README.md

Understanding Customer Conversion with Snowplow Web Event Tracking

Benjamin S. Knight, January 27th 2017

Project Overview

Here I apply machine learning techniques to Snowplow web event data to infer whether trial account holders will become paying customers based on their history of visiting the marketing site. By predicting which trial account holders have the greatest likelihood of adding a credit card and converting to paying customers, we can more efficiently deploy scarce Sales Department resources.

Snowplow is a web event tracker capable of handling tens of millions of events per day. The Snowplow data contains far more detail than the MSNBC.com Anonymous Web Data Set hosted by the University of California, Irvine's Machine Learning Repository. At the same time, we do not have access to demographic data as was the case with the Event Recommendation Engine Challenge hosted by Kaggle. Given the origin of the data, there is no industry-standard benchmark for model performance. Rather, assessing baseline feasibility is a key objective of this project.

Problem Statement

To what extent can we infer a visitor's likelihood of becoming a paying customer based upon that visitor's activity history on the company marketing site? We are essentially confronted with a binary classification problem. Will the trial account in question add a credit card (cc_date_added IS NOT NULL 'yes'/'no')? This labeling information is contained in the 'cc' column within the file 'munged df.csv.'

There is no clear precedent for how effective a model our analysis may ultimately yield, and so a key component of the project is ultimately assessing feasibility of using visitor history on the marketing site to predict conversion. Regarding the most appropriate model, there is no way of knowing which family of algorithms will prove to be most effective in predicting customer conversion. With this in mind, we adopt an "all of the above" approach - applying all algorithms that are feasible given the nature of the classification problem and the structure of the data (see the section entitled 'Algorithms and Techniques' for additional details). That being said, certain families of algorithms are more promising than others (at least initially). Based on findings from Wainer (2016), we predict that a Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel is most likely to yield the best fit.

Metrics

As we discuss later, the data is highly imbalanced (successful customer conversions average 6%). Thus, we are effectively searching a haystack for rare, but exceeedingly valuable needles. In more technical terms, we want to maximize recall as our first priority. Selecting the model that maximizes precision is a subsequent priority. To this end, our primary metric is the F2 score shown below.

$$F_\beta = \frac{(1+\beta^2) * \text{True Positive}}{(1+\beta^2) * \text{True Positive} + \beta^2 * \text{False Negative} + \text{False Positive}}$$

$$\text{where } \beta = 2$$

The F2 score is derived from the F1 score by setting the weight of the \beta parameter to 2, effectively increasing the penalty for false negatives. While the F2 score is the arbiter for ultimate model selection, we also use precision-recall curves to clarify model performance. We have opted for precision-recall curves as opposed to the more conventional receiver operating characteristic (ROC) curve due to the highly imbalanced nature of the data (Saito, 2016).

Data Preprocessing

The raw Snowplow data available is approximately 15 gigabytes spanning over 300 variables and tens of millions of events from November 2015 to January 2017. When we omit fields that are not in active use, are redundant, contain personal identifiable information (P.I.I.), or which cannot have any conceivable bearing on customer conversion, then we are left with 14.6 million events spread across 22 variables.

Table 1: Selected Snowplow Variables Prior to Preprocessing

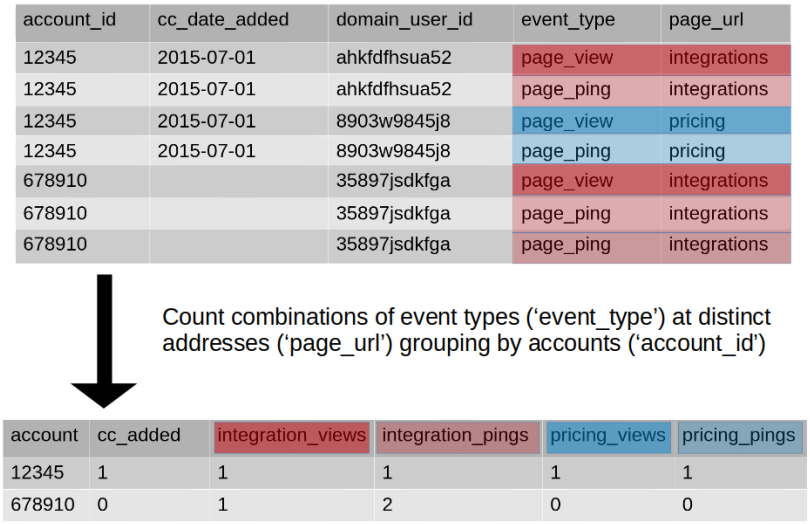
Snowplow Variable Name	Snowplow Variable Description
event_id	The unique Snowplow event identifier
account_id	The account number if an account is associated with the domain userid
reg_date	The date an account was registered
cc_date_added	The date a credit card was added
collector_tstamp	The timestamp (in UTC) when the Snowplow collector first recorded the event
domain_userid	This corresponds to a Snowplow cookie and will tend to correspond to a single internet device
domain_sessionidx	The number of sessions to date that the domain userid has been tracked
domain_sessionid	The unique identifier for the Snowplow cookie/session
event_name	The type of event recorded
geo_country	The ISO 3166-1 code for the country that the visitor's IP address is located
geo_region_name	The ISO-3166-2 code for country region that the visitor's IP address is in
geo_city	The city the visitor's IP address is in
page_url	The page URL
page_referrer	The URL of the referrer (previous page)
mkt_medium	The type of traffic source (e.g. 'cpc', 'affiliate', 'organic', 'social')
mkt_source	The company / website where the traffic came from (e.g. 'Google', 'Facebook')
se_category	The event type
se_action	The action performed / event name (e.g. 'add-to-basket', 'play-video')
br_name	The name of the visitor's browser
os_name	The name of the vistor's operating system
os_timezone	The client's operating system timezone
dvce_ismobile	Is the device mobile? (1 = 'yes')

