Machine learning Engineer Nanodegree

Capstone Project Report

Zhi Deng

Oct 29 2019

Introduction

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/menu challenges (completing required purchases to earn star rewards). Sometimes it also informs customers about limited-time drinks, such as those colorful Instagram Frappuccinos. As a marketing strategy, the goal of these promotional offers is to encourage customers to buy drinks at Starbucks and build loyalty among customers in the long run. Since Starbucks has a variety of products and customers' tastes also vary, it is important to send the right offer to the right customer by building a highly personalized recommendation system.

Problem definition

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. The problem I proposed is to build a model to predict whether a customer will positively respond to a promotional offer.

Data

The data is provided by Starbucks and Udacity, in three JSON files:

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)

portfolio.json

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)

transcript.json

Event log (306648 events x 4 fields)

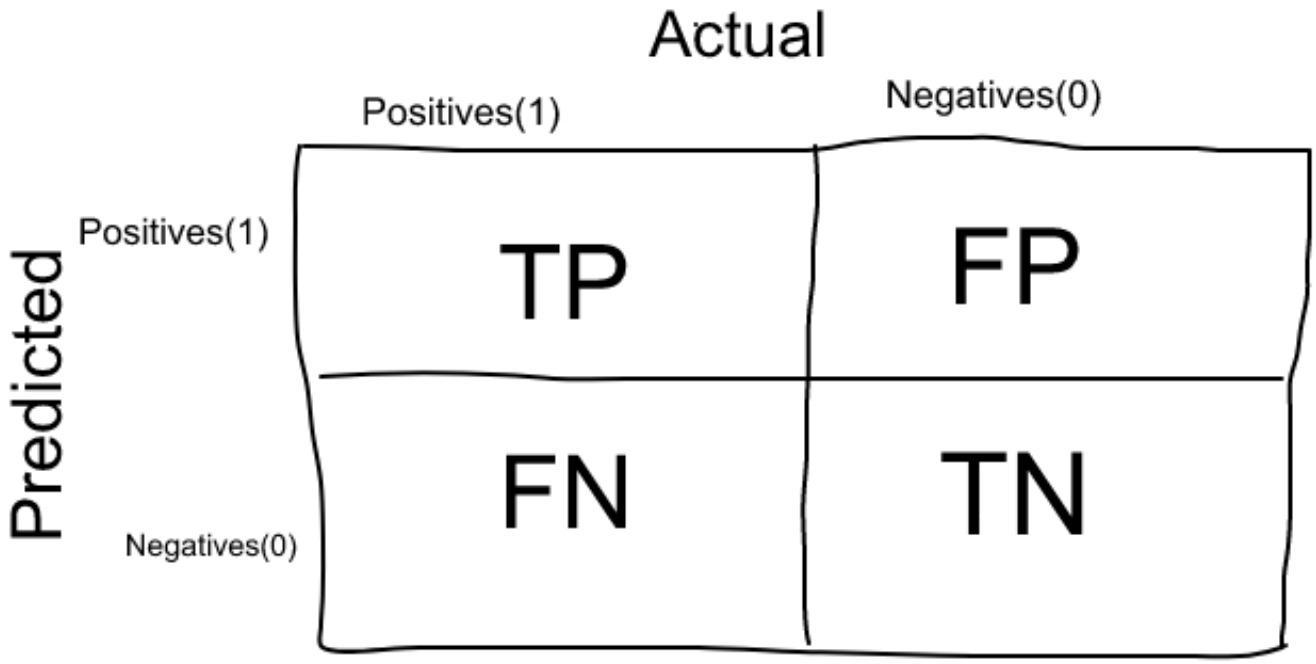
- person: (string/hash)
- event: (string) offer received, offer viewed, transaction, offer completed
- value: (dictionary) different values depending on event type
    - offer id: (string/hash) not associated with any "transaction"
    - amount: (numeric) money spent in "transaction"
- time: (numeric)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

Proposed solution

The transcript will be processed to 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.

Metrics

The performance of solutions will be evaluated on a large unseen dataset. For a binary classification problem, the relationship between prediction and ground truth can be visualized by the confusion matrix,



where all data is sliced into four groups:

- True positive (TP): both prediction and ground truth are positive class
- True negative (TN): both prediction and ground truth are negative class
- False positive (FP): predicted to be positive, but in fact negative
- False negative (FN): predicted to be negative, but in fact positive

Many scalar metrics can be derived from the confusion matrix. Defined by the ratio of correct predictions, accuracy can be expressed as:

- Accuracy = (TP + TN) / (TP + FP + FN + TN)

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 (P) = TP / (TP + FP)
- Recall (R) = TP / (TP + FN)

One can also consider both precision and recall at the same time using F1 score.

