Machine Learning Engineer Nanodegree

Capstone Project

Marcus Winter

February 28, 2021

I. Definition

Project Overview

In this project, we will train and deploy a model that **predicts the completion probability of a customer offer**. For this purpose, we will use simulated data from the **Starbucks Reward mobile app** that contains information about customers, offers, and transactions. One possible use of this model would be a **recommendation system that displays to each customer the offer with the highest probability of being redeemed**. The business value of deploying a recommendation system like this can be huge for marketing-intensive companies like Starbucks. By **tailoring coupons to individual customers**, which generate the highest likelihood of purchase with the lowest incentive, the **expected return on all marketing activities can be maximized**.

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

Here is the schema and explanation of each variable in the files:

portfolio.json

- **id** (string) - offer id

- **offer_type** (string) - type of offer ie BOGO, discount, informational
- **difficulty** (int) - minimum required spend to complete an offer
- **reward** (int) - reward given for completing an offer
- **duration** (int) - time for offer to be open, in days
- **channels** (list of strings)

profile.json

- **age** (int) - age of the customer
- **became_member_on** (int) - date when customer created an app account
- **gender** (str) - gender of the customer (note some entries contain 'O' for other rather than M or F)
- **id** (str) - customer id
- **income** (float) - customer's income

transcript.json

- **event** (str) - record description (ie transaction, offer received, offer viewed, etc.)
- **person** (str) - customer id
- **time** (int) - time in hours since start of test. The data begins at time t=0
- **value** - (dict of strings) - either an offer id or transaction amount depending on the record

Problem Statement

To build the recommendation system we must follow **two consecutive steps**:

Step 1: Build A Predictive Model

We will train and **deploy a model that predicts the completion probability** for a given customer and offer combination. Given various **customer demographics** and **offer characteristics**, we want to predict the probability that a customer completes an offer in its intended validity. We will **compare three different classification models** (Logistic Regression, KNN, Random Forest) and select the best model for the final training and **deployment on Amazon SageMaker**. For the model comparison, we will follow a usual train test split evaluation, meaning that we train each model on the training set and make an **out-of-sample evaluation** on the test set.

Step 2: Build A Recommendation System Upon The Predictive Model

In the second step, we use the predictions from the deployed model for building a **recommendation system**. For all available offers, we predict the completion probability of a given customer and **choose the offer with the highest completion probability** for this customer. Therefore, we must write functions that can **transform a list of customers and offers** into the right format for requesting predictions from the provided endpoint. Based on these predictions the system should **choose the offer with the highest score** for each customer.

Benchmark Model

The benchmark model for the recommender system is **a model that randomly chooses an offer for each customer**. In order to be successful, the completion rate of our recommender system must be higher than that of the random model. The random model is the right choice for the benchmark model if we assume that the **company has not implemented any offer selection system so far**. Then any offer selection strategy that leads to **higher completion rates than a random selection** is **preferable** to the status quo.

Evaluation Metrics

For the **model selection**, we will compare classifiers on the basis of the **AUC (Area Under Curve)** of the ROC Curve. We want to achieve the **best possible discrimination between true positives and false positives** for our recommendation system. The better the classifier is able to discriminate, the better the decisions of the recommendation system will be. Therefore, the AUC metric is a suitable evaluation metric for model selection, because it represents exactly this quality of a classifier.

For the final **recommendations**, we will choose the **completion rate** as evaluation metric. We will simulate the completion rate on the test set if we choose one offer per customer, both with our own recommendation system and the random benchmark model. This is the evaluation metric that **determines whether it would be worthwhile to implement** the recommendation system. If we can achieve a higher completion rate on the test data with our optimized selection strategy than with a random choice, then an implementation might be advisable.

II. Analysis

Data Processing

The input data is contained in **three files**:

portfolio.json - containing offer ids and meta data about each offer (duration, type, etc.)

