Starbucks Project

REPORT

Abdoulaye Diallo
224apps@gmail.com

I. DEFINITION:

A. Domain Background:

Starbucks Corporation is an American coffee company and coffeehouse chain. In reality, the Starbucks app sends out various types of promotional offers to customers, either discounts (BOGO or 50% off during happy hours) or Star Dash challenges (completing required purchases to earn star rewards). Sometimes it also informs customers about limited-time drinks, such as those colorful Instagram Frappuccinos. In a simulated environment, Starbucks sends out three types of offers (BOGO, discount and informational) via multiple channels. Customers' responses to offers and transactions are recorded. In either setting, it is important to send the right offer to the right customer.

B. Proposed solution:

By using the transcript data, We'll extract the responses from customers towards the promotional offers sent to them. With the response as labels, and properties of each customer and offer pair as features, a binary classifier can be trained to predict whether a customer will positively respond to a promotional offer.

C. Problem Statement:

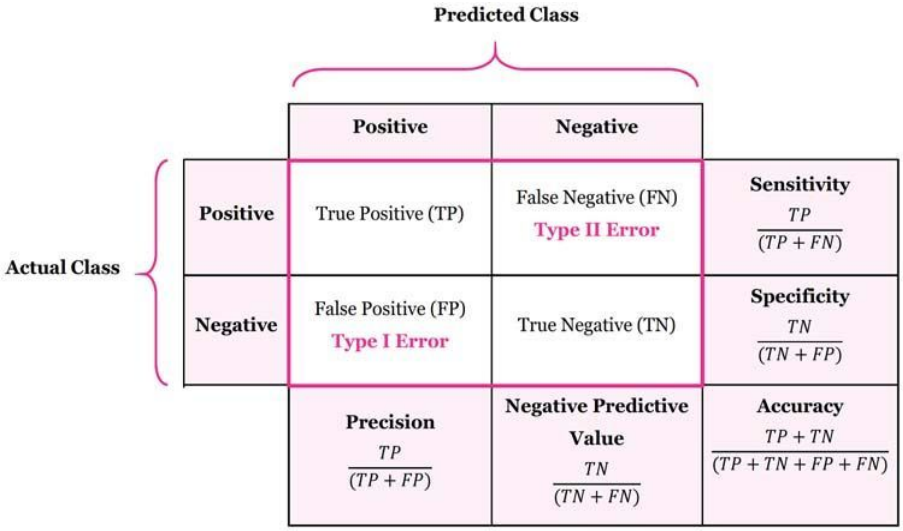
In this project, I will build a model to predict whether a customer will respond to a promotional offer based on the features of customer and offer.

Datasets and Inputs: 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 The data is provided by Starbucks and Udacity.

C. Metrics:

The performance of models will be measured using two metrics, accuracy and F1 score. It will be evaluated on unseen data. Confusion matrix will be used to classify the relationship between the prediction and the ground truth.



Besides the accuracy, we can use F1 Score by combining the precision formula to the sensitivity method.

Accuracy is a good measure when the target classes are nearly balanced. It will become less effective facing highly imbalanced data. For instance, in fraud detection, the positive rate could be lower than 1%. A naive model predicting all instances to be negative will yield an accuracy above 99%, but will never get the job done. In this case, metrics like precision and recall are more sensitive.

Precision noted as P. Recall or Sensitivity noted as R

F1 = 2 * P * R / (P + R)

II. ANALYSIS:

A. Data Exploration:

We used pandas to get the JSON file into variables : portfolio, transcript and profile respectively.

Portfolio: contains features like channels, difficulty, duration, id, offer_type and reward. It has 10 different offers, categorized into 3 types (bogo, discount and informational)

	channels	difficulty	duration	id	offer_type	reward
0	[email, mobile, social]	10	7	ae264e3637204a6fb9bb56bc8210ddfd	bogo	10
1	[web, email, mobile, social]	10	5	4d5c57ea9a6940dd891ad53e9dbe8da0	bogo	10
2	[web, email, mobile]	0	4	3f207df678b143eea3cee63160fa8bed	informational	0
3	[web, email, mobile]	5	7	9b98b8c7a33c4b65b9aebfe6a799e6d9	bogo	5
4	[web, email]	20	10	0b1e1539f2cc45b7b9fa7c272da2e1d7	discount	5
5	[web, email, mobile, social]	7	7	2298d6c36e964ae4a3e7e9706d1fb8c2	discount	3
6	[web, email, mobile, social]	10	10	fafdcd668e3743c1bb461111dcafc2a4	discount	2
7	[email, mobile, social]	0	3	5a8bc65990b245e5a138643cd4eb9837	informational	0
8	[web, email, mobile, social]	5	5	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5
9	[web, email, mobile]	10	7	2906b810c7d4411798c6938adc9daaa5	discount	2

