Capstone Project
Machine Learning Engineer Nanodegree
Brian Cleary
December 14th, 2016

I. Definition

Project Overview

Banco Santander (Bank Santander) is currently interested in improving their personalized product recommendations for their banking customers—a Kaggle Competition. Kaggle provided the relevant data sets from Santander Bank. The training dataset included demographic and customer information dating from Jan 2015 – May 2015, while the Test Set (prediction set) included demographic and customer information for June 2016. Santander Group (parent company) servers more than 100 million customers in the UK, Latin America and Europe. According the Banco Santander's website, "we aim to make your banking life easier by providing convenient and smart ways to spend, save, and manage your money – from basic checking and savings accounts to comprehensive financial solutions."

Per a research report from Aberdeen Group, financial services companies that used predictive analytic saw a 10% increase in identifying new customer opportunities in 2014, compared to a 7% increase in firms not using predictive analytics (TIBCO.) Banco Santander is attempting to improve product recommendations for existing customers to increase the customer experience and increase cross-sell opportunities. I believe that I can train a supervised machine learning algorithm on the training data, then make reasonable predictions for June 2016 (Kaggle Prediction Set).

Problem Statement

Under their current product recommendation system, a small number of Santander's customers receive many recommendations while many others rarely see any resulting in an uneven customer experience. Santander is challenging Kagglers to predict which products their existing customers will purchase in June 2016 based on 1.5 years of data.

Banco Santander has provided 1.5 years of customer behavior data. The data starts with Jan 2015 showing monthly snapshots of customers each month until May 2015. The monthly records identify which of the twenty-four-possible product the customer used that month and some demographic information such age and income. The variables used to train the model are discussed individually in Section II: Analysis.

I will use the Training data set to identify which model best captures the structure of the data to use the model to make general predictions on data that the model has not seen (i.e. June 2016). Santander is only interested in the top 7 product recommendations for June 2016.

Evaluation Metrics

Based on this problem domain, I will need to predict the likelihood of the top seven products for June 2016. By engineering a dependent variable that was a multi-class column for each possible product, I was able utilize supervisory classification algorithms. Probabilistic predictions will allow me to rank-order the predictions before I submit to select the seven most likely products that will be purchased.

During training and evaluation, I will score each model based on the log-loss. I am using log-loss because MAP@7 is not differentiable (XGBoost requirement) and log-loss is. Since I want to compare the same metric during the model building phase, I will use log-loss for all. Once the models are built, I will submit the predictions to Kaggle to get the official MAP@7 score and compare the results of MAP@7 to log loss.

Per Kaggle, logarithmic loss (log-loss) is an error metric used when the goal is to predict whether the dependent variable is true or false with a probability (likelihood) ranging from true (1) to equally true (0.5) to false (0). Per scikit-learn, log-loss can be extended to multiclass problems. The mathematical jargon on scikit-learn is, "let the true labels for a set of samples be encoded as a 1-of-K binary indicator matrix Y, if sample i has label k taken from a set of K labels. Let P be a matrix of probability estimates, with $P_{i,k} = Pr(t_{i,k} =1)$. Then the log loss of the whole set is"

$$L_{\log}(Y, P) = -\log \Pr(Y|P) = -\frac{1}{N} \sum_{i=0}^{N-1} \sum_{k=0}^{K-1} y_{i,k} \log p_{i,k}$$

Per the Kaggle, MAP is just a mean of average precision for all users. In other words, if we have 1000 users, we sum APs for each user and divide the sum by 1000. Each Banco Santander customer is hypothetically interested in some "new" products. We are tasked with recommending 7 items per user. In this completion, MAP@7 indicates the MAP for up to 7 product recommendations per customer. I am not penalized for bad guesses, so submitting all 7 recommendation is preferred; however, order matters (unless I get all right). I will select the best 7 candidates per customer (in order of most likely products to least likely).

Equations for AP and MAP: Average precision at n for the user—P (k) means the precision at cut-off k in the item list, i.e., the ratio of number of users followed, up to the position k, over the number k; P(k) equals 0 when the kth item is not followed upon recommendation; m is the number of relevant nodes; n is the number of predicted nodes. If the denominator is zero, P(k)/min(m, n) is set to zero. The mean average precision is the average of the AP at n for each user.

$$ap@n = \sum_{k=1}^{n} P(k)/min(m, n) \qquad MAP@n = \sum_{i=1}^{N} ap@n_i/N$$

II. Analysis

Data Exploration

Per Kaggle, "Banco Santander has provided 1.5 years of customer behavior data. The data starts at 2015-01-28 and has monthly records of products a customer has, such as "credit card", "savings account", etc. The Training Data set contains 13,647,309 rows of customer information and 48 features and the Kaggle Prediction dataset contains 929,615 rows of

customer information and 48 features. First I will discuss how I will approach the problem and down sample the Training Data set and capture seasonality.

Table I: Data Set features and Santander Products			
Features	**Description**	**Santander Products**	**Description**
fecha_dato	Date of entry: The table is partitioned for this column	ind_ahor_fin_ult1	Saving Account
ncodpers	Customer Code	ind_aval_fin_utl1	Guarantees
ind_empleado	Employee index: A active, B ex employed, F filial, N not employee, P passive	ind_cco_fin_utl1	Current Accounts
pais_residencia	Customer's country residence	ind_cder_fin_utl1	Derivada Account
sexo	Customer's sex	ind_cno_fin_ult1	Payroll Account
age	Age	ind_ctju_fin_ult1	Junior Account
fecha_alta	The date in which the customer became as the first holder of a contract in the bank	ind_ctma_fin_ult1	Más Particular Account
ind_nuevo	New customer Index. 1 if the customer registered in the last 6 months.	ind_ctop_fin_ult1	Particular Account
antiguedad	Customer seniority (in months)	ind_ctpp_fin_ult1	Particular Plus Account
indrel	1 (First/Primary), 99 (Primary customer during the month but not at the end of the month)	ind_deco_fin_ult1	Short-term deposits
ult_fec_cli_1t	Last date as primary customer (if he isn't at the end of the month)	ind_deme_fin_ult1	Medium-term deposits
indrel_1mes	Customer type at the beginning of the month ,1 (First/Primary customer), 2 (co-owner), P (Potential),3 (former primary), 4(former co-owner)	ind_dela_fin_ult1	Long-term deposits
tiprel_1mes	Customer relation type at the beginning of the month, A (active), I (inactive), P (former customer),R (Potential)	ind_ecue_fin_ult1	e-account
indresi	Residence index (S (Yes) or N (No) if the residence country is the same than the bank country)	ind_fond_fin_ult1	Funds
indext	Foreigner index (S (Yes) or N (No) if the customer's birth country is different than the bank country)	ind_hip_fin_ult1	Mortgage
conyuemp	Spouse index. 1 if the customer is spouse of an employee	ind_plan_fin_ult1	Pensions
canal_entrada	channel used by the customer to join	ind_pres_fin_ult1	Loans
indfall	Deceased index. N/S	ind_reca_fin_ult1	Taxes
tipodom	Address type. 1, primary address	ind_tjcr_fin_ult1	Credit Card
cod_prov	Province code (customer's address)	ind_valo_fin_ult1	Securities
nomprov	Province name	ind_viv_fin_ult1	Home Account
ind_actividad_cliente	Activity index (1, active customer; 0, inactive customer)	ind_nomina_ult1	Payroll
renta	Gross income of the household	ind_nom_pens_ult1	Pensions
segmento	segmentation: 01 - VIP, 02 - Individuals 03 - college graduated	ind_recibo_ult1	Direct Debit