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

profile.json - demographic data for each customer

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

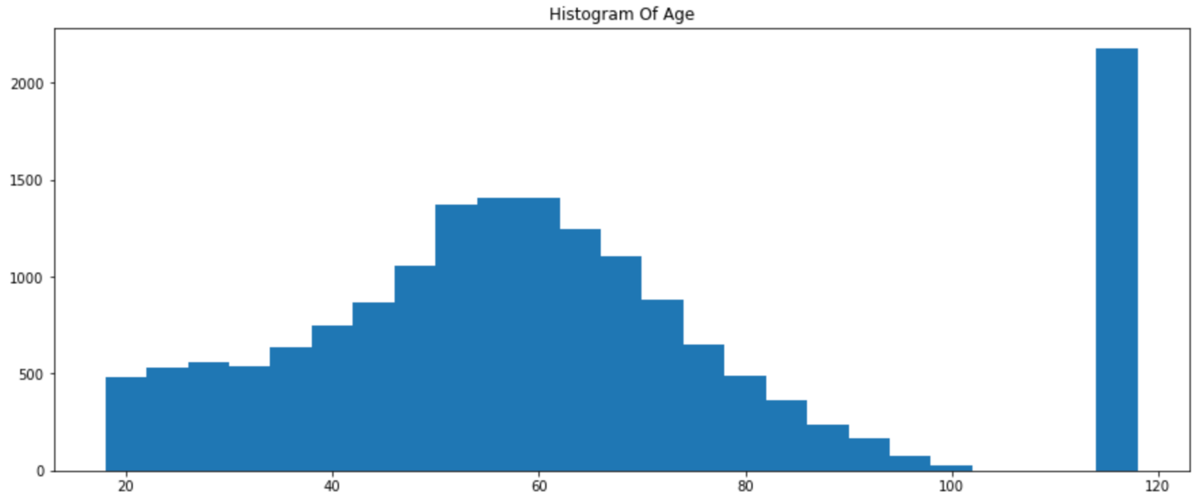
transcript.json - records for transactions, offers received, offers viewed, and offers completed

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

Our first task is to prepare the data from the three data sets for the model training. Since the transaction data is only available in raw form, a main challenge is to **create the target variable** for the classification model in the first place. To do this, we will look at each offer roll out to see if a redemption has occurred within the intended validity of the offer. We will then

prepare customer and offer characteristics as possible model features for each of these observations.

The effort for the preparation of the model features is rather low, because the data is already relatively clean. There are only minor anomalies, such as unavailable age values being stored with a value of 118:



Other minor challenges are to **encode the channel column** (from a single column in list format [web, mobile, social] to individual columns) to or to **calculate the days since registration** from the registration date of customers.

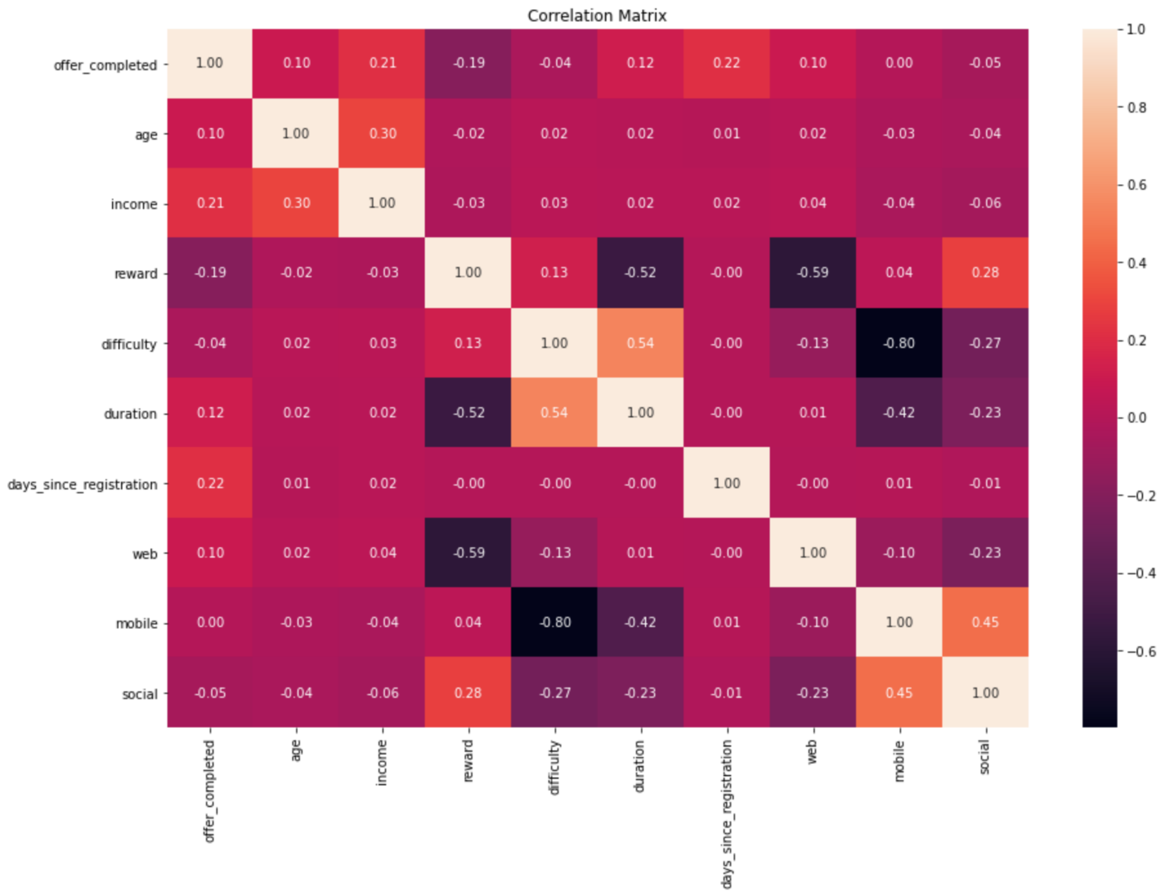
After this preprocessing we come up with a data set that consists of the target label and **13 features**:

