Nadim Kawwa

STARBUCKS CAPSTONE PROJECT

Abstract

Starbucks is an American coffee company and coffeehouse chain. This report explores consumer behavior in relation to an undisclosed product. Information such as customer demographics, spending habits, and offer type are present in separate datasets. By manipulating these features we are able to segment customers, predict spending habits, and suggest offers to customers.

Introduction

64% of Americans age 18 or over said they had a cup of coffee the previous day in 2018 according to the National Coffee Association (1). Among those surveyed, coffee consumed at cafes and other out of home locations totaled at 36%.

Starbucks is an American coffee company and coffeehouse chain traded on NASDAQ under SBUX. In 2018 it accounted for 39.8% of the US coffee chain industry market share and (2). In the same year Starbucks has spent $260.3 million on advertising (3).

The objective of this report is to develop a deeper understanding of Starbucks' customers and their interaction with an unspecified product. The repository for this project can be found on GitHub.

The approach taken in this report is customer centric in order to develop a product that is financially viable (4). For more information on the latter approach, the book is made partially available by Stanfrod's Technology Entrepreneurship course ENGR 145 (5).

Methodology

All solutions must stem from an organized and replicable method. Our initial strategy is to aim for an organized dataset for the users. This dataset should have the user id as row entries (observations) and user attributes in the columns (features). As the next section indicates, the data is given in three separate datasets. Therefore it is imperative to know how they are linked, how to join them (inner, outer, etc…), and provide a justification. By the end of data cleaning we should have a user dataframe with measurable features that we can use for clustering and prediction.

In addition, as we aim to create a recommendation engine the notion of a user-item matrix comes to mind. In this matrix the rows corresponds to users and the columns correspond to the offers. We will also need to determine what to plug in for the entries, if user rating is measure in spending or some other metric.

All data exploration will follow this flow:

- Data types
- Real world interpretation of observation
- Measures of spread
- Missing or inconsistent values

The steps above allow a thorough investigation and detect points of concern early on.

We expect the solutions to the problem statements to be in the form of quantifiable means such as accuracy. We also want to know how our models fail and set aside a validation set when possible to avoid overfitting.

Data Sets

The data sets are made available by Udacity's Data Scientist nanodegree (6):

- 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

Portfolio

Regarding the offer ids the following is given:

- 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

For demographic data:

- age (int) - age of the customer
- became_member_on (int) - date when customer created an app account
- gender (str) - gender of the customer
- id (str) - customer id
- income (float) - customer's income

Transcript

For transaction records:

- 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

Data Exploration & Cleaning

Given the data, the next step is to explore it and draw statistically significant information. This section is also crucial to determine if any data cleaning and/or manipulation is done. Any assumption made in this section will influence the conclusions of this report and therefore impact hypothetical business decisions.

The exploration and cleaning section can be found in the first notebook of the repository.

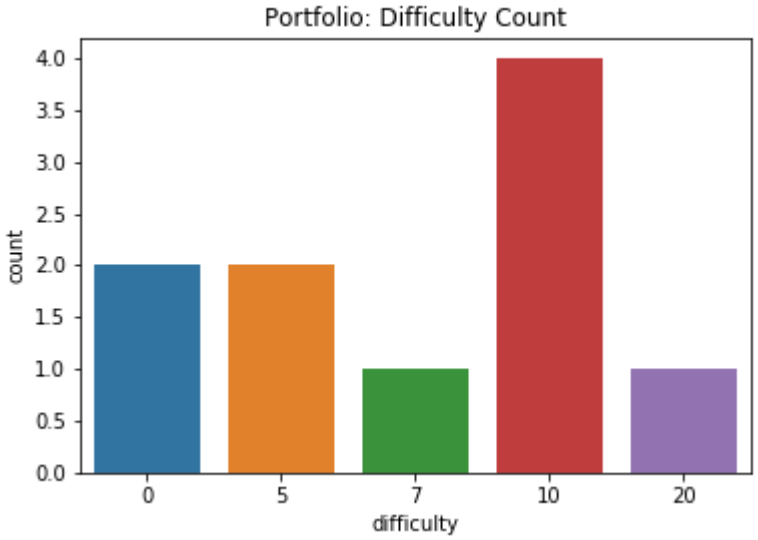
Portfolio

In total there are 10 offers, offer has features that define it. Given that the offers are identified by random characters, we shall refer to them by integers starting from 0. In addition, the offer with the lowest id will correspond to the highest difficulty.

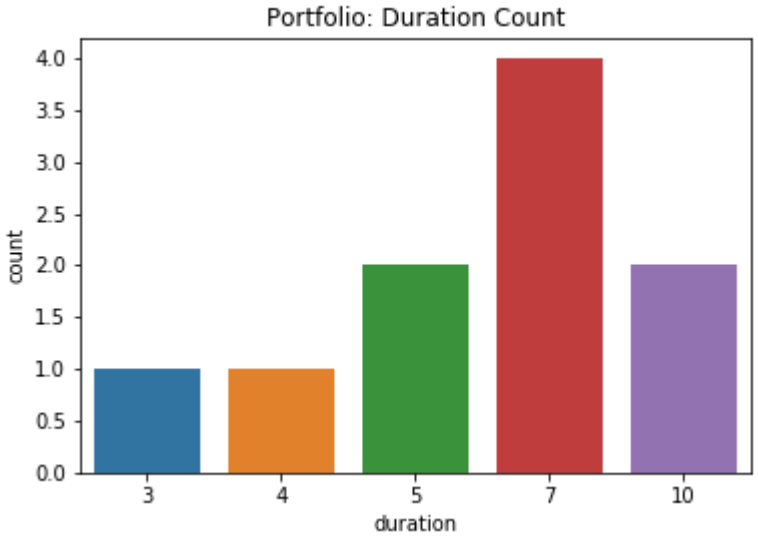
The first feature is the channel of communication and can be a combination of any of the following:

- Web
- Email
- Mobile
- Social

The other features are difficulty, reward, and duration. Each feature is given an integer score, the minimum score being zero. The plots below show the distribution of the counts for each value within a feature.


Portfolio: Difficulty Count
count
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
5
7
10
20
difficulty

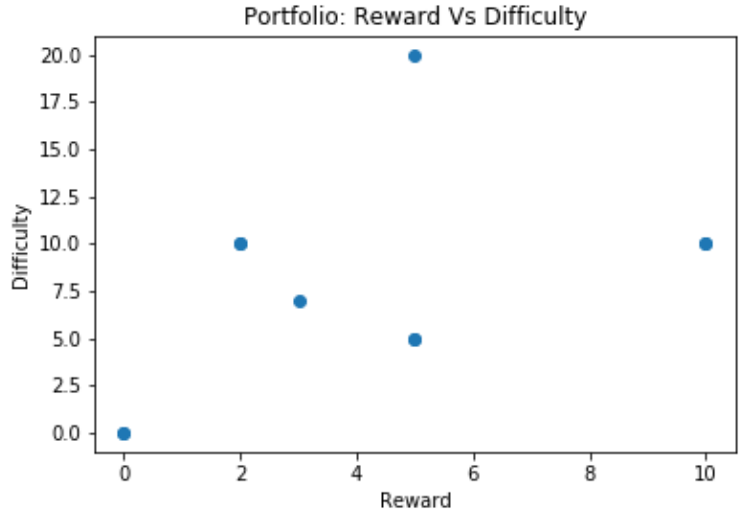

Portfolio: Reward Count
count
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2
3
5
10
reward



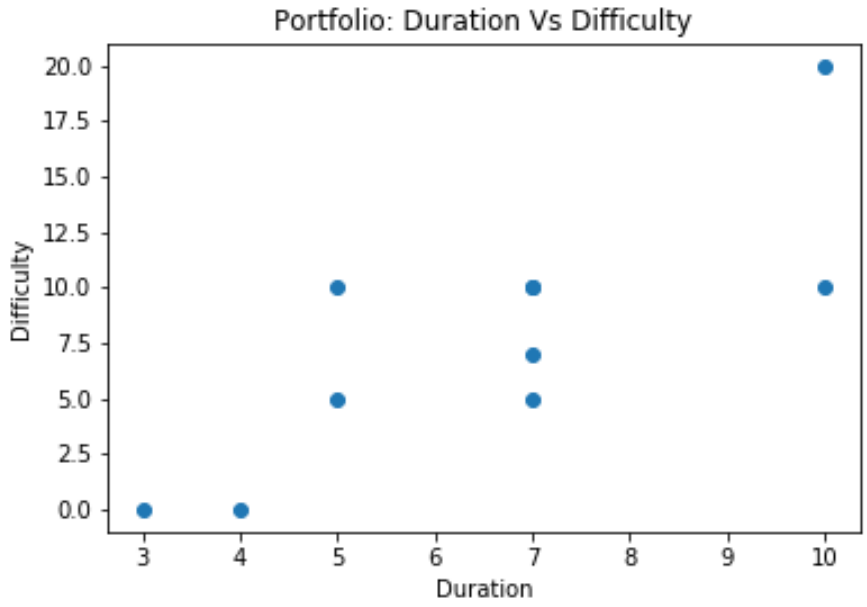
To summarize the plots above in a table:

	Difficulty	Duration	Reward
Min	0	0	0
Max	20	10	10
Mean	7.700000	6.5	4.2
Standard deviation	5.83	2.32	3.58

There appears to be a correlation between the features in the data, as the plot below suggests a somewhat positive correlation between diffiulty and reward.



The correlation is more evident when plotting difficulty versus duration.