I use the phrase 'variable' as opposed to 'feature', since this dataset will need to undergo substantial transformation before we can employ any supervised learning technique. Each row has an 'event_id' along with an 'event_name' and a 'page url.' The event id is the row's unique identifier, the event name is the type of event, and the page url is the URL within the marketing site where the event took place.

The distillation of the raw data into a transformed feature set with labels is handled by the iPython notebook 'Notebook 1 - Data Munging.' In transforming the data, we will need to create features by creating combinations of event types and distinct URLs, and counting the number of occurrences while grouping on accounts. For instance, if '.../pay-ment plan.com' is a frequent page url, then the number of page views on payment plan.com would be one feature, the number of page pings would be another, as would the number of web forms submitted, and so forth. Given that there are six distinct event types and dozens of URLs within the marketing site, then the feature space quickly expands to encompass hundreds of features.

This feature space will only widen as we add additional variables to the mix including geo region, number of visitors per account, and so forth.

Figure 1: Management of Original Categorical Variables into Features



With the raw data transformed, our observations are no longer individual events but indivual accounts spanning the period November 2015 to January 2017. Our data set has 16,607 accounts and 581 features. 290 of these represent counts of various combinations of web events and URLs grouped by account. Next there are two aggregated features - the total number of distinct cookies associated with the account, and the sum total of all Internet sessions linked to that account.There are also 151 features that represent counts of page view events linked to IP addresses within a certain country (e.g. a count of page views from China, a count of page views from France, and so forth).

46 of the features represent counts of page views coming from a specific marketing medium ('mkt medium'). Recall that 'mkt medium' is the type of traffic. Examples include 'partner link,' 'adroll,' or 'appstore.' The 'mkt medium' subset of features is followed by 86 features that correspond to Snowplow's 'mkt source' field. 'mkt source' designates the company / website where the traffic came from. Examples from this subset of the feature space include counts of page views from Google.com ('mkt source google com') and Squarespace ('mkt source square'). There are two additional feature:'mobile pageviews' and 'non-mobile pageviews' that represent counts of page views that took place on mobile versus non mobile devices. I have also included an additional feature derived from these two - the share of page views that took place on a mobile device.

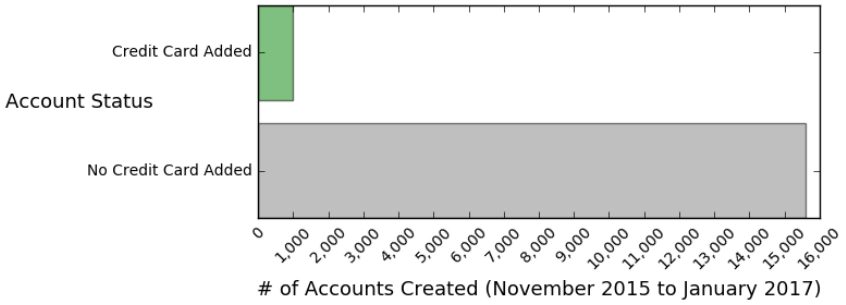
With the aggregations completed, we then take the transformed data and drop all features that are uniformly zero for all observations. Finally, we scale the features using robust scaling.

It bears noting that 'br_name' (the name of the visitor's browser), 'os_name' (the name of the vistor's operating system), and 'os_timezone' (the client's operating system timezone) were not included in the ultimate version of the transformed data. The transformed variables of 'br_name' and 'os_name' were used initially. However, their incorporation added +40 features to the already expansive feature space resulting in inferior performance and so were subsequently dropped.