offer_completed	age	income	reward	difficulty	duration	days_since_registration	web	email	mobile	social	gender_M	gender_O	offer_type_discount
1	33.0	72000.0	5	5	5	461	1	1	1	1	1	0	0
1	33.0	72000.0	2	10	10	461	1	1	1	1	1	0	1
0	NaN	NaN	5	5	5	92	1	1	1	1	0	0	0
1	40.0	57000.0	5	20	10	198	1	1	0	0	0	1	1
1	40.0	57000.0	3	7	7	198	1	1	1	1	0	1	1

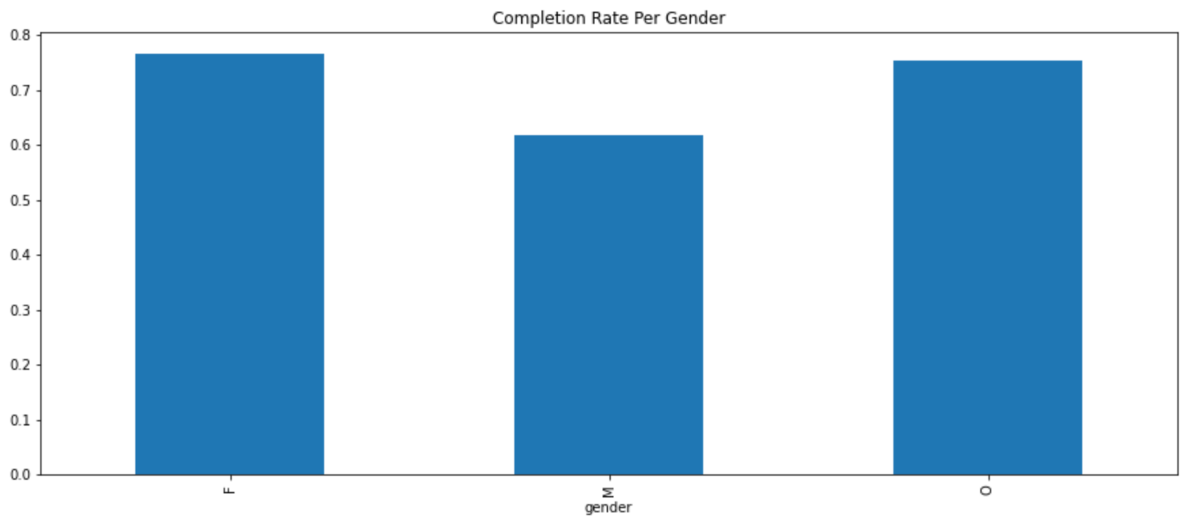
Each row represents one customer and offer combination and comprises both **customer demographics** (e.g. age, income or gender) and **offer characteristics** (e.g. reward, difficulty and duration) to predict the probability of completion.

