Machine Learning Capstone Project

Final Report

Definition

Project Overview

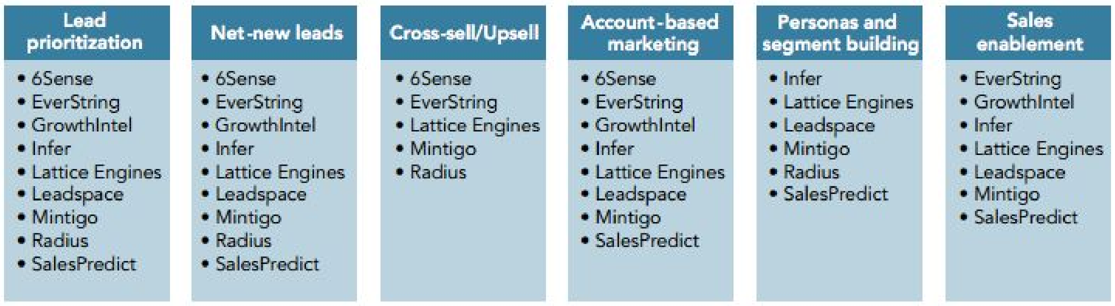
In recent years, technology has made it easier to bridge the gaps between siloed data sources, and provide a unified view of data across platforms, customer relationship management systems and marketing automation software. This enabled a new breed of products based on machine learning that aims to successfully predict whether a lead will convert into a customer.

Companies like Lattice Engines, Fliptop, 6sense and Infer are using machine learning to power sales and marketing teams, enabling them to quickly identify key characteristics of ideal target markets by uncovering purchasing patterns based on historical customer behavior and demographic data. In addition to historical data, some of them are using machine learning to analyze incoming data from external sources, like web searches and social media, to provide an even better understanding of who are the most promising customers.

As shown in the article "Probabilistic Modeling of a Sales Funnel to Prioritize Leads", by Brendan Duncan and Charles Elkan from the Department of Computer Science and Engineering at UCLA, from 2015, machine learning models can result in up to 307% increase in number of successful sales, as well as a dramatic increase in total revenue.

In fact, nearly two thirds of businesses have implemented predictive marketing analytics, including 42% that are expanding or upgrading their implementation within the next 12 months, according to Forresters From Insight to Action: How Predictive Analytics Improves B2B Marketing Outcomes, published in October 2015. Among businesses using predictive marketing analytics, 83% have experienced a positive business impact as a result of their implementation.

Selected predictive marketing analytics platform use cases



Lead prioritization	Net-new leads	Cross-sell/Upsell	Account-based marketing	Personas and segment building	Sales enablement
• 6Sense	• 6Sense	• 6Sense	• 6Sense	• Infer	• EverString
• EverString	• EverString	• EverString	• EverString	• Lattice Engines	• GrowthIntel
• GrowthIntel	• GrowthIntel	• Lattice Engines	• GrowthIntel	• Leadspace	• Infer
• Infer	• Infer	• Mintigo	• Infer	• Mintigo	• Lattice Engines
• Lattice Engines	• Lattice Engines	• Radius	• Lattice Engines	• Radius	• Leadspace
• Leadspace	• Leadspace		• Leadspace	• SalesPredict	• Mintigo
• Mintigo	• Mintigo		• Mintigo		• SalesPredict
• Radius	• Radius		• SalesPredict		
• SalesPredict	• SalesPredict				

Many predictive marketing analytics vendors are rooted in lead scoring but have broadened their capabilities to include predictive modeling, personalization, and product recommendations that push deeper into the purchase funnel. However, none of them are built with a specific market segment in mind. Despite the fact that there is no established market leader among these players, and that the crowded field continues to attract new ventures, there is no specific solution tailored to the education market. This is our motivation.