Data Exploration

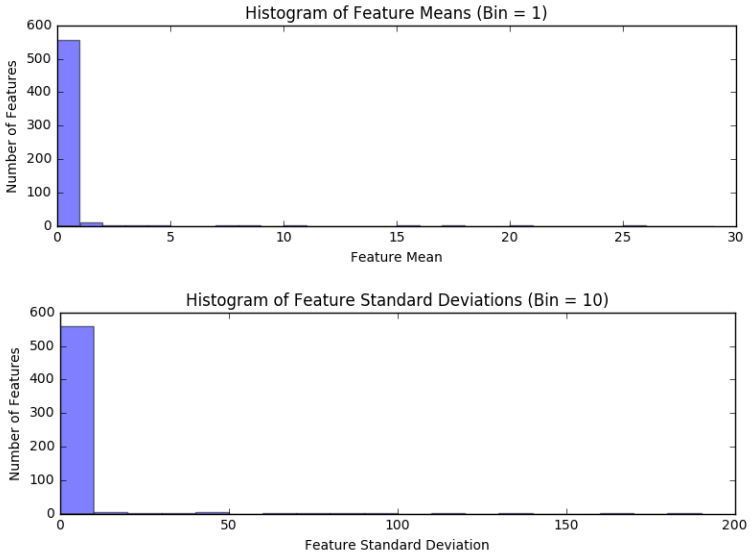
Exploring the transformed data, two features quickly become apparent. First, we can see that the data is highly imbalanced. Only approximately 6% of the labeled accounts show a succesful conversion to paying customer.

Figure 2: Summary Statistics - Distribution of Labels (16,607 Observations)



The second feature of note is that in addition to our feature space being wide with over 500 features, the features themselves are fairly sparse as the histograms below make clear. This is to be expected. The Snowpow features are highly specific. Examples include counts of certain types of events localized within Bangladesh, or the number page views associated with a bit of on-line content that was only made briefly available. As a result, the majority of features are extremely sparse.

Figure 3: Summary Statistics - Means and Standard Deviations of Sparse Feature Space (581 Features)



Benchmark

How do we know if our ultimate model is any good? To establish a baseline of model performance, I implement a K-Nearest Neighbors model within the iPython notebook 'Notebook 3 - KNN (Baseline).' In the same manner as the subsequent model selection, I allocate 90% of the data for training (14,946 observations) and 10% for model testing (1,661 observations). I use the model's default setting of 5 neighbors. I run the resulting model on the test data using 100-fold cross validation. Averaging the 100 resultant F2 scores, we thus establish a benchmark model performance of F2 = 0.04.

Algorithms and Techniques

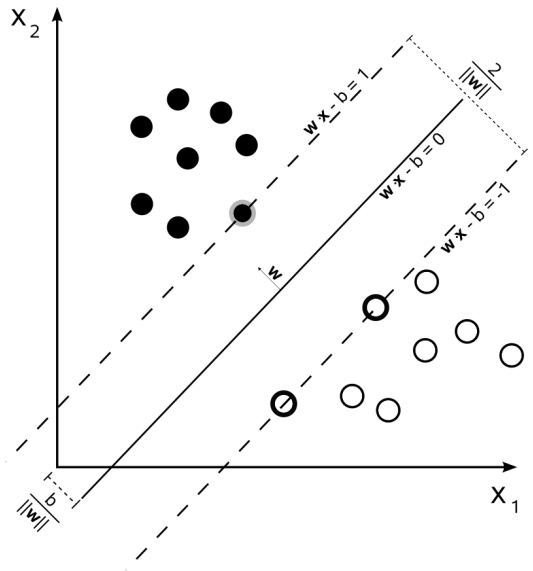
We start our analysis with establishing a benchmark using K-Nearest Neighbors (KNN) before moving on to more sophisticated algorithms. The KNN classifer works by selecting the target observation's n closest neighbors. The target observation is then classifed as being a member of the same class as the majority class within the n-sample. We use Sci-Kit Learn's KNeighborsClassifier which defaults to n = 5. Regarding what qualifies as a 'neighbor,' the KNeighborsClassifier uses the Minkowski distance, which at its default setttings is effectively the Euclidean distance.

Like KNN, logistic regression is computationally inexpensive - a definite strength given the size of the data set (n = 16,607). In addition, logistic regression is uniquely well-suited to the binary nature of the outcome variable. Sci-Kit Learn's LogisticRegression classifier works through maximum likelihood estimation. Through many iterations, the algorithm determines what sequence of weights will, when applied to our 581 features, maximize the likelihood that the pattern of successes and failures seen in the training set will emerge. An added strength of the classifier (of potential interest to future, more in-depth analysis) is that it generates the 'coef_' attribute - a vector of coefficients that can indicate which features are the most useful predictors of the outcome variable.

The second and third algorithms selected are Support Vector Machines (SVM) - the first with a Radial Basis Function (RBF) kernel and the other using a linear function. SVM works by placing multiple hyperplane (support vectors) through the data. The set of hyperplanes that maximizes the distance between the classes is then selected, with the center of the space delimited by the hyperplanes becoming the threshold for classification.