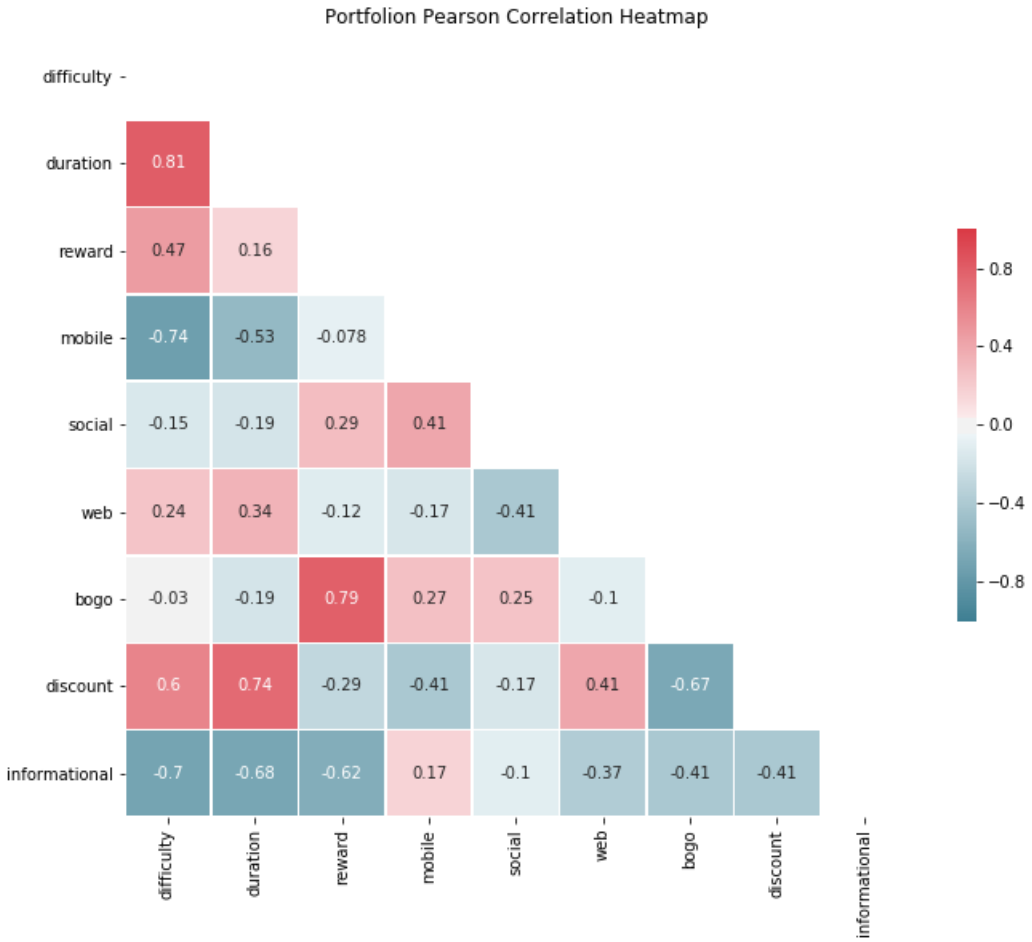
In order to better understand the data we can refer to the heatmap of the Pearson correlation coefficient (7). The correlation coefficient takes a value between -1 and 1, where 1 is total positive linear correlation, 0 is no linear correlation, and −1 is total negative linear correlation.

The diagonal entries are left blank because a feature is always positively correlated to itself. We drop the offer id because it should not have any correlation with the offer. In addition, all offers include email as a means of communication. We conclude that the variance of the feature email is equal to zero. Recall that the correlation coefficient, denoted here by ρ, is calculated as:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$$

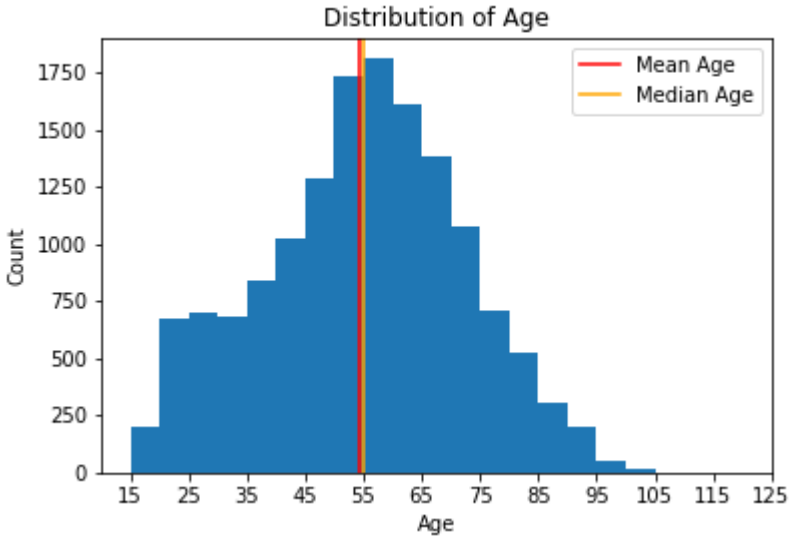
- cov = covariance
- σ_X = standard deviation of X
- σ_Y = standard deviation of Y

Therefore we drop email from the correlation matrix, where. From the heatmap below we can see a strong positive correlation between duration and difficulty, and reward and BOGO type offers. Offers that are informational have the least difficulty, duration, and reward.

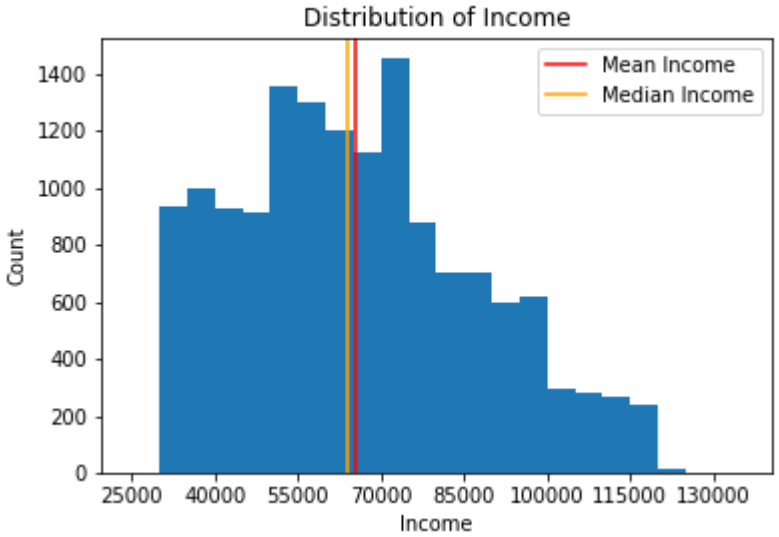


Profile

The plot below shows a histogram of the age distribution among the customers. Note that if the age take the value “118” it means that the age is unknown. The plot below does not include such values. The mean age is around 55 years, the median is also around that number.



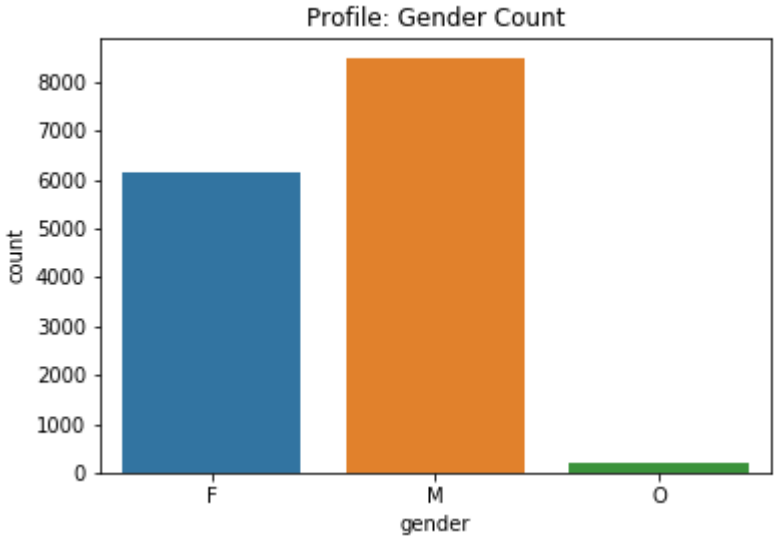
Oddly enough the income follows a similar distribution of mean and median. The average user income is around $65,400 and the median is $64,000.



The gender of the users can take four values:

- M: Male
- F: Female
- O: Other
- None: undeclared

Males appear to be overrepresented among the customers, and are about 60% of declared genders. In future analysis, it will be useful to encode gender with a representative integer and lump undeclared with other.



The final feature is the date of creation the app account. The data does not provide a method to anchor observations relative to some starting point. Indeed the next section regarding transactions will reveal that this information is not relevant. The lack of certainty raises certain questions:

- When did the client close their account?
- Do offers start streaming in at time of account creation?
- What is the frequency of offers? Weekly? Monthly? Holidays?
- What time of day are offers sent?

For the reasons above we decide to discard membership date. In addition, we make the following assumptions about the users:

- Users are drawn from a delimited geographical location
- Users are culturally homogenous
- Data is collected from a cluster of starbucks stores

Transcript

The crux of the report lies in the transcript that records and labels observations for users at a given point in time. Each row lists the event type, the person, the time of occurrence, and some value associated with it.

Exploring the transcript we explode the “value” column and notice that “offerid” and “offer_id” are the same. We will assume that the difference is merely a typo. Thus a transaction value can take on three observations:

- reward
- amount
- offer_id

In order to render the data useful, we will manipulate the transcript into an observation log that identifies the event type for each user under each observation.

