Arvato Bartelsman Customer Segmentation Report

Sebastian Koenig - Udacity Capstone - January 2020

Overview **2**

Metrics 2

Algorithms and Techniques 3

Principal Component Analysis (PCA) 3

Kmeans Clustering 3

Extreme Gradient Boosting (XGB) 3

Summary **3**

Data Exploration and Feature Selection **4**

Customer Segmentation **6**

Data Cleaning 6

Scaling and Dimensionality Reduction 6

Kmeans Clustering 7

Target Segment Analysis 8

Supervised Learning Model **9**

Addressing Data Imbalance 9

Baseline XGBClassifier 9

Intuitive Hyperparameter Tuning 10

Tuned XGB Performance 10

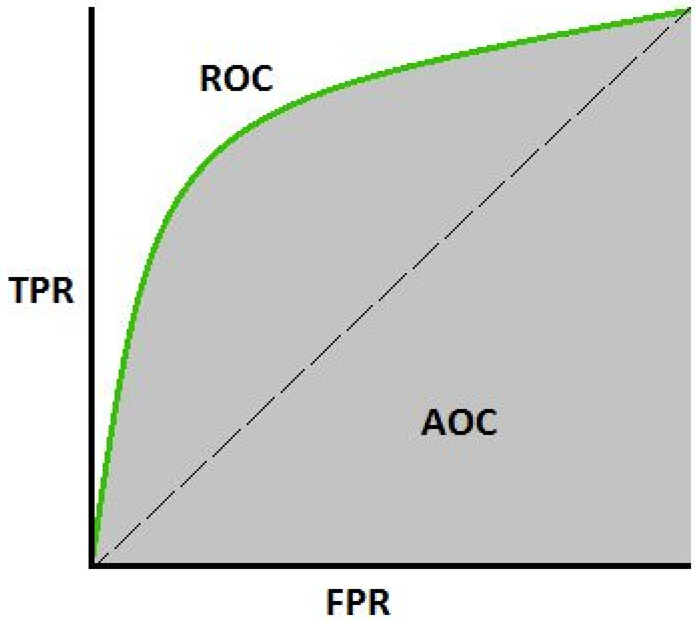
Reexamined Features 10

Kaggle Competition **11**

Overview

This project is essentially a marketing related data analysis task. We are trying to identify what part of the general population is most like the client's current customer base and what individuals would be most likely to become new customers if targeted by a campaign based on data from prior attempts.

Metrics



The datasets provided by Arvato Bertelsman are the result of broad market research consisting of a variety of attributes from classifications based on the household income to the number of children in the home or the types of cars prevalent in the neighborhood.
Our key metric to measure performance of the supervised learning model will be AUC. AUC stands for Area Under The Curve and that curve is the Receiver Operating Characteristics curve. As you can see in the graphic (courtesy of Towards Data-Science), AUC describes the area under the classifiers false positive rate (FPR - False Positives over False Positives and True Negatives) - true positive rate (TPR - True Positives over True Positives and False Negatives) curve. That means it is a measure of how clearly true positives and true negatives are distinguished. An AUC of 1 thus describes a perfect model, while an AUC of 0 would mean a model that perfectly misslables every item. AUC of 0.5 describes a model that can not distinguish between negatives and positives at all. It captures the performance of models on

imbalanced data sets much better than a simpler accuracy score would, just because it is easy to be 99% accurate when 99% of the labels are of one class.
AUC is not recommended when a specific type of classification error needs to be avoided or if the classification threshold, the probability at which a label is predicted as being positive, is relevant, this is however not the case for the given task.

Algorithms and Techniques

Principal Component Analysis (PCA)

PCA is a form of orthogonal linear transformation that sorts all features of a dataset in order of the amount of variance they explain. It is useful for dimensionality reduction, because we can then select a number of components starting with the most significant, which we want to use for further steps. It can also be used to reduce the number of strongly correlated features for algorithms more sensitive to that.