Figure 4: A System of Linear Support Vector Machines Finding the Optimal Separation Between Two Classes

Source: Wikipedia



For the SVM + RBF kernel model, we use Sci-Kit Learn's SVM functionality. The RBF kernel works by creating an additional dimension of information derived fom the squared Euclidean distance of one point vis-a-vis all of the other points. Models using SVM with RBF kernels tend to be perform well with binary data (Wainer, 2016), but are far less expensive than random forest models.

A strength of RBF kernels is that they can accomodate data that is not linearly separable. However, if our data is already linearly separable, then such an approach is not just expensive - it may lead to over-fitting (Webb, 2002, p.138). For this reason, we also use a linear SVM. Linear SVM assumes that the data is linearly separable, and as a result of this assumption, is relatively inexpensive. Linear SVM is also well-suited for the high dimensionality of our data set (581 features).

Implementation

Quantifying Success

One of the edifying aspects of this project was the refinement of the success metric. Given that the task is at its core, a binary classification problem, the initial metric intended was the area under the curve (AUC) as delimited by the receiver operating characteristic ROC. The ROC AUC nicely captures the models' efficacy in terms of the rate of true postives versus false positives. However, the data is highly imbalanced, with only 6% of the customers sucessfully converting to paying accounts. Thus, the challange is less about maximizing the number of true positives relative to the number of false postives, but rather maximizing the share of customer conversions sucessfully capatured while at the same time ensuring the greatest possible precision.

In more technical terms, recall - not precision - became the metric of highest priority. With this in mind, we discarded the ROC curve in favor of a precision-recall curve. A precision-recall curve plots the average precision (the Y-axis) over a given threshold of recall (the X-axis). From this, we could more meaningfully gauge the trade-off between maximizing recall versus precision. However, this new metric also proved to be inadequate. Speaking to the Sales Department prompted us to narrow the success metric even further. Capturing the majority of customer conversion events was the highest priority, and if a member of the Sales Department had to make due with a less accurate model, then so be it. Thus, seeing things from the end user perspective, we settled on the final success metric - the F2 score.

Transforming the Data

With the success metric in hand, the project begins in earnest with transforming the raw, categorical event data into quantitative account-level data. This task is handled by 'Notebook 1 - Data Munging.' After reading in the necessary packages and the data itself, we re-cast the most important fields in order to support the necessary filtering and grouping functions.

Creating a trial account is a prerequisite for becoming a paying customer, and so we start by filtering out visitors that have never created a trial account. We then drop those events that occured after that account's conversion event (cc_date_added). In this way, we confirm that marketing site activity is a predictor of conversion to paying customer and not vice versa. After filtering, our data set is considerably smaller - just shy of two million events.

The next step of data preparation is more labor-intensive. When a visitor visits the site for the first time (e.g. the pricing page), then the URL tracked by Snowplow will read '*https://www.company-name.com/pricing/*.' However, if that visitor is coming via an advertisement - say Google Adwords - then a Urchin Traffic Monitor (UTM) parameter will come into play, inserting itself into the URL. Thus, instead of '*https://www.company-name.com/pricing/*,' Snowplow will see '*https://www.company-name.com/pricing/utm_source=google*.' To further complicate matters, when a trial account is created a subdomain is added to the URL. For example, if Acme Inc. created a trial account and subsequently visited the pricing page, then the recorded URL would appear as '*https://www.company-name/acme_inc.com/pricing/*.'

We could continue, but the core of the problem is that there is not a 1:1 mapping of distinct URLs to distinct locations within the marketing site. To enable any meaningful analysis, UTM parameters, subdomains, and other substrings will need to be removed from Snowplow's recorded URLs. To simplfy matters further, we drop all prefixes including '*www*,' '*https*,' and so forth.

As we proceed to distill marketing site pages and content from the URLS, it becomes clear that there remains an exceedingly large number of distinct URLs (29,245). Our next step is creating features from combinations of URLs and event types, and at this stage we are on track for a feature space ranging into the hundreds of thousands. Fortunately, the overwhelming majority of marketing site activity is captured within a hundred URLs, and so we make the strategic decision to eliminate all but those URLs from the data set.

In the next stage of data transformation, we create new variables from every combination of event type and event location. Snowplow utilizes six different types of events: 'page_view', 'event', 'link_click', 'change_form', 'submit_form', and 'page_ping.' When complete, we should have six hundred new features. We start with, we use the 'get_dummies' function from the Pandas package to turn event_type into six boolean variables. Starting with 'page_view,' we then create six new data objects for each event type. These data objects are event aggregations - sums of the new boolean variables while grouping on account_id and page_url. We then join these objects onto the original data set by mapping on the account_id. Following a similiar process, we create aggregations for the number of distinct visitors (cookies), as well as the number of sessions.

We implement a similiar process with country codes. There are over 200 distinct country codes in the data set, and if we counted every distinct combination of country : event_type (e.g. page pings from Kenya, form submissions from Estonia), then the feature space would increase untenably. Here we make another strategic decision to only include the aggregation of page views from a given country.

