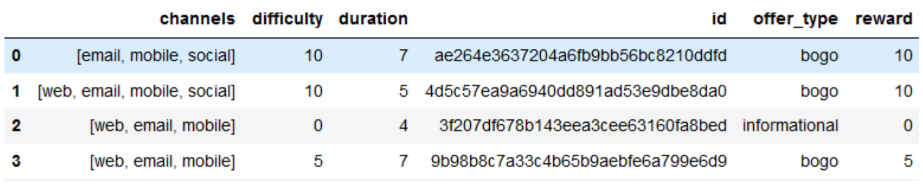
Machine Learning Capstone Project

Definition

Project Overview

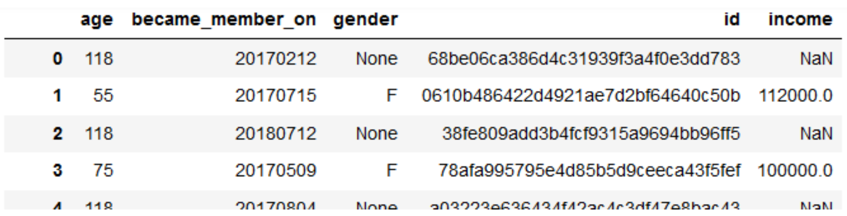
I chose to undertake the Starbucks Capstone project in which students were provided with 3 JSON files containing information about several offers, profiles of members, and a transcript of all event logs. This data is especially insightful for folks in business strategy who are looking to drive revenue by more effectively targeting appropriate demographics/psychographics. While many business professionals make use of data visualization and business intelligence tools like SQL or Tableau to read insights, I thought it might be interesting to go a step further and try to predict individual behavior as opposed to creating aggregated profiles to then target. I was initially curious as to how this would hold up. In particular, I wanted to know if our model might predict more closely the percentage of users who would complete an offer contingent upon them receiving it on the test data when compared with an aggregated sum from the tables. The portfolio dataset consists of 9 different offers with varying types of rewards, duration active, difficulty levels, channels to access it, and type. The profile dataset contained information on a users age, day they became a member, gender, and income. There were lots of missing datapoints. The transcript dataset contained event type, a person id, time since promotion started it occurred, and a value either containing an offer id and reward or an amount for a transaction.

Portfolio data frame



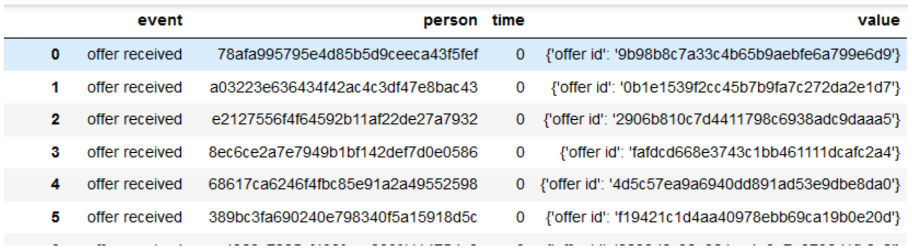
	channels	difficulty	duration	id	offer_type	reward
0	[email, mobile, social]	10	7	ae264e3637204a6fb9bb56bc8210ddfd	bogo	10
1	[web, email, mobile, social]	10	5	4d5c57ea9a6940dd891ad53e9dbe8da0	bogo	10
2	[web, email, mobile]	0	4	3f207df678b143eea3cee63160fa8bed	informational	0
3	[web, email, mobile]	5	7	9b98b8c7a33c4b65b9aebfe6a799e6d9	bogo	5

Profile data frame



	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

Transcript data frame



	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'}
5	offer received	389bc3fa690240e798340f5a15918d5c	0	{'offer id': 'f19421c1d4aa40978ebb69ca19b0e20d'}

Problem Statement

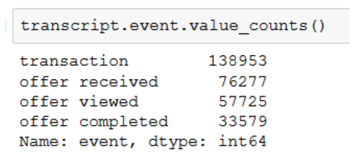
While heuristics are useful in segmenting target audiences for new businesses, might it be useful to take advantage of machine learning algorithms to target customers on a granular level? There might be certain patterns a machine picks up that a data analyst may not see. Similarly, it might improve performance and help us predict precisely who will buy something before they even view an offer or coupon. Through this capstone, I aimed to combine the datasets and run models on them to predict individual decisions. This was a binary classification problem where I tried to predict whether someone would complete an offer or not based on certain factors provided.