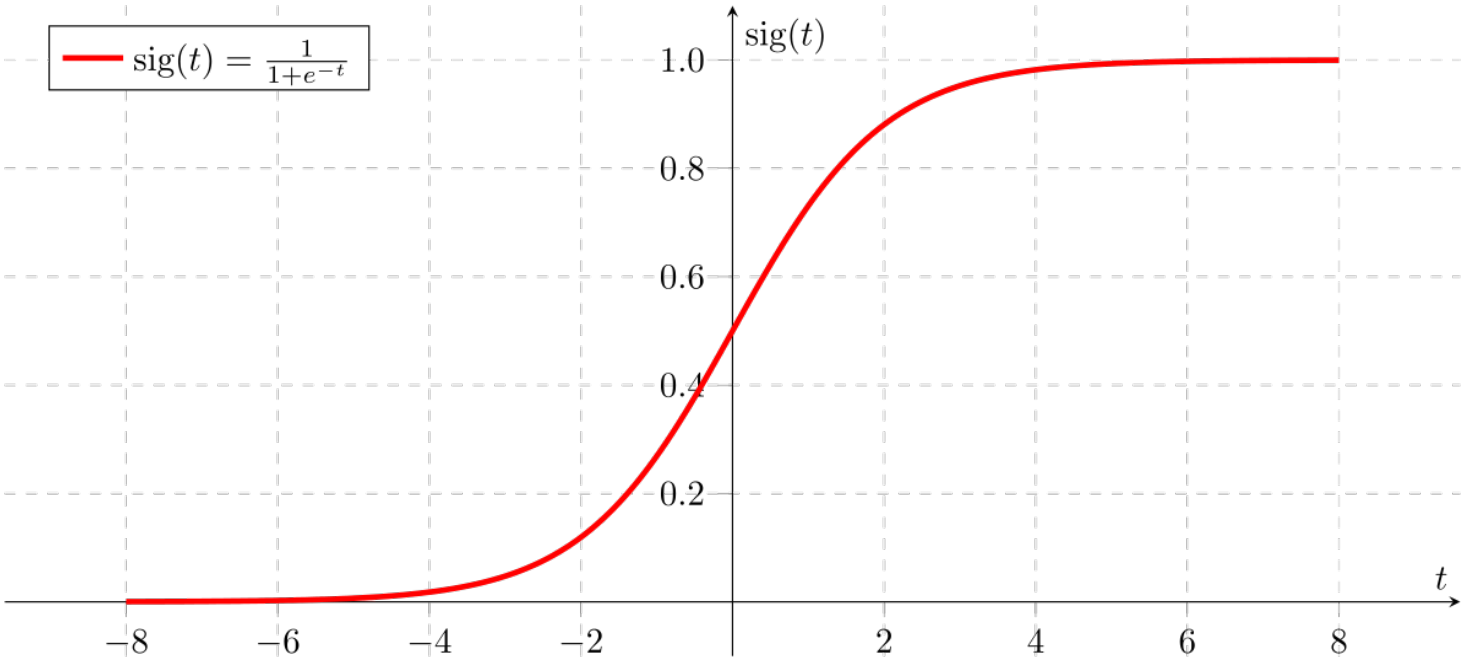
- F1 = 2 * P * R / (P + R)

In this project, I proposed to use accuracy as the primary metric. In case the data is highly imbalanced, F1 score will be used instead.

Benchmark model

A logistic regression will serve as the benchmark model, since it is possibly the most popular algorithm for binary classification problems in industry.

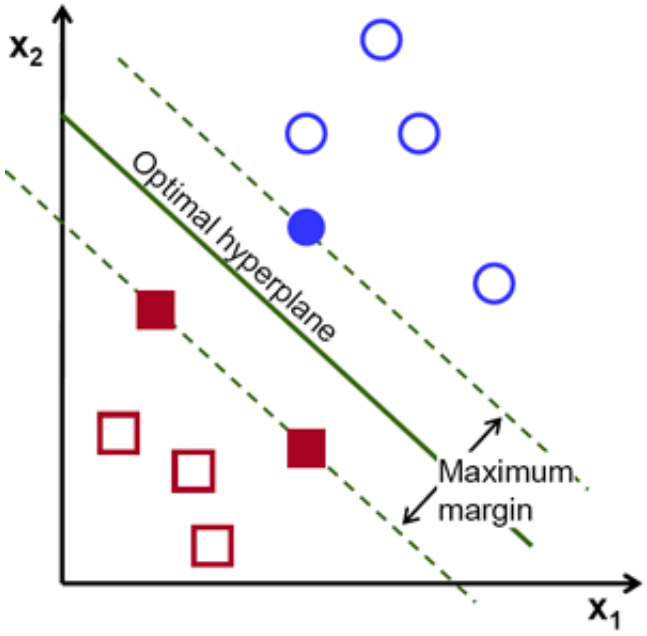
A logistic regression model estimates the probability of an instance belonging to the positive class using the logistic function, with the input being a linear model.