Datasets are public and available at: https://www.kaggle.com/c/santander-product-recommendation/data

I learned with other ML challenges that when predicting sales, it is wise to incorporate seasonality (Kaggle: Rossman). Santander wants to know what product a customer would add in June 2016 only—it is not asking which customers will add a product but if they did, which would they be. Since I am trying to predict products added in June 2016 that were not present in May 2016, I can create a subset of data that shows the customer information as of May 2015 (independent variables) with the products the customer added in June 2015 (dependent variable). I hope that this will allow for good predictions on the subset of data, preserve seasonality and allow my laptop to handle what would otherwise be a very large dataset. Per a Kaggle forum, Breakfast Pirate mentioned how he could score in the ~0.03 range with this technique so I'm confident that it can offer good predictions if implemented correctly.

Since I am looking at what products will be added in June 2016, I will look at what products customers added in June 2015. By checking each row of the Training file, I will create a subset that shows the features and products that the customer had as of May 2015, with a new feature column Y. The newly added Y column will represent which product that customer purchased in June 2015, indexed from 0-23 (the 24 products available; zero based index). This allows me to create a supervised learning problem, and I can train models to predict which product would be added given the set of features in any month.

The Y variable that I added consists of the all products collapsed into a new multi-class variable. If a customer added two or more products, each instance will show up separately on a different row. This method allowed me to cut down on the size of the training set and capture seasonality.

The descriptive statistics for the Train and Test set's numeric features show that some values are erroneous (negative seniority) and that Null values are causing problems. Age and Antiguedad do not show up in the Training set because Python coded them as objects as the Null values were strings and that is the default for Python. Null values will be addressed so the columns will have the correct data type. A look at income reveals that the top earner is much higher than the 3rd quartile—not unusual with income distributions but may want to address these outliers.

Descriptive Statistics on the Training Data

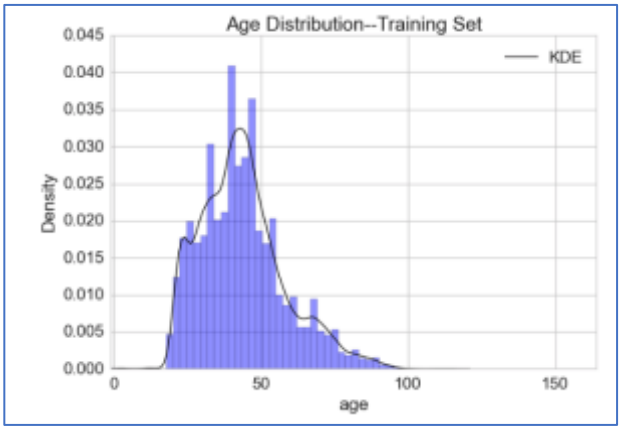
	renta	cod_prov
count	10,852,934	13,553,718
mean	134,254.32	26.57
std	230,620.24	12.78
min	1,202.73	1.00
25%	68,710.98	15.00
50%	101,850.00	28.00
75%	155,955.96	35.00
max	28,894,395.51	52.00

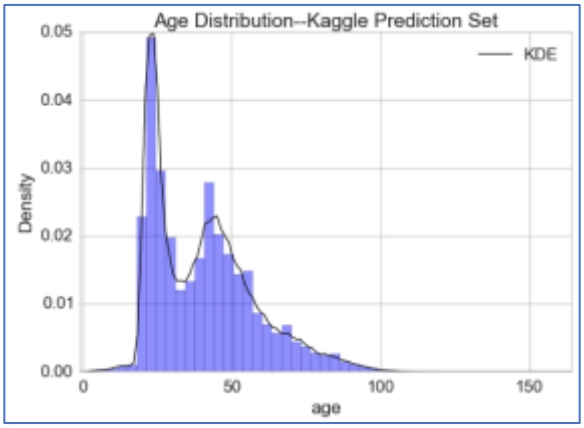
Descriptive Statistics on the Kaggle Prediction Data

	age	antiguedad	cod_prov
count	929,615	929,615	925,619
mean	40.25	77.73	26.55
std	17.19	1,797.82	12.84
min	2.00	(999,999.00)	1.00
25%	25.00	23.00	15.00
50%	39.00	55.00	28.00
75%	51.00	136.00	35.00
max	164.00	257.00	52.00

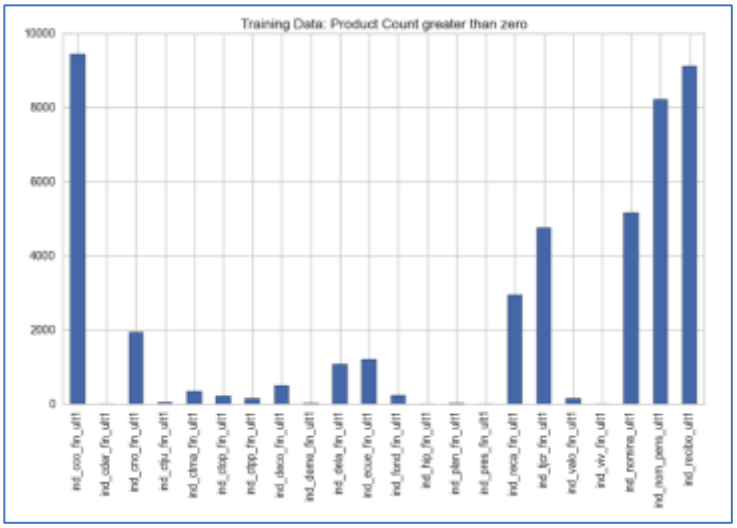
Exploratory Visualization:

When looking at the age distribution of the preprocessed Training Data that represents customers who added a product in June 2015 to the Kaggle prediction set, the distributions are very different. Customers who add products are more likely to be middle aged and the distribution has a mode of around 40-42 years old. However, the clientele of Santander bank is skewed towards the young and has a bimodal distribution.





After the data was preprocessed, I looked at the distribution of the dependent variable. The Y values were distributed unevenly. As we see below, two products are not added at all in June 2016—I will remove them from my product index when I predict which product will be added. Only seven of the remaining products were added more than ~1,500 times.