The next categorical variable to be transformed is 'mkt_medium.' This variable denotes the type of traffic. For example 'cpc' represents cost-per-click, i.e. paid advertising. The medium 'affiliate' represents a visitor coming from a partner website (we can confirm because of the presence of that partner's UTM parameter). 'Organic' represents vistors who come to the marketing site without any external prodding - as in they are typing in the company's name in a search bar and clicking the results. Lastly, Snowplow categorizes visitors coming from LinkedIn, Facebook, or other social networks as 'social.' Just as with the country codes before, we only include the aggregations of page views, ignoring the other event types.

Similiar to 'mkt_medium' is 'mkt_source.' The key difference is that 'mkt_source' represents a specific URL (e.g. Google, Adroll, etc.). After creating and aggregating the boolean variables, we are left with 42 features from the 'mkt_medium' variable. Here a complication arises. The fact that 'mkt_source' is a portion of a URL means that the newly created column headers often contain elements (e.g. spaces) that can interfere with Panda's functionality. To prevent errors, we relabel the columns, dropping odd characters and utilizing underscores.

The final feature transformation is the 'dvce_ismobile' variable. We create a feature representing the count of page views taking place via a mobile device, a count of page views from non-mobile devices, and a feature detailing the percentage of an account's recorded page views to take place on a mobile device.

With feature transformation complete, we create a vector of labels - 'cc' - using conditional logic (is there an associated date when a credit card was added? yes/no). Fortunately, the NULL values in the data set are not an indication of missing data, but rather represent events with counts equal to zero. To prevent errors, we zero fill the data set using the *fillna()* command.

As a final touch, we drop all columns which contain no useful information (i.e. columns only consisting of zeros). In this fashion we condense 14.6 million events into a working data set of 16,607 observations and 581 features.

Exploring the Transformed Data

By using 'Notebook 2 - Exploratory Analysis,' we can get a sense of the scope and structure of the transformed data. After reading in the data and necessary packages, we print such essential information as the number of observations and the number of features. This iPython notebook also creates the visualizations used in the **Data Exploration** sections, including a horizonatal barchart illustrating the imbalanaced nature of the data, as well as histograms of the features' means and standard deviations.

Establishing a Benchmark

It is at this stage that we employ machine learning - our first task being to establish a benchmark of model performance. A detailed discussion of why we chose the algorithms we did is available in the section entitled **Algorithms and Techniques**. To create this benchmark, we use 'Notebook 3 - KNN (Baseline).' After reading in the transformed data and packages, we subset the features into the X_all object, and the labels into the y_all object. Bias can be a potential issue if the features' variance varies significantly. To prevent this, we rescale the features using Scikit-Learn's RobustScaler function. This function removes the median and re-scales the data according to the interquartile range - the range between the 1st and 3rd quartiles. This rescaling technique tends to be more robust to outliers compared to simply subtracting the mean and dividing by the standard deviation.

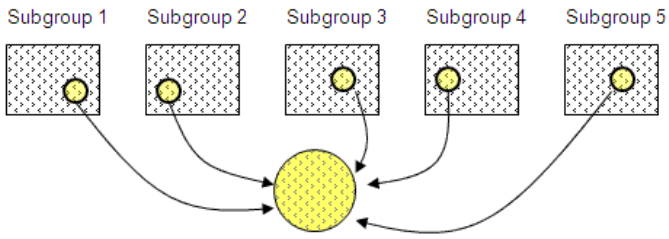
With the features rescaled, we then split the data into training and testing sets using the train_test_split function. For the purposes of replicability, we set the 'random_state' argument to *random_state=1*. Originally, the models were implemented using a 80%:20% split between training and testing data. However, experimenting with the various models showed that increasing the size of the training data set significantly improved model performance. Ultimately, all models were run using a 90%:10% split between training (14,946 observations) and testing (1,661 observations).

Sampling the data into training and testing sets is complicated by the highly imbalanced nature of the data. More often than not, the ratio of converted customers to non-converted customers should be comparable between the training and testing sets. However, with relative few members in the 'converted' class, there is always the possibility of unequal sampling and subsequent bias. To address this concern, I use stratified sampling on the dependent variable - 'cc.'

Stratified sampling works by aggregating observations on the class variable. In effect, instead of randomly taking 90% of the data for the training set, the computer takes 90% of the data where cc = 0, takes 90% of the data where cc = 1, and then combines the results into our training set. With this approach, we can ensure that our training and test sets do not vary in their composition of classes.

Figure 5: Stratified Sampling

Source: Six Sigma Material



With the data partitioned, we then use KNeighborsClassifier to create a classifer, and apply it to the training data. With our model fit, we then use Scikit-Learn's cross_val_score function to score the model's performance when applied to the test data. Cross validation works by taking *k* subsets of the data, deriving a metric of interest from each subset, and then averaging the results. To guard against anomalous results, we use 100 subsets for our cross-validation.