Metrics

Accuracy is a common metric used by people to gauge a model's success. IN addition to accuracy, I wanted to be able to tell whether the model was doing well of its merit or by chance, i.e. would a person guessing at random or someone who guess a single value to better than my model. This can be captured in Cohen's kappa statistic. Additionally, I did observe f1 and logloss though these were not the focus of my interpretation. Additionally, I compared the predicted % of completed offers on the test dataset to those received from a balanced model as well as one trained on imbalanced classes against the percent of people who completed offers within the "understanding" (training + validation) dataset. This was to compare on aggregate performance: if our data shows that 20% of offers received by people were completed and we expect our underlying offers and customers to behave the same in the short term (when the test data set becomes available) then we expect that number to remain steady and become a prediction not just a description.

Analysis

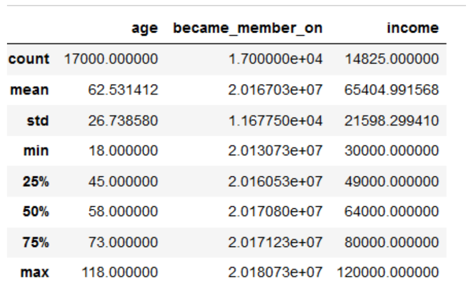
Data Exploration



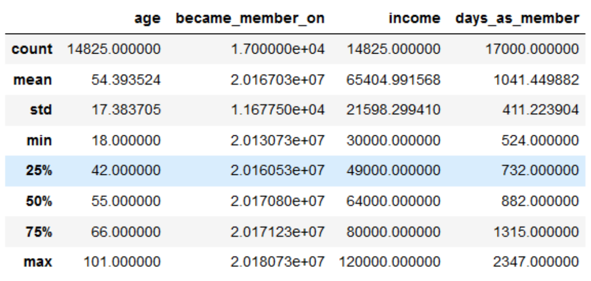
```
transcript.event.value_counts()

transaction        138953
offer received      76277
offer viewed        57725
offer completed     33579
Name: event, dtype: int64
```

Using simple describe functions and value_counts, I was able to get a glimpse of the data. When first looking through the profile data frame, I noticed a significant number of members had not provided information about themselves such as gender or income. Similarly, their ages were coded as 118, which I interpreted as missing (and recategorized as NaN). During each step where I made progress towards merging the dataframes and getting the features into the desired format, I used groupby and describe functions to analyze and understand the data. I found disparities between viewing and completing offers between the genders and by type of offer.



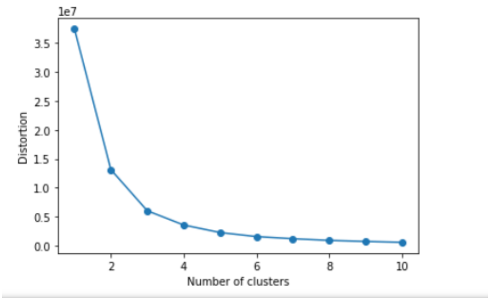
	age	became_member_on	income
count	17000.000000	1.700000e+04	14825.000000
mean	62.531412	2.016703e+07	65404.991568
std	26.738580	1.167750e+04	21598.299410
min	18.000000	2.013073e+07	30000.000000
25%	45.000000	2.016053e+07	49000.000000
50%	58.000000	2.017080e+07	64000.000000
75%	73.000000	2.017123e+07	80000.000000
max	118.000000	2.018073e+07	120000.000000



	age	became_member_on	income	days_as_member
count	14825.000000	1.700000e+04	14825.000000	17000.000000
mean	54.393524	2.016703e+07	65404.991568	1041.449882
std	17.383705	1.167750e+04	21598.299410	411.223904
min	18.000000	2.013073e+07	30000.000000	524.000000
25%	42.000000	2.016053e+07	49000.000000	732.000000
50%	55.000000	2.017080e+07	64000.000000	882.000000
75%	66.000000	2.017123e+07	80000.000000	1315.000000
max	101.000000	2.018073e+07	120000.000000	2347.000000