Unique Product Count: Training Set			
Product	Count	Product	Count
ind_ahor_fin_ult1	0	ind_ecue_fin_ult1	1,219
ind_aval_fin_ult1	0	ind_fond_fin_ult1	246
ind_cco_fin_ult1	9,457	ind_hip_fin_ult1	4
ind_cder_fin_ult1	9	ind_plan_fin_ult1	21
ind_cno_fin_ult1	1,934	ind_pres_fin_ult1	8
ind_ctju_fin_ult1	55	ind_reca_fin_ult1	2,942
ind_ctma_fin_ult 1	349	ind_tjcr_fin_ult1	4,755
ind_ctop_fin_ult1	222	ind_valo_fin_ult1	159
ind_ctpp_fin_ult1	154	ind_viv_fin_ult1	3
ind_deco_fin_ult1	503	ind_nomina_ult1	5,161
ind_deme_fin_ult 1	33	ind_nom_pens_ult 1	8,229
ind_dela_fin_ult1	1,085	ind_recibo_ult1	9,131

Algorithms and Techniques

- Naïve Bayes:

  A probabilistic supervised classifier able to predict the probability of the multiclass Y variable. Naïve Bayes classifier has several strengths: it is easy to implement, fast, well known and understood, and empirically successful. If independence

of attributes holds, NB classifier will converge quicker than discriminative models. Naïve Bayes biggest disadvantage is its simplicity.

At times, model complexity is advantageous when generalizing the relationship of from complex relationships and interactions. Furthermore, the assumption of independence among attributes may not be realistic and models that do not force this assumption on the class structure can better perform when there is high dependence among attributes. Per Haste, even if the individual class density estimates are biased, the posterior probabilities near the decision boundary can withstand considerable bias—the posterior probabilities can be smooth even when the population class densities are not (pg. 210-211). Naïve Bayes does not have hyper parameters to tune.

- Extreme Gradient Boosting (XGBoost)

  XGBoost, created by Tianqi Chen, is optimized for speed and allows the functionality of gradient decent and boosting. XGBoost automatically handles missing data values and offers continued training—boosting a previously fitted model with new data. XGBoost can handle multi-class classification problems using the *multi: softprob* objective.

  XGBoost offers a supervised classifier that is efficient (parallel computation), accurate (performs well on a variety of classification and regression problems), and customizable (allows tuning the hyper-parameters of the model and customized objective functions). Per Brownlee, the "Boosting" refers to the practice of converting a weak learning algorithm to a very good learning algorithm. A "weak learner" is simply one that is better than random chance.

  Adaptive Boosting (AdaBoost) was the first significant algorithm to have success with Boosting and uses decision trees with a single split. AdaBoost puts more weight on difficult to classify instances; then, new weak learners are added at each split, sequentially. The model attempts to improve on the more difficult instances. Thus, AdaBoost increases the focus (i.e. weights) of the samples that are most difficult to classify until the model is successful on these samples.

  Per Brownlee, the “Gradient” aspect of Gradient Boosting uses a statistical framework on a boosting model so that the objective is to minimize the loss of the model by adding weak learners using a gradient decent like procedure. A new weak learner is added iteratively to the existing weak learners (remained unchanged) in an additive fashion as no readjustment is made to previous terms. Gradient boosting is comprised of the loss function, which must be differentiable (which is why I used log-loss rather than MAP@7). Finally, trees are added one at a time as the gradient decent procedure is used to minimize the loss (i.e. follow the gradient). Cross validation is used to put a floor on the model’s tendency to over fit. Given enough time and relaxed constraints, the XGBoost model would be able to predict the training data set perfectly, while not generalizing to new data well.

  XGBoost offers way to ensure the learners remain weak, such as the depth of the tree and number of leaf nodes. In addition, XGBoost offers shrinkage (learning rate), which reduces the I influence of each individual tree---making the model less “greedy”. Furthermore, sub-sampling from the training set (either by row or column) reduces overfitting because this reduces the correlation between trees in the sequence of boosted

models. Finally, using traditional regularization functions (L1 and L2) in the leaf weight values (terminal nodes), helps to smooth learned weights to avoid overfitting.

Table II: XGBoost hyper parameters	
Hyper parameters	**Description**
Objective	specifies the learning objective of the model (multi: softprob). Softprob allows for probabilistic predictions
Max_depth	sets the maximum tree depth (default=3)
Learning_rate	sets the amount of learning per boosted round (default=0.10)
N_estimators	sets the number of boosted trees to fit (default=100)
Silent	whether to print progress while running boosting (default =)
Nthread	specifies the number of parallel threads used to run XGBoost (default=-1; all cores at once)
Gamma	specifies the minimum loss reduction required to make a further partition(default=0)
Min_child_weight	specifies the minimum instance weight needed in a child (default=1)
Max_delta_step	sets the maximum delta step we allow each tree's weight estimation to be (default=0)
Colsample_bytree	sets the ratio of columns available for selection when construction each tree(default=1)
Base_score	sets an initial prediction score of all instances (default=0.5)
Seed	sets a random number seed so that work can be reproduced (default=0)
Missing	sets value for customizable missing values (default=)

- Random Forest:

  An ensemble of decision trees that combine weak to strong learners via random search of features. Per Raschka, Random Forests combine weak learners to build a strong learner via majority vote. The Random Forest algorithm first draws a random bootstrap sample of size N (with replacement). From this bootstrap sample, at each node the algorithm randomly selects D features, splitting the node by maximizing the information gain. Once the Tree is build the algorithm then starts the process over with the bootstrap sample and continues until K trees are built. Once K (the only hyper parameter which tuning is significant) trees are build the algorithm is done and then the model aggregates the prediction by each tree to assign a class label by majority vote. Increasing the hyper parameter K will allow for a larger number of random trees to be built, therefore, increasing the computational demands but also resulting in a more robust model.

  Per Raschka, Random Forests are helpful in the feature selection process. Unlike PCA, Random Forests do not make any assumptions about linearity. Once the model is trained in K trees, all built randomly and ensemble into predictions, we can use the "feature_importances" method to report which features were most important to the model building process. I chose to use Random Forest technique as I have performed PCA analysis before and I wanted to familiarize myself with a new method. Raschka notes that for cases where interpretability is paramount, the feature selection can be misleading for highly correlated features (one will be ranked highly while the other one is ranked low). When the goal is predictive performance, this is not an issue and is one way to drop highly correlated features.

Table III: Random Forrest hyper parameters	
Hyper parameters	**Description**
n_estimators	The number of trees in the forest (default=10)

criterion	The function to measure the quality of a split (default= “gini”)
max_features	The number of features to consider when looking for the best split (default= “auto”)
max_depth	The maximum depth of the tree (default=2)
min_samples_split	The minimum number of samples required to split an internal node (default=1)
min_weight_fraction_leaf	The minimum number of samples required to be at a leaf node
max_leaf_nodes	Grow trees with max_leaf_nodes in best-first fashion(default=None)
bootstrap	Whether bootstrap samples are used when building tree (default=True)
oob_score	Whether to use out-of-bag samples to estimate the generalization error
n_jobs	The number of jobs to run in parallel for both fit and predict(default=1)
Random_state	The seed used by the random number generator (default=None)
verbose	Controls the verbosity of the tree building process (default=0)
warm_start	When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble (default=False)
Class_weight	If not given, all classes are supposed to have weight one. The “auto” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data(optional)

- Logistic Regression:

  Logistic regression was chosen because it is a probabilistic model by design by way of the logistic function. The output of the model transforms the Y-variable inputs onto a Sigmoid curve and the output is interpreted as the probability of that output occurring. One disadvantage to logistic regression is that is assumes the classes are linearly separable, yet it still is one of the most widely used classification algorithms due to ease of implementation and performance. With the OvR (one-vs-rest) technique in Scikit learn, the logistic regression can handle multi-classification problems. Real world applications for class membership probability are weather forecasting—we want to predict want to know how confident we are that it will rain or not.