Before extracting features, certain observations need to be stated:

- The clock starts ticking one a user receives an offer
- The clock does not always start at 0
- The clock does not reset if another offer is received
- The clock does not reset if the offer is completed
- Different users are observed over different timeframes
- If the transaction is done within the influence window and the offer seen, then the amount spent is tied to the offer id
- There are users who have not received any offer whatsoever and still spent money
- The user is under the influence of the first offer viewed which would be first offer sent

For the last point, we will use the schematic table below.

	T0	T1	T2	T3	T4
O1	Not seen	Seen	Seen	Completed	
O2		Not seen	Seen	Seen	Seen
Real Influence		O1	O1 & O2	O2	O2
Simplification		O1	O1	O1	O2

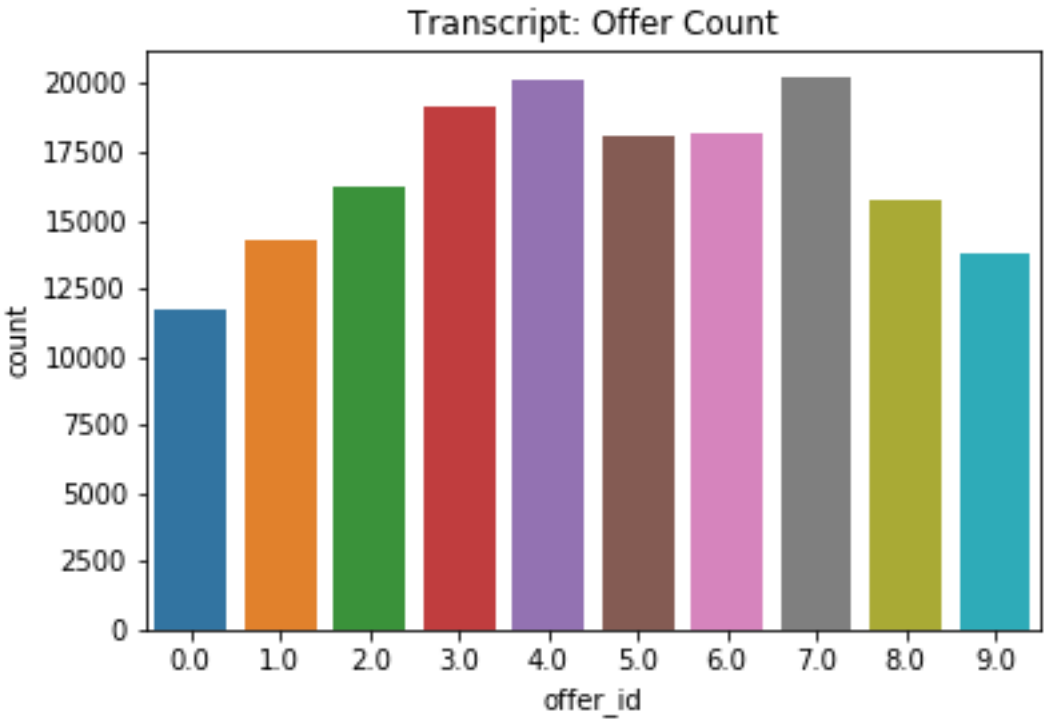
Consider a single user that will receive two offers O1 and O2, such that time is recorded at time T0 when O1 is received. At T0 the user is not under the influence of any offer. Starting T1 the user has seen O1 and therefore any purchases made are done because of O1.

At T1 the user also receives O2 but has not seen it yet. At T2 the offer O2 is seen, however the user is still under the influence of O1. Only after the validity period of O1 can we consider the purchase done because of O2.

This is a simplification deemed necessary for the scope of this report. In reality a more detailed evaluation would include 90 possible offer combinations, and perhaps determine which offer will govern based on metrics such as difficulty, reward, and medium of communication.

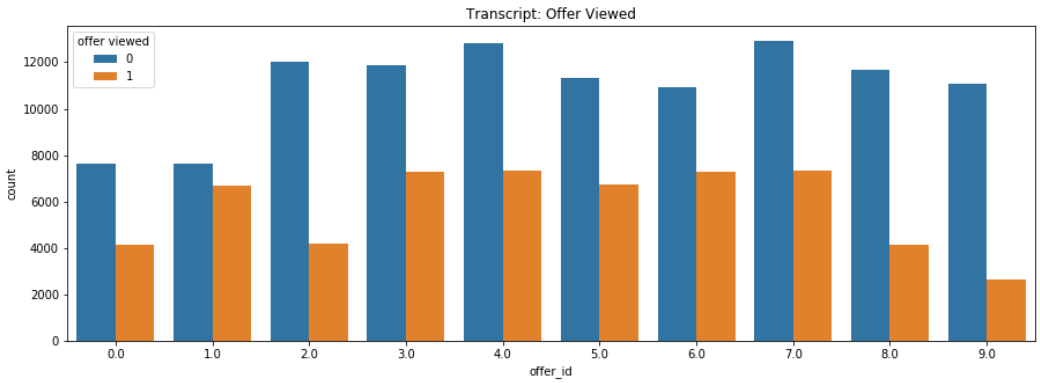
In addition, if the client completes the offer withn the validity period and makes another purchase: Was it because of the offer? For the sake of this report we shall assume that the customer is fully under the influence of the offer for as long as it is not completed.

As a first step we inspect the distribution of offers received by the customers and can see that offers with low to medium difficulty tend to be the most common. In addition the dataset is more or less balanced, for example no offer occurs twice as much as any other.

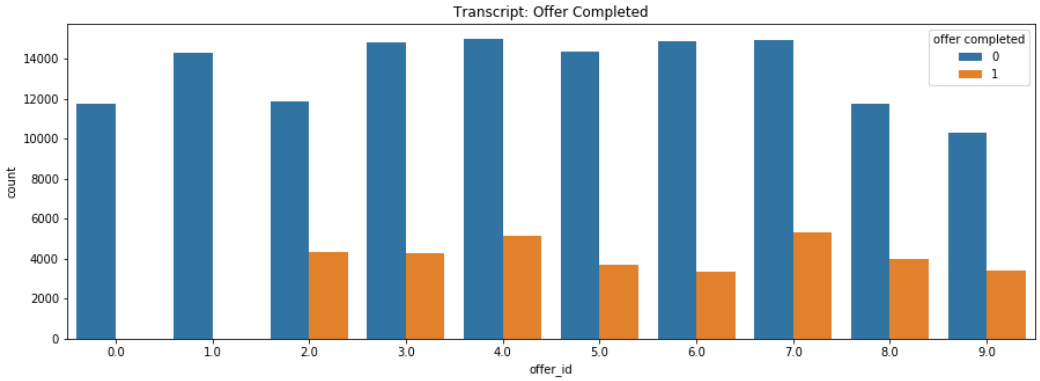


In addition from the plot below we note that some offers are barely viewed such as number 9 which is the most difficult. For the offers that are labeled “viewed” we are yet to determine on which means it was viewed, for example if was first viewed by email or via social media. It is possible that the methods of delivery affect the view rate for offers.

We shall assume for now that all means of communication are equally vaalid in the eyes of the customer. In addition, we shall take into account the influence of offer frequency. Therefore we will distinguish between users who have received for example 2 web offers versus just one.



Consequently the next plot regarding completing should mimic views. Recall that offers 0 and 1 are purely informational, thus there is no real way to measure completion.



Because membership date and time of observation is unknown, the method suggets deriving time based metrics. The data provides the time of seeing the offer, given it has been seen, relative to some arbitrary starting point and the time of completion. Therefore we can determine the average time needed to see and to complete or see an offer

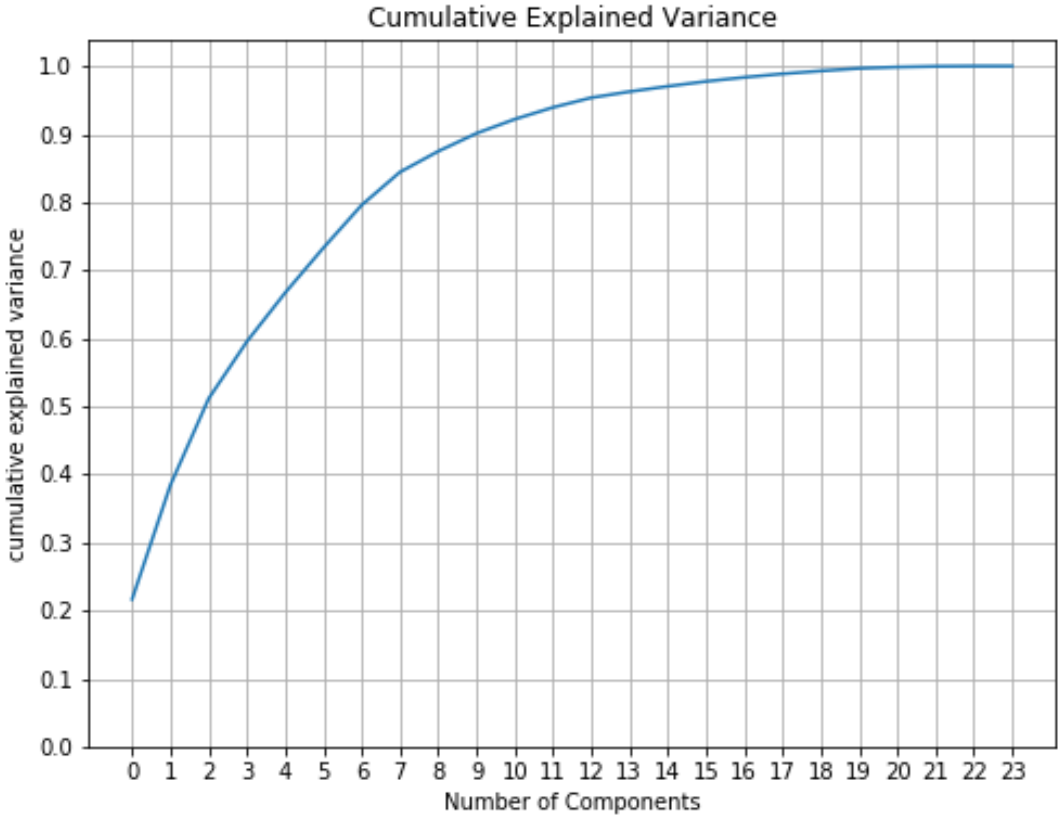
Customer Segmentation

Given the customers, we will attemp to segment them into clusters if possible. The implementation can be found in the second notebook.