As for our "metric of interest," recall that our success metric is the F2 score. To instruct the *cross_val_score* function to derive the F2 score, we use the make_scorer function with 'fbeta_score' as the metric, making sure to set the beta argument to *beta=2*. With our scorer in hand, we derive 100 F2 score and average them together for our benchmark (F2 = 0.04).

Applying More Sophisticated Models

By this point, our basic procedure is established:

- (1.) Read in the data
- (2.) Create separate data objects for the features and labels
- (3.) Rescale the data
- (4.) Create testing and training sets with a 90% to 10%, making sure to use stratified sampling on the labels
- (5.) Create the classifier and train it on the training set
- (6.) Use the newly trained classifer on the testing data set
- (7.) Derive the F2 score one hundred times and take the mean result, repeat for recall and precision

We repeat this workflow in the iPython notebooks 'Notebook 4 - SVM with RBF Kernel,' 'Notebook 5 - Linear SVM,' and 'Notebook 6 - Logistic Regression' - the only varying element being the type of classifier used. As a final note, it quickly became evident that the SVM + RBF kernel model was by far, the most computationally expensive. Including the tuning of hyper-parameters (see below), implementation of the SVM + RBF model took approximately 17 hours.

Refinement

In theory, we should be able to improve upon the baseline models by tuning the models' hyper-parameters. Our primary hyper-parameters of interest are C and gamma for the SVM + RBF model, and just C for the linear SVM model. Recall that C is the penalty parameter - how much we penalize our model for incorrect classifications. A higher C value holds the promise of greater accuracy, but at the risk of overfitting. The selection of the the gamma hyper-parameter determines the variance of the distributions generated by the RBF kernel, with a large gamma tending to lead to higher bias but lower variance.

The C and gamma hyper-parameters can vary by several orders of magnitude, so finding the optimal configuration is no small task. Employing a grid search can be computationally expensive - almost prohibitatively expensive without parallel computing resources. Fortunately, we do not have to exhaustively scan the hyper-parameter space. Rather, we can use Bayesian optimization to find the optima within the hyper-parameter space using surprisingly few iterations.

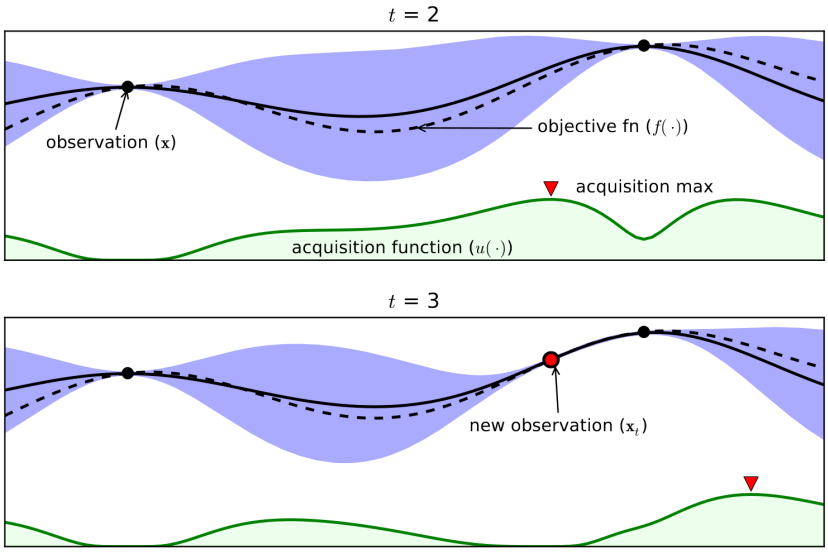
Here I am indebted to Fernando Nogueira and his development of the BayesianOptimization for Python. By means of this package, we are able to scan the hyper-parameter space of the SVM + RBF kernel model for suitable values of C and gammma within the range 0.0001 to 1,000. We are able to scan for suitable values for C within the linear SVM model in similiar fashion.

The below figure illustrates the second and third iterations of this process in a hypothetical unidimensional space - for instance, the hyper-parameter C. Thus, the horizontal axis represents the individual values of C while the horizontal axis represents the metric that we are trying to optimize - in this case, the F2 score.

The true distribution of F1 scores is represented by the dashed line, but in reality is unknown. The dots represent derived F2 scores. The continuous line represents the inferred distribution of F2 score. The blue areas represent aa 95% confidence interval for the inferred distribution, or in other words, represent areas of potential information gain.

Figure 6: An Acquisition Function Combing a Unidimensional Space for Two Iterations

Source: Bayesian Optimization and Its Applications Part II, gauravbharaj, April 28, 2014



The Bayesian optimizer resolves the perennial dilemma between exploration and optimization by use of an acquisition function, shown above in green. The red triangle denotes the global maximum of the acquisition function, with the subsequent iteration deriving the F2 score for that value of C. Note how the acquisition function derives high value from regions of relatively low information (the exploration impetus), yet achieves even greater values when in the vicinity of known maxima of the inferred distribution (the optimization impetus).

