dummies function. Similar to the treatment done for income column, we'll scale reward, difficulty and duration columns later using the MinMaxScalar.

For the transcript dataset, we will clean the dataset and separate them into offers and transactions dataset – since it would make our life easier when we check for effectiveness of the offers. The only cleaning that we'll do here is to one-hot encode the event column.

Next step would be create an effective offer label for the dataset. For BOGO and discount offers, an effective offer would be defined if the events are defined in this chronological order: ***offer received > offer viewed > transaction > offer completed.***

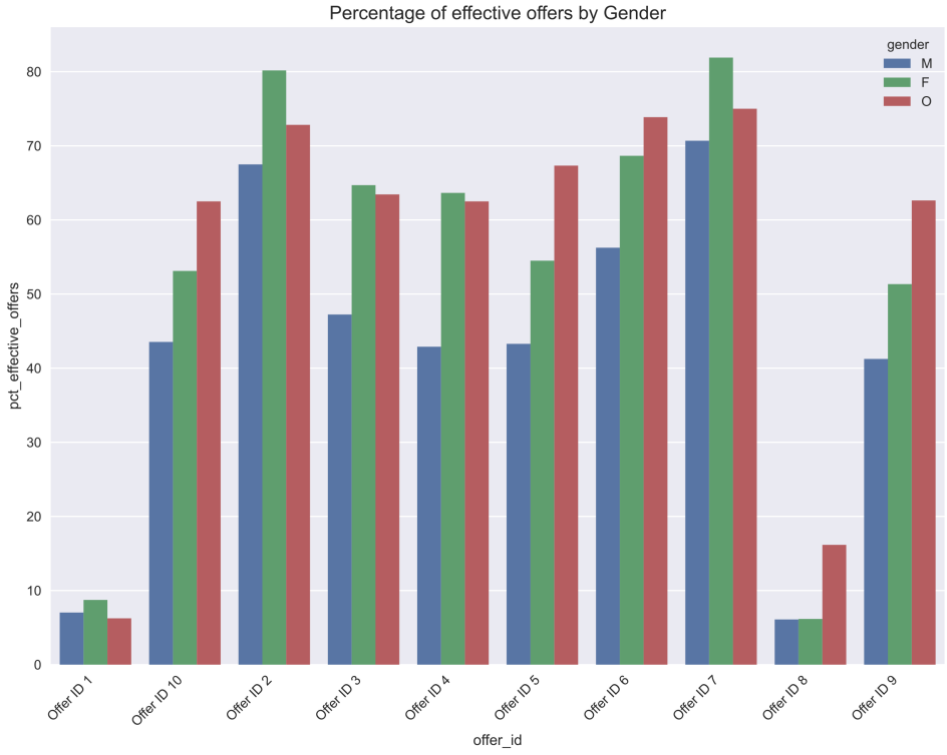
We will employ a simple algorithm which will loop through each customer and their offers and check whether the offer was effective (used) or not. In the end, we will combine all the results to form a clean dataset with target class labels.

Algorithm:

We will iterate over each customer's profile. First, select a customer's profile. Second, select offer data for a specific customer. Third, select transactions for a specific customer. Fifth, I apply the following procedure for each offer that a customer receives:

1. Initialize the current offer id
2. Look-up the offer description of the current offer from the cleaned portfolio dataset
3. Get the offer duration value and the time when an offer is received – calculate the offer end time using start + duration
4. Check if any customer transactions fall within the valid offer time window (between start and end time)
5. Check whether the user views and completes the offer within the valid offer time window (between start and end time)
6. Determine whether the current offer was successful (for an offer to be successful a customer must view and complete it)
7. Select customer transactions that occurred within the current offer valid time window and cumulate the amount spent
8. Initialize a dataframe to store all information for the offer along with a Boolean label for effective/ineffective offer classification.

Once we have the labelled dataset, let's check the distribution of the effective offers.



This shows the out of the total offers for each gender, the conversion rate was highest for the female gender – the other gender has only 212 customers out 14820 total customers (17000 – 2175 nulls – 5 users with no offer data, just transactions) hence the number looks high.

Finally, we standardize numeric columns with continuous data (like income, duration, reward, and difficulty) to a [0,1] range by applying minimum / maximum scaling to avoid model bias. We also drop some extra columns that are not really features for predicting an effective offer – like offer_id, customer_id, amount, time and informational offer type.