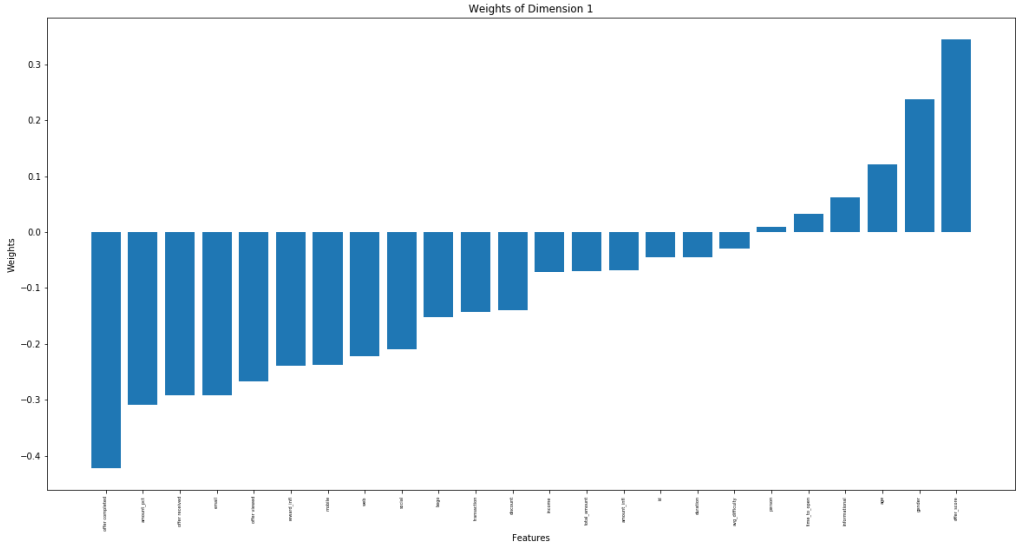
We will follow the methodology recommended by Muller and Guido (8) for this section. Since we now have several features, it is useful to apply principal component analysis (9) to reduce the dimensionality of the data.

As a first step we impute the missing values, particularily for income, with the median. This will inject some unwanted variance in the data, however the distribution of the features shows that it should not have a major effect.

The plot below shows that 12 features account for more than 95% of the variance in the data. We can therefore reduce our space to this number.



Consequently we can explore the weights of the first dimension, i.e. the eigenvector with the highest eigenvalue:



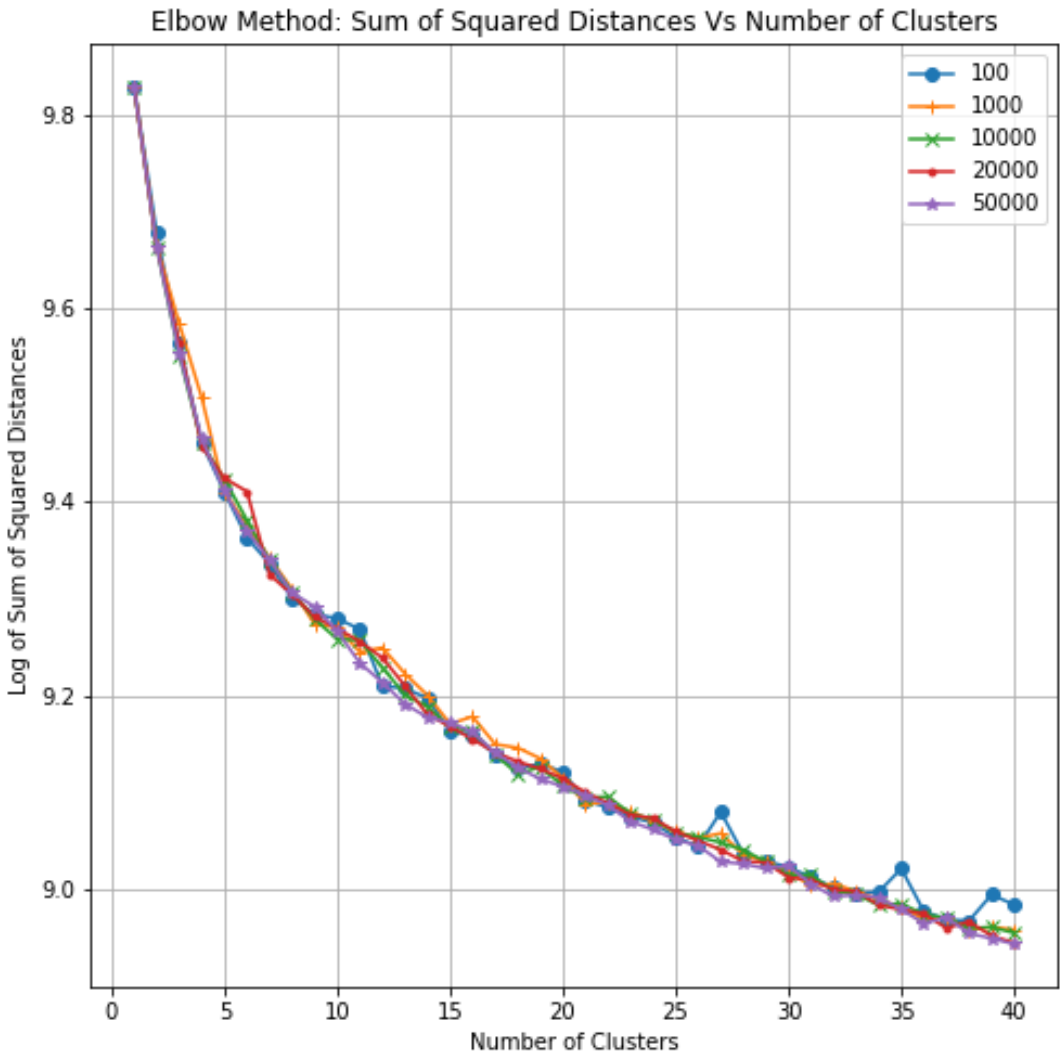
The 5 most positive weights for the 1st component in ascending order:

1.	time_to_open	0.0334
2.	informational	0.0628
3.	age	0.1218
4.	gender	0.2373
5.	offer_score	0.3447

The 5 most negative weights for the 1st component in descending order:

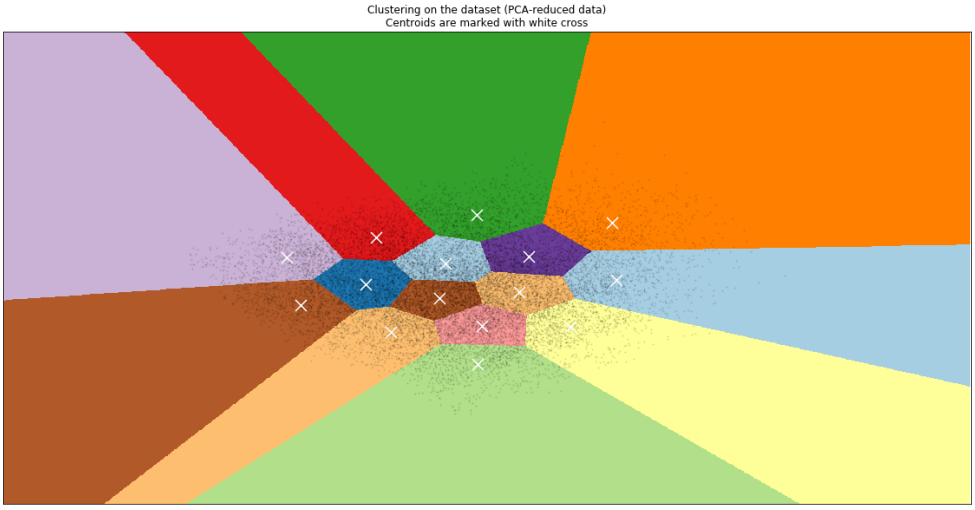
1.	offer completed	-0.4227
2.	amount_pct	-0.3086
3.	offer received	-0.2920
4.	email	-0.2920
5.	offer viewed	-0.2672

After reducing the data, we mini-batch k-means (10) in order to cluster the data using small random batches. Per the plot below and using the elbow method (11), we can guess the number of batches to be around 15.



The plot below shows how the clusters would appear in a 2 dimensional space. Since k-means works best with spherical datasets, it might not be the best fit for the data.

The silhouette score (12) is around 0.1 therefore clustering the data as it is can be improved upon.



Predict User Behavior

This section seeks to evalute how many times a user will see an offer. This part is not offer specific, however it offers a way to get to know the customers. Reaching out to create an offer has a price tag on it, and it is useful to know which customers are those willing to open them.

This part deals with supervised learning and can be found in the third notebook. The data is split into training and validation data. Training data is used to train the model, validation is used to measure its performance.