Problem Statement

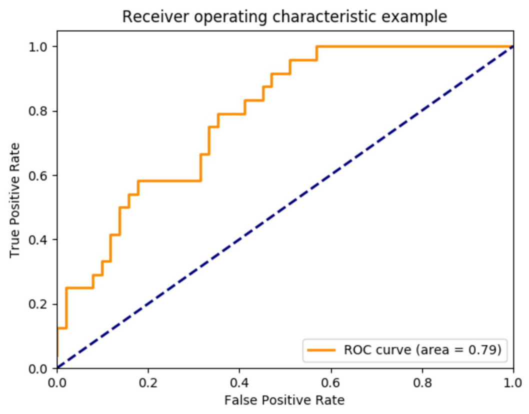
In June 2016 Udacity started to operate in Brazil. Since then, we have seen exponential growth in our Nanodegree students: average compounded weekly growth is 10-12% for the past 8 months. Our sales funnel is composed by the following stages:

1. Awareness: User discovered us through social network ad campaigns, PR, live events, or organically (SEO), and landed in our website;
2. Lead: User created a free account;
3. Marketing-qualified lead (MQL): User watched webinar(s), visited specific pages, opted-in our newsletter, followed Udacity in social network(s), and/or started a free course;
4. Sales-qualified lead (SQL): User visited our checkout more than once, made inquires about how our Nanodegree programs work, and/or tried to pay (unsuccessfully);
5. Student in trial: User started Nanodegree trial;
6. Paying student: User successfully concluded the trial and/or (directly) enrolled in a Nanodegree.

Although we currently use marketing automation, we are not using any user behaviour data to drive them toward the end of the funnel: we simply do promotional email campaigns to generate urgency to purchase. Usually, they are a one-size-fits-all campaigns targeted to MQLs. This raises the question:

> Given all behaviour data we collect from our users through the sales funnel, can we train an algorithm to successfully predict whether or not they are potential Udacity Nanodegrees students?

Metrics



Our dataset is very imbalanced: only aprox. 10% of our samples are Udacity Nanodegree students. In order to avoid accuracy paradox, we will use **ROC AUC score** to choose the best algorithm, and then to tune its parameters.

ROC AUC score means area under the ROC curve. ROC stands for Receiver Operating Characteristic, because it was created by Radar Engineers in World War-II. However, in machine learning the curve has nothing to do with radar signals or signal receivers. In an ROC curve, we plot ‘True Positives‘ on Y-axis and ‘True Negatives‘ on

X-axis: the greater the area under the curve, the better our model is.

Analysis

Data Exploration

In order to tackle the proposed problem, we used the datasets in the following table. Important to highlight: emails, names, or any other data that could identify users were not used.

Dataset name	Description	Rows	Total fields	Field examples
auth_user	Contains key data about users	60,859	11	id, last_login, username, first_name, last_name, email, date_joined
payment_app_product	Contains key data about Nanodegrees	26	12	id, name, code, price
payment_app_subscription	Contains all subscription data from paying students	5,760	39	id, access_until, cancel_requested, status, register_date, credit_card_retries, chosen_payment_type_code, product_id, user_id, first_payment_date, cohort_id, full_amount, instalment_amount
frontend_brazil.pages	Contains data from all website visits	4,707,359	47	anonymous_id, category, context_ip, name, path, referer, sent_at, timestamp, title, url, user_id, context_campaign_medium
frontend_brazil.identifies	Contains data that links visitors with user accounts	113,630	34	received_at, anonymous_id, user_id
frontend_brazil.tracks	Contains data from all events tracked in marketing website	1,092,373	31	received_at, event, event_text, timestamp, user_id, anonymous_id, context_ip
brazil_events.event_sign_up	Contains data from all webinars	26,931	54	email, enrollment_date, event, event_title, event_start_date, event_end_date, slug, user_id, user_interests, event_type
analytics_tables.course_enrollments	Contains data from all free course enrollments		15	user_id, course_key, join_time, leave_time, course_title, course_level

We gathered, cleaned and prepared all data in the **new_features.ipynb** Jupyter Notebook file. We have chosen to focus on **Digital Marketing students that enrolled in any moment after 2017-04-01 and before 2017-08-10**. We made it for one reasons: Digital Marketing Nanodegree is the largest program Udacity has in Brazil, with over 50% students - therefore, we have more data. We used as total universe users who are registered and visited Digital Marketing Nanodegree Overview Page (DMND NDOP) at least once in this period.