Kmeans Clustering

Kmeans Clustering is an unsupervised learning technique which groups data points into a user defined number of clusters based on how similar they are. We first have to define the number of clusters we want our data grouped into. Kmeans then creates that number of center points (k number of means) and groups all data points around the closest of these selected centers.

Extreme Gradient Boosting (XGB)

XGB is a refined form of Gradient Boosting, a machine learning technique which uses an ensemble of weak models, in this case decision trees, to produce a stronger predictor. Boosting is a sequential ensemble technique where subsequent models reduce the effect of poor models while boosting the effect high performing models have on the final prediction. Gradient Boosting refers to that same boosting technique enhanced by gradient descent (in XGBs case Stochastic Newtonian). Extreme Gradient Boosting improves on that formula by employing parallel processing, tree-pruning (the removal of under performing leaves/branches or even entire trees), and finally several regularization tools to reduce overfitting.

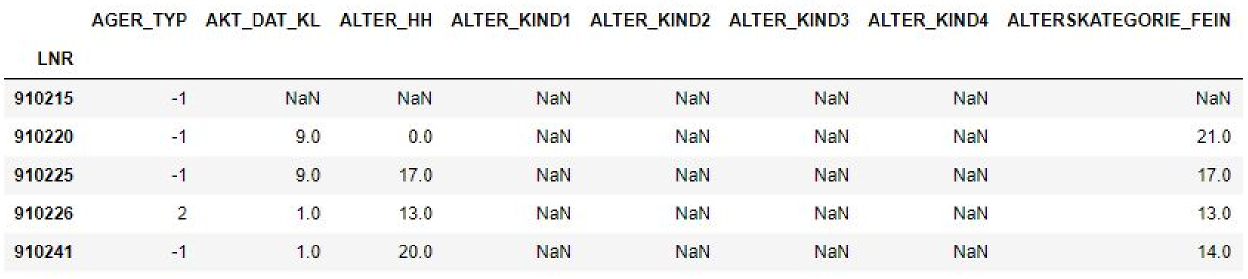
Summary

This report includes both customer segmentation using Kmeans clustering and the use of a supervised XGBClassifier to identify those individuals who are most likely to be responsive to an advertisement campaign.
Data exploration revealed that some steps had to be taken to reduce noise. This included ensuring uniform encoding of missing data, removal of sections with unusually large amounts of missing values, and the selection of useful features. Kmeans clustering pointed towards a target

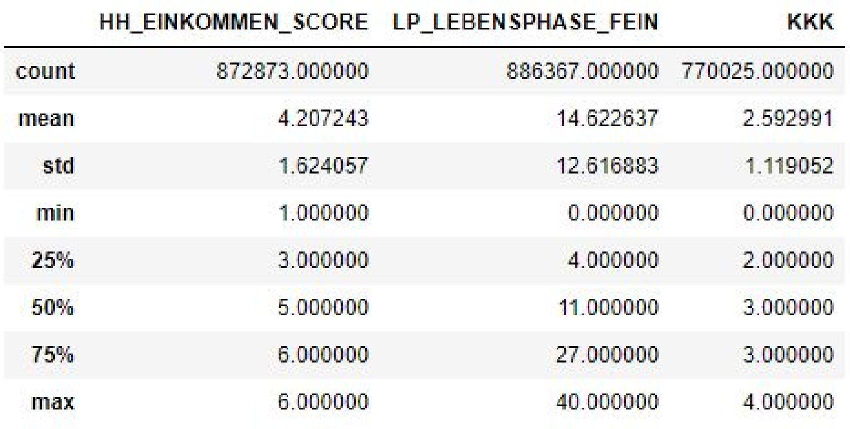
segment roughly 30% of the general population with similar attributes as 60% of the clients customer base.
The training set for the supervised learning task proved highly imbalanced, I therefore resampled data with SMOTEENN (automatic over and undersampling). The XGBClassifier delivered an AUC of 0.996+ on the resampled training data but only about 0.627 on the unaltered development set. Hyperparameter tuning seemed the next sensible step. After testing some values manually I found an XGBClassifier that performed only marginally better which revealed that feature engineering would be more crucial. I first discovered that our oversampling actually caused the algorithm to overfit, XGB is remarkably resilient to imbalanced data and my attempt to train it on balanced data meant that it produced false positives.
I then recleaned the Training data, leaving a larger selection of remaining features and achieved significantly better results. I finally decided to run a randomized hyperparameter search to see if a better selection of hyperparameters than those intuitively selected could be found. The final model performed better than all other XGB models but failed to beat the Google AutoML performance.