For preprocessing of the data we shall use MinMax scaling which transforms the data into a range from 0 to 1 such that 1 is the maximum possible value. Note here that the scaling is fitted only on the training data. The validation data is transformed by the scaler of the training data. A failure to do so will cause data leakage and cause the model to overfit.

Model evaluation is per the following metrics:

- Accuracy
- Precision
- Recall
- F1 Score

Accuracy is defined as:

$$Accuracy = \frac{Number\ of\ True\ Predictions}{Total\ Number\ of\ Predictions}$$

For the remaining metrics, we must first present a sample confusion matrix (13) in the table below.

Negative Class	TN	FP
Positive Class	FN	TP
	Predicted Negative	Predicted Positive

This simplification considers a binary supervised learning example. The elements on the diagonal represent correct predictions. The utility of this matrix is to see if we are overestaming a certain type of user behavior.

We can rewrite accuracy as:

$$accuracy = \frac{TN + TP}{TN + TP + FP + FN}$$

From the matrix we can define precision and recall:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

The f1 score is then the harmonic mean of precision and recall, it varies between 1 for perfect precision and recall at worst is equal to 0.

$$f1_{score} = 2\frac{precision \times recall}{precision + recall}$$

In the case of multiclass classification, the metrics are taken as one versus many. Therefore the label we are predicting for is the “true” class, anything else is “false”.

In addition the f1 score can be calculated in three methods (14):

- Micro: Calculate metrics globally by counting the total true positives, false negatives and false positives
- Macro: Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account

- Weighted: Calculate metrics for each label, and find their average weighted by support (the number of true instances for each label). This alters 'macro' to account for label imbalance; it can result in an F-score that is not between precision and recall

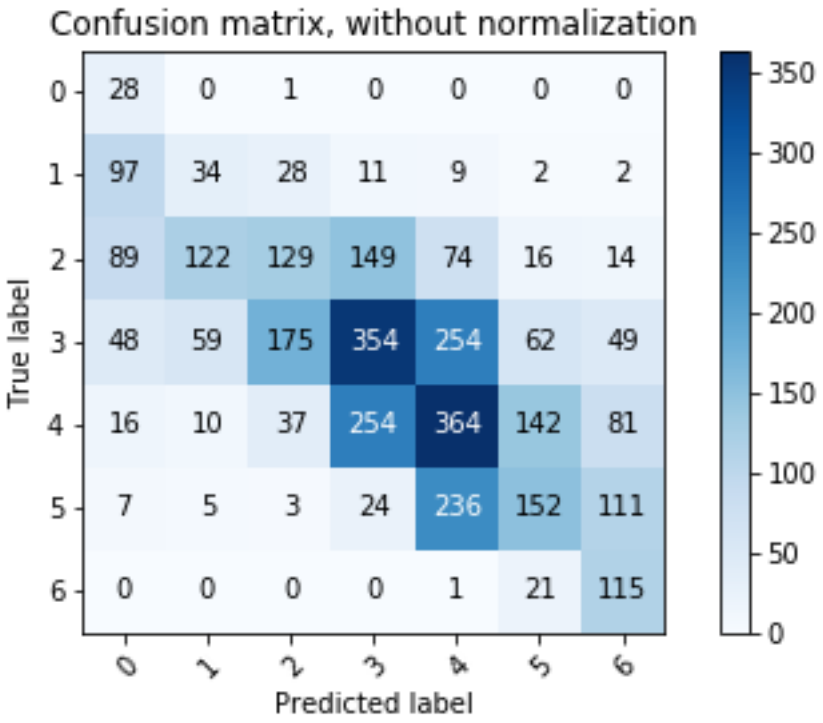
As a final indication of model performance, we consider our baseline to be random guessing. In this case we have 7 possible labels. A completely random model would have an accuracy of:

$$random\ accuracy = \frac{1}{7} \approx 0.142$$

Any model that fails to beat the baseline will be eliminated.

Naïve Bayes

Naïve Bayes (15) is the first supervised learning algorithm to consider. The model is very fast to implement and predict, it is also very useful as a baseline model. The confusion matrix for Naïve Bayes is show in the plot below.



The evaluation report yields:

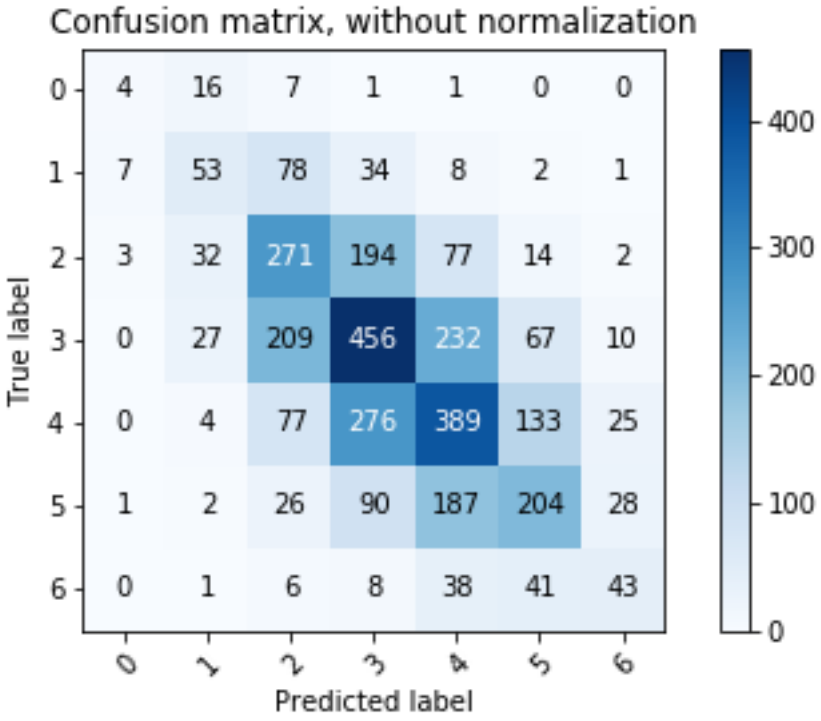
		precision	recall	f1-score	support
	0	0.1	0.97	0.18	29
	1	0.15	0.19	0.16	183
	2	0.35	0.22	0.27	593
	3	0.45	0.35	0.39	1001
	4	0.39	0.4	0.4	904
	5	0.38	0.28	0.33	538
	6	0.31	0.84	0.45	137
micro	avg	0.35	0.35	0.35	3385
macro	avg	0.3	0.46	0.31	3385
weighted	avg	0.38	0.35	0.35	3385

With an accuracy of 34.7%, it is twice as good as as random prediction.

K Nearest Neighbors

K Nearest Neighbors (KNN) is another simple algorithm(16). Building this model only consists of storing the training set. To make a prediction for a new data point, the algorithm finds the point in the training set that is closest to the new point. Then it assigns the label of this training point to the new data point. The k in k-nearest neighbors signifies that instead of using only the closest neighbor to the new data point, we can consider any fixed number k of neighbors in the training.

This model yields an accuracy of 41.9%, improving upon Naïve Bayes. The confusion matrix is shown below:



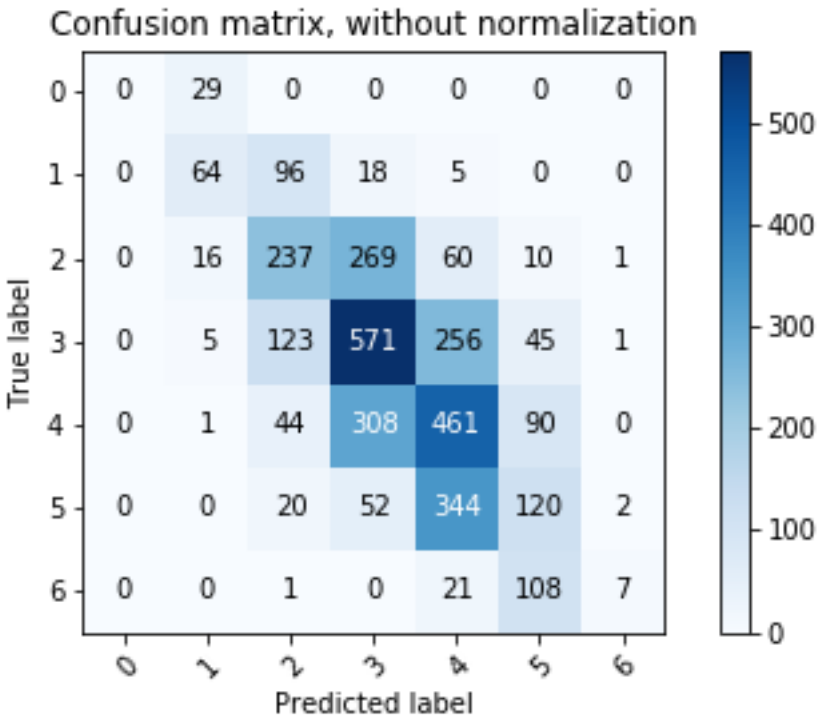
Reporting on the metrics:

		precision	recall	f1-score	support
	0	0.27	0.14	0.18	29
	1	0.39	0.29	0.33	183
	2	0.4	0.46	0.43	593
	3	0.43	0.46	0.44	1001
	4	0.42	0.43	0.42	904
	5	0.44	0.38	0.41	538
	6	0.39	0.31	0.35	137
micro	avg	0.42	0.42	0.42	3385
macro	avg	0.39	0.35	0.37	3385
weighted	avg	0.42	0.42	0.42	3385

Logistic Regression

Logistic regression (17) is another simple model to consider. It scales well to large datasets and highly dimensional data. This model and subsequent ones in this section are sensitive to regularization of the data. By default the model uses L2 regularization.