Then, we split the combined offer effectiveness data into training and test sets prior to assessing the accuracy and F1-score of a naive model that assumes all offers were successful. The ratio of 90/10 was used for splitting between training and test sets – a higher value was chosen for the split since the amount of data is not huge and more training data is needed to prevent model overfitting.

```
Number of total features: 29
============================

results of the split
============================
Training set has 59850 samples.
Testing set has 6651 samples.

class distribution
============================
y_train class distribution
----------------------------
effective_offer
0                 0.528354
1                 0.471646
dtype: float64

y_test class distribution
----------------------------
effective_offer
0                 0.528342
1                 0.471658
dtype: float64
```

2.3 Benchmark

We now access the accuracy and F1-score of a naive model that assumes all offers were successful.

```
{'NaivePredictor': {'test': {'accuracy': 0.4716583972334987,
                             'f1score': 0.6409889660809155},
                    'train': {'accuracy': 0.4716457811194653,
                              'f1score': 0.6409773155612072}}}
```

As seen from the above image, the analysis suggests that the naive model accuracy was 0.471 and its F1-score was 0.640.

2.4 Model Implementation

We will now implement machine learning models to evaluate predictions for effective offers. As defined in the proposal, we will implement 4 models – Logistic Regression, Random Forest, Gradient Boosting and Support Vector classifiers. We have also included an additional model – K-Nearest Neighbours for comparison purposes.

We will perform a random search of logistic regression, random forest, gradient boosting, SVC and KNN models to select the one that had the highest training data accuracy and F1-score (beta of 0.5). This will be done through the Random Search Cross Validation on model hyperparameter space. Grid search is a brute force way of finding the optimal parameters because it trains and test every possible combination.

Why did we choose Random Grid Search rather than the standard Grid Search? While it's possible that RandomizedSearchCV will not find as accurate of a result as GridSearchCV, it surprisingly picks the best result more often than not and in a fraction of the time it takes GridSearchCV would have taken. Given the same resources, Randomized Search can even outperform Grid Search[2]. Generally GridSearchCV works best with small datasets (< 5000 samples) and lots of computational resources. With large data sets, the high dimensional data that we have (60,000 x 29) will greatly slow down computation time and be very costly.

RandomizedSearchCV is better suited for this since we ran the model on a local machine, so we have limited computational power. Also since the number of iterations is explicitly defined by us, RandomizedSearchCV works faster.

RandomizedSearchCV select the best hyperparameter combination for each model which saves us time on manually tuning the model. For example, for Random Forest, these are the best fit hyperparameters that were selected.

```
Random Forest Classifier best fit hyperparameters:
====================================
{'max_depth': 10,
 'max_features': 'sqrt',
 'min_samples_leaf': 4,
 'min_samples_split': 5,
 'n_estimators': 400}
```

[2] https://blog.usejournal.com/a-comparison-of-grid-search-and-randomized-search-using-scikit-learn-29823179bc85

3. Model Evaluation and Validation

3.1 Model Performance

Machine Learning Algorithms	Train - Accuracy	Train - F1 Score	Test - Accuracy	Test - F1 Score
NaivePredictor	0.471646	0.640977	0.471658	0.640989
KNeighborsClassifier	0.750276	0.737863	0.713427	0.701628
LogisticRegression	0.721988	0.716402	0.725455	0.721392
SVC	0.727703	0.715956	0.727409	0.715608
GradientBoostingClassifier	0.738329	0.705083	0.728011	0.693960
RandomForestClassifier	0.740451	0.733230	0.730717	0.723141

These results suggest that a random forest model had the best train and test data accuracy compared to gradient boosting, logistic regression, SVC and KNN and a naive predictor that assumed all customer offers were successful – There is one instance on the training dataset where KNN model has a higher accuracy than even Random Forest model. However, it underperforms the Random Forest and other models (except the naive predictor) – which means that the higher training accuracy can be attributable to model overfitting.

Some insights on the models –

1. A logistic regression model constructs a linear decision boundary to separate successful and unsuccessful offers. However, the exploratory analysis of customer demographics for each offer suggests that this decision boundary should be non-linear. Therefore, an ensemble method like random forest or gradient boosting should perform better.
2. We observe that the gradient boosting model underperforms the random forest model (with a low F1 score – relative to the other models).