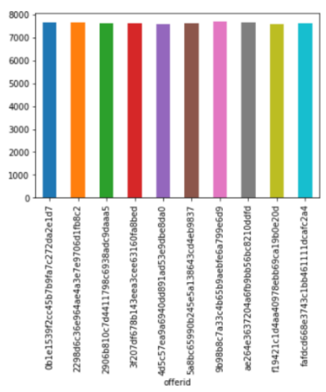
Exploratory Visualization

I tried to get a better sense of the distribution of our users' income, time as member and age. This was helpful for me in deciding how to deal with these features when creating my "model ready" data frame. To better understand how offers were received, viewed, and completed, I made some plots. It seems that while offers were sent in equal numbers to users, they were not viewed equally, nor acted upon in equal measure. There were two offers which no one had completed. Upon closer inspection, it turns out they were informational offers without any reward. Therefore, it could have been that there wasn't a way to encode them or tell if users had completed them. Conversely it could mean that they were not meant to be "completed".

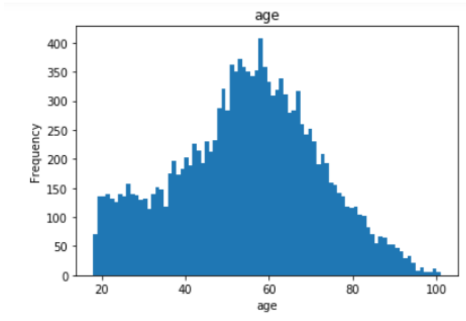


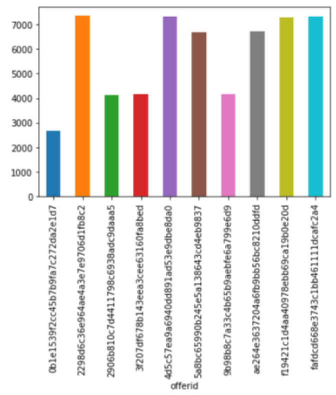
I also used seaborn to create a correlation chart to help me during feature selection.

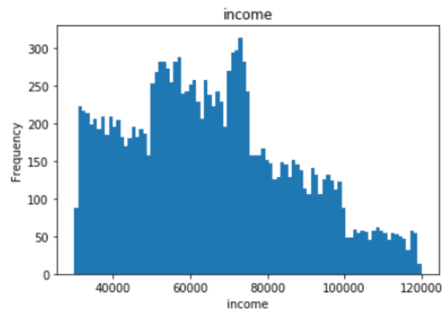
Algorithms and Techniques

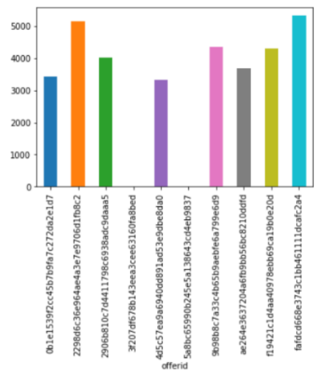


The problem I chose was a classification one. I decided to test out multiple algorithms on my training and validation data sets. However, even before running trained supervised classification models on the validation or test data, I used k-means to visualize potential clusters (between k=1 and 11) for several columns in my merged data frame. I used the elbow method with distortion to get the number of potential clusters I wanted to create when preparing my data for the model. I considered other methods of creating intervals but decided use the cut function from pandas and replace numericals with labels from bins I created. These bins were equal in size which did stay on my mind given how the data for the columns in question were distributed. The reward, difficulty and duration columns had few unique data points and I initially considered converting them into dummy variables or normalizing them for analysis.