The model returns an accuracy of 43.1%, slightly improving upon KNN. The confusion matrix is shown below. Notice something peculiar: it fails to detect cases where the user has seen no offer and cases where all offers are seen. This can be seen in the classification report for labels 0 and 6 where the f1 score is highlighted in red.



		precision	recall	f1-score	support
	0	0	0	0	29
	1	0.56	0.35	0.43	183
	2	0.45	0.4	0.43	593
	3	0.47	0.57	0.51	1001
	4	0.4	0.51	0.45	904
	5	0.32	0.22	0.26	538
	6	0.64	0.05	0.09	137
micro	avg	0.43	0.43	0.43	3385
macro	avg	0.41	0.3	0.31	3385
weighted	avg	0.43	0.43	0.42	3385

Support Vector Machines

Support Vector Machines (SVM) is an alogirthm that uses linear classification using what is called a kernel trick(18) to draw a decision boundary between observations. For this more complex algorithm, we will make use of scikit's pipeline feature to sequentially apply a list of transformations and a final estimator(19). Moreover, hyperparameter tuning is needed to optimize the results. For this algorithm and subsequent ones, we will use grid search(20) for the following paramters:

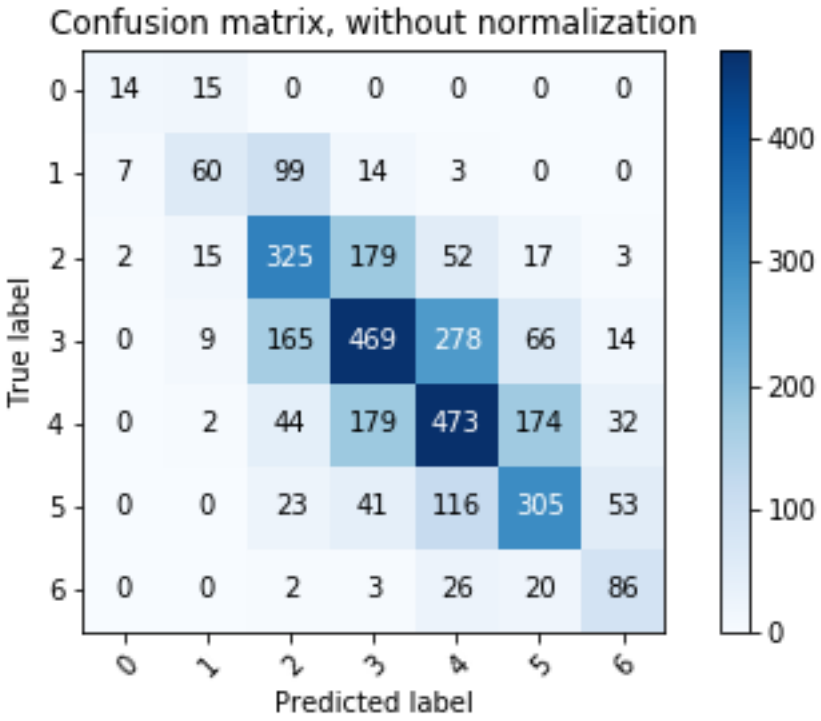
- Penalty paramter C
- Kernel type
- Kernel coeffient Gamma

With no tuning the accuracy is 46%, after grid search it reaches to 51.2%. The confusion matrix and report for the optimized SVC are shown below. We notice that contrary to logistic regression, SVM is now better adapt at predicting instances of no offers viewed (label 0) and all offers viewed (label 6).

It is also worth noting that the time complexity (21) of SVM varies between the following values:

$$O_{SVM} \in [n_{features} \times n_{samples}^2 , n_{features} \times n_{samples}^3]$$

As the algorithm gets more complex, results can improve, and training time increases. An algorithm might be deployed to production in the future, therefore it is critical to know how intensive it will be on the hosting server and the estimated wait time on the consumer side.



		precision	recall	f1-score	support
	0	0.61	0.48	0.54	29
	1	0.59	0.33	0.42	183
	2	0.49	0.55	0.52	593
	3	0.53	0.47	0.5	1001
	4	0.5	0.52	0.51	904
	5	0.52	0.57	0.54	538
	6	0.46	0.63	0.53	137
micro	avg	0.51	0.51	0.51	3385
macro	avg	0.53	0.51	0.51	3385
weighted	avg	0.52	0.51	0.51	3385

Random Forest Classifier

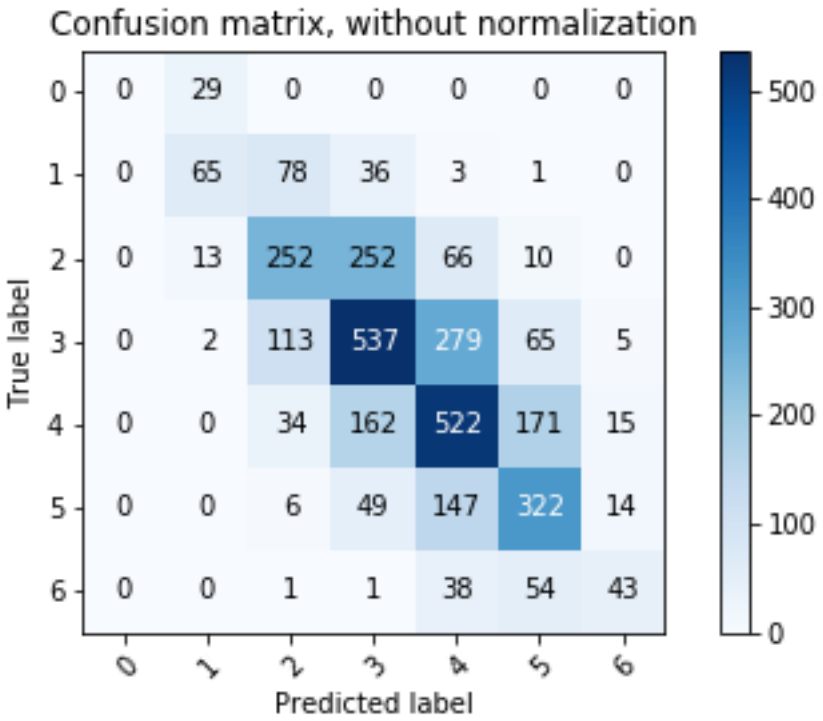
Random decision forest (RF) is an ensemble learning method for classification (22). RF builds many decision trees and averages the results. A drawback of random forests is that they do not perform well on very high dimensional or sparse data. Another limitation comes from computing power that might not be handily available. The most important paramter is the number of estimators, i.e. the number of trees. With grid search the following parmaters are optimized:

- Number of estimators
- Maximum depth of the tree
- Minimum number of samples required to split an internal node
- The minimum number of samples required to be at a leaf node
- To use or not use the solution of the previous call to fit and add more estimators

Our accuracy is 51.4%, a slight improvement from SVMs. However our time complexity is significantly less:

$$O_{Random\ Forest} = n_{features} \times n_{samples} \times \log\left(n_{samples}\right)$$

The confusion matrix and report are shown below. Note that similar to logistic regression, it performs poorly where no offer is seen.



		precision	recall	f1-score	support
	0	0	0	0	29
	1	0.6	0.36	0.45	183
	2	0.52	0.42	0.47	593
	3	0.52	0.54	0.53	1001
	4	0.49	0.58	0.53	904
	5	0.52	0.6	0.55	538
	6	0.56	0.31	0.4	137
micro	avg	0.51	0.51	0.51	3385
macro	avg	0.46	0.4	0.42	3385
weighted	avg	0.51	0.51	0.51	3385

Multi Layer Perceptron

Neural networks are the final kind of classifcation algorithms on the list. A multi layer perceptron (MLP) is a class of feed forward articicial neural networks (23). This method is memory intensive and requires the mos time to train (24). The time complexity is given by:

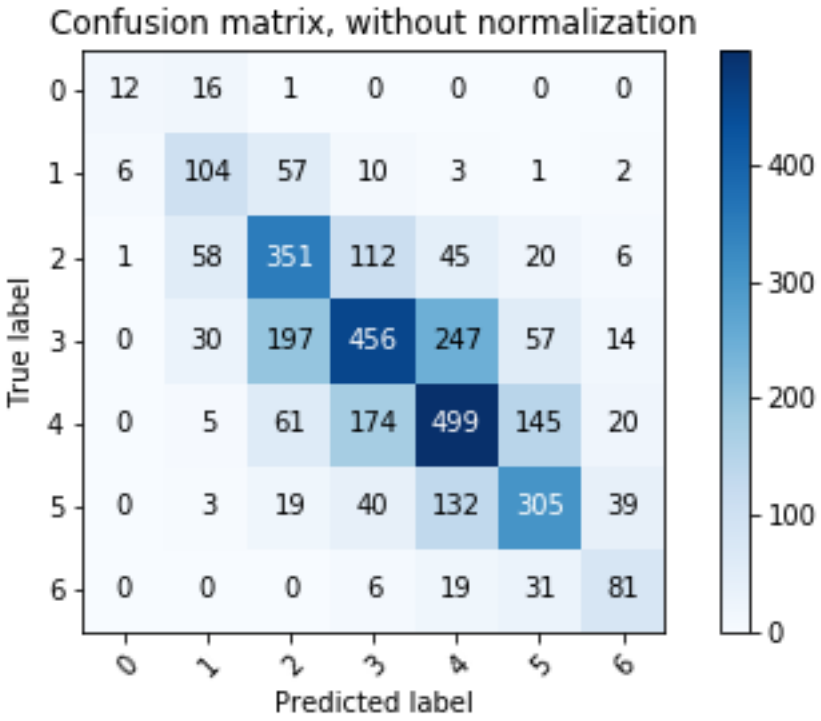
$$O_{MLP} = n \times m \times h^k \times o \times i$$

- n = number of samples
- m= number of features
- h= number of neurons
- o= number of output neurons
- i = number of iterations

For hyperparameter tuning we will optimize:

- Number of neurnons in hidden layers
- L2 penalty parameter
- Learning rate

The MLP outforms all algorithms with an accuracy of 53.4% and in terms of 1 score as shown in the confusion matrix and report below.



		precision	recall	f1-score	support
	0	0.63	0.41	0.5	29
	1	0.48	0.57	0.52	183
	2	0.51	0.59	0.55	593
	3	0.57	0.46	0.51	1001
	4	0.53	0.55	0.54	904
	5	0.55	0.57	0.56	538
	6	0.5	0.59	0.54	137
micro	avg	0.53	0.53	0.53	3385
macro	avg	0.54	0.53	0.53	3385
weighted	avg	0.54	0.53	0.53	3385

Supervised Learning Conlcusion

In order to pick the best classifer, the first metric will be accuracy. This rules out Naïve Bayes, K Nearest Neighrors, and Logistic Regression.