Table IV: Logistic Regression hyper parameters

Hyper parameters	**Description**
penalty	Used to specify the norm used in the penalization. The newton-cg and lbfgs solvers support only l2 penalties (‘l1’ or ‘l2’)
dual	Dual formulation is only implemented for l2 penalty with liblinear solver
C	Inverse of regularization strength, smaller values specify stronger regularization(default=1.0)
fit_intercept	specifies if a constant (a.k.a. bias or intercept) should be added the decision function (default: True)
intercept_scaling	Useful only if solver is liblinear (default: 1)
class_weight	Over/under samples the samples of each class per the given weights(optional)
max_iter	Useful only for the newton-cg and lbfgs solvers. Maximum number of iterations taken for the solvers to converge.
random_state	The seed of the pseudo random number generator (default=None)
solver	Algorithm to use in the optimization problem. {‘newton-cg’, ‘lbfgs’, ‘liblinear’}
tol	Tolerance for stopping criteria (optional)

multi_class	Multiclass option can be either 'ovr' or 'multinomial' (ovr', 'multinomial')
verbose	For liblinear and lbfgs: any positive number for verbosity

Benchmark Model (Naive Bayes)

Per a Michael Littman, a Udacity lecturer, Naïve Bayes is often a good first model to run because it provides a reasonable baseline to compare other models against. I will use the Naïve Bayes classifier as a benchmark because it has several strengths: it is easy to implement, well known and understood, and empirically successful. If independence of attributes holds, NB classifier will converge quicker than discriminative models. Naïve Bayes biggest disadvantage is its simplicity. Naïve Bayes received a MAP@7 score of 0.0102542. I used this as the benchmark.

III. Methodology

Data Preprocessing:

1) Recode categorical feature duplicates

Although some labels in the Training set (i.e. ind_nuevo) could be represented at binary, the Prediction Kaggle set has unknown values. Since the data frames must be the same, I will not recode these instances as binary so that the features and labels match exactly. Only Indrel_lmes needs recoding for duplicate.

2) Impute N/A values and recode numeric features

Null values for numeric colums were originally recoded to -1 in the prepressing function. This allowed me to count the missing values and maintain the numeric data type for the column. Then the missing values for the four numeric columns were imputed: {Age: mode, Income: median, Sales Channel: mode, Seniority: mode}. For age and income, I also reclassified extreme outliers so that the min/max age was 20 and 90 and the maximum income recoded to $1.5M.

- Impute N/A values and recode Age of customer:
  I grouped under 20s as 20yrs old and over 90s as 90 yrs. Old to address the outliers. The Training set had 8 missing values with I set to 40---the age most prevalent for people that added products.
- Impute N/A values and recode Renta: Customer income:
  Many missing values in both the Training (6,193) and Kaggle prediction set (227,965). I recoded the missing values to median and capped high earners at 1.5 euros.
- Impute N/A values and recode Cod_prov (Sales Channel):
  Replace the null values in Sales channel with the mode from the Training set.
- Impute N/A values and recode Antiguedad (Seniority):
  Only 8 people are Null for seniority in the training set. Recoded to zero.

3) Recode categorical null values to "Unknown_Value":

Unlike numerical missing values, which I must either impute or delete, categorical missing values can be recoded to a new label so that no information is lost. Null values (N/A,

NaN and “”) were reclassified as “Unknown_Value”. This approach was chosen because when creating dummy variables, the “Unknown_Value” will still provide information to the ML algorithms—it might be possible that these data points cluster.

4) Scaling Features: Numeric Values

I choose to standardize my variables through the Standard Scaler object in Python rather to normalize the data with Python’s Min/Max scaler. Standardization centers the features columns at mean zero and standard deviation of one. Per Raschka, standardization can be more practical than normalization when using Logistic Regression and SVM.

Many linear models initialize the weights to zero and by using standardization, the feature columns take on a Gaussian distribution and make it easier for many machine learning models to learn the weights. Additionally, standardization retains outlier information rather than forcing a hard limit on the range of values.

5) One-Hot Encoding: Dummy Variables

The data contains several categorical feature columns and machine learning (ML) algorithms require numeric variables. Since none of these categorical features are ordinal, I cannot pass a value to them as the ML algorithms will presume they are ordinal. Per Rashka, this mistake is “one of the most common when dealing with categorical data”.

Instead, I used One-Hot Encoding to create a dummy feature for each unique value in the categorical feature column with the “get_dummies” method. After I performed "get_dummies", I checked to see that all columns in the Training and Kaggle prediction set were exactly the same--a requirement for machine learning. They were not, so I had to fix the problem by creating the same columns in each data frame (values were all zero). This created a very sparse matrix and is one reason I performed feature selection later with Random Forest. Finally, I sorted the columns alphabetically.

6) Split Train Data Set into Train/Validation Set for model building

I chose to use stratified K-fold cross validation to randomly split the Original Training Data into Train and Evaluate data sets. Stratification ensures that all classes are represented proportionally in each split. I split the data into three parts (k=3). Each split is called a fold, hence the name. K-fold provides better results when compared to arbitrary sub setting of the original training data. This is because when the model is being trained it evaluated multiple times on different data, though it takes longer.

Since this problem is an unbalanced multi-classification problem, stratified cross validation enforces class distributions so that one-fold does not have all the minority class labels, which would throw off the evaluation process. This is not actually done until the model building phase in my code.

7) Feature Importance & Selection (Random Forrest)

Per Raschka, Random Forests are helpful in the feature selection process. Unlike PCA, Random Forests do not make any assumptions about linearity. Once the model is trained in K trees, all built randomly and ensemble into predictions, we can use the "feature_importance" method to report which features were most important to the model building process. Total percent contribution of the top 25 features is 89.5%. I then subset both datasets by the top 25 features, cutting out 330 sparse columns that only contributed 10.5% to the model.

Implementation (XGBoost and Logistic Regression)

XGBoost and Logistic Regression algorithms were compared to the benchmark, Naïve Bayes, which scored 0.0102542 for MAP@7. First, I initialized an XGBoost model with default parameters, except for objective and seed. The objective was set to "multi: softprob" to allow for probabilistic predictions and the seed was set to allow for reproducible results. Similar hyper parameter adjustments were made for the Logistic Regression; {multi_class: 'ovr', random_state:43}. StratifiedKFold (k=3) was used to split the Training Data into Training and Evaluation subsets for model building. Grid Search objects were created for XGB and Logistic Regression.