For the purposes of Bayesian optimization, we used 20-fold cross validation with a custom scoring function to maximize the F2 scores. The optimization yielded values of C = 998 and gamma = 0.2 for the SVM + RBF model, C = 335 for the linear SVM model, and C = 585 for the logistic regression model.

Results

The optimal model in terms of F2 score, recall, but also precision was the linear SVM model with hyper-parameter tuning via Bayesian optimization. The linear SVM model was so successful that the non-optimized version was the second best performing model. The linear SVM model achieved a mean F2 score of 0.25 versus 0.04 for the benchmark KNN model. For recall, the linear SVM achieved a mean score of 0.33. In other words, the model sucessfully found a third of the valuable needles within our haystack. The model also achieved a mean precision of 0.14 - effectively tying with the hyper-parameter tuned logistic regression model. To put this in context, a sales representative engaged in blind guessing which acccounts would convert to paying customers would be hard pressed to be accurate more than 6% of the time (the rate of customer conversion).

Table 2: Comparision of Performance Metrics Averaged from 100-Fold Cross Validation

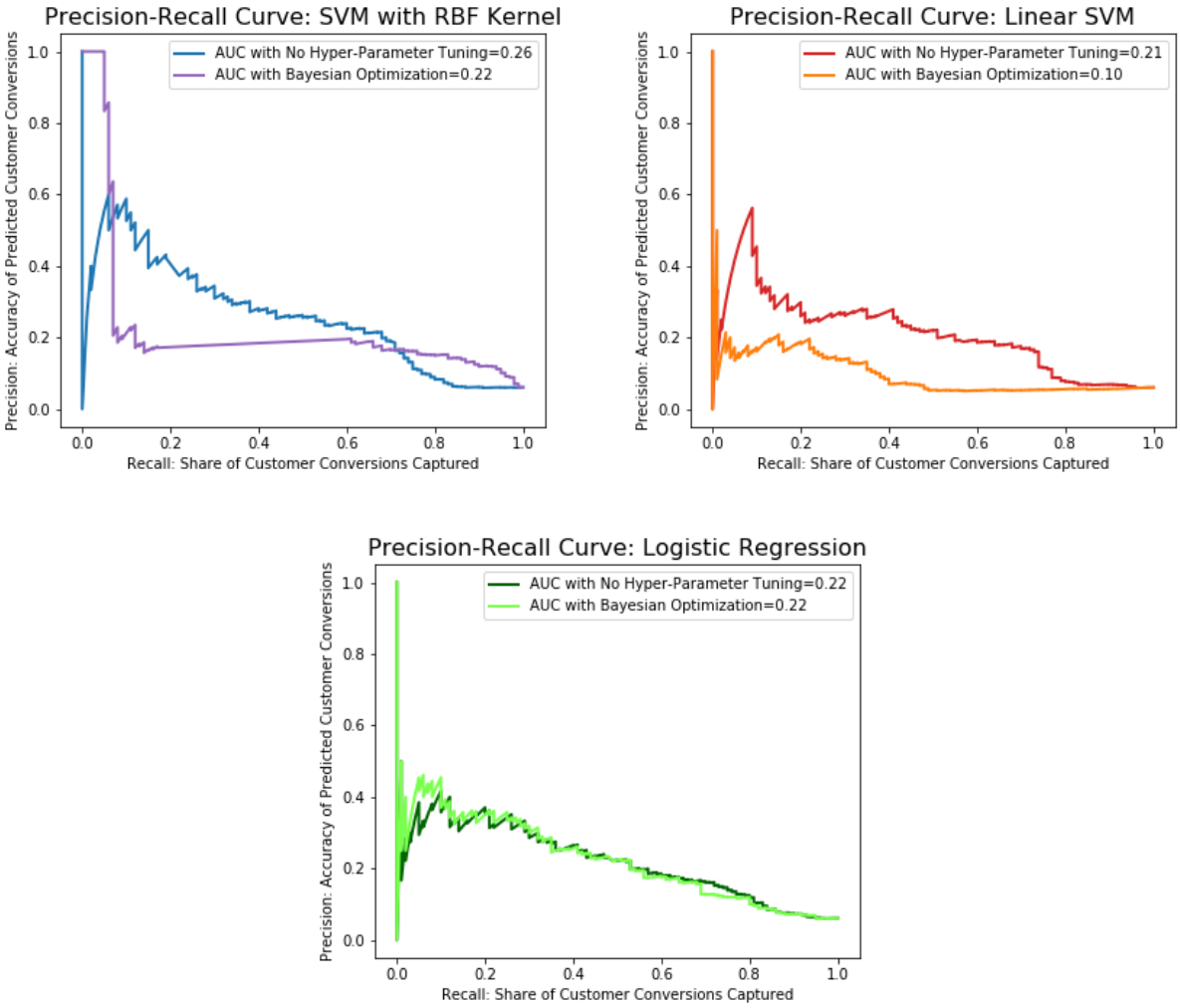
Model Used	F2 Score	Recall	Precision
K-Nearest Neighbors (Baseline)	0.04	0.04	0.04
Logistic Regression	0.16	0.19	0.13
Logistic Regression with Hyper-Parameter Tuning	0.15	0.18	0.14
Support Vector Machines with RBF Kernel	0.00	0.00	0.00
Support Vector Machines with RBF Kernel and Hyper-Parameter Tuning	0.03	0.03	0.03
Linear Support Vector Machines	0.16	0.20	0.13
Linear Support Vector Machines with Hyper-Parameter Tuning	**0.25**	**0.33**	**0.14**

