Wilson Chan

Starbucks Capstone

Udacity Machine Learning Engineer

PROBLEM STATEMENT
Does the reward size of the coupon influence the decision of whether the customer uses it or not?
If so, how big of a reward does an offer need to be to entice the customer to use the offer to make a transaction?

DATASETS AND INPUTS
The datasets used in for this project can be retrieved from Udacity's Machine Learning Engineer Nanodegree Capstone Project.

portfolio.json: Provides various offers along with their details

- 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: Holds data from people who have either received Starbucks offers or have made a transaction with Starbucks in the past

- 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: Contains information about actions made on transactions and offers

- 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

Development Stage 1: Load, Explore and Visualize Data

First I loaded the dataframe in order to get a feel for them. *Portfolio* was quite small as it contained purely the 10 unique offers. On the other hand, *Profile* and *Transcript* had a much larger set of data. This was where the bulk of the data wrangling was going to be happening.

Since my problem statement was related to offer completion, I instantly removed all transcript data from *Transcript*.

```
portfolio
```

	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.head()
```

	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.head()
```

	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

Development Stage 2: Clean and Pre-process Data

This is where the forming of the complete dataframe took place, along with the dropping of columns I intuitively believed to be unnecessary.

Combine all 3 *.json* files into a single DataFrame and drop useless columns

```
df = transcript_offers.merge(profile, how='inner', left_on='person', right_on='id').drop(['person', 'id'], axis=1)
df = df.merge(portfolio, how='inner', left_on='value', right_on='id').drop(['value', 'id'], axis=1)
df = df.drop('became_member_on', axis=1)
df.head()
```

	event	time	gender	age	income	reward	channels	difficulty	duration	offer_type
0	offer received	0	F	75	100000.0	5	[web, email, mobile]	5	7	bogo
1	offer viewed	6	F	75	100000.0	5	[web, email, mobile]	5	7	bogo
2	offer completed	132	F	75	100000.0	5	[web, email, mobile]	5	7	bogo
3	offer received	408	M	68	70000.0	5	[web, email, mobile]	5	7	bogo
4	offer viewed	420	M	68	70000.0	5	[web, email, mobile]	5	7	bogo

Pre-processing occurred here, where I filled in incomplete features, dropped categorical features, normalized numerical features, and mapped labels to 0 and 1 as this is a binary classification problem.

```
# We can see that gender and income features both have nan values
df.isnull().sum()
```

```
event             0
time              0
gender        18776
age               0
income        18776
reward            0
channels          0
difficulty        0
duration          0
offer_type        0
dtype: int64
```

```
df.drop(['gender', 'channels', 'offer_type'], axis=1, inplace=True)
df.head()
```

	event	time	age	income	reward	difficulty	duration
0	offer received	0	75	100000.0	5	5	7
1	offer viewed	6	75	100000.0	5	5	7
2	offer completed	132	75	100000.0	5	5	7
3	offer received	408	68	70000.0	5	5	7
4	offer viewed	420	68	70000.0	5	5	7

```
from sklearn.preprocessing import MinMaxScaler

numerical_features = ['time', 'age', 'income', 'difficulty', 'duration', 'reward']

scaler = MinMaxScaler((0, 1))
df_pca = pd.DataFrame(scaler.fit_transform(df[numerical_features].astype(float)))
df_pca.index = df.index
df_pca.columns = numerical_features
df_pca.head()
```

	time	age	income	difficulty	duration	reward
0	0.000000	0.57	0.777778	0.25	0.571429	0.5
1	0.008403	0.57	0.777778	0.25	0.571429	0.5
2	0.184874	0.57	0.777778	0.25	0.571429	0.5
3	0.571429	0.50	0.444444	0.25	0.571429	0.5
4	0.588235	0.50	0.444444	0.25	0.571429	0.5

	event	time	age	income	reward	difficulty	duration
0	0	0.000000	0.57	0.777778	0.5	0.25	0.571429
1	0	0.008403	0.57	0.777778	0.5	0.25	0.571429
2	1	0.184874	0.57	0.777778	0.5	0.25	0.571429
3	0	0.571429	0.50	0.444444	0.5	0.25	0.571429
4	0	0.588235	0.50	0.444444	0.5	0.25	0.571429