Algorithms beyond the benchmark

Support vector machine (SVM)

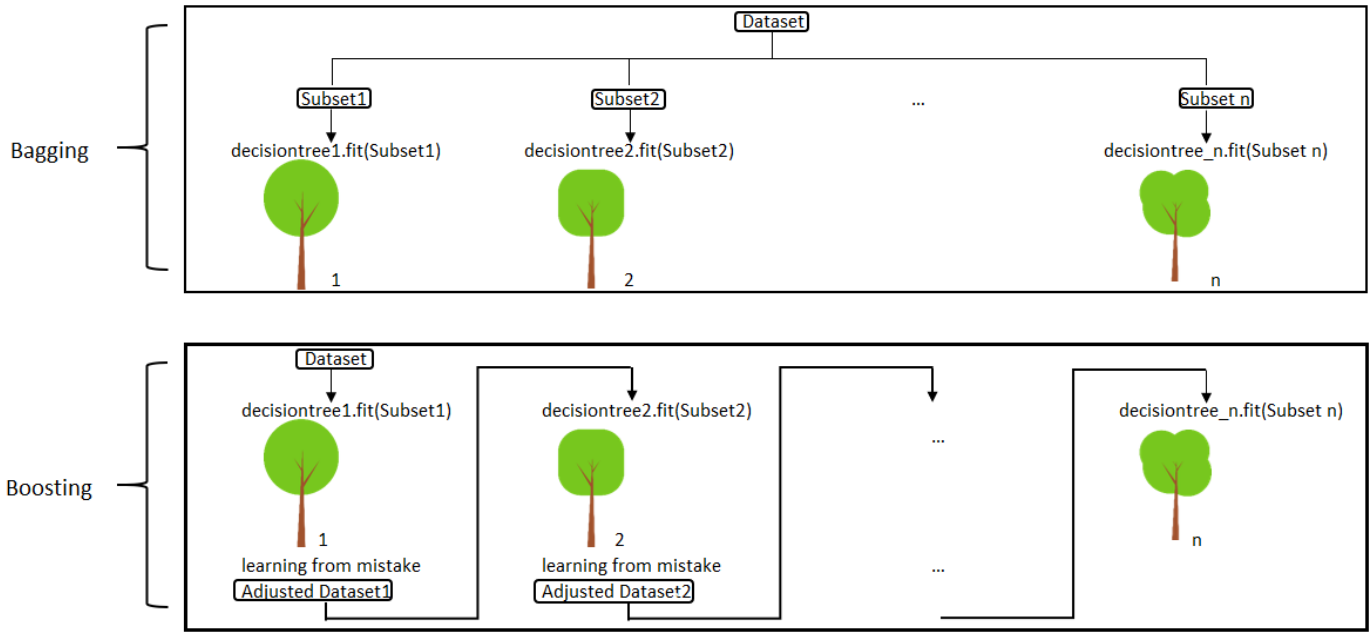
The objective of the support vector machine algorithm is to find a hyperplane in feature space that distinctly classifies the data points. For classification problems, the optimal hyperplane stays as far away from the closest training instances as possible to maximize the margin.



Decision tree and basic ensemble

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.



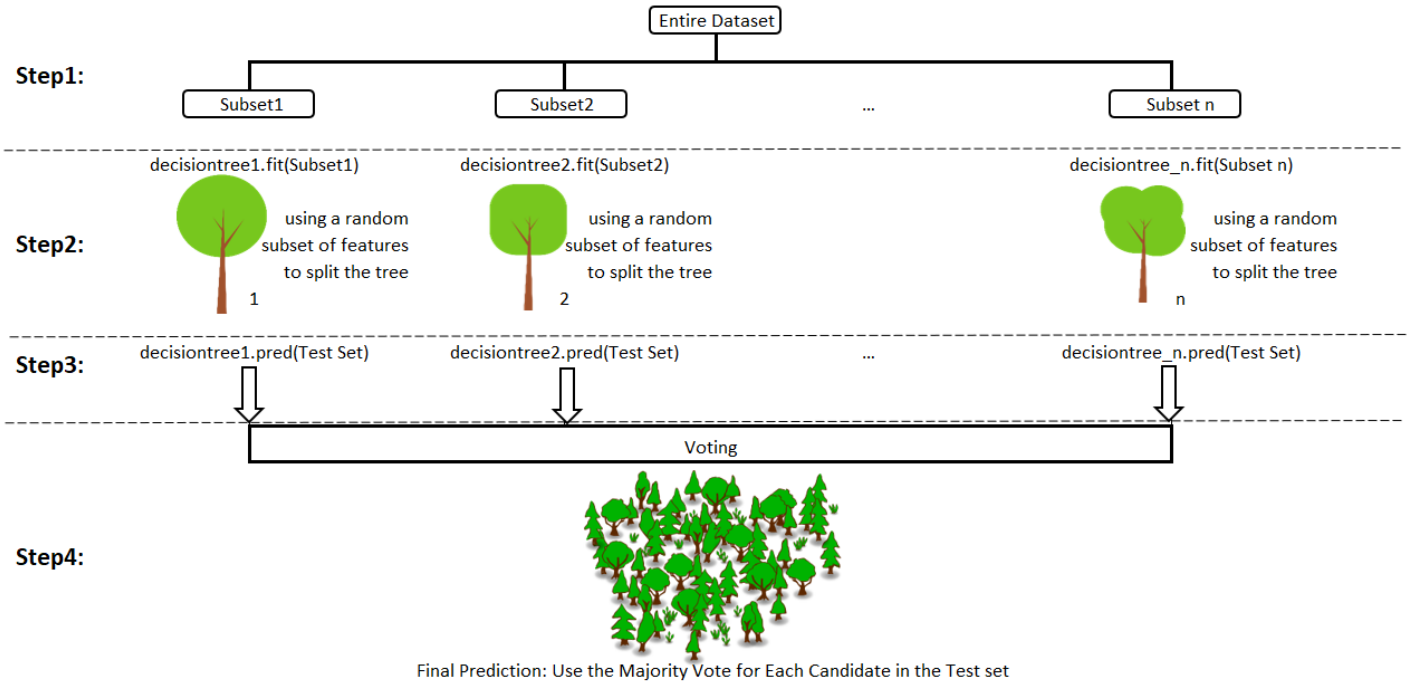
- Bagging: Training a bunch of individual models in a parallel way. Each model is trained by a random