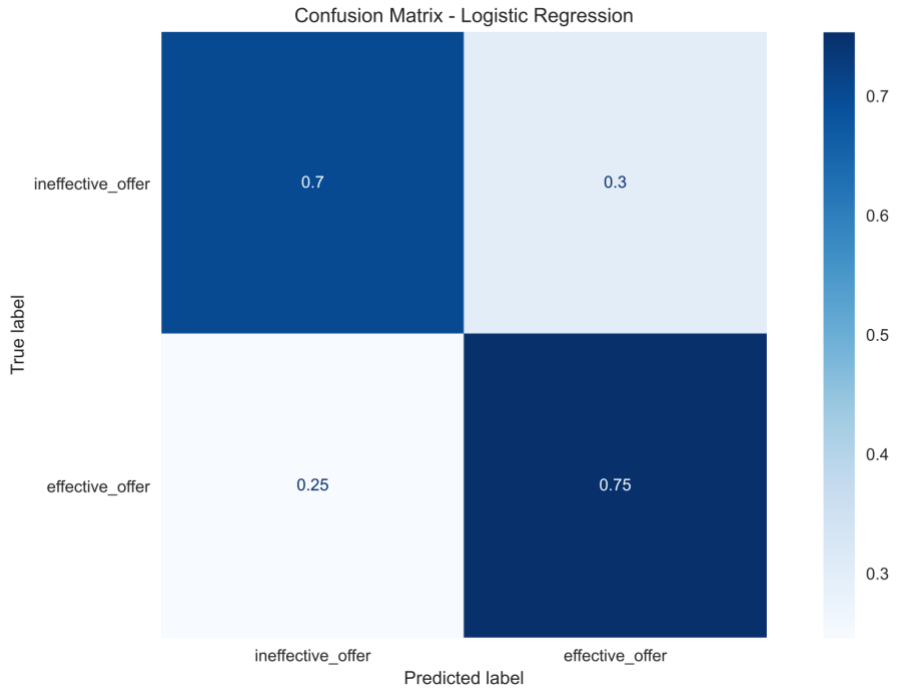
   Why is that? Well – Conceptually, both random forest and gradient boosting models are a combination of multiple decision trees. A random forest classifier randomly samples the training data with replacement to construct a set of decision trees that are combined using majority voting.

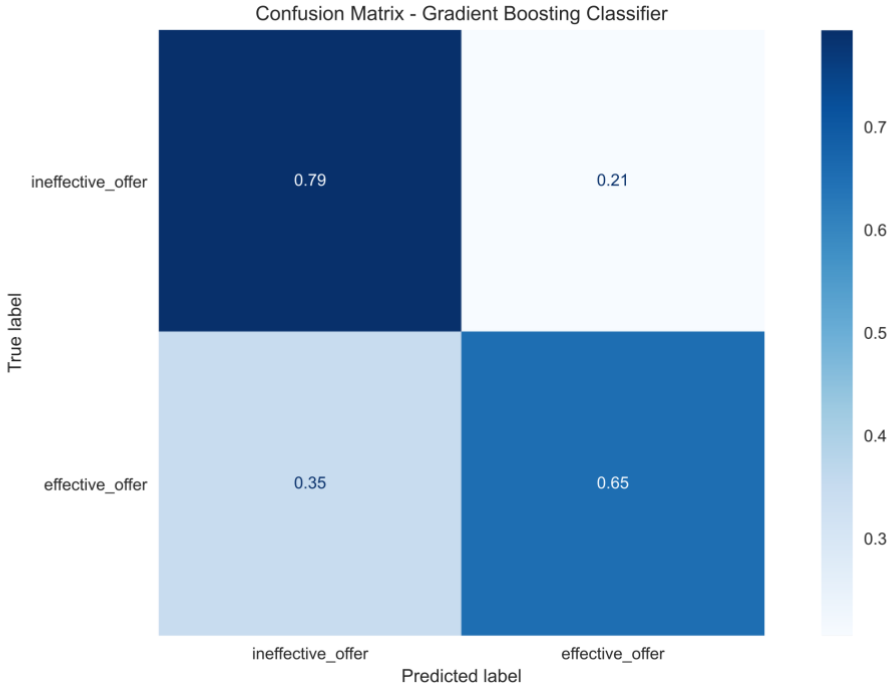
In contrast, gradient boosting iteratively constructs a set of decision trees with the goal of reducing the number of misclassified training data samples from the previous iteration. A consequence of these model construction strategies is that the depth of decision trees generated during random forest model training are typically greater than gradient boosting weak learner depth to minimize model variance.