Development Stage 3: Visualize Data

To get a feel of what the data was like, I calculated a few percentages including:

- Offers completed: $44.02\,\%$
- Transactions that involved offers: $31.87\,\%$

Development Stage 4: Split Data into Train/Test Datasets

Having pre-processed the data, I then split the data into training and testing datasets, in order to prepare it for the machine learning algorithm.

I allocated 30% of the data to be for testing, and ended up with 117306 training samples, and 50275 testing samples.

```
from sklearn.model_selection import train_test_split

# 70% of the data will be training, and 30% of the data will be testing
X_train, X_test, y_train, y_test = train_test_split(df.drop('event', axis=1),
                                                    df['event'],
                                                    test_size=0.3,
                                                    random_state=0,
                                                    shuffle=True)

print("Training set has {} samples.".format(X_train.shape[0]))
print("Testing set has {} samples.".format(X_test.shape[0]))
```

```
Training set has 117306 samples.
Testing set has 50275 samples.
```

Development Stage 5: Reduce Dimensionality using PCA and find Vectors of Maximal Variance

This is where I reduced the number of features to only the features that were most relevant.

I defined various helper functions to use to perform PCA: *pca_results*, *plot_component*, and *scree_plot*.

Initially, I applied PCA without restricting the number of principal components to calculate so that I could get a view of how much variance each component actually accounted for.

This gave me 7 principal components, the same number of dimensions as my original data, along with what fraction of which features each of the components were made out of.

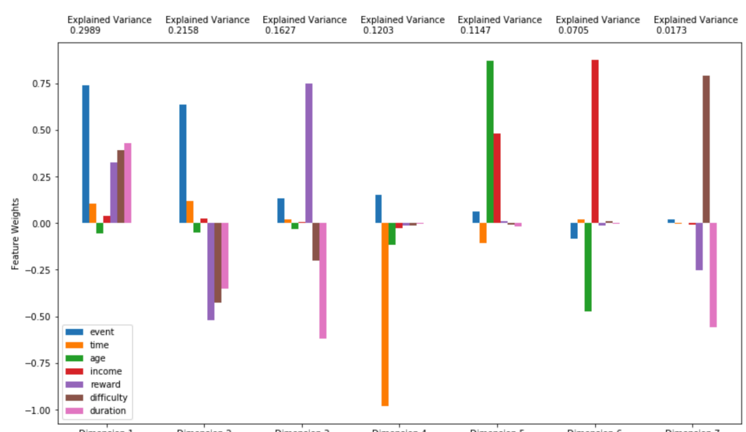
We can see that *reward* and *event* are both strongly present in the first 2-3 principal components. This verifies our hypothesis that reward size does dictate whether or not someone will complete an offer.

Note: The reverse, completing an offer indicates a fairly big reward size, is also plausible.

	0	1	2	3	4	5	6
0	-0.204490	-0.172259	-0.000782	0.431856	0.334149	0.268811	-0.164446
1	-0.203608	-0.171270	-0.000605	0.423612	0.333240	0.268989	-0.164469
2	0.552522	0.486117	0.134047	0.399784	0.377238	0.189607	-0.146714
3	-0.154326	-0.109677	0.011093	-0.111654	0.052010	0.021914	-0.162543
4	-0.152561	-0.107698	0.011446	-0.128143	0.050190	0.022269	-0.162589
...	...	...	...	...	...	...	...
167576	-0.139737	-0.007683	-0.146793	-0.327206	-0.029887	0.262289	-0.035757
167577	0.597868	0.628931	-0.015848	-0.177903	0.033217	0.179177	-0.017518
167578	-0.129865	0.004334	-0.137410	-0.284534	-0.392976	0.258041	-0.034258
167579	-0.126336	0.008290	-0.136704	-0.317511	-0.396615	0.258752	-0.034350
167580	0.622737	0.657764	-0.003465	-0.275384	-0.345339	0.177949	-0.016411