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

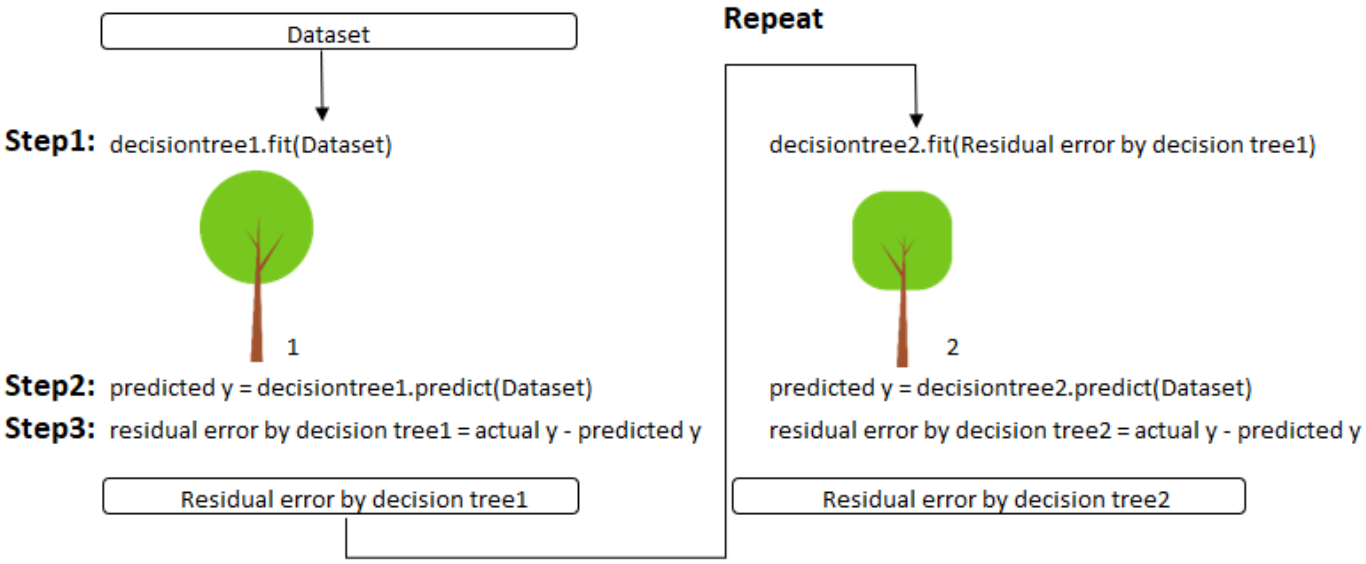
Random forest

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



Gradient boosting

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.



Data analysis

`portfolio`

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

`portfolio` contains 10 different offers, categorized into 3 types (BOGO, discount and informational).

`profile`

gender	age	id	became_member_on	income
	118	68be06ca386d4c31939f3a4f0e3dd783	20170212	nan
F	55	0610b486422d4921ae7d2bf64640c50b	20170715	112000
	118	38fe809add3b4fcf9315a9694bb96ff5	20180712	nan
F	75	78afa995795e4d85b5d9ceeca43f5fef	20170509	100000
	118	a03223e636434f42ac4c3df47e8bac43	20170804	nan

`profile` contains information of each customer. There are notable NaNs found for gender, age and income. Strategies to deal with these NaNs will be discussed in the next section.

`transcript`

Take a look at the transcript of a particular customer.