Although MLPs and Random Forest outform SVC they have some drawbacks. MLP is memory intensive and might not perform well in real time. On the other hand RF is poor with outlier cases, where users do not open any offer.

Therefore the best algorithm for supervised learning apperas to be SVC.

Building a Recommendation Engine

In the final part of this project we will seek to build a recommendation engine, the fourth notebook is available on GitHub. The objetive of this section is to see how we can suggest a product to a user, even with little to no prior information about their purchases.

Most Popular Recommendation

Between June 2017 and June 2018, Amercians drank more than 67,000,000 lattes, making it the most popular coffee order in the USA (25). Should the first recommendation for a user always be the most popular item?

Correspondingly for the product at hand, should we always suggest the most popular product? We can measure popularity of an offer by two means:

- The most viewed offer
- Monetary return by view

Although not the most complex method, it is simple and easy to implement. It assumes that the individual's taste is similar to that of the masses. Conversly it offers little benefit in terms of novelty and serendipity.

Collaborative Filtering: User-User Similiarity

A more involved way to suggest products is to measure similarity between users. The underlying assumption behind collaborative filtering is that users who are similar in terms of age, social status, media consumption, and other features will tend to like the same proucts (26).

To measure similarity we can make use of three methods:

- Euclian Distance
- Cosine Similarity

- Pearson Correlation

Euclidian distance measures the ordinary straight line distance between two points in an euclidian space(27).

For two vectors x, and y the Euclidian distance is given by:

$$EUC(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Pearson's correlation is a measure of linear correlation between two variables (28). The coefficient is given by:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$$

Such that cov(X,Y) is the covariance and σ is the variance.

A key mathematical property of Pearson's correlation coefficient:

It is invariant under separate changes in location and scale in the two variables. That is, we may transform X to a + bX and transform Y to c + dY, where a, b, c, and d are constants with b, d > 0, without changing the correlation coefficient. (This holds for both the population and sample Pearson correlation coefficients.)

Note also that if the attribute vectors are normalized by subtracting the vector means, the measure is called the centered cosine similarity and is equivalent to the Pearson correlation coefficient (29).

Therefore for user-user similarity, the preferred measure of similarity is the Pearson correlation.

In implemntation we do the following:

- Compute similarity between one user and the rest
- Rank users from most to least similar (excluding original user)
- Determine offers user has not seen
- Propose unseen offer from most similar user

Matrix Factorization

The last part of remmendation systems presents matrix factorization as a way to decompose a user-item matrix into the product of two lower dimensionality rectangular matrices (30). In

particular we will make use of FunkSVD and its variation of singular value decomposition (31)(32).

As a first step we modify our transcript data to obtan a user-item matrix where the users are the rows, the offer id is the columns, and the values are how many times the offer was seen:

		offer_id									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
person	**0**					1			2	1	1
	1										
	2		1	1							
	3		1	1	1						
	4	1									2

The table above represents a reality of user-item tables, that most values are blank. Indeed we are unable to use singular value decomposition as it requires all values to be known.

If all values in the matrix are known, SVD decomposes an *nxm* matrix in into matrices of the following dimensions:

$$U_{n\times k}$$
$$\Sigma_{k\times k}$$
$$V^T_{k\times m}$$

- n= number of users
- k= number of latent features
- m= number of items

With FunkSVD we first fill the user and item matrices with random values and proceed as follows

- search the item matrix for a rating that already exist.
- Take the dot product between the row associated with the user from U and the column associated with the item from V.
- Calculate error as:

$$Error = (y_{true} - y_{pred})^2$$

- In the user matrix take the derivative of the error with respect to each value identified by *u*:

$$\frac{\delta}{\delta u_i}(y - uv)^2 = -2(y - uv)v_i$$

- In the item matrix take the derivative of the error with respect to each value identified by *v*:

$$\frac{\delta}{\delta v_i}(y - uv)^2 = -2(y - uv)u_i$$

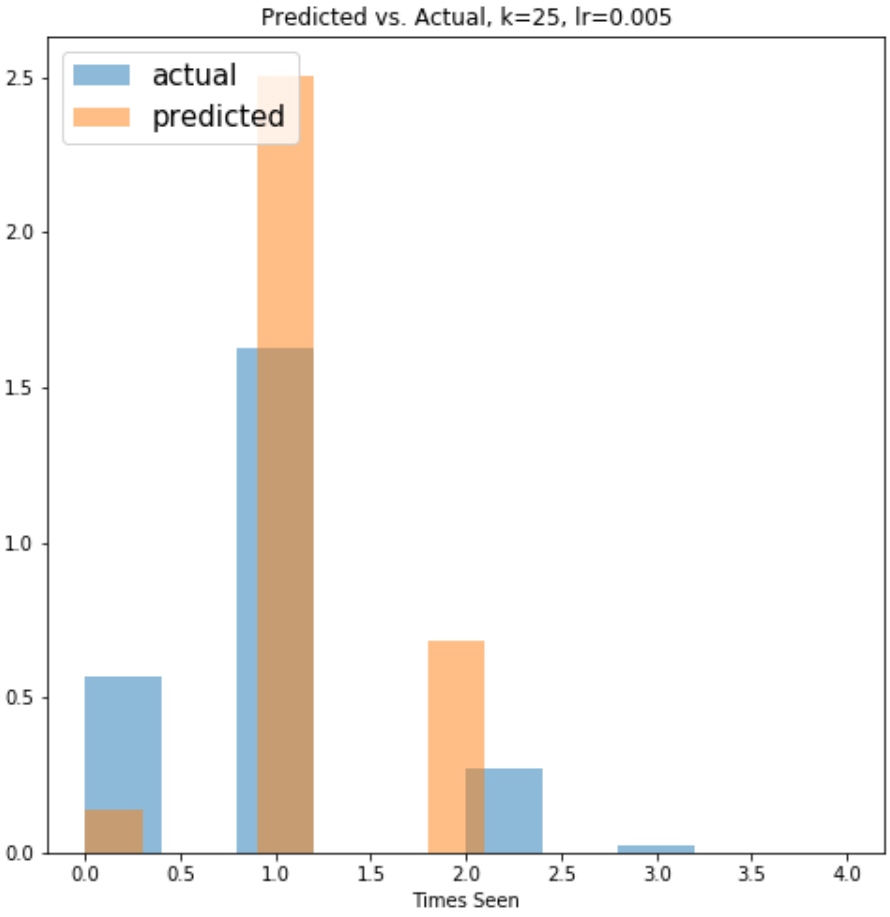
- Update values in matrix with in the opposite direction to minimize error and apply learning rate *α*:

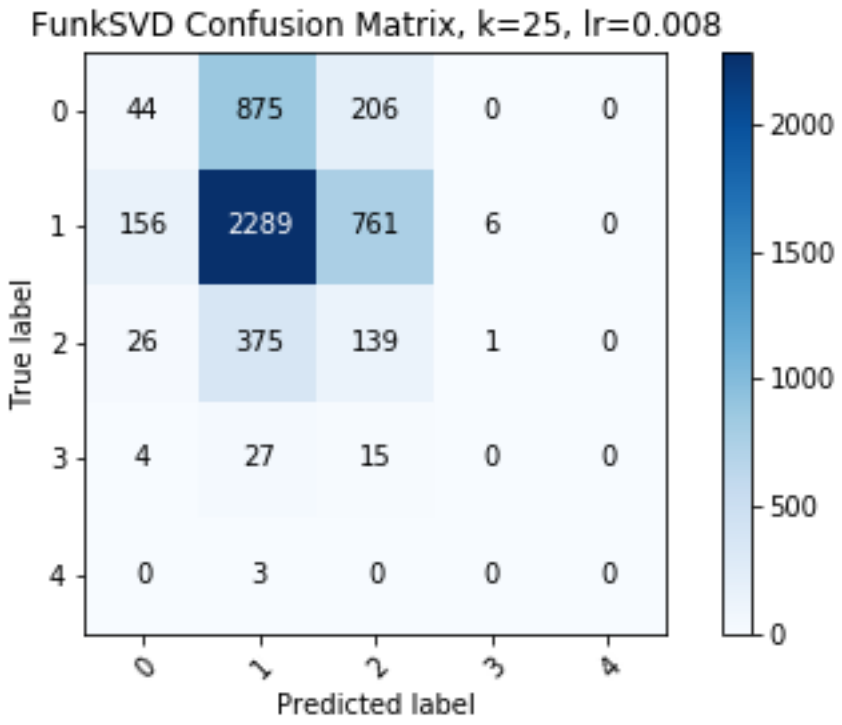
$$u^i_{new} = u^i_{current} + 2\alpha(y - uv)v^i_{current}$$