It is striking how the AUC scores for the precision-recall curves imply a very different performance ranking than what the F2 scores report. Looking to the figures below, we can see that the linear SVM with hyper-parameter tuning actually has the lowest AUC of any of the curves (AUC = 0.10). These AUC scores are based upon average precision as opposed to recall.

However, it is recall, not precision, that is our priority here. Nevertheless, it is worthing bearing in mind that more often than not, the price for greater recall is precision and vice versa.

The results suggest that our winning model - linear SVM with Bayesian optimization - represents a reasonably robust result. Our success metrics were derived from 100-fold cross validation, and so our F2 score, precision, and recall scores are unlikley to be the results of a statistical fluke. More intuitively, the hyper-parameter tuned linear SVM model had the smallest C value of any of the models above (C=335), the implication being that the linear SVM model should be less likely to overfit vis-a-vis the competing models.

Figure 7: Precision-Recall Curves of All Three Algorithms with and without Hyper-Parameter Tuning



Conclusion

This project sought to differentiate soon-to-be paying customers from non-paying account-holders based solely on their activity history on the marketing site. This is a tall order, and while the linear SVM did achieve a F2 score of 0.25 (a more than five-fold improvement vis-avis the KNN benchmark) in practical terms the model is not yet successful enough to be used as part of the sales process. Should the company see a drastic influx of new leads, then our linear SVM model can conceivably be used to prioritize sales outreach. However, in the mean time we should re-examine the model for areas of improvement.

A major challange of this project is the high dimensionality of the data combined with the sparse feature space. Dimensionality reduction via principal component analysis (PCA) was attempted, but yielded inferior model performance and was discontinued. Nevertheless, dimensionality reduction could be used to open up an array of robut models such as random forests. A possible alternative to PCA is linear discriminant analysis (LDA). Unlike PCA, LDA emphasizes discrimination between classes. Moreover, given the large number of observations available to us, PCA is generally less likely to perform well relative to LDA (Martinez and Kak, 2001).

LDA coupled with random forest modeling is one potential area to pursue. Another, more radical aproach would be to reconceive the problem not as one of supervised classification, but rather outlier detection. Given that the accounts of interest only make up 6% of the data set, there is an argument to be made that such observations are in fact - outliers. In pratical terms, we could leverage Sci-Kit Learn's One Class SVM funcationality. In this fashion, future extensions might take a factorial approach, varying between random forest versus One Class SVM and LDA-reduced data versus no dimensionality reduction.

Perhaps the greatest potential for improving accuracy involves re-considering how we go about data transformation and sampling. With the original data transformation, we made the decision to aggregate page views by country code to create approximately 150 geographic predictors. There is no reason to believe that aggregating page views by country should yield greater accuracy than aggregating page pings by country. It is likley worth assessing whether a different set of geographic predictors might yield dividends.

References

- Alexandr, A., Indyk, P. “Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions”, Foundations of Computer Science, 2006. FOCS ‘06. 47th Annual IEEE Symposium.
- Bawa, M., Condie, T., Ganesan, P. “LSH Forest: Self-Tuning Indexes for Similarity Search”, WWW ‘05 Proceedings of the 14th international conference on World Wide Web Pages 651-660.
- Bishop, Christopher M. Pattern Recognition and Machine Learning, Chapter 4.3.4.
- Martinez, A., Avinash C. Kak. "PCA versus LDA," IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(2), 2001: doi:10.1.1.144.5303.
- Pedregosa et al. "Scikit-learn: Machine Learning in Python," JMLR 12, pp. 2825-2830, 2011.
- Wainer, Jacques. "Comparison of 14 Different Families of Classification Algorithms on 115 Binary Datasets," June 6, 2016. Retrieved from https://arxiv.org/pdf/1606.00930v1.pdf.
- Saito T., Rehmsmeier M. "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets," PLoS ONE 10(3), 2015: doi:10.1371.
- Schmidt, Mark and Nicolas Le Roux and Francis Bach: Minimizing Finite Sums with the Stochastic Average Gradient.
- Smola, A., Bernhard Schölkopf “A Tutorial on Support Vector Regression”, Statistics and Computing archive Volume 14 Issue 3, August 2004, p. 199-222.