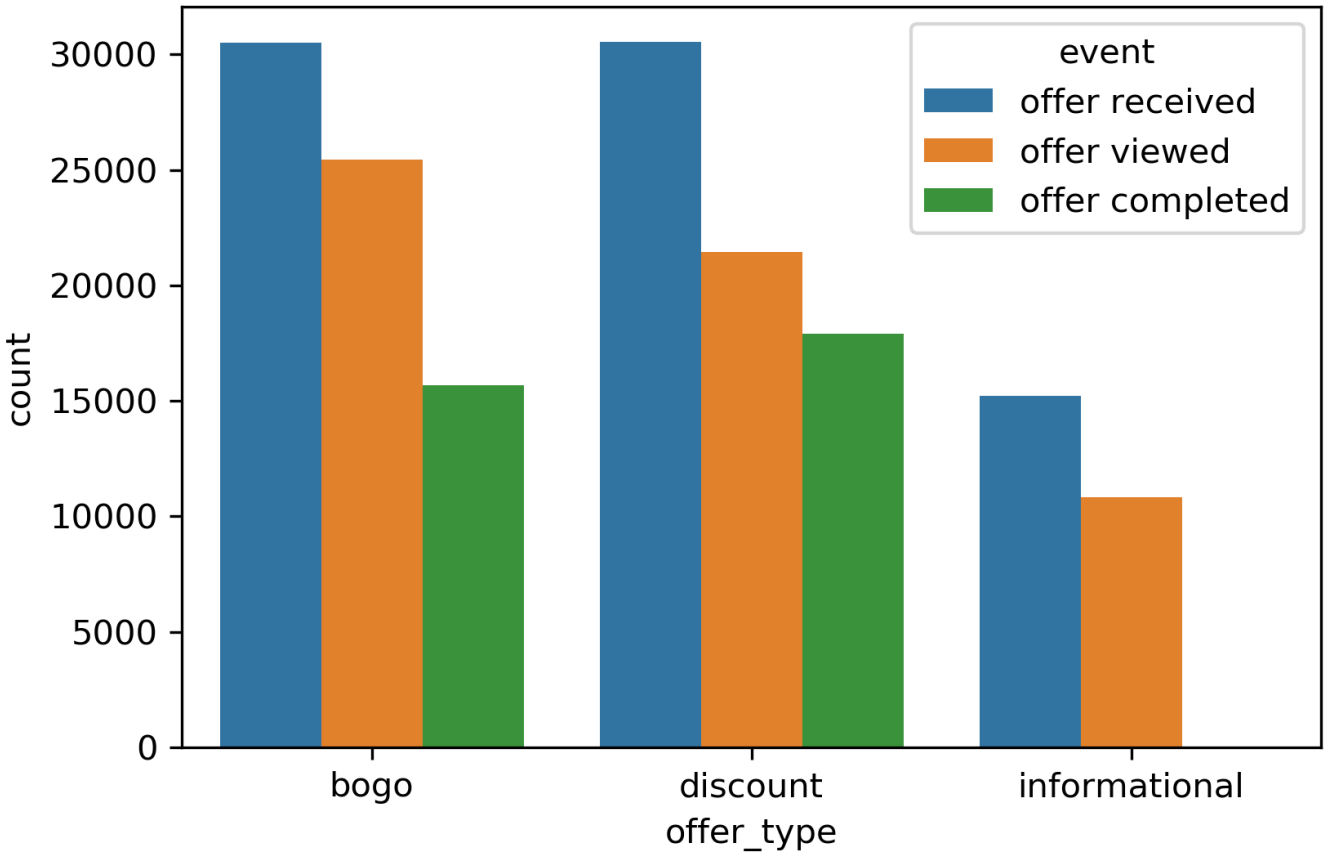
person	event	value	time
78afa995795e4d85b5d9ceeca43f5fef	offer received	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	0
78afa995795e4d85b5d9ceeca43f5fef	offer viewed	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	6
78afa995795e4d85b5d9ceeca43f5fef	transaction	{'amount': 19.89}	132
78afa995795e4d85b5d9ceeca43f5fef	offer completed	{'offer_id': '9b98b8c7a33c4b65b9aebfe6a799e6d9', 'reward': 5}	132
78afa995795e4d85b5d9ceeca43f5fef	transaction	{'amount': 17.78}	144
78afa995795e4d85b5d9ceeca43f5fef	offer received	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	168
78afa995795e4d85b5d9ceeca43f5fef	offer viewed	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	216
78afa995795e4d85b5d9ceeca43f5fef	transaction	{'amount': 19.67}	222
78afa995795e4d85b5d9ceeca43f5fef	transaction	{'amount': 29.72}	240
78afa995795e4d85b5d9ceeca43f5fef	transaction	{'amount': 23.93}	378
78afa995795e4d85b5d9ceeca43f5fef	offer received	{'offer id': 'ae264e3637204a6fb9bb56bc8210ddfd'}	408
78afa995795e4d85b5d9ceeca43f5fef	offer viewed	{'offer id': 'ae264e3637204a6fb9bb56bc8210ddfd'}	408
78afa995795e4d85b5d9ceeca43f5fef	offer received	{'offer id': 'f19421c1d4aa40978ebb69ca19b0e20d'}	504
78afa995795e4d85b5d9ceeca43f5fef	transaction	{'amount': 21.72}	510
78afa995795e4d85b5d9ceeca43f5fef	offer completed	{'offer_id': 'ae264e3637204a6fb9bb56bc8210ddfd', 'reward': 10}	510
78afa995795e4d85b5d9ceeca43f5fef	offer completed	{'offer_id': 'f19421c1d4aa40978ebb69ca19b0e20d', 'reward': 5}	510
78afa995795e4d85b5d9ceeca43f5fef	transaction	{'amount': 26.56}	534
78afa995795e4d85b5d9ceeca43f5fef	offer viewed	{'offer id': 'f19421c1d4aa40978ebb69ca19b0e20d'}	582