167581 rows × 7 columns

	Explained Variance	event	time	age	income	reward	difficulty	duration
Dimension 1	0.2989	0.7376	0.1050	-0.0535	0.0407	0.3241	0.3893	0.4287
Dimension 2	0.2158	0.6366	0.1177	-0.0513	0.0248	-0.5219	-0.4271	-0.3505
Dimension 3	0.1627	0.1309	0.0210	-0.0334	0.0074	0.7474	-0.2002	-0.6186
Dimension 4	0.1203	0.1493	-0.9811	-0.1187	-0.0264	-0.0125	-0.0151	-0.0058
Dimension 5	0.1147	0.0631	-0.1083	0.8689	0.4783	0.0085	-0.0073	-0.0189
Dimension 6	0.0705	-0.0831	0.0211	-0.4736	0.8764	-0.0147	0.0089	-0.0014
Dimension 7	0.0173	0.0182	-0.0027	0.0012	-0.0107	-0.2522	0.7909	-0.5571



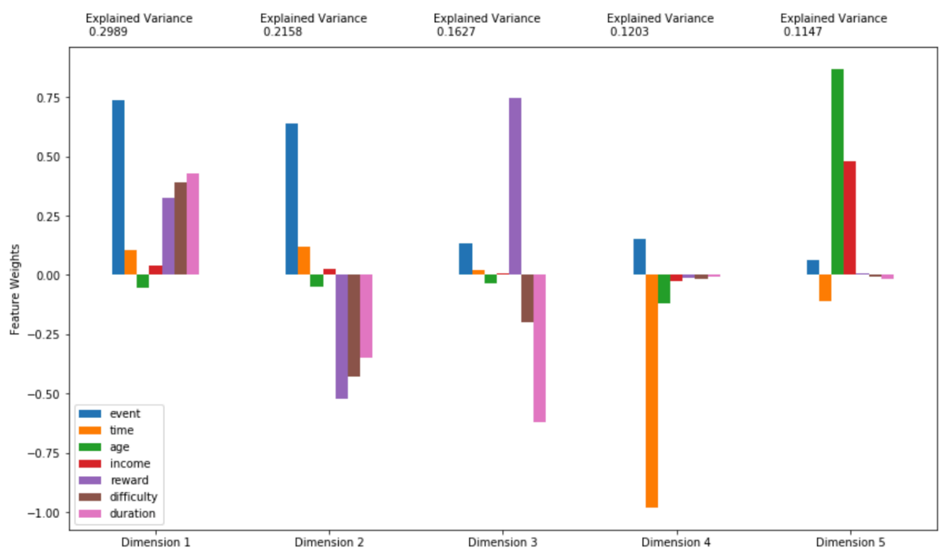
Wilson Chan

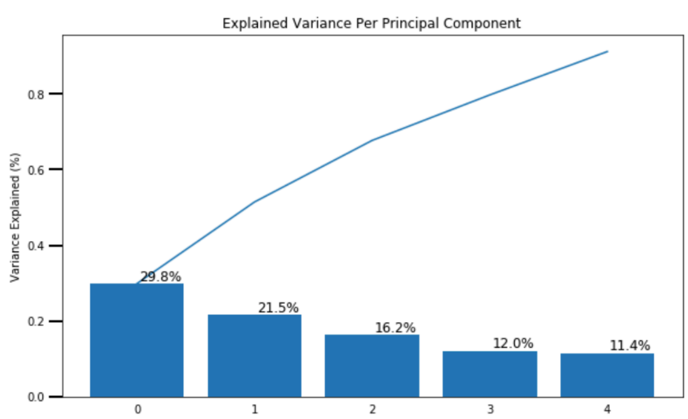
Visually, I could see that the top 5 principal components accounted for the majority of the variance. So I re-applied PCA while only deciding to retain the top 5 components. This left me with a cumulative variance of ~91%.