Data Exploration and Feature Selection

The data set consisted of just over 890,000 entries with 360+ features.

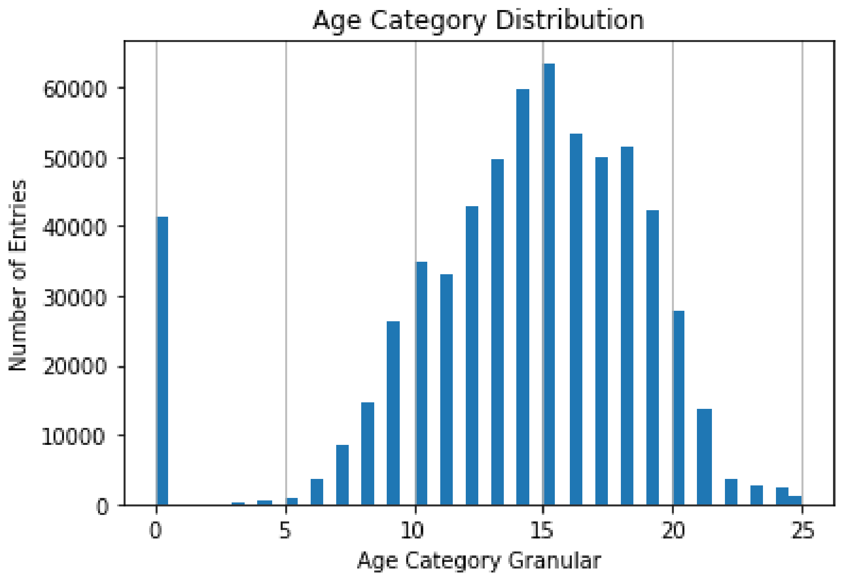


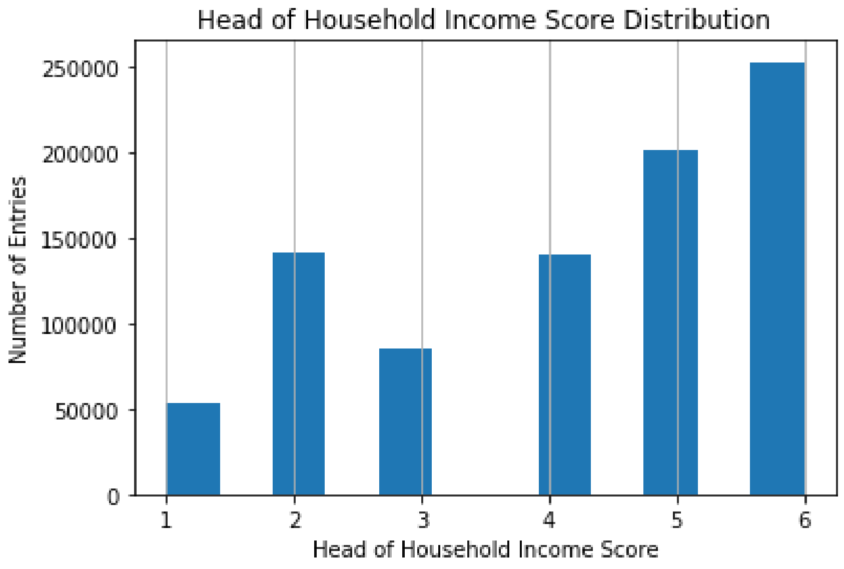
	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN
LNR								
910215	-1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
910220	-1	9.0	0.0	NaN	NaN	NaN	NaN	21.0
910225	-1	9.0	17.0	NaN	NaN	NaN	NaN	17.0
910226	2	1.0	13.0	NaN	NaN	NaN	NaN	13.0
910241	-1	1.0	20.0	NaN	NaN	NaN	NaN	14.0