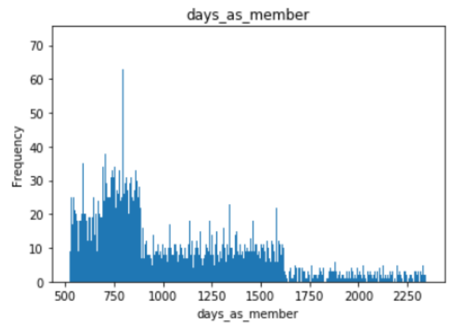




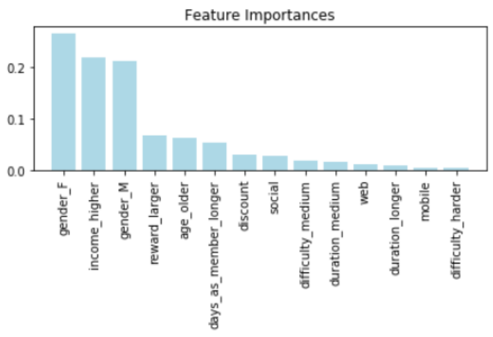




The other algorithm I ran before touching the validation and test sets was random forest with 1000 decision trees. I used the random forest algorithm primarily to discover variable importance. I created two simple plots (one was the Python Machine Learning Packt Community Experience Distilled book) to visualize and compare variable importance. Using the random forest in such a way gave me some insight



into which variables were contributing the most information to our model.



When it came to running models on the validation set, I chose 6 different models to compare. Though I've heard many pros and cons about different machine learning algorithms under different circumstances, I've found it best to always try multiple. For this project, I was especially interested in the performance of the base algorithsm with default settings and so I ran them each a few times. I eventually removed my SVM (svm) model as it was taking incredibly wrong and as I discovered like logistic classifier was simply guessing 0 for every instance. Realzing my class imbalance problem and unwilling to use methods like kfold cross validation on a fairly large dataset, I decided to implement weight balancing to handle the imbalanced classes on the algorithms which accepted this as a parameter (Random forest, decision tree classifier, and logistic regression). I should mention, I also employed knn and Naïve Bayes. My knn algorithm came up short and took longer to execute than all others but the SVM which I removed after the first iteration. The best performing models on this data with regards to accuracy, kappa, and f1 were the logistic regression, random forest classifier, and decision trees.

Random Forest is an ensemble method and drawing inspiration from the max_features parameter, I decided to have each model run multiple times with different number of training features each time which were determined by our variable importance. I was shocked at how much better models performed with so few features compared to multiple and this realization made me jot some notes on how to handle this better in future cases via dimensionality reduction through PCA or LDA.

Benchmark

As mentioned, my benchmark was the simple predictive statement I made based upon aggregating the first joined "understanding" data frame: namely that 23.64% of all people who received an offer would accept it. I decided to use this because the objective of my models was to accurately predict whether someone would complete an offer or not based on a set of criteria regarding the offer and information about themselves. Initially, I believed this scaled up well to answering that question as I would simply have to be closer to the "true" or % of test data offer completed. I did use accuracy scores, f1, cohen's kappa and analyzing confusion matrices to evaluate how I had or hadn't met my goals. This experience taught me about how important choosing appropriate benchmarks and tailoring to optimize for surpassing that benchmark is. As mentioned, to improve accuracy, I had to deal with class imbalance. I also realized that accuracy was not a good measure for me particularly because without a baseline, it masked the actual performance of my model.