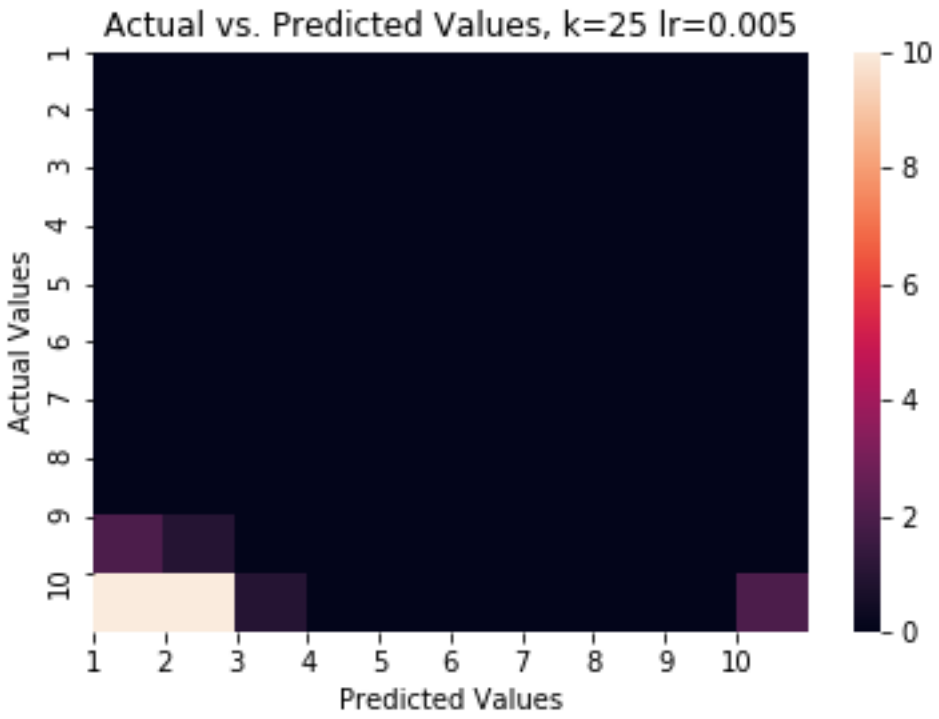
$$v^i_{new} = v^i_{current} + 2\alpha(y - uv)u^i_{current}$$

- Search the next non missing value in user-item matrix
- Repeat for *n* iterations

After grid search, the best learning rate is 0.005 coupled with k=25 latent features.
The plots below show a histogram of actual versus predicted, a confusion matrix, and a heatmap for the validation data.


Predicted vs. Actual, k=25, lr=0.005
actual
predicted
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Times Seen


FunkSVD Confusion Matrix, k=25, lr=0.008
True label
Predicted label
0
1
2
3
4
44 875 206 0 0
156 2289 761 6 0
26 375 139 1 0
4 27 15 0 0
0 3 0 0 0
2000
1500
1000
500
0


Actual vs. Predicted Values, k=25 lr=0.005
Actual Values
Predicted Values
1 2 3 4 5 6 7 8 9 10
10
8
6
4
2
0

Reflection

Throughout the project the majority of time was spent on figuring out an efficient way to clean the transcript data. My main concern was how do I make this information readable should I come back to it, and how do I append it to my main user dataframe. The data is messy and represents a realistic scenario where it must be wrangled and polished before being used.

I was also aware that having mapping dictionaries is immensly useful since it allows to map two separate dataframes consistently. This came in handy when it came to join the tables since the shared key was obvious. In addition I found it useful to prepare a handful of helper functions to reduce clutter in the main notebooks.

Therefore early on I knew I wanted a user dataframe and a user-item matrix and this guided me in the first part of the project. For the unsupervised learning, implementation was relatively simple thanks to sklearn's libraries. Once the feature space was reduced, implementing K-means was a matter of a few lines of code.

I first ran a quick prototype model for some algorithms with the default parameters to get a feel of running time and performance. In addition my first intuition was to look at accuracy since desipte its simplicity it is still the first metric to inspect. I then looked into how I can improve upon my results with grid search and cross validation. Having pipelines in place meant I could train the data without worrying about data leaking.

The final part was the most interesting for me. Data science can appear to be limited to prediction and the business applications seemed limited. Earlier in the course I implemented a case of customer segmentation and saw the business potential. Building a recommendation engine enters the domain of suggestions and calls for more creativity. I believe there is huge potential for this field since it allows businesses to connect with potential users. In addition it is cross-disciplinary since it involves input from product managers, marketing, sales, and a customer centric approach.

Some lessons I got from this project was to leave a lot of notes and comments in the code. In addition I imagined if I presented my work to a technical audience such as colleagues or an outsider such as a salesperson, this affected the kind of language I used. Hence for each step I made sure to explain the intent with a global overview, and follow up with an in-depth explanation if necessary along with sources.

Possible Improvements

The first variance we injected in the model was replacing the missing values with the median. Alternatively we could see how our dataset would vary if we decide to drop these observations with a Kolmogorov-Smirnov test (33).

Furthermore, our predictions in the supervised learning part can be improved upon using feature selection. Indeed it is advisable per Muller and Guido (8) apply pre-pruning for decision tree classifiers since random forests don't tend to perform well on high dimensional data. We can obtain the dominant features by looking into the coefficients for the logistic regression, therefore reducing the feature space. Moreover it is possibe to use the data after PCA transformation for this task. With these changes we expect the random forest model to perform better at least in terms of accuracy and compilation time.

For unsupervised learning only one clustering algorithm was used. It is possible to use DBSCAN or agglomorative clustering and compare silhouette scores. In addition we can make use of the gap statistic proposed by Tibshirani et al (34). The gap statistic uses the clustering ouput of any algorithm and compares the change in within-cluster dispersion with that expected under an appropriate reference null distribution. We might obtain a more representative number of clusters by varying the algorithm used and the measure of correctness.

For the recommendation engine by collaborative filtering, we can make use of Kendall's Tau or Spearman's Rho. Pearson's correlation assumes that both variables are normally distributed and a straight line relationship between the two variables. Although we can't measure accuracy for this kind of recommendation, it useful to see if any offer appears across most possible suggestions.

Conclusion

Predicting consumer spending habits is not a trivial task, this report has made several assumptions about those habits. Namely that purchases that are seen influence spending and that no two offers interact.

We have seen how we can predict customers seeing an offer with supervised learning algorithms while keeping in mind time complexity.

Finally, the report offers a business application by proposing different methods to suggest offers to users.

Acknowlegments

The data for this report was provided with the generosity of Starbucks. Special thanks to the wonderful staff of Starbucks on 3rd Street for their consistently amazing vanilla lattes and the much needed caffeine boosts.

References

1) https://www.reuters.com/article/us-coffee-conference-survey/americans-are-drinking-a-daily-cup-of-coffee-at-the-highest-level-in-six-years-survey-idUSKCN1GT0KU
2) https://www.statista.com/topics/1246/starbucks/
3) https://notesmatic.com/starbucks-marketing-and-advertising-expenses/
4) Blank, Steven G. *The Four Steps to the Epiphany Successful Strategies for Products That Win*. S. Blank, 2013.
5) https://web.stanford.edu/group/e145/cgi-bin/winter/drupal/upload/handouts/Four_Steps.pdf
6) https://www.udacity.com/course/data-scientist-nanodegree--nd025
7) https://en.wikipedia.org/wiki/Pearson_correlation_coefficient
8) Müller, Andreas C., and Sarah Guido. *Introduction to Machine Learning with Python* Ji Qi Xue Xi Ru Men. Dong Nan Da Xue Chu Ban She, 2017.
9) https://en.wikipedia.org/wiki/Principal_component_analysis
10) https://www.geeksforgeeks.org/ml-mini-batch-k-means-clustering-algorithm/
11) https://en.wikipedia.org/wiki/Elbow_method_(clustering)
12) https://en.wikipedia.org/wiki/Silhouette_(clustering)
13) https://en.wikipedia.org/wiki/Confusion_matrix
14) https://scikit-learn.org/stable/modules/model_evaluation.html#from-binary-to-multiclass-and-multilabel
15) https://en.wikipedia.org/wiki/Naive_Bayes_classifier
16) https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
17) https://en.wikipedia.org/wiki/Logistic_regression
18) https://en.wikipedia.org/wiki/Support-vector_machine
19) https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
20) https://en.wikipedia.org/wiki/Hyperparameter_optimization#Grid_search
21) https://en.wikipedia.org/wiki/Time_complexity
22) https://en.wikipedia.org/wiki/Random_forest
23) https://en.wikipedia.org/wiki/Multilayer_perceptron
24) https://scikit-learn.org/stable/modules/neural_networks_supervised.html#complexity
25) https://squareup.com/townsquare/coffee-day-2018
26) https://en.wikipedia.org/wiki/Collaborative_filtering
27) https://en.wikipedia.org/wiki/Euclidean_distance
28) https://en.wikipedia.org/wiki/Pearson_correlation_coefficient
29) https://www.researchgate.net/post/Can_someone_differentiate_between_Cosine_Adjusted_cosine_and_Pearson_correlation_similarity_measuring_techniques
30) https://en.wikipedia.org/wiki/Matrix_factorization_(recommender_systems)
31) https://sifter.org/~simon/journal/20061211.html
32) https://en.wikipedia.org/wiki/Singular_value_decomposition
33) https://en.wikipedia.org/wiki/Kolmogorov–Smirnov_test
34) https://statweb.stanford.edu/~gwalther/gap