**Goals**

* build predictive model
* explain the key indicators of future redemption behaviors

**Problems**

* Class imbalance
  + only 17% of the data has future redemptions
* Data are in time series column-wise

**Key Metrics**

* Since future redemptions are mostly negative, model accuracy will always be high
* focus on:
  + increasing true positives
  + improving recall and f1-score

**Feature Engineering**

Since the data was in time series column-wise, I decided to use feature engineering to extract important pieces of information from each row. The feature extraction functions that I created are as follows:

* fix\_columns
  + Convert 'gap' to 0, 'engage' to 1, and verify to '2'. Keep session the same to extract app\_minutes later
* days\_since\_last
  + calculates the number of days since last activity
* engagement\_ratio
  + calculates the ratio of days that have engaged activity
* verify\_ratio
  + calculates the ratio of days that have verify activity
* app\_minutes
  + for the session data, uses regular expression to add the minutes spent on the app
* inactive\_days
  + returns the count of inactive days in the last 14 days
* last\_action
  + displays the last action of the user
* utilization\_ratio
  + calculates the ratio of utilization of the app to the total days

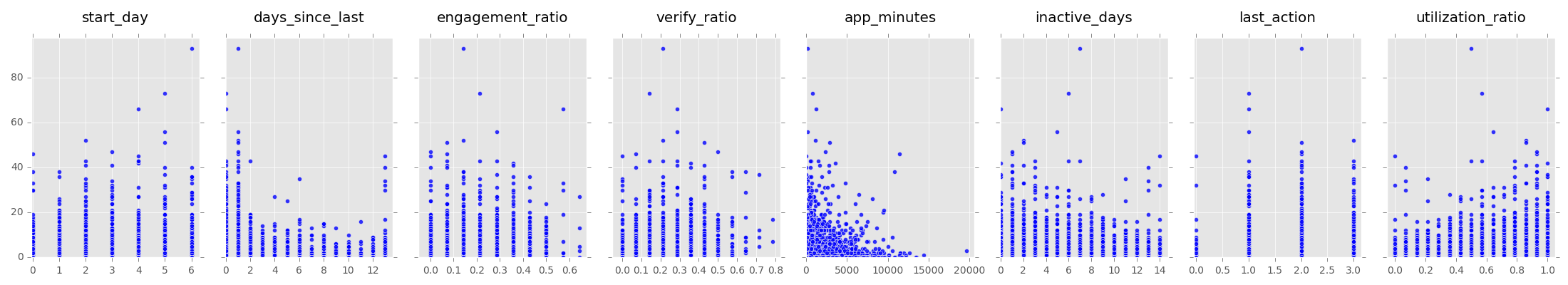
The feature functions were packaged in feature\_functions.py and mapped to the DataFrame via the following:

for func in functions:

df[func.\_\_name\_\_] = df[day\_columns].apply(lambda x: func(x), axis=1)

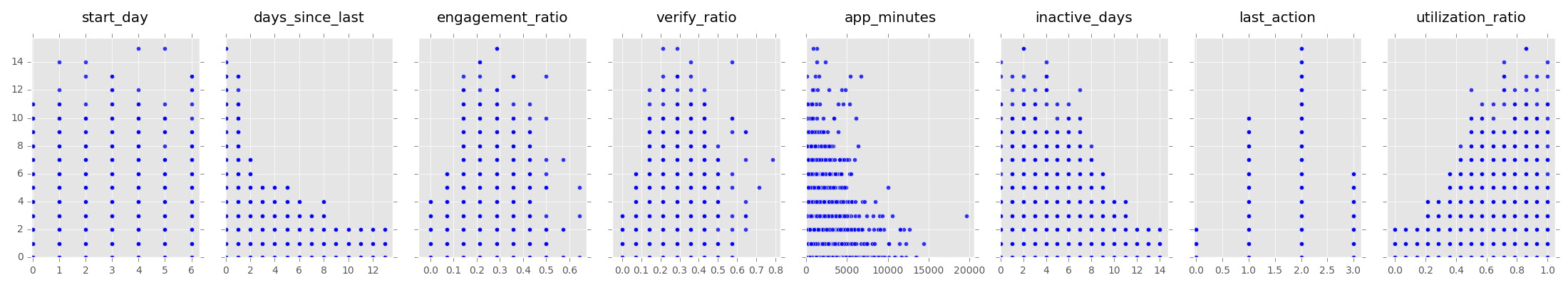
**EDA**

scatter plot of future\_redemptions to engineered feature

**[](https://github.com/estenssoros/ibotta_test/blob/master/part_2/images/outlier_scatter_matrix.png)**

**Outlier Removal**

For the discrete engineered features, there appears to be outliers. Using pandas and (mean + 3 \* standard deviation), I removed future\_redemptions outliers categorically for each feature.

**[](https://github.com/estenssoros/ibotta_test/blob/master/part_2/images/scatter_matrix.png)**

**Predictive Modeling**

4 different models were fit on the binary classification for future\_redemptions.

* random forrest (rf)
* gradient boosted random forrest (gbr)
* adaptive boosted random forrest (adr)
* k-nearest neighbors (knn)

I used sklearn's gridsearchcv to tune the hyperparamaters for gbr and adr however, random forest out performed both boosted models and the knn implementation.