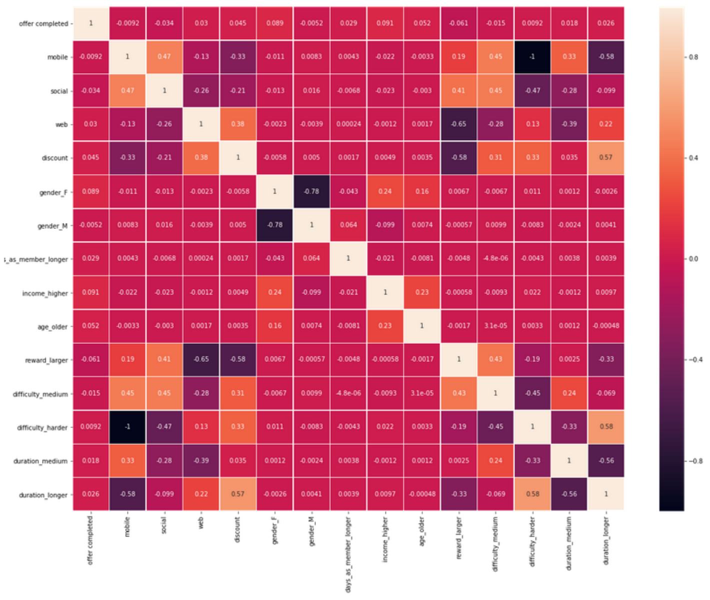
Methodology

Data Preprocessing

I ended spending a significant chunk of my time on preprocessing. Initially, when I noticed I had a significant number of missing values on hand, I became a bit concerned. I realized soon that the missing values were missing together, namely age, income and gender might be missing for the same row. This told me that the member had not given their personal information. When faced with this problem, one has several choices. The first step is to determine whether values are missing completely at random, missing at random, or missing not at random. From there it's you can determine whether to drop missing values, "ignore" them, or impute new values for them. In R, two powerful tools are the mice and Amelia packages. In this case, I to handle the missing data another slightly insidious way. I converted those variables with significant missing values into categorical ones and then created dummies.

Once I did this, I noticed certain variables were true for every case and thus unhelpful for prediction. Namely offer received was true for every row in my data frame built upon aggregating every combination of peronid offerid possible. One major assumption I made was that the same offer wasn't shown twice to the same person during this period. Another change I made during preprocessing was to drop all the transaction rows, the offer viewed column (as every instance of someone completing an offer meant they viewed it), and the other gender column. I took the principles of the k-1 approach to dummy variables and dropped of the "lowest" column from each set of dummies I created.

I also dropped off every row corresponding to an informational offer in getting my joined data frame ready for the model. I did this as informational offers were not helpful in that people did not complete them. As part of feature selection for my models, I used Random Forest and Seaborn to determine and visualize variable importance. I wanted to avoid having too many dummy variables and my target was to keep the number of variables below 20. Initially, during my first run I had around 40 and managed to bring that number down by dropping features which were accounted for or unimportant as determined by Random Forest.



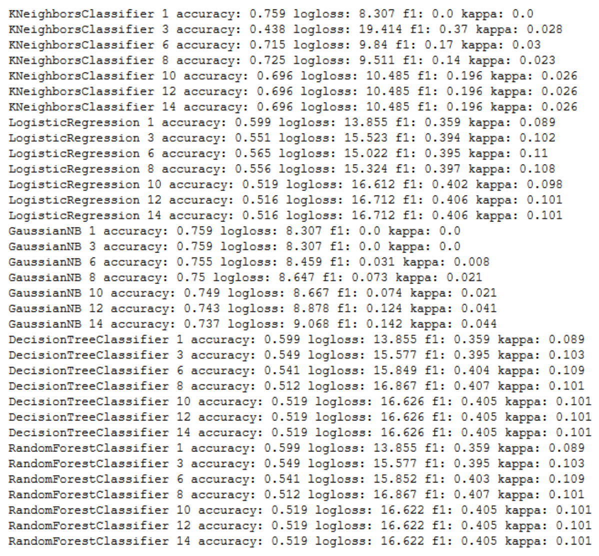
Implementation

I used the sklearn library, matplotlib, pandas, and numpy for this project and worked primarily in the workspace assigned to the project using Jupyter Notebook. One of the biggest challenges I overcame early on was grouping the dataset in such as way as to encapsulate unique combinations of gender and offerid as well as returning the furthest along someone had gotten with respect to the offer for that particular offer. The data wrangling and preprocessing as well as feature selection took up a significant amount of time. I went back into previous sections of my notebook to update the list of features which I dropped, changes I'd make to others and how to handle certain cases. With regards to classification, I wanted to change variables into forms where it was the presence and not the degree which mattered which is why I chose to categorize rather than scale or normalize certain features.

I was unable to use the xgboost algorithm in the workspace I was working in, nor could I import the balanced_accuracy metric which I got around by using weight=balanced for the models which I could, including the one I ended up using on my test dataset.

