**Campaign Performance Predictive Model**  
By Jessica Otaguro

**Introduction**

The Amazon Advertising Platform (AAP) allows advertisers to reach Amazon customers across the web through display and video ads. AAP utilizes Amazon’s first party shopping data to understand customer behaviors, allowing advertisers to target desired customers based on lifestyle, browsing habits, location, demographics, and more. Customers then can be reached on both Amazon O&O and third party web sources, on either desktop, tablet, and mobile browsers, as well as on mobile apps.

On AAP we are able to optimize towards a variety of key performance indicators (KPI) for our advertisers including return on ad spend (ROAS), click through rate (CTR), and detail page view rate (DPVR). ROAS is the most common and highest revenue-generating KPI on AAP and will be the only KPI used in this model.

***Can we predict ROAS based on campaign and product attributes   
such as campaign budget, time of year, product rating, and price?***

**Data**

Ad data was pulled via the Advertising Data Warehouse for 2017 US purchase campaigns. Campaigns under a certain spend and sales threshold were also removed. If a campaign tracked more than five products (ASINs), five ASINs were selected at random to represent the campaign. Product data (price, rating, number of reviews, and category) is not easily accessible by AAP and was therefore gathered using a Python script that scraped each product’s detail page (Figure 1). In some cases, details were not found on the page because the item was out of stock or removed from the site. For such campaigns, the median across the entire data set was inputted for price, reviews, and rating. These fields were then averaged across ASINs per campaign. Two categorical fields (advertiser and product category) were transformed into dummy variables so that they could be incorporated into the model, resulting in an x (data is private) row by 999 column data frame.



Figure 1. Code to pull product rating from Amazon.com



Figure 2. Code converting field data types, creating dummies, and setting the variables

**Data Dictionary**

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Data Type | Description | Example |
| advertiser\_name | Object (string) | Advertiser running the campaign | Toyota |
| ad\_campaign\_id | Numeric (integer) | Campaign unique identifier | 335234235 |
| startmonth | Numeric (integer) | Month of campaign start date | 2 |
| endmonth | Numeric (integer) | Month of campaign end date | 5 |
| campaignlength | Numeric (integer) | Length of campaign (End date – start date in days) | 31 |
| retargeting | Numeric (integer) | Yes/no if a retargeting line exists in the campaign | 1 |
| category | Object (string) | Highest level product category | Electronics |
| price | Numeric (float) | Average price of campaign ASINs | 19.95 |
| reviews | Numeric (float) | Average number of reviews | 386.4 |
| rating | Numeric (float) | Average rating (1-5 possible) | 4.3 |
| spend | Numeric (float) | Total ad spend | 21442.12 |
| sales | Numeric (float) | Total retail sales attributed to campaign | 44002.06 |
| dailyspend | Numeric (float) | Ad spend per day (spend / campaign length) | 596.43 |
| roas | Numeric (float) | Return on ad spend (sales / spend) | $9.64 |

Figure 3. Fields in the data set

**Modeling**

After building various models and using different combinations of features, the model with the lowest root mean squared error (RMSE) was the BaggingRegressor using the DecisionTreeRegressor as the base estimator. This ensembling estimator fits random subsets of the data using the decision tree estimator, and then averages their predictions together, thus reducing variance and producing a more accurate model. The model was measured for accuracy using the mean absolute error (MAE) and the root mean squared error (RMSE).