I used the 25 top features selected in the preprocessing stage, then compared this to models run with all 355 features. I used the log-loss scoring function to evaluate the models during Grid Search (Training and Evaluation Data sets). I previously ensured that the two data frames were identical in every way. I than ran predictions with the best hyper parameters found in Grid Search for each model. Since the predictions were probabilities for the multi-classification problem, I had to transform these probabilities to products and rank-order the probabilities so that the largest were shown first. After I completed this with "argsort", I indexed the probabilities to the Santander product list, making sure not to include the two products that were not added in June 2015. After selecting the top seven products per customer for the prediction set, I merged these to a Pandas data frame with the customer IDs so that I could submit the predictions to Kaggle. Kaggle scored my predictions using MAP@7.

I faced several challenges during this stage. XGBoost is not part of Scikit learns stock library of ML algorithms. I first learned how to create an XGBoost environment within Python 2.7. Other complications included learning how to use the XGB scikit learn wrapper API so that I would have access to grid search functionality. Originally, I passed the data to an XGBoost DMatrix and fit the function with XGBoost, not the XGBoostClassifier. I learned that I had to use the scikit-learn API (XGBoostClassifier) if I wanted to use Grid Search—it took me awhile to understand why the code would not run with Grid Search.

Rather than baby sit the laptop during the Refinement stage, waiting for the model to finish and then rerun the model with different hyper parameters, I chose to learn how to use Grid Search with XGBoost. Considerable time was spent studying the different hyper parameters for XGBoost so that I would understand which hyper parameters to modify and in what direction to change the values. Tuning hyper parameters will be discussed in the Refinement section.

The biggest challenge was getting the data preprocessed before any ML algorithms were even considered. Munging the data required that I write code that could subset, transform and visualize the data. The most challenging was the pre-processing stage to get the features and Y variable in good shape before analysis. Once the data munging was complete, switching from XGBoost to Logistic Regression was straightforward. Since Logistic Regression Classifier is part of the scikit-learn library, all the documentation and examples are detailed and very helpful. Logistic Regression was very easy in comparison. I have used Logistic Regression on other problems so I did not go through the same learning curve experienced with XGBoost.

Another challenge was using the SVM classifier. I ran a Support Vector Classifier too, but it used copious amounts of RAM—it locked up my laptop several times. Since the SVC algorithm did not converge (a warning issued) and took vast amounts of RAM, I concentrated on improving XGBoost and Logistic Regression via Grid Search in the Refinement stage.

Refinement

For model refinement, I passed different values to multiple hyper parameters for XGBoost and Logistic Regression within Grid Search. Grid Search allows for an exhaustive, brute force search over several combinations of each hyper parameters value located in the Grid Search dictionary.

Per Raschka, hyper parameters can be adjusted to optimize model performance by the data scientist (i.e. tree depth). Hyper parameters contrast with model parameters as model parameters are learned from the training data (i.e. weights in logistic regression). Grid search allowed me to evaluate model performance on a variety of values for specific hyper parameters. Since this is an exhaustive brute-force search, a range of values were used for the hyper parameters I felt were most likely to contribute to an improved model. This was part intuition and part research via online resources. No absolute rule exists for this tuning process but heuristics are helpful (Raschka, 2015).

I do not suggest my models will be optimized globally, as I could not specify all possible combinations of hyper parameter values; however, the tuning process allowed me improve model performance minimally at least. Table V shows the different combinations I used for Grid Search—starting with Naïve Bayes as the benchmark, then showing the best XGBoost and Logistic Regression models sorted by best log-loss score. The models that scored the best log-loss for each model and for the Features Used column were submitted to Kaggle—these have MAP@7 score highlighted in peach. The final model and best overall MAP@7 is highlighted in blue.

Table V: Scoring Results During Grid Search

Features Used	Algorithm	Default	Grid Search	Log-Loss	MAP@7
Top 25	Naive Bayes	N/A	N/A	N/A	0.01025
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 100, 'solver': 'lbfgs', 'fit_intercept': False}	-1.5199	0.02537
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 150, 'solver': 'lbfgs', 'fit_intercept': True}	-1.5202	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2',	params: {'C': 0.75, 'max_iter': 150, 'solver': 'lbfgs',	-1.5202	

		random_state=43, solver='lbfgs', tol=0.0001, verbose=0	'fit_intercept': False}		
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 100, 'solver': 'newton-cg', 'fit_intercept': False}	-1.5202	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 150, 'solver': 'newton-cg', 'fit_intercept': False}	-1.5202	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 200, 'solver': 'newton-cg', 'fit_intercept': False}	-1.5202	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 200, 'solver': 'lbfgs', 'fit_intercept': False}	-1.5203	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 100, 'solver': 'newton-cg', 'fit_intercept': True}	-1.5206	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 150, 'solver': 'newton-cg', 'fit_intercept': True}	-1.5206	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 200, 'solver': 'newton-cg', 'fit_intercept': True}	-1.5206	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100,	params: {'C': 0.75, 'max_iter': 200, 'solver': 'lbfgs',	-1.5206	

		multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	'fit_intercept': True}		
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 100, 'solver': 'lbfgs', 'fit_intercept': True}	-1.5207	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 150, 'solver': 'lbfgs', 'fit_intercept': True}	-1.5211	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 100, 'solver': 'newton-cg', 'fit_intercept': False}	-1.5215	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 150, 'solver': 'newton-cg', 'fit_intercept': False}	-1.5215	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 150, 'solver': 'lbfgs', 'fit_intercept': False}	-1.5215	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 200, 'solver': 'newton-cg', 'fit_intercept': False}	-1.5215	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 200, 'solver': 'lbfgs', 'fit_intercept': False}	-1.5215	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1,	params: {'C': 1, 'max_iter': 200, 'solver': 'lbfgs',	-1.5215	

		max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	'fit_intercept': True}		
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 100, 'solver': 'newton-cg', 'fit_intercept': True}	-1.5218	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 150, 'solver': 'newton-cg', 'fit_intercept': True}	-1.5218	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 200, 'solver': 'newton-cg', 'fit_intercept': True}	-1.5218	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 100, 'solver': 'lbfgs', 'fit_intercept': False}	-1.5221	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1.25, 'max_iter': 150, 'solver': 'lbfgs', 'fit_intercept': False}	-1.5222	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1.25, 'max_iter': 200, 'solver': 'lbfgs', 'fit_intercept': False	-1.5225	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1.25, 'max_iter': 200, 'solver': 'lbfgs', 'fit_intercept': True}	-1.5226	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False,	params: {'C': 1.25, 'max_iter': 100,	-1.5226	

		fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	'solver': 'newton-cg', 'fit_intercept': False}		
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1.25, 'max_iter': 150, 'solver': 'newton-cg', 'fit_intercept': False}	-1.5226	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1.25, 'max_iter': 200, 'solver': 'newton-cg', 'fit_intercept': False}	-1.5226	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1.25, 'max_iter': 100, 'solver': 'newton-cg', 'fit_intercept': True}	-1.5230	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1.25, 'max_iter': 150, 'solver': 'newton-cg', 'fit_intercept': True}	-1.5230	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1.25, 'max_iter': 150, 'solver': 'lbfgs', 'fit_intercept': True}	-1.5230	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1.25, 'max_iter': 100, 'solver': 'lbfgs', 'fit_intercept': False}	-1.5233	
All Features	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1.25, 'max_iter': 100, 'solver': 'lbfgs', 'fit_intercept': True}	-1.5251	

Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 150, 'solver': 'lbfgs', 'fit_intercept': False},	-1.5266	0.02622
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 200, 'solver': 'lbfgs', 'fit_intercept': False},	-1.5266	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 100, 'solver': 'lbfgs', 'fit_intercept': False},	-1.5266	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 150, 'solver': 'newton-cg', 'fit_intercept': False},	-1.5266	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 200, 'solver': 'newton-cg', 'fit_intercept': False},	-1.5266	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 100, 'solver': 'newton-cg', 'fit_intercept': True},	-1.5269	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 150, 'solver': 'newton-cg', 'fit_intercept': True},	-1.5269	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 200, 'solver': 'newton-cg', 'fit_intercept': True},	-1.5269	

Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 200, 'solver': 'lbfgs', 'fit_intercept': True},	-1.5269	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 150, 'solver': 'lbfgs', 'fit_intercept': True},	-1.5269	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0		-1.5269	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 1, 'max_iter': 100, 'solver': 'lbfgs', 'fit_intercept': True},	-1.5269	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 100, 'solver': 'lbfgs', 'fit_intercept': True},	-1.5270	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 150, 'solver': 'lbfgs', 'fit_intercept': True},	-1.5271	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 100, 'solver': 'lbfgs', 'fit_intercept': False},	-1.5271	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 150, 'solver': 'lbfgs', 'fit_intercept': False},	-1.5271	

Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 200, 'solver': 'lbfgs', 'fit_intercept': False},	-1.5271	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 200, 'solver': 'lbfgs', 'fit_intercept': True},	-1.5271	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 100, 'solver': 'newton-cg', 'fit_intercept': True},	-1.5271	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 150, 'solver': 'newton-cg', 'fit_intercept': True},	-1.5271	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 200, 'solver': 'newton-cg', 'fit_intercept': True}	-1.5271	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 100, 'solver': 'newton-cg', 'fit_intercept': False},	-1.5271	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 150, 'solver': 'newton-cg', 'fit_intercept': False},	-1.5271	
Top 25	Logistic R.	C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=100, multi_class='ovr',penalty='l2', random_state=43, solver='lbfgs', tol=0.0001, verbose=0	params: {'C': 0.75, 'max_iter': 200, 'solver': 'newton-cg', 'fit_intercept': False},	-1.5271	

All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.05, 'colsample_bytree': 1.0, 'max_depth': 3}	-1.6260	0.02618
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 3}	-1.6403	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.05, 'colsample_bytree': 1.0, 'max_depth': 3}	-1.6408	0.02617
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.05, 'colsample_bytree': 1, 'max_depth': 3}	-1.6469	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 3}	-1.6551	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 3}	-1.6586	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 3}	-1.6601	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10,	params: {'n_estimators':	-1.6686	

		max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	100, 'learning_rate': 0.075, 'colsample_bytree': 0.75, 'max_depth': 3},		
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.05, 'colsample_bytree': 1, 'max_depth': 3}	-1.6687	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.05, 'colsample_bytree': 1, 'max_depth': 3}	-1.6780	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 3}	-1.6801	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 3}	-1.6826	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 3}	-1.6864	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.075, 'colsample_bytree': 0.75, 'max_depth': 3}	-1.6924	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1,	params: {'n_estimators': 175, 'learning_rate': 0.05,	-1.7053	

		objective='multi:softprob', seed=43, silent=True,subsample=1	'colsample_bytree': 1, 'max_depth': 3}		
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 3}	-1.7064	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 3}	-1.7085	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 3}	-1.7093	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 3}	-1.7136	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 3}	-1.7311	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 3}	-1.7408	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.075, 'colsample_bytree': 0.75, 'max_depth': 3}	-1.7460	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10,	params: {'n_estimators':	-1.7503	

		max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	175, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 3}		
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.075, 'colsample_bytree': 0.75, 'max_depth': 3}	-1.7564	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 3}	-1.7633	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 3}	-1.7666	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 3}	-1.7872	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.075, 'colsample_bytree': 0.75, 'max_depth': 3}	-1.7874	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 3}	-1.7999	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 3}	-1.8017	

All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 3}	-1.8052	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 3}	-1.8075	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 6}	-1.8173	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 6}	-1.8416	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 3}	-1.8449	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 3}	-1.8494	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.05, 'colsample_bytree': 1, 'max_depth': 6}	-1.8576	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1,	params: {'n_estimators': 140, 'learning_rate': 0.05,	-1.8577	

		objective='multi:softprob', seed=43, silent=True,subsample=1	'colsample_bytree': 0.75, 'max_depth': 6}		
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.075, 'colsample_bytree': 0.75, 'max_depth': 6}	-1.8728	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.05, 'colsample_bytree': 1, 'max_depth': 6}	-1.8822	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.05, 'colsample_bytree': 1, 'max_depth': 6}	-1.8896	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 6}	-1.8899	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 6}	-1.8989	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 9}	-1.9024	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.075, 'colsample_bytree':	-1.9163	

			0.75, 'max_depth': 6}		
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.05, 'colsample_bytree': 1, 'max_depth': 6}	-1.9172	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 6}	-1.9231	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.05, 'colsample_bytree': 1, 'max_depth': 6}	-1.9304	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.075, 'colsample_bytree': 0.75, 'max_depth': 6}	-1.9307	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 6}	-1.9356	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 9}	-1.9406	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 6}	-1.9421	

All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 6}	-1.9449	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 6}	-1.9468	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 9}	-1.9519	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.075, 'colsample_bytree': 0.75, 'max_depth': 6}	-1.9680	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.075, 'colsample_bytree': 0.75, 'max_depth': 9}	-1.9694	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 6}	-1.9774	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 6}	-1.9794	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1,	params: {'n_estimators': 100, 'learning_rate':	-1.9819	

		n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	0.05, 'colsample_bytree': 1, 'max_depth': 9}		
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.075, 'colsample_bytree': 0.75, 'max_depth': 6}	-1.9859	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 6}	-1.9866	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 6}	-1.9880	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 6}	-1.9956	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 9}	-1.9959	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 9}	-1.9992	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 6}	-2.0031	

All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 6}	-2.0166	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.05, 'colsample_bytree': 1, 'max_depth': 9	-2.0189	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.075, 'colsample_bytree': 0.75, 'max_depth': 6}	-2.0304	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.05, 'colsample_bytree': 1, 'max_depth': 9}	-2.0310	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 9}	-2.0362	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 6}	-2.0398	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.075, 'colsample_bytree': 0.75, 'max_depth': 9}	-2.0406	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1,	params: {'n_estimators': 175, 'learning_rate': 0.1,	-2.0427	

		objective='multi:softprob', seed=43, silent=True,subsample=1	'colsample_bytree': 1, 'max_depth': 6}		
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 6}	-2.0448	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 9}	-2.0470	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.05, 'colsample_bytree': 0.75, 'max_depth': 9}	-2.0494	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 6}	-2.0544	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.05, 'colsample_bytree': 1, 'max_depth': 9}	-2.0659	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 9}	-2.0854	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.075, 'colsample_bytree': 0.75, 'max_depth': 9}	-2.0876	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10,	params: {'n_estimators':	-2.0882	

		max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	140, 'learning_rate': 0.05, 'colsample_bytree': 1, 'max_depth': 9}		
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 6}	-2.0884	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 9}	-2.0964	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 9}	-2.0984	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 6}	-2.1023	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 9}	-2.1060	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 9}	-2.1061	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.05, 'colsample_bytree': 1, 'max_depth': 9}	-2.1313	