Refinement

As mentioned previously, I used random forest for feature importance. I also ran each of my classifiers on a different number of features to see how they would react as I increased the number the models were trained on. I was incredibly intrigued by the loss of accuracy in some cases, despite an increase in kappa in those very same cases. I ran both balanced and unbalanced version ofs of these algorithms and kept the results of both for the randomforest I ran on my test dataset. For the knn I trained, I changed the number of nearest neighbors from 5 to 3 though early on, I had decided I would choose between a



```
KNeighborsClassifier 1 accuracy: 0.759 logloss: 8.307 f1: 0.0 kappa: 0.0
KNeighborsClassifier 3 accuracy: 0.438 logloss: 19.414 f1: 0.37 kappa: 0.028
KNeighborsClassifier 6 accuracy: 0.715 logloss: 9.84 f1: 0.17 kappa: 0.03
KNeighborsClassifier 8 accuracy: 0.725 logloss: 9.511 f1: 0.14 kappa: 0.023
KNeighborsClassifier 10 accuracy: 0.696 logloss: 10.485 f1: 0.196 kappa: 0.026
KNeighborsClassifier 12 accuracy: 0.696 logloss: 10.485 f1: 0.196 kappa: 0.026
KNeighborsClassifier 14 accuracy: 0.696 logloss: 10.485 f1: 0.196 kappa: 0.026
LogisticRegression 1 accuracy: 0.599 logloss: 13.855 f1: 0.359 kappa: 0.089
LogisticRegression 3 accuracy: 0.551 logloss: 15.523 f1: 0.394 kappa: 0.102
LogisticRegression 6 accuracy: 0.565 logloss: 15.022 f1: 0.395 kappa: 0.11
LogisticRegression 8 accuracy: 0.556 logloss: 15.324 f1: 0.397 kappa: 0.108
LogisticRegression 10 accuracy: 0.519 logloss: 16.612 f1: 0.402 kappa: 0.098
LogisticRegression 12 accuracy: 0.516 logloss: 16.712 f1: 0.406 kappa: 0.101
LogisticRegression 14 accuracy: 0.516 logloss: 16.712 f1: 0.406 kappa: 0.101
GaussianNB 1 accuracy: 0.759 logloss: 8.307 f1: 0.0 kappa: 0.0
GaussianNB 3 accuracy: 0.759 logloss: 8.307 f1: 0.0 kappa: 0.0
GaussianNB 6 accuracy: 0.755 logloss: 8.459 f1: 0.031 kappa: 0.008
GaussianNB 8 accuracy: 0.75 logloss: 8.647 f1: 0.073 kappa: 0.021
GaussianNB 10 accuracy: 0.749 logloss: 8.667 f1: 0.074 kappa: 0.021
GaussianNB 12 accuracy: 0.743 logloss: 8.878 f1: 0.124 kappa: 0.041
GaussianNB 14 accuracy: 0.737 logloss: 9.068 f1: 0.142 kappa: 0.044
DecisionTreeClassifier 1 accuracy: 0.599 logloss: 13.855 f1: 0.359 kappa: 0.089
DecisionTreeClassifier 3 accuracy: 0.549 logloss: 15.577 f1: 0.395 kappa: 0.103
DecisionTreeClassifier 6 accuracy: 0.541 logloss: 15.849 f1: 0.404 kappa: 0.109
DecisionTreeClassifier 8 accuracy: 0.512 logloss: 16.867 f1: 0.407 kappa: 0.101
DecisionTreeClassifier 10 accuracy: 0.519 logloss: 16.626 f1: 0.405 kappa: 0.101
DecisionTreeClassifier 12 accuracy: 0.519 logloss: 16.626 f1: 0.405 kappa: 0.101
DecisionTreeClassifier 14 accuracy: 0.519 logloss: 16.626 f1: 0.405 kappa: 0.101
RandomForestClassifier 1 accuracy: 0.599 logloss: 13.855 f1: 0.359 kappa: 0.089
RandomForestClassifier 3 accuracy: 0.549 logloss: 15.577 f1: 0.395 kappa: 0.103
RandomForestClassifier 6 accuracy: 0.541 logloss: 15.852 f1: 0.403 kappa: 0.109
RandomForestClassifier 8 accuracy: 0.512 logloss: 16.867 f1: 0.407 kappa: 0.101
RandomForestClassifier 10 accuracy: 0.519 logloss: 16.622 f1: 0.405 kappa: 0.101
RandomForestClassifier 12 accuracy: 0.519 logloss: 16.622 f1: 0.405 kappa: 0.101
RandomForestClassifier 14 accuracy: 0.519 logloss: 16.622 f1: 0.405 kappa: 0.101
```

tree algorithm or logistic regression given the speed and results — namely accuracy and kappa — these provided in this case compared with the other options available. I changed the number of trees for my randomforest and experimented with 100, 300, and 1000. As I wasn't receiving any significant deltas on the score metrics by leaping from 300 to 1000 trees while also spending a lot more time, I decided to revert to 300, a happy medium.