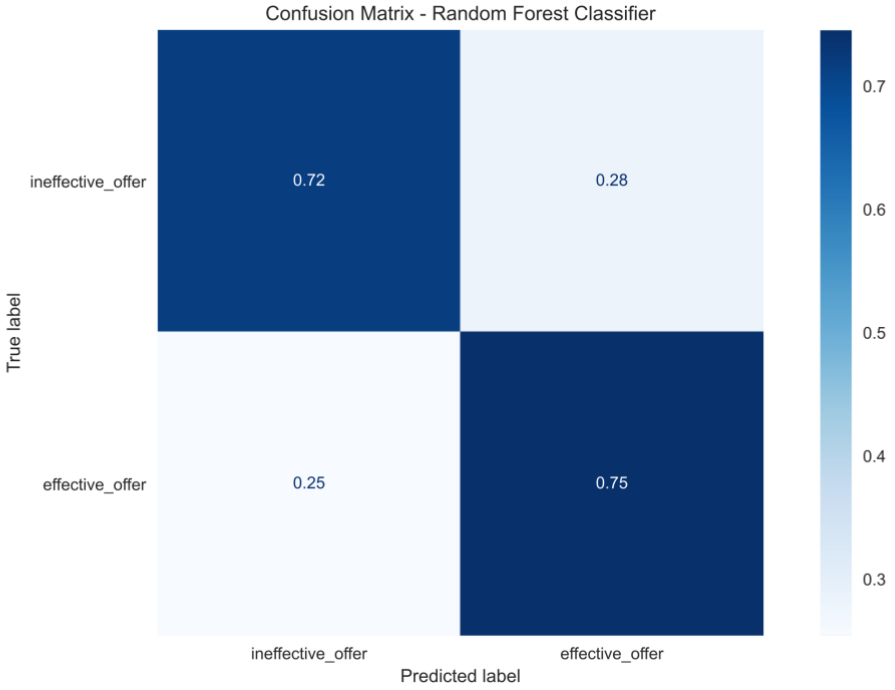
Typically, gradient boosting performs better than a random forest classifier. However, gradient boosting may overfit the training data and requires additional effort to tune. A random forest classifier is less prone to overfitting because it constructs decision trees from random training data samples. Also, a random forest classifier's hyperparameters are easier to optimize than gradient boosting.

3. The results indicate that Random Forest and Logistic Regression model have done the best at not misclassifying negative events as positive (meaning, misclassifying people on which offers are ineffective, as people on which offers would be effective).

   Gradient Boosting has the highest rate of misclassifying negative events as positive - however, it also has the lowest rate of misclassifying positive events as negative (from the confusion matrix).




Confusion Matrix - Gradient Boosting Classifier
ineffective_offer
effective_offer
True label
0.79
0.21
0.35
0.65
ineffective_offer
effective_offer
Predicted label
0.7
0.6
0.5
0.4
0.3


Confusion Matrix - Random Forest Classifier
ineffective_offer
effective_offer
True label
0.72
0.28
0.25
0.75
ineffective_offer
effective_offer
Predicted label
0.7
0.6
0.5
0.4
0.3

Overall, terms of both accuracy and F1 score, the Random Forest model has the best performance, albeit with a narrow margin over the other models, with KNN model performing best on the training dataset but fails to have a high accuracy over the test dataset.

We also performed a random search for the best fit hyperparameters from a set of values so that we select the best performing parameters for each model. Even then, as you can see, we had a very narrow band of accuracy figures for each model - ranging from 72% to 74%. We know from other examples over the internet that the ensemble methods like Random Forest and Gradient Boosting should show a better performance over the other models.

This means that model tuning might not help much here, so we should look into improving the feature selection of the model, i.e. removing sparse features.