Data Exploration

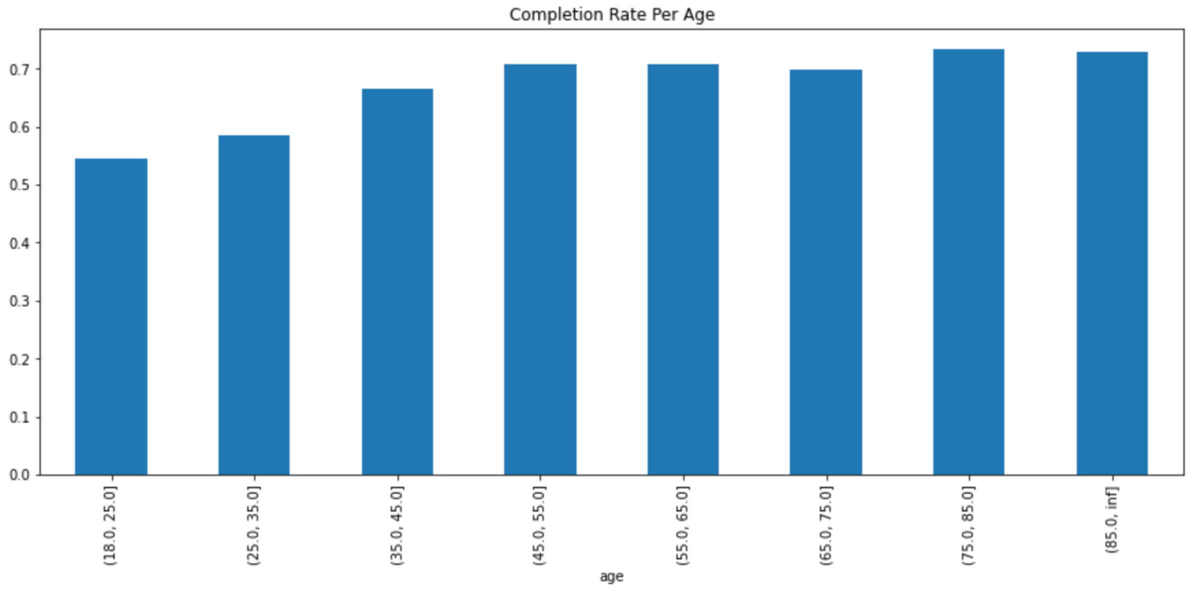
With this data set we can visualize the correlations between some of the features and its corresponding label:



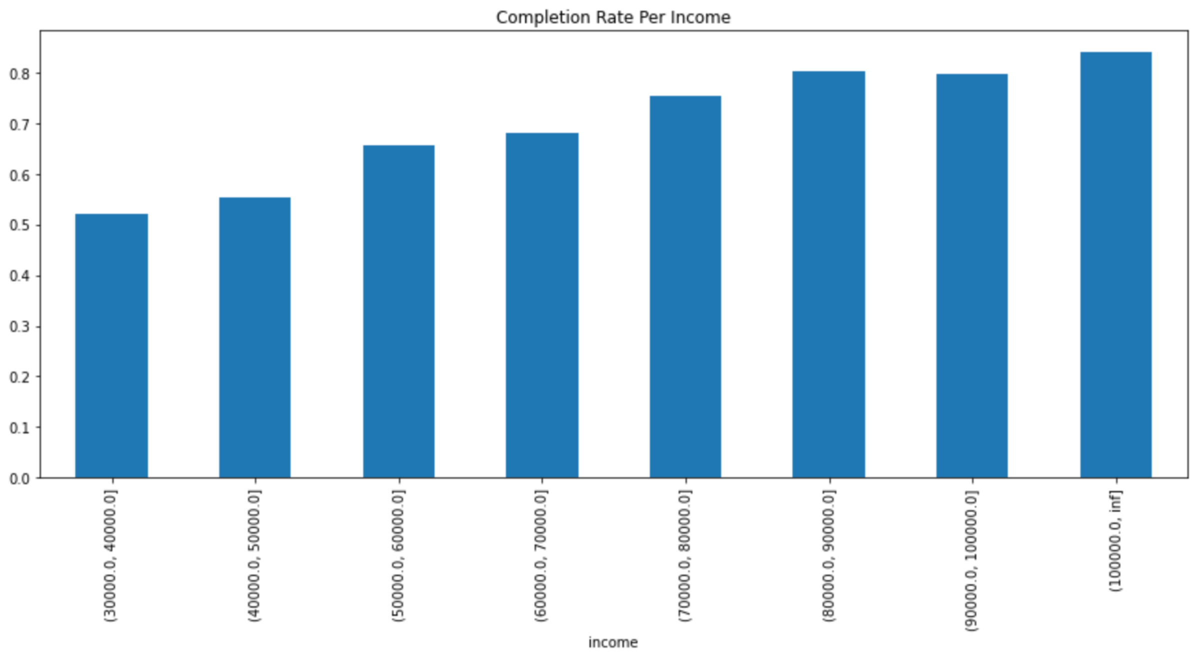
From the overall correlation matrix, we can infer that **`age`**, **`reward`** and **`days_since_registration`** show the **strongest correlation** with the completion probability.