Profile: contains customer information. There are notable NaNs found for gender, age and income. Strategies to deal with these NaNs will be displayed below.

	age	became_member_on	gender	id	income
0	118	20170212	None	68be06ca386d4c31939f3a4f0e3dd783	NaN
1	55	20170715	F	0610b486422d4921ae7d2bf64640c50b	112000.0
2	118	20180712	None	38fe809add3b4fcf9315a9694bb96ff5	NaN
3	75	20170509	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0
4	118	20170804	None	a03223e636434f42ac4c3df47e8bac43	NaN
5	68	20180426	M	e2127556f4f64592b11af22de27a7932	70000.0
6	118	20170925	None	8ec6ce2a7e7949b1bf142def7d0e0586	NaN
7	118	20171002	None	68617ca6246f4fbc85e91a2a49552598	NaN
8	65	20180209	M	389bc3fa690240e798340f5a15918d5c	53000.0

All these rows with NaNs have gender and income as NaN, and age of 118. Imputation is required for gender, age and income.A ge and gender will filled with the median of its respective numerical fields and 'U' is applied for an unknown gender. Since the three fields are either all missing or full, only one column 'profile_nan' is added as an indication of missing values.

```
]: profile.loc[profile.isnull().any(axis=1), 'age'] = None
   profile['profile_nan'] = 0
   profile.loc[profile.isnull().any(axis=1), 'profile_nan'] = 1
   profile['gender'].fillna('U', inplace=True)
   profile['age'].fillna(profile['age'].median(), inplace=True)
   profile['income'].fillna(profile['income'].median(), inplace=True)
```

```
]: profile.head()
```

]:

	age	became_member_on	gender	id	income	profile_nan
0	55.0	20170212	U	68be06ca386d4c31939f3a4f0e3dd783	64000.0	1
1	55.0	20170715	F	0610b486422d4921ae7d2bf64640c50b	112000.0	0
2	55.0	20180712	U	38fe809add3b4fcf9315a9694bb96ff5	64000.0	1
3	75.0	20170509	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0	0
4	55.0	20170804	U	a03223e636434f42ac4c3df47e8bac43	64000.0	1

Transcript: Transcript records all events during this 30-day simulation.

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

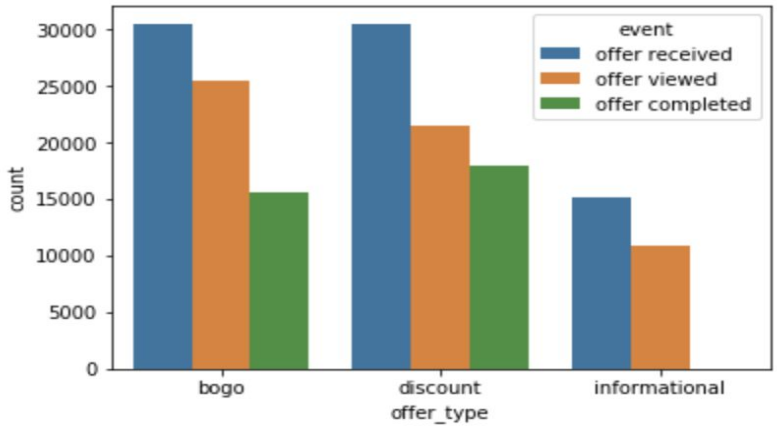
There are four types of events recorded in transcript, transactions and three types of offer activities. By looking at the activities of one customer, the meaning behind each offer activity becomes clear.

- offer received' marks the time when Starbucks sends the offer to the customer, and the offer becomes effective instantly regardless whether the customer has viewed it or not.
- offer viewed' marks the time when customer views the offer or becomes aware of it. Note that the customer can still view the offer even if he/she has already completed without knowing it, as long as the time is still within the offer's duration.
- offer completed' marks the time when the customer makes qualified transaction meeting the requirement. The dict in the 'value' column has an additional key 'reward', which is associated with the offer id in portfolio.

To further explore offer activities, split transcript to transactions and offer activities, and extract out transaction amount and offer id from transactions and offer activities, respectively.

```
offer_log = transcript[transcript['event'].str.startswith('offer')]
offer_log.loc[:, 'offer'] = offer_log['value'].apply(lambda c: c['offer id'] if 'offer id' in c else c['offer_id'])
offer_log.drop(['value'], axis=1, inplace=True)
trans_log = transcript[transcript['event'] == 'transaction']
trans_log.loc[:, 'amount'] = trans_log['value'].apply(lambda c: c['amount'])
trans_log.drop(['value'], axis=1, inplace=True)
```

```
ax = sns.countplot(x='offer_type', hue='event', data=offer_types)
```



It is noteworthy that informational offers do not have completion events, possibly because there is no reward to claim. Based on the understanding of offer activities, here are the proposed rules for processing transcript to label responses:

- A positive response requires both customer's awareness (viewed) and offer completion. The former must occur before the latter.
- For informational offers, the response is determined from the number of transactions within the effective time window between viewed time and offer expiration. No transactions simply indicate a negative response.