3.2 Feature Importance

Machine learning works on a simple rule – if you put garbage in, you will only get garbage to come out. By garbage here, we mean noise in data.

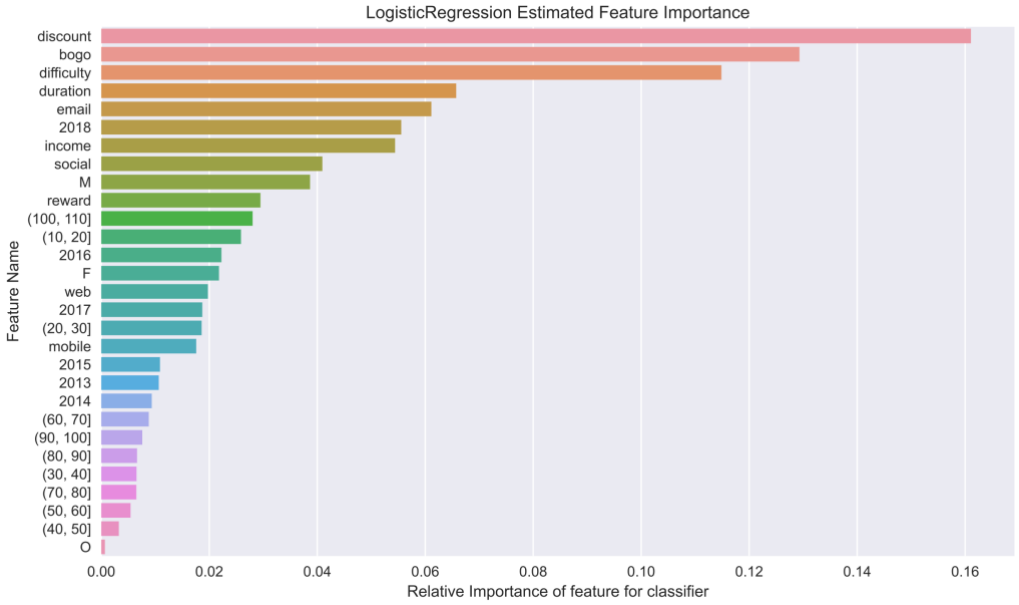
Feature importance refers to techniques that assign a score to input features based on how useful they are at predicting a target variable. The main reasons to use feature selection are:

- It enables the machine learning algorithm to train faster.
- It reduces the complexity of a model and makes it easier to interpret.
- It improves the accuracy of a model if the right subset is chosen.

We will now check the features by relative importance for each of our models.

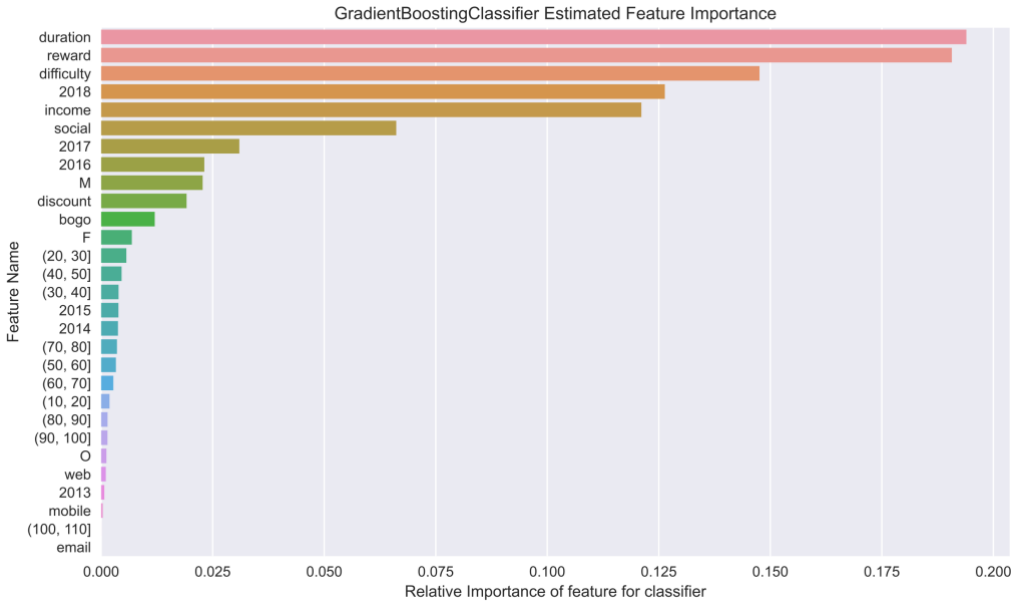
1. **Logistic Regression:** We can retrieve the coeff_ property that contains the coefficients found for each input variable. These coefficients can provide the basis for a crude feature importance score. This assumes that the input variables have the same scale or have been scaled prior to fitting a model. For Logistic Regression, all that the transform method is doing is looking at which coefficients are highest in absolute value[3].