By looking at activities of one customer, one can easily find four types of events recorded in `transcript`, transactions and three types of offer activities.

- '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`.

```
>>> 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)
```

After filtering out transactions using the code above, I further analyzed offer activities with different offer types considered. From the count plot of events for different types of offers, one can easily observe that informational offers do not have completion events, possibly because there is no reward to claim. This is a key observation, as it will ultimately determine how `transcript` will be processed for different types of offers.



Here are the **rules** I propose 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.

All the offer-customer pair will be traced from all the 'offer received' events.

```
>>> 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. There are chances that multiple offers are effective in one customer's account since the offer durations are generally longer than the minimum interval. Processing `transcript` by person and unique offer id may reduce such complexity, but it is still tricky if the person has more than one

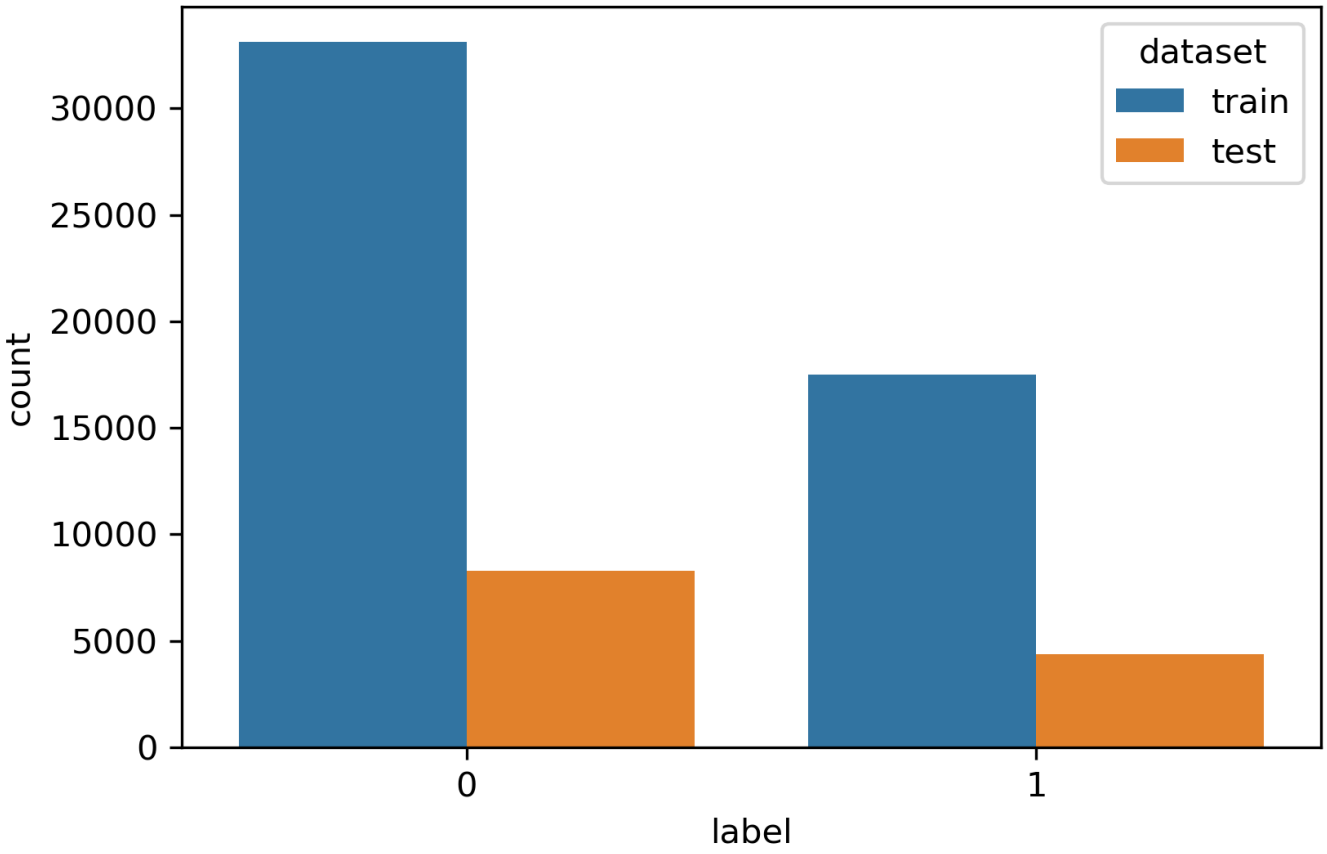
active offers with the same id.