Use the 'offer received' event as the starting point for labeling. Explore more details about the offers sent.

```
offer_receive = offer_log[offer_log['event'] == 'offer received']
```

```
offer_receive['time'].unique() / 24
```

```
array([ 0.,  7., 14., 17., 21., 24.])
```

During the 30-day simulation, Starbucks only sends out six waves of offers. The time interval between offer waves varies from three days to a week.

```
tmax = transcript.iloc[-1]['time']
tmax / 24
```

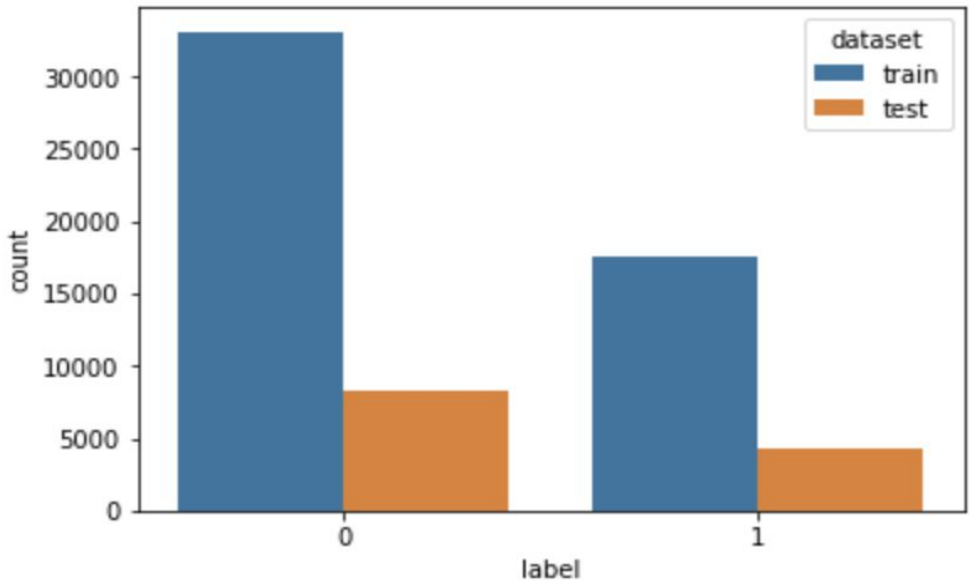
```
29.75
```

the final wave of offers arrives on the 24th day, leaving less than six days in transcript. Some offers have longer duration and the final response remains unknown if they are not completed by the final time mark in transcript. Those offers will be dropped during processing.

```
]: offer_receive.groupby(['person', 'offer']).count()['event'].value_counts()
```

```
]: 1    51570
   2    10523
   3     1124
   4       66
   5        5
   Name: event, dtype: int64
```

As a final remark, a unique customer-offer pair may have different responses since there are customers getting the same offer for more than once. Their responses may not necessarily remain the same. However, a model predicting customer-offer match should yield a unique output based on the input. To simplify the problem, a customer-offer pair is labeled as positive as long as the customer responses to the offer positively for once.



After finishing data processing and putting together data for modeling, I did another round of EDA before jumping into training the classifier. The target label (positive as 1 and negative as 0) has a 2:1 ratio in both training and test set, i.e., the data is not highly imbalanced. In this case, **accuracy** alone is good enough to measure the model performance, and F1 score will not be used.

```
train_X.columns
```