	0	1	2	3	4
0	-0.204490	-0.172259	-0.000782	0.431856	0.334149
1	-0.203608	-0.171270	-0.000605	0.423612	0.333240
2	0.552522	0.486117	0.134047	0.399784	0.377238
3	-0.154326	-0.109677	0.011093	-0.111654	0.052010
4	-0.152561	-0.107698	0.011446	-0.128143	0.050190
...	...	...	...	...	...
167576	-0.139737	-0.007683	-0.146793	-0.327206	-0.029887
167577	0.597868	0.628931	-0.015848	-0.177903	0.033217
167578	-0.129865	0.004334	-0.137410	-0.284534	-0.392976
167579	-0.126336	0.008290	-0.136704	-0.317511	-0.396615
167580	0.622737	0.657764	-0.003465	-0.275384	-0.345339

167581 rows × 5 columns

	Explained Variance	event	time	age	income	reward	difficulty	duration
Dimension 1	0.2989	0.7376	0.1050	-0.0535	0.0407	0.3241	0.3893	0.4287
Dimension 2	0.2158	0.6366	0.1177	-0.0513	0.0248	-0.5219	-0.4271	-0.3505
Dimension 3	0.1627	0.1309	0.0210	-0.0334	0.0074	0.7474	-0.2002	-0.6186
Dimension 4	0.1203	0.1493	-0.9811	-0.1187	-0.0264	-0.0125	-0.0151	-0.0058
Dimension 5	0.1147	0.0631	-0.1083	0.8689	0.4783	0.0085	-0.0073	-0.0189





I interpreted the top 3 components to get a feel for what they were composed of.

```python
# Check first principal component
principal_component_feature(0, components_df)
```

	age	income	time	reward	difficulty	duration	event
0	-0.053457	0.040687	0.104973	0.324137	0.389299	0.428705	0.737605

```python
# Check second principal component
principal_component_feature(1, components_df)
```

	reward	difficulty	duration	age	income	time	event
1	-0.521887	-0.427105	-0.350467	-0.051346	0.024828	0.117713	0.636615

```python
# Check third principal component
principal_component_feature(2, components_df)
```

	duration	difficulty	age	income	time	event	reward
2	-0.618552	-0.200228	-0.033435	0.007405	0.021005	0.130945	0.747356

Principal Component 1

- Most negative feature: *age*

- Most positive feature: *event*

It appears that someone who's younger is less likely to complete an offer.

By the same token, someone who's older is more likely to complete an offer.

Principal Component 2

- Most negative feature: *reward*

- Most positive feature: *event*

This is the component that correlates with our hypothesis.

It seems that the lower the reward, the less likely someone will complete an offer.

Also, the greater the reward, the more likely someone will complete an offer.

Principal Component 3

- Most negative feature: *duration*

- Most positive feature: *reward*

This component is also very interesting.

It tells us that the shorter an offer is available for use, the smaller the reward for that offer actually is.

Similarly, the longer an offer is available for use, the larger the reward for that offer actually is.

Findings

The last two components are the most interesting. They suggest that the longer rewards may also have a significant impact on whether the consumer's makes a transaction or not.

The first component gives us insight on some details on the customers.

From component 3, we see that offers that last longer are correlated with a greater reward.

From component 2, we see that a greater reward is correlated with a greater chance the offer will be used.

From component 1, we see that a greater chance an offer will be used, the more likely the customer is more senior.

Development Stage 6: Test and evaluate several supervised learning algorithms with default parameters and pick the one with the highest accuracy and f-score

I created a training and predicting pipeline in order to quickly train and predict several models consecutively.