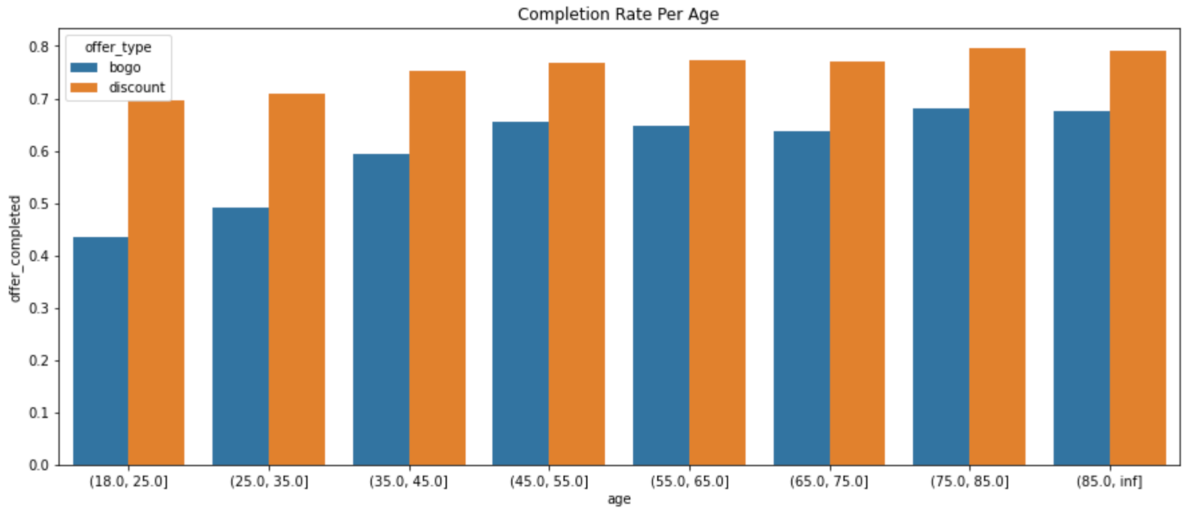
Men show lower completion rates than women and other.



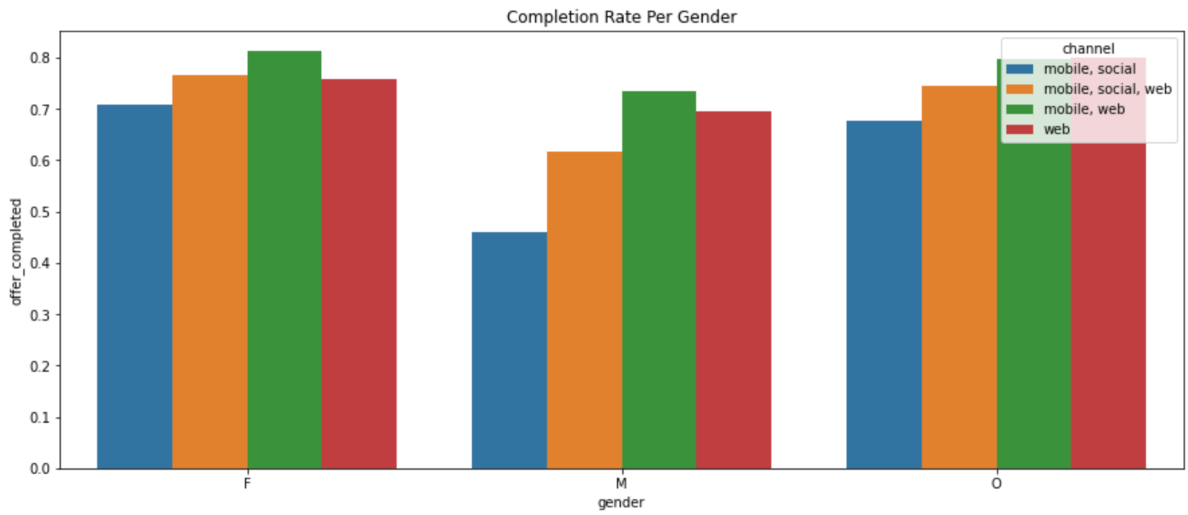
Users under 45 show lower completion rates than higher age groups.