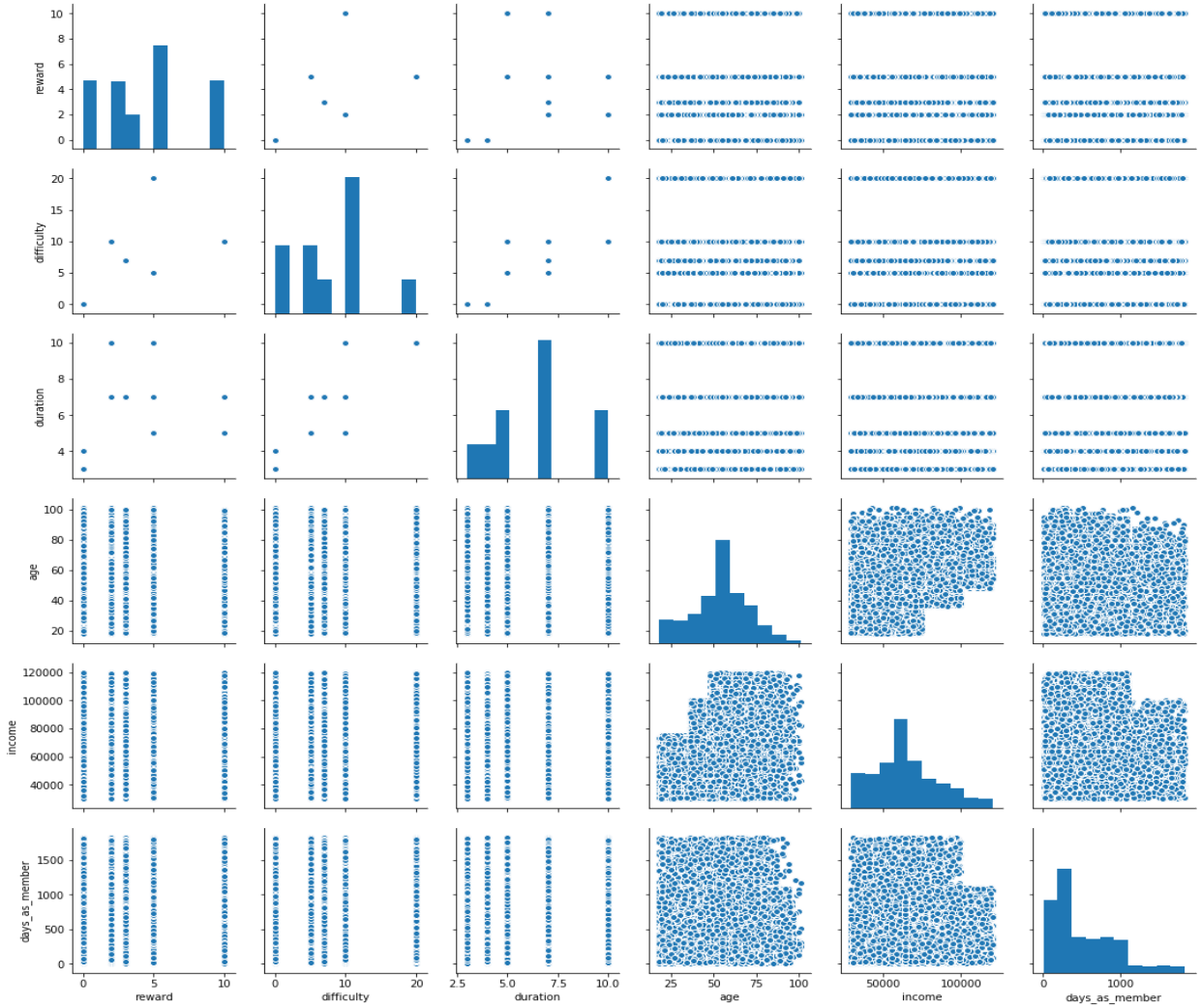
```
Index(['difficulty', 'duration', 'reward', 'channel_web', 'channel_mobile',
       'channel_social', 'type_bogo', 'type_discount', 'type_informational',
       'age', 'income', 'profile_nan', 'days_as_member', 'gender_F',
       'gender_M', 'gender_O', 'gender_U'],
      dtype='object')
```

All features can be grouped into two types, binary (0, 1) labeled or numerical.

Visualize the distribution and pair relationship of numerical columns.

```
axes = sns.pairplot(train_X[['reward', 'difficulty', 'duration', 'age', 'income', 'days_as_member']])
axes.savefig(os.path.join(fig_dst, 'num_feature_pair.png'), dpi=300)
```