	HH_EINKOMMEN_SCORE	LP_LEBENSPHASE_FEIN	KKK
count	872873.000000	886367.000000	770025.000000
mean	4.207243	14.622637	2.592991
std	1.624057	12.616883	1.119052
min	1.000000	0.000000	0.000000
25%	3.000000	4.000000	2.000000
50%	5.000000	11.000000	3.000000
75%	6.000000	27.000000	3.000000
max	6.000000	40.000000	4.000000

As you can see in the two snapshots above, the data had significant portions of missing information and comprised mostly of scores, and other numerically encoded categorical data such as information about the individuals life phase and purchasing power.





As you can see above the data was collected on a wide range of individuals, in both age and economic status. However the age category histogram also shows a clear problem with the data: Large sections of missing data, in this case encoded as a 0.

Looking at the explanation of the features provided in an excel sheet, I realized that a significant portion of those features dealt primarily with the prevalence of makes and models of cars owned by the consumer or prevalent in their local area. This seemed of little relevance as our client is a mail order company. Obviously this data could give some indirect insight into the socio-economic conditions of the household, but I do not believe it to be significant enough given the inclusion of far more direct metrics in the dataset. Data concerned with cars was thus removed from consideration. I also removed any other entries which were not explained in the provided excel sheet and thus not useful for the segmentation report. There is not much point in including a feature when it cannot be understood what it means if it was significant.