[3] Source: https://stackoverflow.com/questions/24255723/sklearn-logistic-regression-important-features



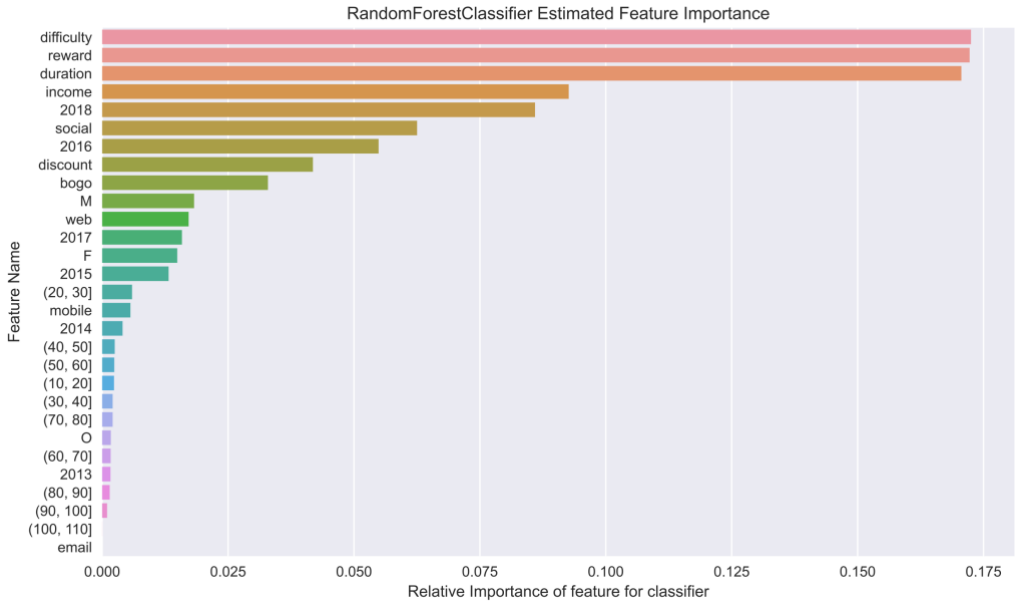
2. **Gradient Boosting:** The feature importance for a GB model is calculated as sum up the feature importance of the individual trees, then divide by the total number of trees[4]. For importance of each individual tree, iterate through the nodes of the tree. As long as you are not at a leaf node, calculate the weighted reduction in node purity from the split at this node, and attribute it to the feature that was split on. Then, when done, divide it all by the total weight of the data (in most cases, the number of observations).

   We can directly extract the feature importance from the model estimator using the feature_importances_ property of the decision tree classifiers.



[4] https://stats.stackexchange.com/questions/162162/relative-variable-importance-for-boosting

3. **Random Forest:** The scikit-learn default method of assigning feature importance to a Random Forest classifier is through Gini importance (or mean decrease impurity)[5]. tree-based strategies used by random forests naturally ranks by how well they improve the purity of the node. This mean decrease in impurity over all trees (called Gini impurity). Nodes with the greatest decrease in impurity happen at the start of the trees, while notes with the least decrease in impurity occur at the end of trees. Thus, by pruning trees below a particular node, we can create a subset of the most important features. Similar to the implementation for GB model, we can directly extract the feature importance from the model estimator using the feature_importances_ property of the decision tree classifiers.



Let's look at the most important features for the Random Forest Model since it's the best performing model. The top five features based on their importance for an effective offer are:

- Offer difficulty (how much money a customer must spend to complete an offer)
- Offer reward
- Offer duration
- Customer income
- Whether a customer created an account on the Starbucks rewards app in 2018

[5] https://mljar.com/blog/feature-importance-in-random-forest/

This makes sense as the customer also tries to maximize his value from purchase relative to cost of the product, reward, and income. There is higher probability of an offer being completed if the duration for the offer is longer.