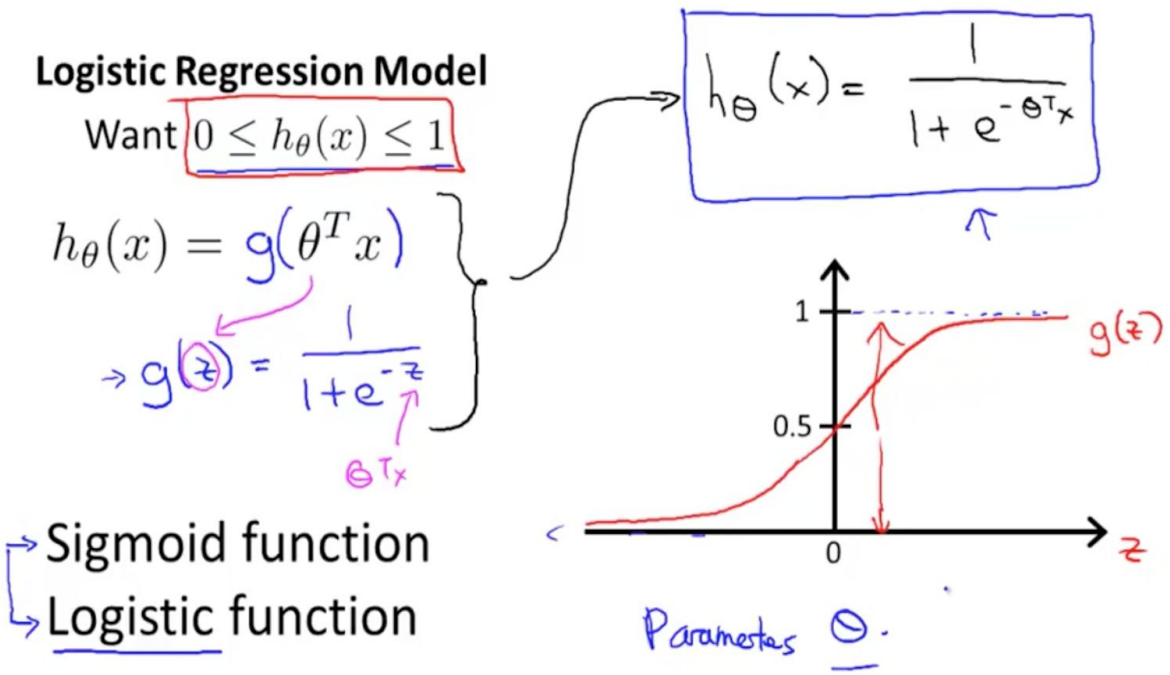
There are 17 features available from customer or offer characteristics. These features can be grouped into two types, binary labeled (including some one-hot encoded ones) and numerical.



A straightforward observation from the pair plot of numerical features is that all offer attributes are discrete while customer ones are continuous. Moreover, no clear correlation can be observed between any pair of customer attributes. Therefore, the features will be used as is, with a bit of preprocessing to scale them in case the algorithm is distance based (e.g., SVM).

B. Benchmark model:

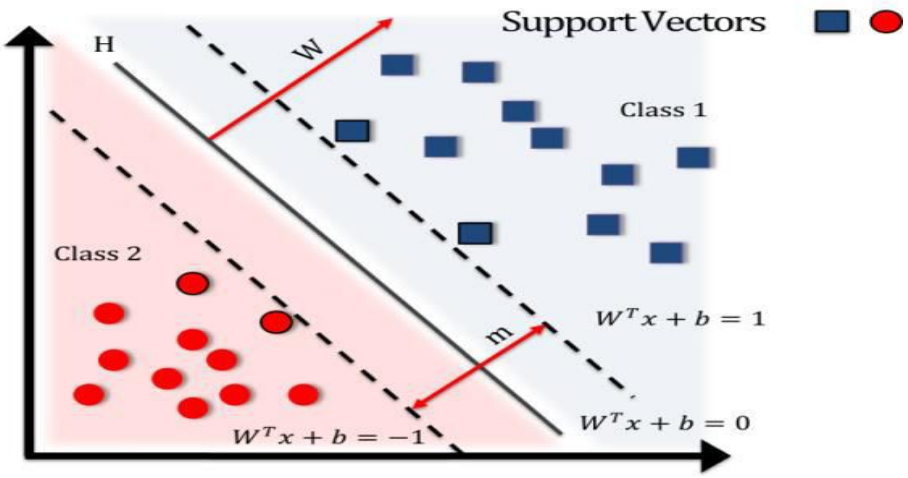
A logistic regression model will serve as the benchmark model in this project. Logistic regression is possibly the most popular algorithm for binary classification problems in industry



Besides the benchmark, we will also use the following algorithms:

Support vector machine(SVM):