The **completion rate increases with the income** of customers.



Discount offers show similar completion rates for different age groups, whereas **bogo offers are clearly less favorable to young people**.



Male customers are obviously **less responsive to offers that are not advertised on the web** channel.

Algorithms and Techniques

We will compare **three algorithms** for solving the classification problem:

- **Logistic Regression**: Logistic regression is based on the well-known linear regression model. Binary logistic regression analysis examines the relationship between the **probability that the dependent variable takes the value 1** and the independent variables. That is, it **does not predict the value of the dependent variable**, but the probability that the dependent variable takes the value 1.

- **KNN**: The k-nearest neighbor algorithm is a classification technique in which the **class assignment** for an observation is made **considering its k nearest neighbors**. In other words, an observation is assigned to the **majority class of the k samples that are most similar to it**.
- **Random Forest**: A random forest **combines the results of many different decision trees** to make the best possible decisions. Besides considering multiple decision trees (bagging), the random forest adds **further randomness** to the model **as the trees grow**. Instead of searching for the most important feature as a node is split, it searches for the **best feature among a random subset of features**.

I made a deliberate decision **not to use more complex model types such as neural networks** in this project because we are operating on **relatively small feature sets** with no more than 13 variables. From my experience, the above models are sufficiently capable of capturing the **comparatively simple relationships** between user demographics and offer characteristics. At the same time, the algorithms chosen are **different enough** to represent multiple **alternative classification approaches**.

Benchmark

For benchmarking the recommendation system, we **filter the test set to customers that got at least two different offers**. There are **2489 customers** in the test set that fulfil this criterium. We then choose one offer per customer (based on two different strategies) and calculate the corresponding completion rate.

The benchmark strategy is to **choose a random offer per customer**. This strategy results in **completion rate of 60.1%** for the given group of customers:

```
# Completion rate of random model
(
    test_more_than_1_offer.reset_index()
    .groupby('person')
    .apply(lambda df: df.sample(1))
    .label.mean()
)
```

```
0.6010445962233829
```

This is the number by which our classification model will be judged. In order to be successful, the **completion rate** of our recommender **system must be higher than that of the random model**.

III. Methodology

For the following sections, we will start with the dataset that we already cleaned up in the previous chapter:

offer_completed	age	income	reward	difficulty	duration	days_since_registration	web	email	mobile	social	gender_M	gender_O	offer_type_discount
1	33.0	72000.0	5	5	5	461	1	1	1	1	1	0	0
1	33.0	72000.0	2	10	10	461	1	1	1	1	1	0	1
0	NaN	NaN	5	5	5	92	1	1	1	1	0	0	0
1	40.0	57000.0	5	20	10	198	1	1	0	0	0	1	1
1	40.0	57000.0	3	7	7	198	1	1	1	1	0	1	1

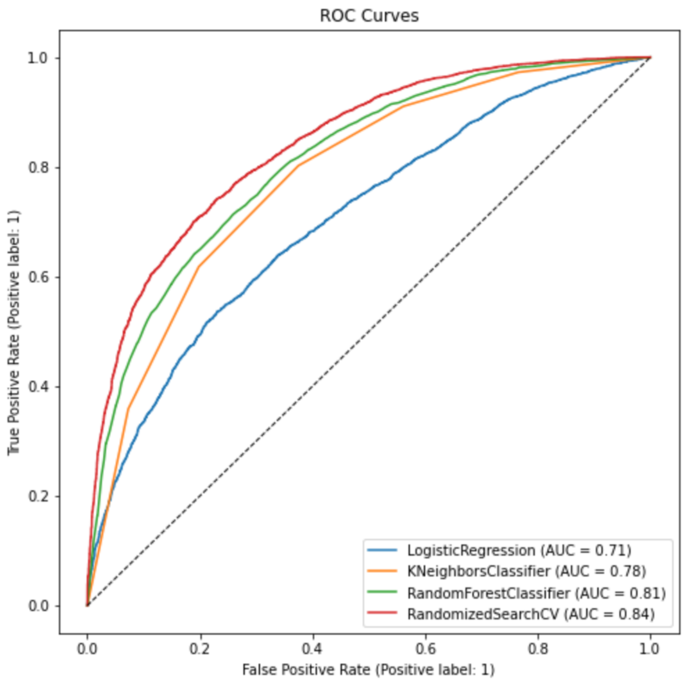
Each row represents one customer and offer combination and comprises both **customer demographics** (e.g. age, income or gender) and **offer characteristics** (e.g. reward, difficulty and duration) to predict the probability of completion.