I decided to test 5 different models on 1000 samples:

- Gaussian Naive Bayes
- Decision Tree Classifier
- Support Vector Machine
- K-Nearest Neighbor
- Stochastic Gradient Descent

The F-score for the AdaBoost Classifier on the test set was the highest at *0.462* and accuracy at *0.814*. I decided tune this model's hyperparameters to hopefully improve it.

```python
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import SGDClassifier

# Initialize models to test
gaussian_model = GaussianNB()
tree_model = DecisionTreeClassifier()
svc_model = SVC()
adaboost_model = AdaBoostClassifier()
knearest_model = KNeighborsClassifier()

# Dictionary to hold results
results = {}
sample_size = 1000

# For all models
for model in [gaussian_model, tree_model, svc_model, adaboost_model, knearest_model]:

    # Get model name
    model_name = model.__class__.__name__
    results[model_name] = train_predict(model, sample_size, X_train, y_train, X_test, y_test)

    print('{0} accuracy on test data: {1:0.3f}'.format(model_name, results[model_name]['accuracy_test']))
    print('{0} f-beta score on test data: {1:0.3f}\n'.format(model_name, results[model_name]['fscore_test']))
```

```
GaussianNB trained on 1000 samples!
GaussianNB accuracy on test data: 0.792
GaussianNB f-beta score on test data: 0.111

DecisionTreeClassifier trained on 1000 samples!
DecisionTreeClassifier accuracy on test data: 0.779
DecisionTreeClassifier f-beta score on test data: 0.432

SVC trained on 1000 samples!
SVC accuracy on test data: 0.798
SVC f-beta score on test data: 0.000

AdaBoostClassifier trained on 1000 samples!
AdaBoostClassifier accuracy on test data: 0.814
AdaBoostClassifier f-beta score on test data: 0.462

KNeighborsClassifier trained on 1000 samples!
KNeighborsClassifier accuracy on test data: 0.767
KNeighborsClassifier f-beta score on test data: 0.256
```

Development Stage 7: Tune model hyperparameters with mode promising algorithm

I then tuned AdaBoost's hyperparameters in an attempt to improve it.

I used Grid Search to test hyperparameters 25, 50, and 75 for *n_estimators* and 1, 0.1, and 0.001, for *learning_rate*.

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer

# Initialize model of choice
model = AdaBoostClassifier(random_state=0)

# Create parameter list to tune
parameters = {
    'n_estimators': [25, 50, 75],
    'learning_rate': [1, 0.1, 0.001]
}

# Make f-beta scoring object
scorer = make_scorer(fbeta_score, beta=0.5)

# Perform grid search on classifier using scorer as the scoring method (using all CPUs)
grid_search_obj = GridSearchCV(model, param_grid=parameters, scoring=scorer, n_jobs=-1)

# Find optimal parameter by fitting object to training data
grid_search_fit = grid_search_obj.fit(X_train, y_train)

# Note the best estimator
best_model = grid_search_fit.best_estimator_

# Make predictions using unoptimized and optimized model
predictions = model.fit(X_train, y_train).predict(X_test)
best_predictions = best_model.predict(X_test)
```

Development Stage 8: Evaluate model using accuracy and f-beta score

Finally, the first evaluation is performed.

Unoptimized AdaBoostClassifier Model:

Accuracy on testing data: 0.8170

F-score on testing data: 0.4638

Optimized AdaBoostClassifier Model:

Final accuracy score on the testing data: 0.8271

Final F-score on the testing data: 0.5405

Development Stage 9: Compare model performance against a naive predictor that always predicts a customer to respond to an offer

I then compared the optimized model to a naive predictor in order make sure it was at least sufficient by this metric.

The naive model always outputted 1. This was an arbitrary choice.

It ended up with an accuracy of 0.2 and f-beta score of 0.2385.

Development Stage 10: Concluding thoughts

The optimized AdaBoost Classifier was only 1% better than the unoptimized model in terms of accuracy, and 17% better in terms of F-Beta Score. Not negligible, but not ground-breaking either.

In terms of comparison with the naive model, it was far better by ~400% in accuracy, and ~225% in terms of F-Beta Score - a significant improvement.