These are the general steps we followed in this phase:

1. First, we retrieved the data from **32,544 visitors** who came to DMND NDOP in the period.
2. We then retrieved who became paying student in this period - the target feature we wanted our model to predict. We stored that in the column **is_paying_student**, with **0 or 1**. We were able to collect data from **2,952** paying students.
3. Then, we started to gather data, that would be used to train our models. Our first hypothesis was that the likelihood of a student to enroll was influenced by how recent they had an account registered at Udacity. So, retrieved the joining date of each account and created **age_in_days** feature.
4. Doing webinars was one of our key strategies to nurture leads. So, we counted how many webinars each user watched, and saved this in **webinar_enrollments** feature.
5. Similarly, we counted how many times each user enrolled in a free course, and saved this in **course_enrollments** feature.
6. Then, we counted how many times each user visited each one of our **top 25 pages** in our website. We saved this data in 25 columns, each one with the prefix **is_**: the column **is_home** counted how many times a user visited our home page, while the column **is_checkout** counted how many times a user visited our checkout, for example.
7. We also counted how many visits were done in a mobile device, saving that in **is_mobile** column.
8. Then, we counted how many times each visited was generated from our **top 15 referrers**. We saved this data in 15 columns, each one with the prefix is_referrer_: the column **is_referrer_google** counted how many times a user came to our website from Google, while the column **is_referrer_facebook** counted how many times a user came from Facebook, for example.
9. One of the key strategies we used to nurture leads was email marketing. We counted how many times each user opened our emails, saving that in **opened_emails** column.
10. Finally, we reordered the 46 columns and saved the data in **data_prepared.csv** file.

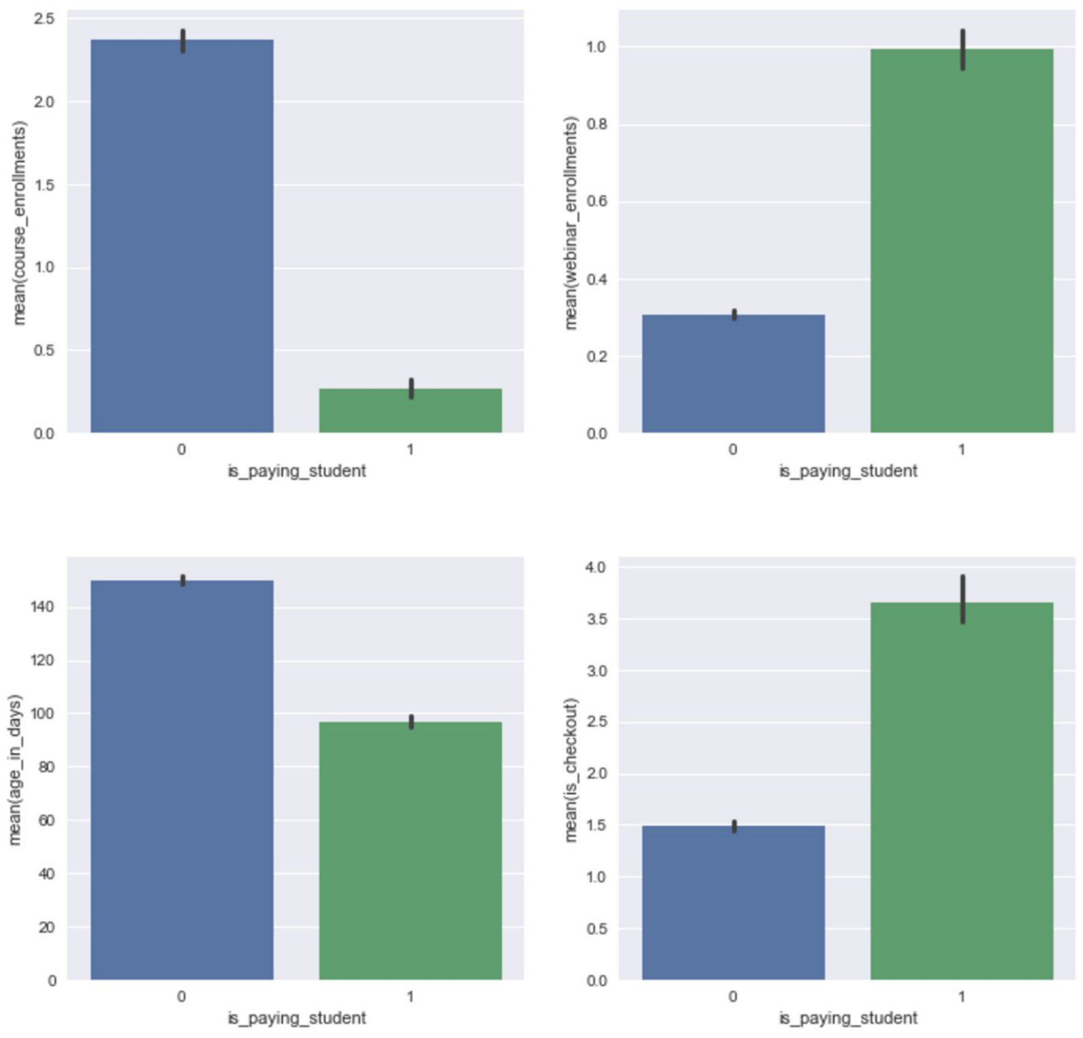
Before start training models, we checked statistics of our dataset main features. In the table below you can see a sample of that. For a complete description of all feature, please check **capstone_final_project.ipynb**.