All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 9}	-2.1321	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 9}	-2.1436	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 9}	-2.1528	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 100, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 9}	-2.1632	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 9}	-2.1724	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 9}	-2.1828	
All Features	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 9	-2.1947	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.075, 'colsample_bytree': 1, 'max_depth': 9	-2.2228	

Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 140, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 9}	-2.2380	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.1, 'colsample_bytree': 0.75, 'max_depth': 9}	-2.2387	
Top 25	XGBoost	base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.10, max_delta_step=0, max_depth=3, min_child_weight=1, n_estimators=100,nthread=-1, objective='multi:softprob', seed=43, silent=True,subsample=1	params: {'n_estimators': 175, 'learning_rate': 0.1, 'colsample_bytree': 1, 'max_depth': 9	-2.2897	

IV. Results

Model Evaluation and Validation

The final model that I chose is the Logistic Regression model trained on only 25 features and the hyper parameters shown below (Table VI). Tuning this model allowed for the best MAP@7 per Kaggle—it nearly had the lowest-log loss too. Logistic Regression seems to identify the structure of the data slightly better than XGBoost because the top 60 models were Logistic Regression (based on log-loss). It should be noted that the difference in MAP@7 score for the best Logistic Regression and XGBoost models was minor—only 0.00004.

Logistic Regression scored best when using only the top 25 features, whereas XGBoost did better with all features. Like before, the differences are very minor but the results do confirm what I have read about XGBoost's ability to "ignore" extraneous features and focus only on features that were useful.

For Logistic Regression, I used Grid Search to explore the C, max iteration, solver and fit_intercept hyper parameters. The best Logistic Regression model identified the following combination of hyper parameters during Grid Search: "C" = 1, "max_iter" = 150, "solver" = lbfgs" and "fit_intercept" = False. The two that were not defaults are max_iter and fit intercept. The hyper parameters have interaction effects, but it seems that allowing a higher maximum iteration did not cause the model to over fit the data. It should be noted that the best score on the MAP@7 was not the absolute lowest log-loss, although log-loss was an excellent proxy for it.

For XGBoost, I used Grid Search to explore the n_estimators, learning_rate, colsample_bytree and max_depth. The best XGBoost model identified the following combination of hyper parameters during Grid Search: "n_estimators" = 100, "learning_rate" = 0.05, "colsample_bytree" = 0.75 and "max_depth" =3. The two parameters that are not defaults are learning_rate and colsample_bytree. I found it interesting that lowering the learning rate did not result in a jump in the n_estimators or depth as I had expected. Since the max_depth of "3"

was consistent for many of the top XGBoost models, it appears that allowing the trees to grow more than "3" only allows the model to suffer from overfitting.

Grid Search was useful for tuning the hyper parameters but the combination of possible values and the interaction between them makes the process very time consuming. I would like to learn about using Bayesian Analysis to use a more statistical approach to tuning the hyper parameters—Grid Search is a brute-force method. This model's results can be trusted because stratified KFold cross validation ensured that the Training data was split into 3 train and evaluation subsets, which had proportional representation of the dependent variable. This matters because machine learning algorithms can over fit the data and must be addressed in any problem.

Table VI: Best Model Logistic Regression (Top 25 features)

Hyper parameters	Description
penalty	l2'
dual	False
C	1
fit_intercept	False
intercept_scaling	1
class_weight	None
max_iter	150
random_state	43
solver	lbfgs
tol	0.0001
multi_class	ovr'
verbose	0

Justification

Both the Logistic Regression and XGBoost models proved better than the benchmark model for the scoring metric MAP@7 (Table V). Choosing Logistic Regression for the best model was because the model had the highest MAP@7 score and the top 60 models were Logistic Regression when ranked by log-loss. The down sampling technique, training on May & June 2015 data proved a smart way to work with limited computing resources, while still capturing the seasonal nature of sales data.

The final model and solution could be used to help Santander market to existing and potential customers based on historical data. Customers would appreciate the bank anticipating their future product needs and would be less inclined to tune-out marketing materials if they are relevant. From this standpoint, I am happy with the model and the performance; however, the problem is not solved as the Public Leaderboard shows that top score is 0.0309636 and my top score is 0. 0262154.

This is a global competition and has a $60,000 prize for the winner. I expected the top scores to represent contestants with significant expertise in machine learning. Although I'm not in the top 1% of contestants yet, I feel the model has improved and the results show the advantages of machine learning. The top score gives me something to strive for as I continue to learn new techniques and strategies. I have summarized the best models for each algorithm below:

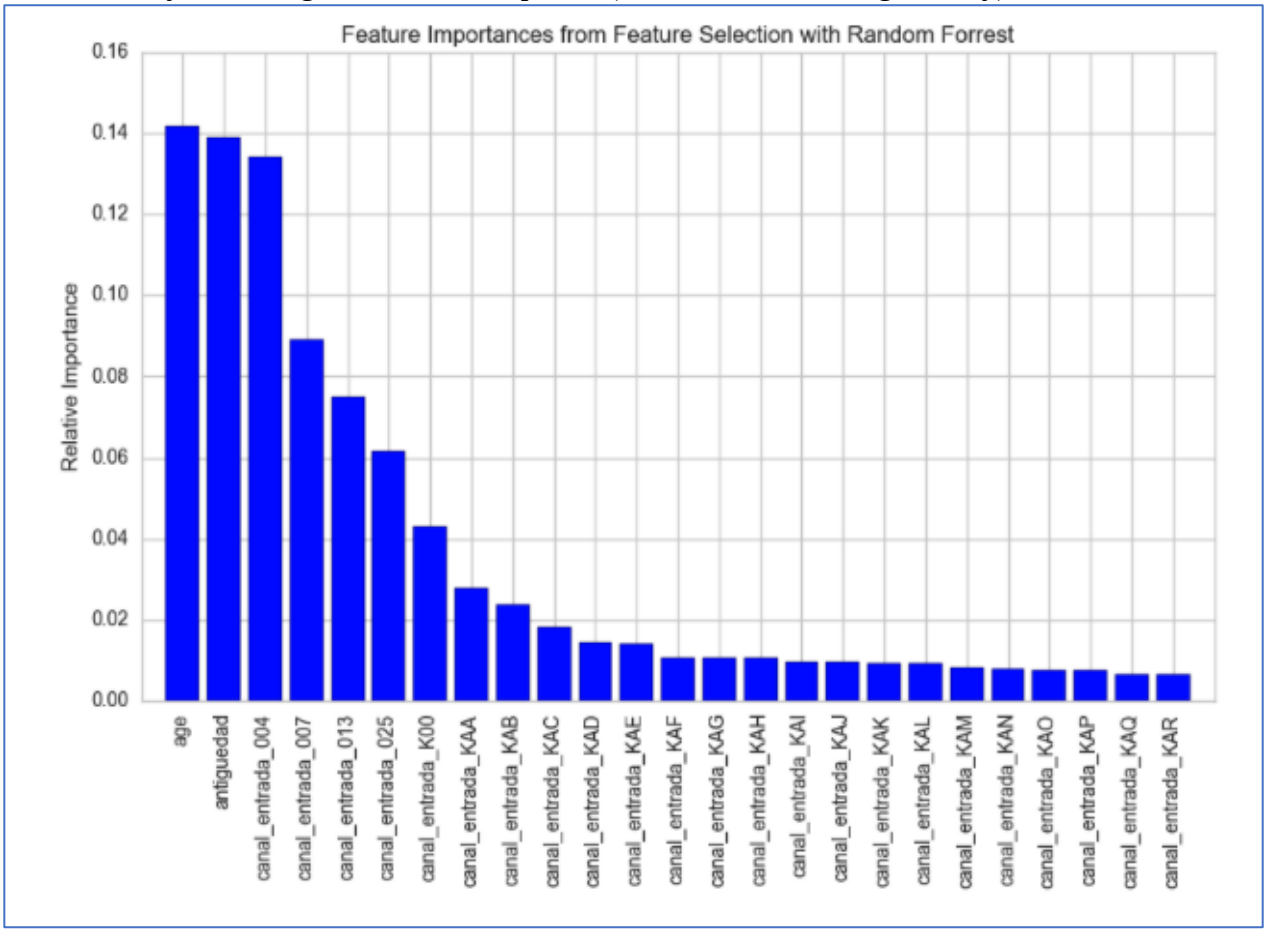
Table VII: Benchmark vs. XGBoost Final

Best Algorithm	MAP@7
Logistic Regression	0.0262154
XGBoost Best Model	0.0261797
Naive Bayes Benchmark	0.0102542

V. Conclusion

Visualization

With parsimony as a guide, I wanted to show a final visualization on the relative importance of the top 25 features from the feature extraction process. The final model used only the top 25 features. The top three features contribute more than 10%. Santander could help themselves by collecting additional data points (i.e. social networking activity).



Reflection

Looking back on this project, I realize how much I have learned. The first challenge was determining how I would recommend products for Santander's customers. I researched product recommendation engines, but the information I found indicated that there were no libraries in Scikit-learn to work from. "Crab" is still a work in progress and is the best I could find in terms of a product recommendation engine library. I settled on making the project a multi-classification problem by collapsing the products added in June 2015 that were not "owned" in May 2015. Fortunately, this dropped the rows from ~13M to ~45K. This matters as my RAM was unable to handle the Support Vector Classifier and I am not sure XGBoost or Logistic regression could have handled 13M rows.

Preprocessing the data was needed as the Kaggle files were not ready to be passed to the ML algorithms. This included recoding duplicate categorical features, imputing Null numeric values, recoding Age outliers, recoding Null categorical values to "Unknown Value", scaling numeric features, and splitting the Training Data set into a Train and Evaluate set. I used One-Hot encoding to create Dummy Variables for the categorical features and then made sure the columns matched exactly because some class labels were present in the Train Data Set but not the Kaggle Prediction set. I used Random Forest to identify the 25 most important features that represented ~90% of the importance of the data. This allowed me to drop 330 mostly sparse feature columns. None of the dropped features had a relative importance of more than 0.639% (approximately 1/2 a percent).

I used Naive Bayes as a quick benchmark model for making predictions as I have read that it is a speedy and relatively strong first algorithm to use. I used XGBoost because of empirical evidence that the algorithm is both accurate and fast. I found it was both. Logistic Regression was surprisingly good and very quick. Both models were chosen based on the problem at hand--supervised classification problems that required multi-class probability objectives. I was under the impression that XGBoost would significantly surpass Logistic Regression, but both models performed similar in both scoring metrics, log-loss and MAP@7. Logistic Regression narrowly won the title of “best model” as a tuned model had the highest MAP@7 score.

I found that the data munging was challenging and tedious. The final model does not fit my expectations because I thought that XGBoost would have performed much better than Logistic Regression—especially after tuning hyper parameters. However, both Logistic Regression and XGBoost scored significantly better than the Naïve Bayes benchmark model. This is in line with my hypothesis. The best performance was Logistic Regression with the top 25 features and the hyper parameters shown in Table VI. I read that XGBoost is robust to extraneous features as it will simply not use them when building a model. This is consistent with my experience and explains why it did better when I let it learn from all the columns, after all XGBoost is designed to excel with many weak-learners during the boosting process.

Improvement

From reviewing the Kaggle Leaderboard, I still have a long way to go to make the top score. This is a good and bad thing. It is good because I am still learning and will attempt to continue to improve on my score. I'm ranked in the top ~30% of submissions (435 of 1410 competitors). I believe further research can be done on Time Series data, such as lag features.

Additionally, I would like to determine if running the model on more than two months of information could improve predictions. I am curious if more information helps or hurts the model performance as some months may be helpful in predictions, while other months may only

add noise. I would like to ensemble several models (i.e. combine XGBoost and Logistic Regression) as I have read that this can improve model performance. Since different algorithms model different mathematical structures, an ensemble can create a more generalized final model that does well on predicting new data. Finally, I will research Python's pipeline functionality as I believe this will allow for more efficient code when working with several models for the ensemble.

Works Cited:

Brownlee, Jason. “XGBoost with Python”. N.p. 2016. Print.

Kaggle. “Mean Average Precision” Kaggle.com. Web. N.d. Available at: https://www.kaggle.com/wiki/MeanAveragePrecision

Kaggle. “Santander Product Recommendation” Kaggle.com. Web. 26 Oct. 2016. Available at: https://www.kaggle.com/c/santander-product-recommendation/details/evaluation

H. Liu, X. Yin and J. Han, "An Efficient Multi-Relational Naive Bayesian Classifier Based on Semantic Relationship Graph", ACM MRDM, 2005.

Hastie, Trevor, Trevor Hastie, Robert Tibshirani, and J H. Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. New York: Springer, 2001. Print.

McKinney, Wes. *Python for Data Analysis.* California: O’Reilly, 2013. Print

Raschka, Sebastian. *Python Machine Learning.* UK: Packt Publishing, 2015. Print.

TIBCO Software Inc. “Predictive Analytics for Financial Services Firms: Forecasting the Future.” Tibco.com. Web. 27 Jan. 2014. Available at: http://www.tibco.com/blog/2014/01/27/predictive-analytics-for-financial-services-firms-forecasting-the-future/

The Financial Brand. “10 Branch Banking Innovation Strategies for 2016.” thefinancialbrand.com. Web. 24 Nov. 2015 Available at: https://thefinancialbrand.com/55561/branch-banking-innovation-strategies/

Z. Zygmunt. “What you wanted to know about Mean Average Precisionn” fastml.com. Web. 08 Aug. 2012. Available at: http://fastml.com/what-you-wanted-to-know-about-mean-average-precision/