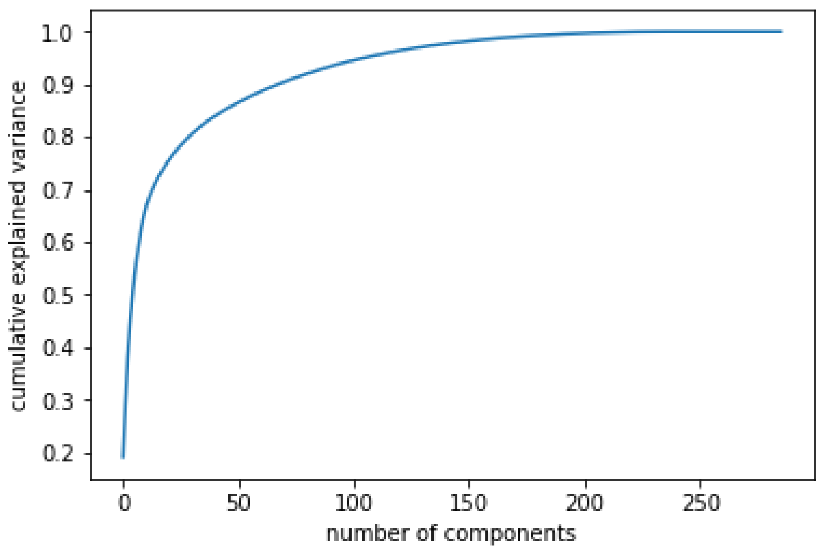
Customer Segmentation

Data Cleaning

Data exploration revealed that quite a few of the values in our data actually encode missing information. Ensuring that missing information is uniformly encoded was vital to the project. Therefore I created a checkbook from the above mentioned explanatory excel sheet identifying which values encode missing information for each feature, then iterated over the whole dataset to uniformly encode all missing information as NaN.
In a second step we analyzed which features had large gaps and removed them from consideration. We then performed the same elimination for those individual entries which were highly incomplete. To ensure that both the general population and customer data would be processed the same way we used the remaining features of the general population data as a filter for the customer data.
Finally we looked at the remaining features to figure out which ones represented non-sequential categorical data. Sequential categorical data or binary data that is numerically encoded can be used in learning algorithms, but non-sequential categorical data, even when encoded numerically, needs to be reencoded because the algorithm will treat numbers close in value as related even if the underlying categories have no relation to one another. A list of these features was created and one-hot encoding applied, generating a new binary feature for each category.
Finally we defined a cleaning function to automate the cleaning process for all other datasets.

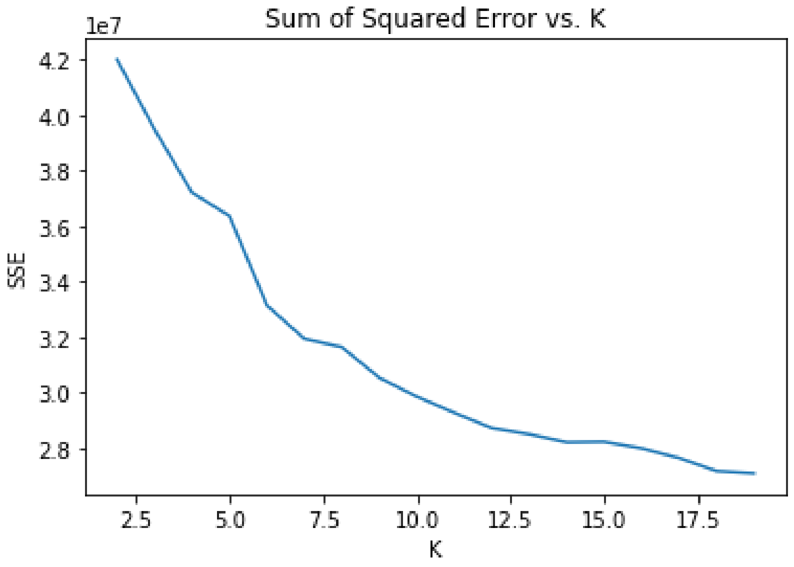
Scaling and Dimensionality Reduction

I used a simple imputer to replace all remaining missing values in the data set with the median value for the feature then re-scaled it to ensure that all values fall into the same range. As a final step in the preparation I used principal component analysis (PCA) to reduce the dimensionality of the dataset without losing significant variance.
As visible in the graph about 200 features explain close to 100% of the variance in the data, thus applied PCA dimensionality reduction to include only the most significant 100 features.

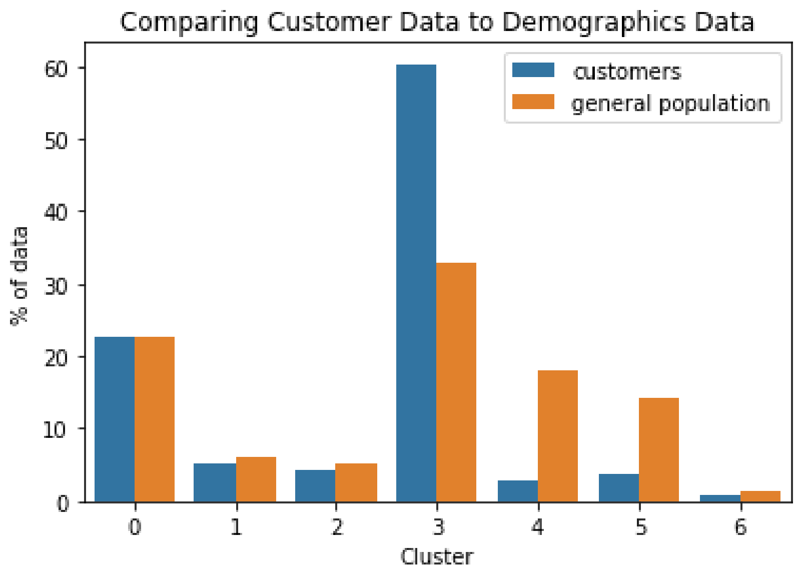


Kmeans Clustering

To determine how many clusters our data should be grouped into we used MiniBatchKmeans to iterate over each possibility between two and twenty clusters, looking at the sum of squared error as a proxy for the relative intra-cluster distance between data points.