For SVC and KNN models, we cannot compute feature importance in an easy way. Feature Importance for fitted SVM classifier can be extract using the same method we used for Logistic Regression - features with highest importance are the ones with coefficients from the coef_ property that are highest in absolute value. However, it only works for SVM classifiers with linear kernel - but our best fitted model uses a 'rbf' kernel. For other kernels, it is not possible because data is transformed by the kernel method to another space, which is not related to input space[6].

Similarly, feature importance is not defined for the KNN Classification algorithm. There is no easy way to compute the features responsible for a classification for a KNN fitted model. A hacky, brute force method based of shuffling of features can work like this[7] –

- Evaluate the model accuracy based on the original dataset
- For each feature in the dataset:
- Make a copy of the dataset
- Randomly shuffle the current target feature
- Evaluate the model accuracy based on the dataset with the shuffled feature
- Compute the difference in accuracies - this is feature importance, where higher is better

[6] https://stackoverflow.com/questions/41592661/determining-the-most-contributing-features-for-svm-classifier-in-sklearn
[7] https://github.com/scikit-learn/scikit-learn/issues/8898

4. Conclusion and Future Improvements

All our models except Gradient Boosting are predicting the positive case (i.e. where an offer is effective) more accurately compared to predicting the negative cases (i.e. where an offer is ineffective), which is expected given the uneven classes. We are not much concerned with the misclassification cases since we don't mind sending people more offers than they would have liked; we would rather not miss anyone on which an offer would have been effective.

Since all our models had an accuracy > 70% which we set out to beat, we can say with confidence that we have successfully achieved our objectives. Given the exhaustive analysis, we would select Random Forest as the best model to use for predicting effective offers for the Starbucks rewards data.

We also looked at important features for each algorithm and the top features for Random Forest are very intuitive which reinforces our decision that it is the best model out of the five models tested.

Obviously, as any model in the world, our models can be further refined and improved in a number of ways to increase predictive accuracy over the test dataset. Here are top four recommendations from my experience to increase the model performance:

1. **Drop features**: To improve model performance, we can try to drop some dummy variables/one-hot encoded features and see how it will affect the model performance.

   There is also an ongoing debate on the use of one hot encoding when using tree and regression models. For regression classification models (e.g. logistic regression), we should typically remove one level of the variable in order to prevent multicollinearity between variables. Typically, we should not run into this issue with tree-based models like random forest or gradient boosting.

   However, there is some debate as to whether one should do it or not. According to some articles[8], it is generally not advisable to encode categorical variables as they would

[8] https://roamanalytics.com/2016/10/28/are-categorical-variables-getting-lost-in-your-random-forests/

generate sparse matrices. The resulting sparsity virtually ensures that continuous variables are assigned higher feature importance.

A single level of a categorical variable must meet a very high bar in order to be selected for splitting early in the tree building. This can degrade predictive performance.

2. **Introduce Polynomial features**: A low accuracy score for the models is likely due to the model underfitting more likely since we have very sparse features, we can try to transform the features into polynomial form (using PolynomialFeatures from sklean.pre_processing module) to further improve model performance.

3. **Using more data:** More data with a better distribution of effective/ineffective offers would have helped achieve a higher accuracy, as is the case with most machine learning algorithms.

4. **Feature Selection using feature importance scores:** Feature importance scores can be fed to a wrapper model, such as the SelectFromModel[9] class, to perform feature selection.

   This method can be very useful datasets like the one we used, which have sparsity of features, where selecting only a subset of the most interesting features from the dataset.

4.1 Reflection

A key takeaway from this project is how big a role understanding the underlying data set as well as cleaning/transforming plays in a real life data science project. In addition, it also helped me understand the importance of tweaking intricate parameters (and their math!) behind each machine learning model and how they can affect/alter the final result.

Thanks Udacity for the code snippets to get me started on the project and Starbucks for providing the data used by this project.

This project could not have been possible without the help of reference code and explanations from Kaggle, Machine Learning Mastery, Stack Overflow, and Towards Data Science.

[9] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html