```
>>> tmax = transcript.iloc[-1]['time']
>>> tmax / 24
29.75
```

In addition, 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
```

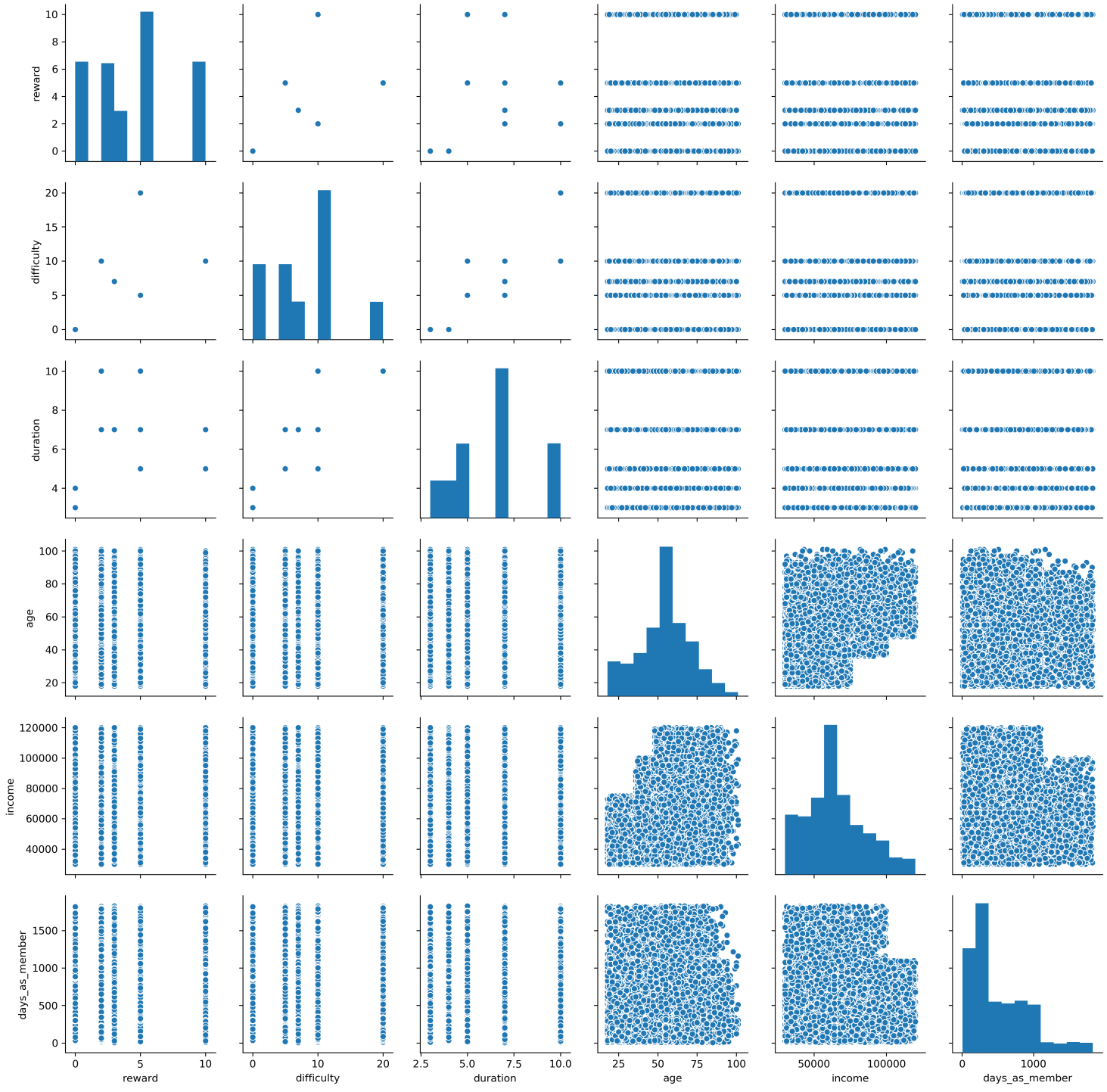
As a final remark for data processing, a unique customer-offer pair may have different responses since there are customers getting the same offer for more than once. Their responses may not necessarily remain the same. However, a model predicting customer-offer match should yield a unique output based on the input. Here I made a simple assumption that a customer likes an offer as long as a positive response is collected for more than 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(['reward', 'difficulty', 'duration', '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')
```

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).

Solution implementation

Data cleaning

1. Handle missing values in `profile`

Based on my observations, all the rows with NaNs in `profile` have gender and income as NaN, and age of 118. As already mentioned in the data source, 118 is the encoded missing value for age. Therefore, imputation is required for gender, age and income. The median of numerical fields (age and income) is used, and another label ('U' for unknown) is applied for gender. Finally, since the three fields are either all missing or full, only one column 'profile_nan' is added as an indication of missing values. `profile` after imputation is shown below.

gender	age	id	became_member_on	income	profile_nan
U	55	68be06ca386d4c31939f3a4f0e3dd783	20170212	64000	1
F	55	0610b486422d4921ae7d2bf64640c50b	20170715	112000	0
U	55	38fe809add3b4fcf9315a9694bb96ff5	20180712	64000	1
F	75	78afa995795e4d85b5d9ceeca43f5fef	20170509	100000	0
U	55	a03223e636434f42ac4c3df47e8bac43	20170804	64000	1

2. 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 dataframe 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)
```