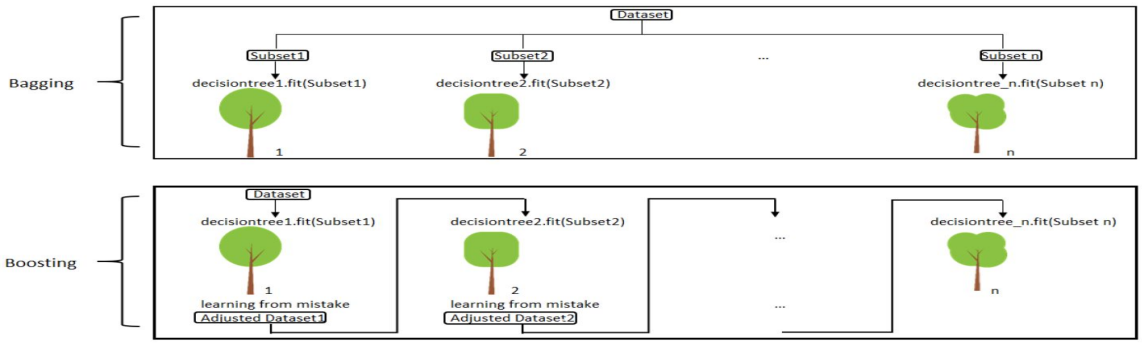
SVM is another learning algorithm used in classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.



Decision Trees:

A decision tree is a flowchart-like structure in which each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label.

A single decision tree is rarely used in machine learning due to its instability. Small variations in the training data could lead to tremendous changes in the structure of optimal decision tree. In practice, decision trees are applied as the base estimator of ensemble methods, such as bagging and boosting.

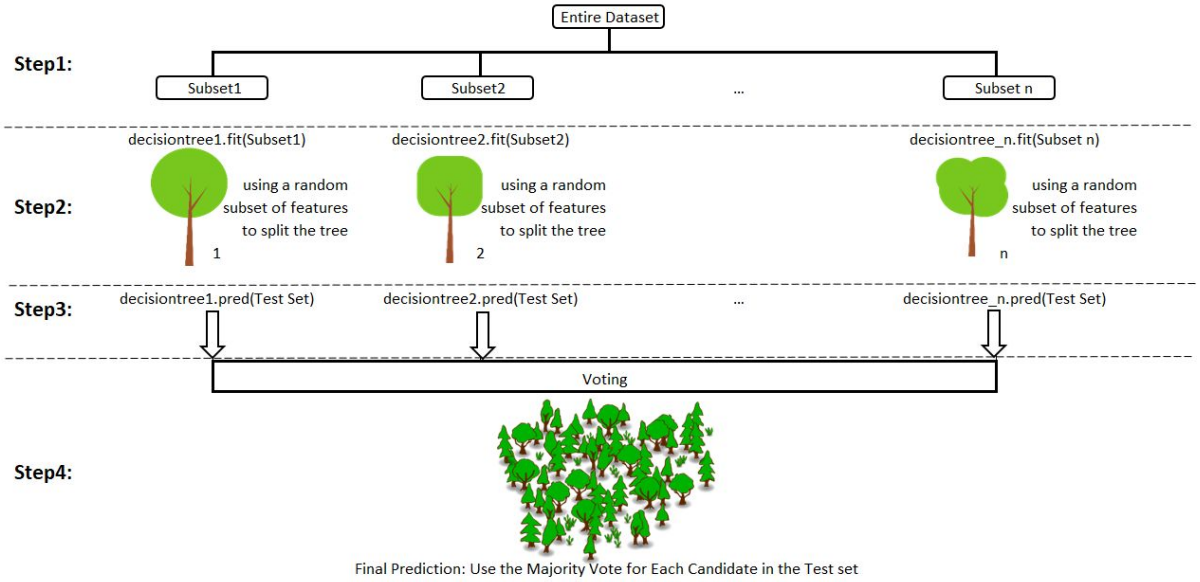


Bagging: Training a bunch of individual models in a parallel way. Each model is trained by a random subset of the data

Boosting: Training a bunch of individual models in a sequential way. Each individual model learns from mistakes made by the previous model

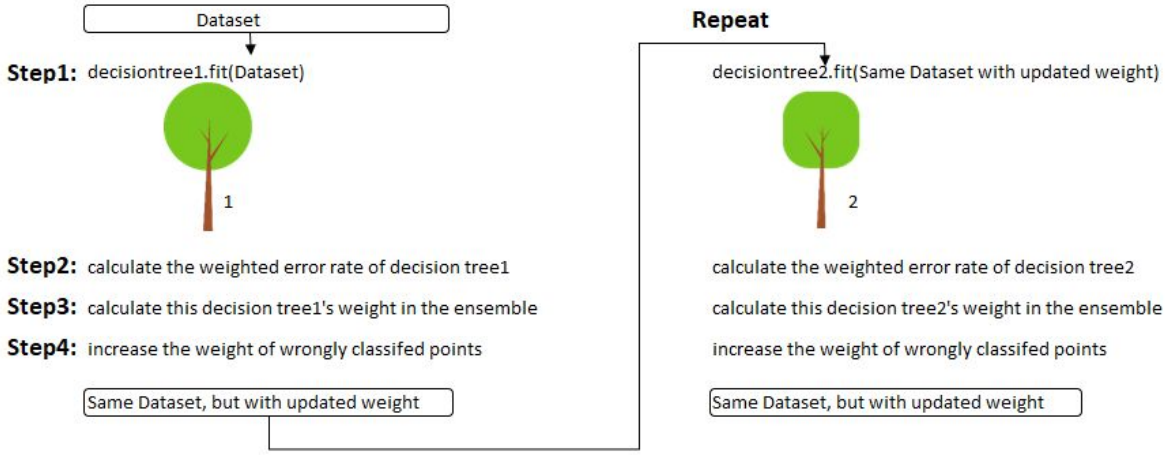
Random Forest:

Random forest is an extension of decision tree bagging with additional randomness in selection subset of features.



Gradient Descent:

In gradient boosting, a new decision tree is added to the ensemble by learning from the residual directly. It is an extremely popular solution in online machine learning competitions.



III. Methodology:

A. Data Cleaning :

a. **To handle the Nans value on the profile dataset**, Imputation is required for gender, *age* and *income*. *Age* and *gender* will be filled with the median of its respective numerical fields and 'U' is applied for an unknown *gender*. Since the three fields are either all missing or full, only one column 'profile_nan' is added as an indication of missing values. After the result we get this:

	age	became_member_on	gender	id	income	profile_nan
0	55.0	20170212	U	68be06ca386d4c31939f3a4f0e3dd783	64000.0	1
1	55.0	20170715	F	0610b486422d4921ae7d2bf64640c50b	112000.0	0
2	55.0	20180712	U	38fe809add3b4fcf9315a9694bb96ff5	64000.0	1
3	75.0	20170509	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0	0
4	55.0	20170804	U	a03223e636434f42ac4c3df47e8bac43	64000.0	1

b. **Process transcript to label responses**:

Implement a label_response method to process transcript with one particular customer-offer pair as the unit. First, add a 'positive_response' column to offer_receive with starting value 0 (negative), and then loop label_response through all appearing customers and offers sent to each one of them to label positively responded offers to 1. A few key ideas in addition to the proposed **rules** reflected in label_response :

- Use a queue to store active offers represented by a list [t_receive, t_expire, t_viewed] since offers have FIFO nature. Go through the offer activities by time, and pop offers completed or expired.
- Treat informational offers differently. Besides the way a positive response is determined, the way to decide which offer is viewed when multiple offers (with same id) active. Label earlier BOGO, discount offers as viewed as offer chasers will complete them first to maximize their rewards. On the other hand, label later informational offers as viewed to extend the effective time window.

Once the loop finishes, group offer_receive by 'person' and 'offer' using logic OR. After dropping other columns, a three-column data frame appears to be the skeleton of the final dataset.

```
response = offer_receive.groupby(['person', 'offer']).any().reset_index()
response.drop(columns=['event', 'time'], inplace=True)
response.head()
```

	person	offer	positive_response
0	0009655768c64bdeb2e877511632db8f	2906b810c7d4411798c6938adc9daaa5	False
1	0009655768c64bdeb2e877511632db8f	3f207df678b143eea3cee63160fa8bed	False
2	0009655768c64bdeb2e877511632db8f	5a8bc65990b245e5a138643cd4eb9837	False
3	0009655768c64bdeb2e877511632db8f	f19421c1d4aa40978ebb69ca19b0e20d	False
4	0009655768c64bdeb2e877511632db8f	fafdcd668e3743c1bb461111dcafc2a4	False

B. Data Preprocessing:

Encode categorical variables and assemble dataset :

- For the portfolio, expand channels into four columns, drop the email option and use one-hot encoding for ‘offer-type’
- For the profile, convert ‘became_member_on’ to membership days, then, use on-hot encoding for ‘gender’

Then, join both, processed portfolio and profile onto the skeleton and drop all hash columns (customer and offer id). Create a test set with 20% data for final metric evaluation

Scale numerical variables: Implement NumericalTransformer to preprocess numerical columns, leveraging on the scikit-learn api. The discrete ones are scaled using MinMaxScaler , while the continuous ones are scaled using StandardScaler . Note that this preprocessing step is completely optional for tree-based algorithms since they are non-distance based.

3. Model Evaluation and Validation:

Select optimal algorithm

Use out-of-the-box classifiers to select optimal classification algorithm. The score is taken from the mean of 5-fold cross validation accuracy.