To build the recommendation system we proceed in two steps:

Step 1: Build A Predictive Model

Here we train and **deploy a model that predicts the completion probability** for a given customer and offer combination. We do a rough **pre-screening of the three classification models** (Logistic Regression, KNN and Random Forest) and select the best model for final training and **deployment on Amazon SageMaker**.

For the model selection we compare the three classifiers on the basis of the **AUC (Area Under Curve)** of the ROC Curve. Here the **Random Forest** (with some slightly optimized hyperparameters) shows the **highest AUC (0.84)** by far:



With that we deploy the model pipeline on Amazon SageMaker:

We **split** the preprocessed data set into **training and test set**:

```
# Make train test split
train, test = train_test_split(data, test_size=0.3, random_state=0)
print("train shape: ", train.shape)
print("test shape: ", test.shape)
```

```
train shape:  (27878, 14)
test shape:  (11948, 14)
```

We **upload both data sets to S3**:

```
# Upload data to S3
data_dir = 'data'
bucket = sagemaker_session.default_bucket()
prefix = 'sagemaker/starbucks_rewards'

train.to_csv(os.path.join(data_dir, 'train.csv'), header=False, index=False)
test.to_csv(os.path.join(data_dir, 'test.csv'), header=False, index=False)

train_location = sagemaker_session.upload_data(os.path.join(data_dir, 'train.csv'), key_prefix=prefix)
test_location = sagemaker_session.upload_data(os.path.join(data_dir, 'test.csv'), key_prefix=prefix)
```

We instantiate and **fit a custom scikit-learn estimator**:

```
# Set directory to save model artifacts
s3_output_path = "s3://{}/{}/output".format(bucket, prefix)

# Instantiate the sklearn estimator
estimator = SKLearn(
    sagemaker_session=sagemaker_session,
    role=role,
    entry_point='train.py',
    source_dir='src',
    py_version='py3',
    framework_version='0.23-1',
    instance_count=1,
    instance_type='ml.c4.xlarge',
    output_path=s3_output_path
)
```

```
%%time

# Train estimator on S3 training data
estimator.fit({'train': train_location})
```

The `train.py` file does not only contain the training script, but also **a custom `predict_fn`** function in order **to retrieve prediction probabilities instead of binary labels**:

```
def predict_fn(input_data, model):
    """Predict probabilities of belonging to the positive class"""

    classes = model.classes_
    pred_prob = model.predict_proba(input_data)

    return pred_prob[:,np.argwhere(classes==1)].squeeze()
```

This is a requirement for recommending the best offer per customer in the next step.

Step 2: Build A Recommendation System Upon The Predictive Model

Next we illustrate the use case of a recommendation system for offers. For all available offers, we want to predict the completion probability of a given customer and **choose the offer with the highest completion probability** for this customer.

First we **deploy the estimator** from the previous step:

```
%%time

# Deploy model and assign to variable for making predictions
predictor = estimator.deploy(
    initial_instance_count=1,
    instance_type='ml.t2.medium'
)
```

We can use the deployed endpoint for **making predictions on new data**:

```python
pred_proba = predictor.predict(X_test)
pred_proba
```

```
array([0.42470315, 0.64550168, 0.02014354, ..., 0.062982  , 0.23421987,
       0.96312812])
```

The responsible product team can now use these predictions to build the recommendation system for offers.