It appeared sensible to try both 7 and 14 clusters as reduction in intra-cluster distance dropped off significantly after those two values. Ultimately I elected to use only 7 clusters as we found a cluster that accounted for the majority of the customer data while providing us a target segment of the general population, and the more granular 14 clusters did not provide a smaller target window resembling a similar or larger portion of the customer data.



Target Segment Analysis

After clustering I extracted the portion of the general population data belonging to the target cluster (3) and saved it indexed by LNR which serves as a unique identifier for every data point. Using that data in combination with the value explanation spreadsheet we can get some general insight into what type of individual would be most likely to become a new customer.
The following is a table giving the average values for our target group along a few selected list of features as an example:

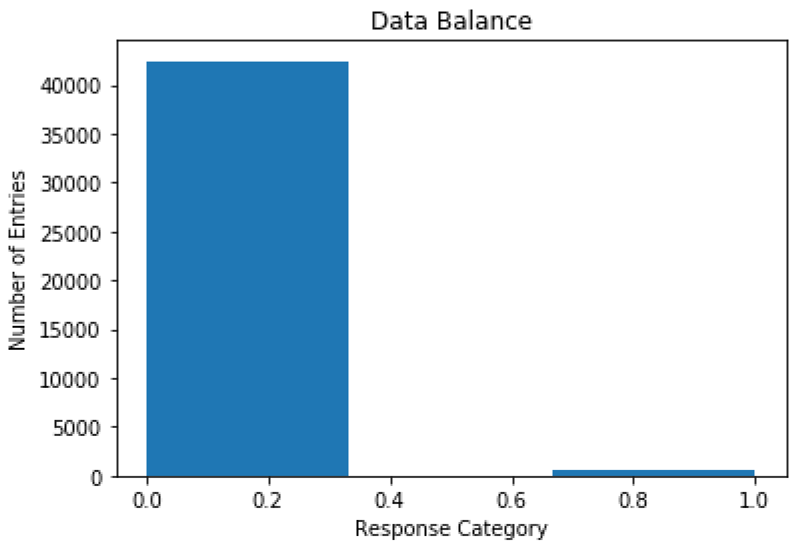
Affluence	Established Middle Class
Age	45-60
Household size	2
Movement	Mainstream
Location	40-50km outside of urban centers, West Germany

This appears to be people around age 50, usually two person households living on average 40 to 50 km outside of the nearest urban center. They tend to have slightly above average purchasing power.

Supervised Learning Model

Knowing that Extreme Gradient Boosting (XGB) algorithms have performed well on similar problems in the past, I decided to use a basic XGB as the baseline and see how much performance can be improved by a combination of intuitive manual and randomized algorithmic hyperparameter tuning.

Addressing Data Imbalance



The primary difficulty of this task is the extreme imbalance of the dataset. Of the almost 43,000 data points only 532 are positive responses. Training data this imbalanced incentivises many algorithms to label everything uniformly as the dominant class. To address this I resampled the data using a combination of synthetic over- and undersampling to create a more balanced data set.

Baseline XGBClassifier

To establish a baseline I trained an XGBClassifier on the resampled data and evaluated it on a development set roughly 10% of the original training data provided, which was not resampled. XGB fit the training data well (AUC 0.996+) but performed less well on the evaluation set (AUC 0.627). This algorithm was clearly overfitting the training data. I decided to first tune hyperparameters manually to see if the issue was one of tuning or feature engineering.