Results

Model Evaluation and Validation

The final model I chose was a random forest classifier. A random forest is an ensemble method using decision trees and employing techniques such as feature reduction during splits. Random Forest consistently proved among the high performers on the validate data I fed it. The random_seed set on my randomforest was the same as every other instance where I used a random seed (for replicability) at 123.

```
clf = RandomForestClassifier(n_estimators=300, n_jobs= -1, random_state=123,\
                                              max_features='sqrt', class_weight="balanced")
```

I chose 300 estimators and as is suggested for classification, chose to use sqrt for max features. The image shown above is for the model I ran accounting for class imbalance. I ran another model (with a higher accuracy) without the class weight set to balanced. I examined both when evaluating the model against the benchmark I had chosen.

Justification

My accuracy for the imbalanced data was a 75.78%. With the balancing, accuracy fell down to 54%. During validation, Random Forest had the highest kappa stats on par with logistic regression. I found that the accuracy was not impressive. In fact, just guessing 0 100% of the time would yield roughly the same accuracy, and that is what occurred with the imbalanced data. Also the kappa was low at around 0.11, indicating that we didn't perform much better than someone who just guessed. I also sought to compare the % of people it predicted would complete an offer once they'd received to the 23.64% I'd found in my understanding (train + validation) data prior. It seems like the random forest algorithm(imbalanced) only predicted 2 people would complete an offer once having received it. This was radically different with the balancing weight model which believed that nearly 53% of people who received an offer would complete it. The truth was far closer to the prediction made earlier at 24.21%. That being said, there were certainly some interesting discoveries along the way including the far high completion rate of women to most offers than men. Going further, while the false positives in our data would be problematic from a prescriptive POV (if you decide to act on this data and send this individual an offer and they don't complete it, how might this impact attrition?) Conversely, false negatives which lead to you not sending an offer that would have been completed is leaving money on the table. Our imbalanced model produced a lot more false negatives than false positives. Let's talk about different approaches to potentially pursue in the future and improve the model. From a business standpoint, our decisions may prevent us from generating revenue but they'll help us avoid attrition or bother the consumer. We can nickname this a "conservative" model which doesn't want to bother consumers with excessive offers/advertising.

Further Steps

What Did I Learn

Data wrangling, preprocessing and feature engineering/selection take a lot of time relative to model fitting. Most of my time was spent getting data into the right format and trying to visualize it through tables, functions, or plots. Additionally, accuracy is not necessarily a robust measure of success. If I hadn't chosen another statistic like kappa, I might have used an un assuming GaussianNB (Naïve Bayes) and had a kappa of 0 with an accuracy explained by pure chance. I also learned that tailoring your data format to the results you're trying to achieve is incredibly important. Representing features in the right format is important, as is handling missing values. While I didn't impute missing values, that's certainly an option. If the data was determined to be missing completely at random, it could have been dropped without consequence. Different algorithms are better able to handle different tasks. For example, while support vector machines are quite stable and can handle both regression and classification, it can take a relatively long time to train and is difficult to interpret. Conversely, decision trees are easier to understand for us mortals and even logistic regression classifier can return probabilities we can view as confidence within a decision.

Potential Next Steps

In the future, it may be interesting to try preprocessing a different way such as normalizing or standardizing numeric data instead of turning them into dummies after categorizing. One could also simply drop all missing values to avoid dealing with them or impute either mean, median or random values (this is how some algorithms handle missing values). Reducing feature dimensions through PCA might help us optimally capture variance. Using k-fold cross validation is another useful method to improve your models. With certain machine learning models, hyper parameter tuning can yield drastically different results. There are endless possibilities in data mining on what questions to answer, how to answer them, and what to do with your data to get it to tell you something. I look forward to continue exploring different possibilities within data mining and machine learning.