Logistic regression (benchmark)

```
In [12]: from sklearn.linear_model import LogisticRegression
         logistic = make_pipeline(NumericalTransformer(), LogisticRegression())
         cross_val_score(logistic, train_X, train_y, scoring='accuracy', cv=5, n_jobs=-1).mean()
Out[12]: 0.7365592954908515
```

Support vector machine

```
In [13]: from sklearn.svm import LinearSVC
         svm = make_pipeline(NumericalTransformer(), LinearSVC(dual=False))
         cross_val_score(svm, train_X, train_y, scoring='accuracy', cv=5, n_jobs=-1).mean()
Out[13]: 0.7346828966249379
```

Random Forest

```
In [14]: from sklearn.ensemble import RandomForestClassifier
         cross_val_score(RandomForestClassifier(n_estimators=100), train_X, train_y, scoring='accuracy', cv
         =5, n_jobs=-1).mean()
Out[14]: 0.7374480946484973
```

Gradient boosting

```
In [25]: from sklearn.ensemble import GradientBoostingClassifier
         cross_val_score(GradientBoostingClassifier(), train_X, train_y, scoring='accuracy', cv=5, n_jobs=-
         1).mean()
Out[25]: 0.7594311595545019

In [26]: from xgboost import XGBClassifier
         cross_val_score(XGBClassifier(), train_X, train_y, scoring='accuracy', cv=5, n_jobs=-1).mean()
Out[26]: 0.7580288289089329

In [27]: from lightgbm import LGBMClassifier
         cross_val_score(LGBMClassifier(), train_X, train_y, scoring='accuracy', cv=5, n_jobs=-1).mean()
Out[27]: 0.7637567343733975
```

Gradient boosting implemented in LightGBM is the winner.

Hyperparameter tuning:

Tuning hyperparameters for gradient tree boosting itself is a high dimensional problem. Here I followed a guide on hyperparameter tuning on XGBoost to break it down to a few steps of low dimensional grid search:

- Keep a high *learning_rate* (0.1) until the final step. Find optimal parameters controlling the growth of an individual tree (num_leaves and max_depth).
- Tune other parameters related to tree growth (min_data_in_leaf and min_sum_hessian_in_leaf).
- Tune "random forest" parameters (bagging_fraction and feature_fraction).
- Tune regularization parameters (lambda_l1 and lambda_l2).
  Try lowering learning_rate and more rounds of iterations to see if the score further improves.

The entire process is leveraged on the cross validation API from LightGBM, with early stopping enabled and number of iterations returned. Here is the final optimal set of hyperparameters, and default values are used for those not mentioned.

```
In [40]: lgbm = LGBMClassifier(num_leaves=127, max_depth=14, n_estimators=98)
```

IV. Results:

After training both benchmark and optimal models on the entire training set, evaluate their performance on the test set.

Logistic regression (benchmark)

```
In [12]: from sklearn.linear_model import LogisticRegression
         logistic = make_pipeline(NumericalTransformer(), LogisticRegression())
         cross_val_score(logistic, train_X, train_y, scoring='accuracy', cv=5, n_jobs=-1).mean()

Out[12]: 0.7365592954908515
```

```
In [27]: from lightgbm import LGBMClassifier
         cross_val_score(LGBMClassifier(), train_X, train_y, scoring='accuracy', cv=5, n_jobs=-1).mean()

Out[27]: 0.7637567343733975
```

To validate model robustness with no additional data, reshuffle the training and test data with different random states and evaluate performance on the new test data using the model trained from the new training data. The accuracy score seems stable regardless of how the dataset splits, suggesting the solution is robust.

```
In [44]: all_data = pd.read_csv(os.path.join(data, 'all_data.csv'))
         y = all_data['positive_response'].astype(int)
         X = all_data.drop(columns=['positive_response'])
```

```
In [45]: from sklearn.model_selection import train_test_split

         scores = []
         for seed in [8, 24, 13, 10, 41, 35, 7, 30]:
             train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.2, random_state=seed)
             lgbm.fit(train_X, train_y)
             lgbm_y = lgbm.predict(test_X)
             scores.append(accuracy_score(lgbm.predict(test_X), test_y))
         scores
```

```
Out[45]: [0.7622847211249802,
          0.7657607836941065,
          0.7595986727761099,
          0.7721598988781798,
          0.7664717964923369,
          0.7639437509875178,
          0.7649707694738506,
          0.7659977879601833]
```

Justification:

Overall, the optimal LightGBM model has minor improvement in predicting accuracy comparing with the benchmark logistic regression model. Both models solve the problem, predicting whether a customer will positively responds to a promotional offer based on their characteristics. The benchmark model is already a good starting point, and the improvement may seem marginal at first glance. But its impact will be amplified considering the business revenue of Starbucks, a worldwide coffee retailer.

References:

- Wikipedia:
- Scikit-learn
- Accuracy Score.
- F1 Score
- Support Vector Machine
- Decision Tree
- Towards Data Science