Intuitive Hyperparameter Tuning

I have trained some XGBClassifiers for past projects and decided to use a few combinations of hyperparameters that served me well on similar datasets in the past, choosing the following:

```
XGBClassifier(max_depth = 10, eta = 0.1, n_estimators = 50, min_child_weight = 1, subsample
= 0.5, colsample_bytree = 0.8, gamma = 0, reg_alpha = 0.01, reg_lambda = 5, seed = 1,
n_jobs=-1, objective = 'binary:logistic')
```

Tuned XGB Performance

Our tuned XGB performed a lot better on the un-resampled training data (AUC ~0.681 to ~0.876), but didn't improve on the test data (AUC ~0.62). We were able to improve fit without further over fitting but didn't manage to reduce overfitting either. Training the optimized model on imbalanced data reduced overfitting a little, but not significantly enough (validation set AUC ~0.64).
At this point additional feature engineering appeared to be the only option for further progress. We likely removed features important to model performance in cleaning. While our cleaning approach served us well for the Kmeans clustering it is possible that some of the features not explained in the DIAS Attributes Values 2017 excel file are important to model performance.

Reexamined Features

I decided to reclean the data manually, leaving a lot more features, specifically those not explained in the DIAS Attributes Values 2017 excel sheet. They were not useful for segmentation as we would not have been able to explain what they signified even if they had produced better defined clusters, but they might improve our ability to predict customer responses. Indeed even our model optimized for the original feature selection performed significantly better (Validation Set AUC of 0.8+). I decided to test sklearn's RandomSearchCV algorithm to tune hyperparameters hoping to produce a final optimized model for our new feature set.

We ended up with this estimator:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
        colsample_bynode=1, colsample_bytree=0.5, eta=0.01, gamma=1,
        learning_rate=0.1, max_delta_step=0, max_depth=9,
        min_child_weight=10, missing=None, n_estimators=100, n_jobs=-1,
        nthread=None, objective='binary:logistic', random_state=0,
        reg_alpha=1, reg_lambda=5, scale_pos_weight=1, seed=1,
        silent=None, subsample=1.0, verbosity=1)
```

After randomly exploring this parameter grid:

```
xgb_param_grid= {
    'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'eta': [0.01, 0.05, 0.1, 0.2],
    'n_estimators': [50, 75, 100, 150, 200, 375, 500],
    'min_child_weight': [1, 5, 10],
    'subsample': [0.5, 0.8, 1.0],
    'colsample_bytree': [0.5, 0.8, 1.0],
    'gamma': [0, 0.1, 0.5, 1, 2],
    'reg_alpha': [0, 0.001, 0.01, 0.1, 1],
    'reg_lambda': [0, 0.01, 0.5, 1, 2, 5]}
```

As you can see the primary difference between our intuitively tuned XGB classifier and the result of the random search is a slightly reduced max depth (10 to 9), doubling the number of estimators to 100, and significantly increased regularization parameters min child weight from 1 to 10, gamma from 0 to 1, alpha from 0.1 to 1.
I ran a final grid search on the regularization parameters but could not improve performance further.

Kaggle Competition

Finally we cleaned the test data to conform to our revised feature engineering, imputed missing values and applied the same scaling as we did for the training data, then had our final XGBClassifier predict label probabilities, rather than actual labels as required by the Kaggle competition. Unfortunately we did not achieve the AUC we hoped and also failed to beat Google AutoML which achieves AUCs of 0.802+. We only got up to AUC 0.78791 with our XGBClassifier. In other words, the most sensible workflow for this project appears to be using a state of the art pre-build classifier like Google AutoML rather than to create one from scratch. I can see that with a lot of time or luck it could be possible to engineer features further to improve performance, but this would represent a lot of effort for marginal gain.

Model	XGBClassifier	TunedXGB	FinalXGB	Google AutoML
Kaggle Competition AUC	0.598	0.621	0.788	0.802