person	offer	positive_response
0009655768c64bdeb2e877511632db8f	2906b810c7d4411798c6938adc9daaa5	False
0009655768c64bdeb2e877511632db8f	3f207df678b143eea3cee63160fa8bed	False
0009655768c64bdeb2e877511632db8f	5a8bc65990b245e5a138643cd4eb9837	False
0009655768c64bdeb2e877511632db8f	f19421c1d4aa40978ebb69ca19b0e20d	False
0009655768c64bdeb2e877511632db8f	fafdcd668e3743c1bb461111dcafc2a4	False

Pre-processing

3. Encode categorical variables and assemble dataset

Before merging `portfolio` and `profile` onto the skeleton, further process them to encode columns not suitable as inputs for a model, and drop the original columns. These include:

`portfolio`

- Expand channels into four columns representing different media where the offer is sent. Drop the email column since all offers have this option.
- Use one-hot encoding for 'offer_type'.

`profile`

- Convert 'became_member_on' (date) to membership days counted from a later date (2018-08-01).
- Use one-hot encoding for 'gender'.

Finally, join 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.

4. 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.

Modeling

5. Select optimal classification 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)

```
>>> 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()
0.7365592954908515
```

Support vector machine with linear kernel

```
>>> 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(
)
0.7346828966249379
```

Non-linear kernels are not considered since the algorithm scales poorly with number of instances.

Random forest

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> cross_val_score(RandomForestClassifier(n_estimators=100), train_X, train_y, scori
ng='accuracy', cv=5, n_jobs=-1).mean()
0.7380603720566018
```

Gradient boosting

```
>>> from sklearn.ensemble import GradientBoostingClassifier
>>> cross_val_score(GradientBoostingClassifier(), train_X, train_y, scoring='accuracy
', cv=5, n_jobs=-1).mean()
0.7594311595545019
```

```
>>> from xgboost import XGBClassifier
>>> cross_val_score(XGBClassifier(), train_X, train_y, scoring='accuracy', cv=5, n_jo
bs=-1).mean()
0.7580288289089329
```

```
>>> from lightgbm import LGBMClassifier
>>> cross_val_score(LGBMClassifier(), train_X, train_y, scoring='accuracy', cv=5, n_j
obs=-1).mean()
0.7637567343733975
```

Gradient boosting implemented in LightGBM is the algorithm for the optimal model. Here LightGBM is chosen over XGBoost not only because of better performance, but also the early stopping feature. As a strategy to significantly reduce training time, both XGBoost and LightGBM support early stopping. However, with early stopping enabled, LightGBM always returns the optimal ensemble while XGBoost only returns the final ensemble.

6. Tune hyperparameters

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:

1. 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`).
2. Tune other parameters related to tree growth (`min_data_in_leaf` and `min_sum_hessian_in_leaf`).
3. Tune "random forest" parameters (`bagging_fraction` and `feature_fraction`).
4. Tune regularization parameters (`lambda_l1` and `lambda_l2`).
5. 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.

```
>>> lgbm = LGBMClassifier(num_leaves=127, max_depth=14, n_estimators=98)
```

Results

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

Model	LogisticRegression	LightGBM
Accuracy	0.737	0.766

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 how the dataset splits, suggesting the solution is robust.

```
>>> 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, rando
m_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
[0.7622847211249802,
 0.7657607836941065,
 0.7595986727761099,
 0.7721598988781798,
 0.7664717964923369,
 0.7639437509875178,
 0.7649707694738506,
 0.7659977879601833]
```

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

- Starbucks - Wikipedia
- Performance Metrics for Classification problems in Machine Learning
- Logistic Regression - Detailed Overview
- Support Vector Machine - Introduction to Machine Learning Algorithms
- Decision tree - Wikipedia
- Decision Tree Ensembles - Bagging and Boosting
- LightGBM Python API
- LightGBM Parameters
- Complete Guide to Parameter Tuning in XGBoost with codes in Python