	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
opened_emails	31991	29	49	0	3	11	35	1211
age_in_days	31991	145	116	1	60	112	196	417
webinar_enrollments	31991	0.37	0.93	0	0	0	0	21
course_enrollments	31991	2.2	5.2	0	0	0	2	128
is_checkout	31991	1.7	4.5	0	0	0	2	306
is_mobile	31991	1.0	5.9	0	0	0	0	210
count_visits	31991	50	88	1	10	23	55	4130
is_ndop	31991	8.8	20.4	0	1	4	10	1583
is_catalog_all	31991	4.6	8.8	0	0	2	5	427
is_home	31991	10.9	30.3	0	1	3	10	2948
is_referrer_infomoney	31991	0.16	0.73	0	0	0	0	49
is_signin	31991	3.3	7.0	0	0	1	4	393

Exploratory Visualization

We drew some interesting conclusions from exploratory visualization, as seen in the charts below. We saw, as expected, that leads who became paying students ended up accessing over 2x more the checkout page vs. who didn't convert. We also saw, as expected, that leads who became paying students ended up watching close to 3x more webinars than who didn't convert.

We also observed, not so obviously, that the older an account is, less likely the lead is to convert. Also, the more a lead enrolls in a free course, the less likely it is for him/her to become a Nanodegree student.



Algorithms and Techniques

We used all key supervised learning key methods, not only base estimators (e.g. Stochastic Gradient Descent, Naive Bayes, Nearest Neighbors) but also ensemble methods (e.g. Gradient Tree Boosting), and compare all results - using scikit-learn implementation.

Also, we used 75% of the data for training and 25% of the data for testing for validation, and cross-validation as well.

Now, let's provide an introductory overview of these classifiers, with a more detailed description of Gradient Tree Boosting - our selected model.

Stochastic Gradient Descent Classifier

SGD Classifier is a generalized linear classifier that will use Stochastic Gradient Descent as a solver, i.e. the cost (difference between observed class and predicted class) is minimized using Stochastic Gradient Descent algorithm. SGD is stochastic approximation of the gradient descent optimization method. In a Gradient Descent method, the set of parameters is changed through multiple iterations in a way that it gradually converges to the optimal solution starting from a random one.

Gradient descent works by the iterative application of a function that moves the parameters in the direction of their optimum values. To apply this function, we need to know the gradient of the cost function with the current parameters. This method is often called batch gradient descent, because each update to the coefficients happens inside an iteration over all the data in a single batch.

With very large amounts of data, each iteration can be time-consuming and waiting for convergence could take a very long time. An alternative method of gradient descent is called stochastic gradient descent or SGD. In this method, the estimates of the coefficients are continually updated as the input data is processed. The difference is that the parameters are updated over each iteration taking into consideration only one sample or a mini-batch - a random smaller subset of the overall data. The advantage stochastic gradient descent offers over batch gradient descent is that it can arrive at good estimates in just one iteration over the dataset.