Given the necessary **customer data**..

```python
customers = [
    {
        'id': '0009655768c64bdeb2e877511632db8f',
        'age': 33,
        'gender': 'M',
        'income': 72000,
        'days_since_registration': 461},
    {
        'id': '00116118485d4dfda04fdbaba9a87b5c',
        'age': np.nan,
        'gender': np.nan,
        'income': np.nan,
        'days_since_registration': 92},
    {
        'id': '0011e0d4e6b944f998e987f904e8c1e5',
        'age': 40,
        'gender': 'O',
        'income': 57000,
        'days_since_registration': 198}
]
```

.. and **a list of potential offers**..

```python
offers = [
    {
        'id': 'ae264e3637204a6fb9bb56bc8210ddfd',
        'type': 'bogo',
        'web': 1,
        'email': 1,
        'social': 1,
        'mobile': 0,
        'reward': 10,
        'difficulty': 10,
        'duration': 7},
    {
        'id': 'f19421c1d4aa40978ebb69ca19b0e20d',
        'type': 'bogo',
        'web': 1,
        'email': 1,
        'social': 1,
        'mobile': 1,
        'reward': 5,
        'difficulty': 5,
        'duration': 5},
    {
        'id': '0b1e1539f2cc45b7b9fa7c272da2e1d7',
        'type': 'discount',
        'web': 1,
        'email': 1,
        'social': 0,
        'mobile': 0,
        'reward': 5,
        'difficulty': 20,
        'duration': 10}
]
```

.. we can **predict the completion rate for each offer**..

```
probas = get_success_probabilities(customers, offers)
pprint.pprint(probas)
```

```
{'0009655768c64bdeb2e877511632db8f': {'0b1e1539f2cc45b7b9fa7c272da2e1d7': 0.9426955139133467,
                                      'ae264e3637204a6fb9bb56bc8210ddfd': 0.5648851131888629,
                                      'f19421c1d4aa40978ebb69ca19b0e20d': 0.9340771182491163},
 '00116118485d4dfda04fdbaba9a87b5c': {'0b1e1539f2cc45b7b9fa7c272da2e1d7': 0.053495241758556036,
                                      'ae264e3637204a6fb9bb56bc8210ddfd': 0.02977202504887236,
                                      'f19421c1d4aa40978ebb69ca19b0e20d': 0.10867103658455653},
 '0011e0d4e6b944f998e987f904e8c1e5': {'0b1e1539f2cc45b7b9fa7c272da2e1d7': 0.68744225190439,
                                      'ae264e3637204a6fb9bb56bc8210ddfd': 0.5578886429082346,
                                      'f19421c1d4aa40978ebb69ca19b0e20d': 0.6036967331975791}}
```

.. and therefore **choose the best offer per customer**:

```
choices = get_best_offer_for_customer(probas)
pprint.pprint(choices)
```

```
{'0009655768c64bdeb2e877511632db8f': '0b1e1539f2cc45b7b9fa7c272da2e1d7',
 '00116118485d4dfda04fdbaba9a87b5c': 'f19421c1d4aa40978ebb69ca19b0e20d',
 '0011e0d4e6b944f998e987f904e8c1e5': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}
```

IV. Results

The benchmark model for the recommender system is **a model that randomly chooses an offer for each customer**. In order to be successful, the completion rate of our recommender system must be higher than that of the random model.

For the comparison we **filter the test set to customers that got at least two different offers**. There are **2489 customers** in the test set that fulfil this criterium. We then choose one offer per customer (based on two different strategies) and calculate the corresponding completion rate.

The first strategy (benchmark) is to **choose a random offer per customer**. This strategy results in **completion rate of 60.1%** for the given group of customers.

```
# Completion rate of random model
(
    test_more_than_1_offer.reset_index()
    .groupby('person')
    .apply(lambda df: df.sample(1))
    .label.mean()
)
```

```
0.6010445962233829
```

The second strategy (recommendation system) is to **choose the offer with the highest prediction** from our classification model. This strategy results in a **completion rate of 66.5%**. This is an **uplift of 6.4%** compared to the benchmark model.

```python
# Completion rate of recommendation system
(
    test_more_than_1_offer.reset_index()
    .groupby('person')
    .apply(lambda df: df.sort_values('prediction', ascending=False).head(1))
    .label.mean()
)
```

```
0.6645239051828044
```

Since this is an evaluation on an independent data set, both metrics **represent the expected performance value**. Because the uplift of 6.4% is relatively large, too, we can conclude altogether that **our recommendation system leads to higher completion rates than the benchmark model**.

V. Conclusion

In this project, we trained and deployed a p**rediction model for the completion probability of customer offers**. We showed that a **recommendation system** that chooses the best offer per customer **can increase the completion rate** over all customers **by up to 6.4%**. I think that this is a promising result, considering that the original dataset contains **relatively few offers** and only **a fraction of the available customer information**. For example, it might be worthwhile to derive further features from customers' transaction data, e.g. about their historical purchasing behavior or reactions to offers).