Accuracy metrics:  
OOB = 0.35  
MAE = 2.01  
RMSE = 4.88  
Null RMSE = 6.65

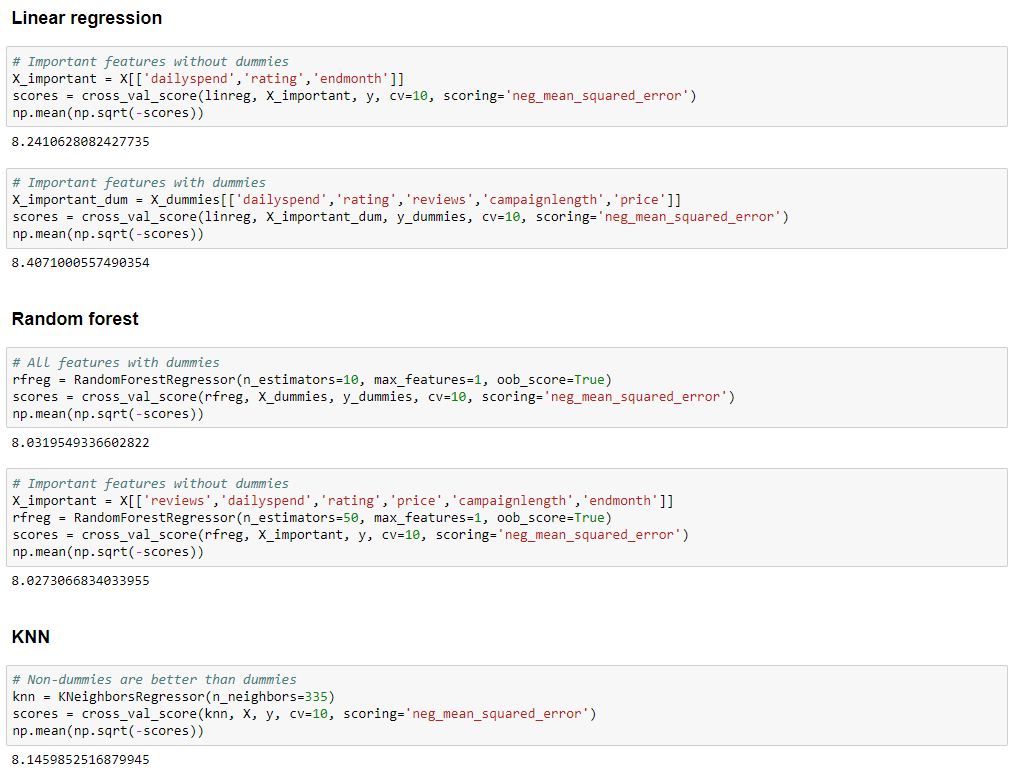


Figure 4. Code for trying out different estimators

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Features incl. dummies | | Features not incl. dummies | |
| All | Important | All | Important |
| LinearRegression | 128,447,239.61 | 8.41 | 8.44 | 8.24 |
| RandomForestRegressor | 7.67 | 8.21 | 7.94 | 7.92 |
| KNeigborsRegressor | 8.15 | 8.15 | 8.15 | 8.15 |
| BaggingRegressor(DecisionTreeRegressor) | 4.88 | 6.65 | 6.61 | 6.61 |
| BaggingRegressor(KNeigborsRegressor ) | 7.29 | 7.29 | 7.36 | 7.37 |
| BaggingRegressor(RandomForestRegressor) | 5.29 | 6.72 | 6.71 | 6.72 |
| BaggingRegressor(LinearRegression) | 25,581,479.72 | 7.44 | 7.67 | 7.44 |

Figure 5. RMSE of different models and features.

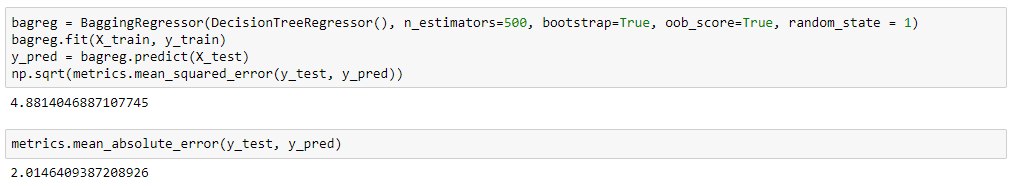


Figure 6. Code of selected model, a bagging regressor

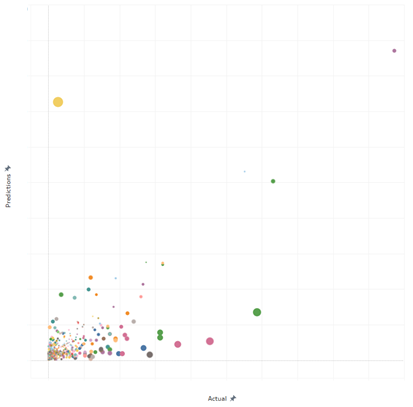


Figure 7. Each circle is a campaign that we tested the model on. X-axis is the actual ending ROAS and y-axis is the prediction using the model. Size of the bubble represents the error (distance between actual and predicted values).

**Conclusion and Next Steps**

This model was able to predict return on ad spend fairly accurately for our testing set of campaigns, but can be improved and can evolve into a more accurate, scalable, and extensive model with the below next steps.

* + Remove outliers from data set, hone features, and try additional estimators to make model more accurate
  + Build distinct models for additional KPIs, regions, and entities
  + Test model accuracy on future campaigns
  + Incorporate additional features such as supply sources, segments, bids, etc.
  + Capture only hero ASIN instead of all tracked ASINs
  + Ingest retail product data systematically
  + Build a tool on top of the script so that users can input campaign details and receive a prediction