**Class Imbalance**

Since only 17% of the data contained positive classification, I used numpy to sample with replacement (bootstrap) the positive classes and append to the data frame. The data was then split into training and testing groups to train the models.

df\_redeemed = df[df['redeemed'] == True]

k = len(df) - len(df\_redeemed)

up\_sample = df\_redeemed.iloc[np.random.randint(0, len(df\_redeemed), size=k)]

df = df.append(up\_sample, ignore\_index=True)

**The following charts show classification scoring for true classification:**

**Initial modeling**

| **model** | **precision** | **recall** | **f1-score** |
| --- | --- | --- | --- |
| rf | 0.56 | 0.41 | 0.47 |
| gbr | 0.63 | 0.41 | 0.51 |
| adr | 0.63 | 0.45 | 0.52 |
| knn | 0.53 | 0.38 | 0.44 |

**Removed outliers**

| **model** | **precision** | **recall** | **f1-score** | |
| --- | --- | --- | --- | --- |
| rf | 0.55 | 0.45 | 0.50 |
| gbr | 0.64 | 0.50 | 0.56 |
| adr | 0.64 | 0.50 | 0.56 |
| knn | 0.56 | 0.45 | 0.50 |

**Removed outliers and up-sampled imblanced class**

| **model** | **precision** | **recall** | **f1-score** |
| --- | --- | --- | --- |
| rf | 0.91 | 0.96 | 0.93 |
| gbr | 0.83 | 0.86 | 0.84 |
| adr | 0.83 | 0.84 | 0.84 |
| knn | 0.83 | 0.94 | 0.88 |

**Feature importance**

Using the random forest model with removed outliers and up-sample of the imbalanced class, we can use the random forest class' .feature\_importances\_ to see how each feature was split during the fit of the trees.

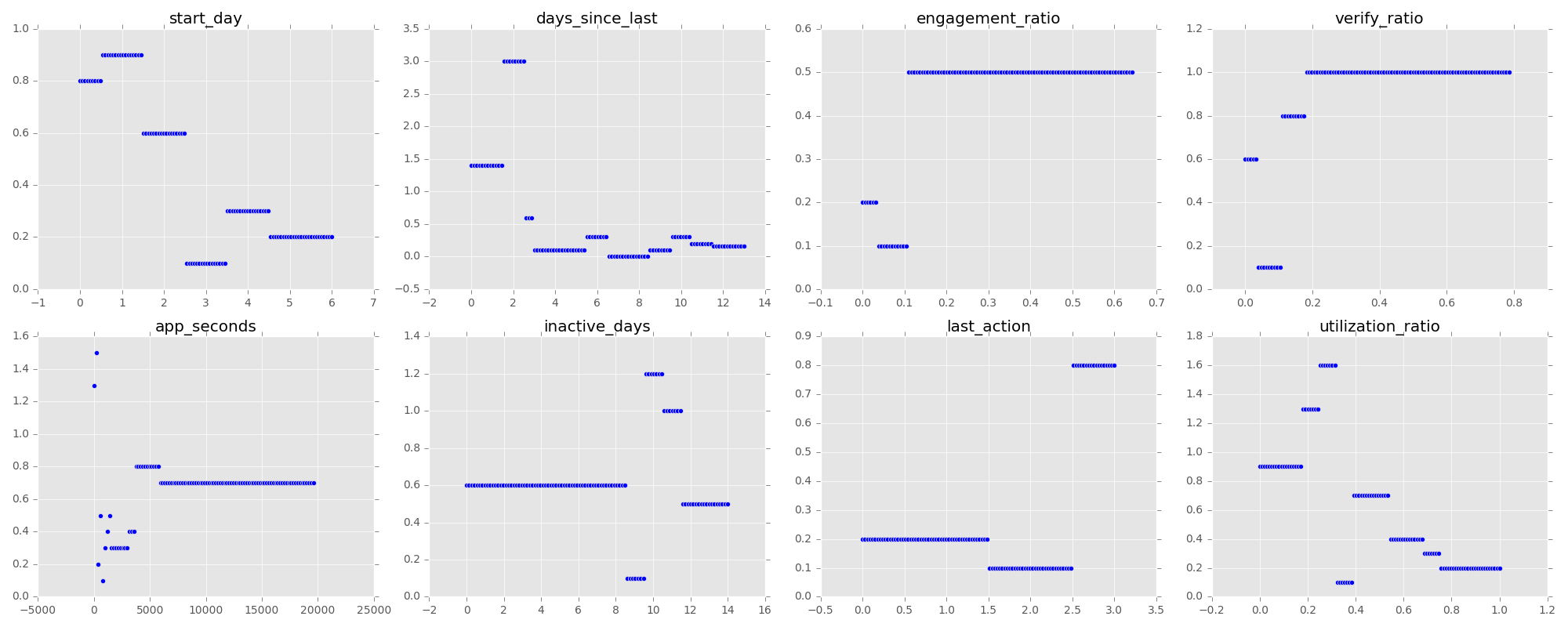
* start\_day - 10.85%
* days\_since\_last - 21.95%
* engagement\_ratio - 4.55%
* verify\_ratio - 15.27%
* app\_minutes - 17.89%
* inactive\_days - 19.69%
* last\_action - 2.36%
* utilization\_ratio - 7.44%

This shows us that for determining if a user will make a future redemption, the most important features to consider are the days since last action, minutes spent on the app, and number of inactive days in the last 2 weeks.

However, this does not allow us to predict how many future redemptions the user will have. We can instead run a simple Random Forrest Regressor model on the data that we have created.

**The resulting R squared score was 88.6%**

I used the trained model and iterated over the range of possible variables for each feature holding the other columns fixed at their mean value. The result shows us changing each variable, impacts the value of future redemptions.

**[](https://github.com/estenssoros/ibotta_test/blob/master/part_2/images/feature_ranges.png)**

Using the previously stated feature importances, we see that if we wish to improve chances of future redemption, we should target:

* users who have days since last use are greater than 8
* app\_seconds that are less than 1000
* inactive days that are lower than 10

­­­