Naive Bayes Classifier

Naive Bayes classifier is a simple but powerful probabilistic classifier based on the application of Bayes' theorem with strong (naive) independence assumptions between the features. This technique is particularly suited when the dimensionality of the inputs is high. Despite its simplicity, Naive Bayes can often outperform more sophisticated classification methods. Naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters.

Naive Bayes is the simplest form of Bayesian network, in which all attributes are independent given the value of the class variable. Intuitively, since the conditional independence assumption that it is based on is almost never hold, its performance may be poor. It has been observed that, however, its classification accuracy does not depend on the dependencies; i.e., naive Bayes may still have high accuracy on the datasets in which strong dependencies exist among attributes.

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality. On the flip side, although naive Bayes is known as a decent classifier, it is known to be a bad estimator.

Nearest Neighbors Classifier

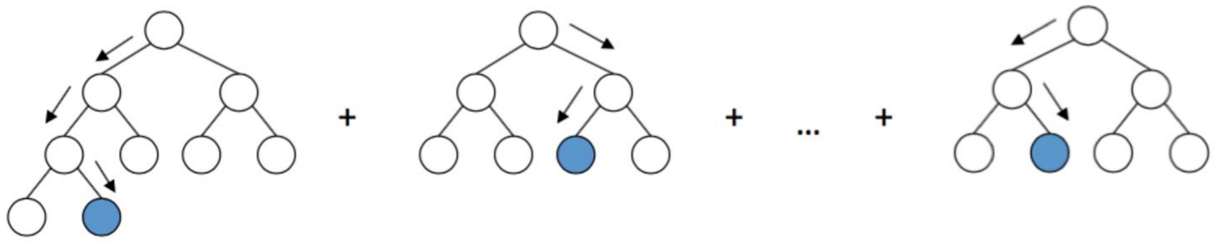
Neighbors-based classification is a type of instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

This method achieves consistently high performance, without a priori assumptions about the distributions from which the training examples are drawn. It involves a training set of both positive and negative cases. A new sample is classified by calculating the distance to the nearest training case; the sign of that point then determines the classification of the sample. The k-NN classifier extends this idea by taking the k nearest points and assigning the sign of the majority. It is common to select k small and odd to break ties (typically 1, 3 or 5). Larger k values help reduce the effects of noisy points within the training data set, and the choice of k is often performed through cross-validation.

The nearest neighbour algorithm is quite simple, but very computationally intensive.

Gradient Tree Boosting Classifier

Gradient tree boosting classifier produces a prediction model in the form of an ensemble of decision trees (weak prediction models). It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. This is one of the most powerful techniques for building predictive models.



The idea of boosting came out of the idea of whether a weak learner can be modified to become better. A weak hypothesis or weak learner is defined as one whose performance is at least slightly better than random chance. The idea is to use the weak learning method several times to get a succession of hypotheses, each one refocused on the examples that the previous ones found difficult and misclassified.

Gradient boosting involves three elements:

1. A loss function to be optimized.
2. A weak learner to make predictions.
3. An additive model to add weak learners to minimize the loss function.

In classification problems, usually logarithmic loss or ROC AUC score are used as functions to be optimized.

To make predictions, decision trees are used as weak learners. They are built in a greedy manner: the best split points are chosen based on Gini scores, for example. Also, the decision trees are constrained in different ways (e.g. maximum number of layers, nodes, splits or leaf nodes), to ensure they will remain weak.

The trees are added one at a time; once added, a tree does not change. While adding trees, this method uses gradient descent algorithm to minimize the loss. Before adding a tree to the model, the algorithm parametrizes the tree and modify the parameters until it is moving into the right direction (i.e. reducing the residual loss). Then, the new tree is added to the existing sequence of trees, improving the final output of the model.

Benchmark

Udacity US created a model to predict if a student would start a trial. We used that as benchmark, as it is a comparable approach used to the same purpose.

After extracting similar data from US data sources, the team from headquarters treated the data, divided into test/train set (⅓ of elements go to the train set) and ran the training set over the following algorithms: various support vectors machines with different parameters and kernels, stochastic gradient descent, linear and logistic regression.

They compared the results against accuracy (over the test dataset), where SVM and logistic regression gave approximately 72% and other ~64%. The highest value was our benchmark.

Methodology

Here is the step-by-step walkthrough of the process for preparing the data:

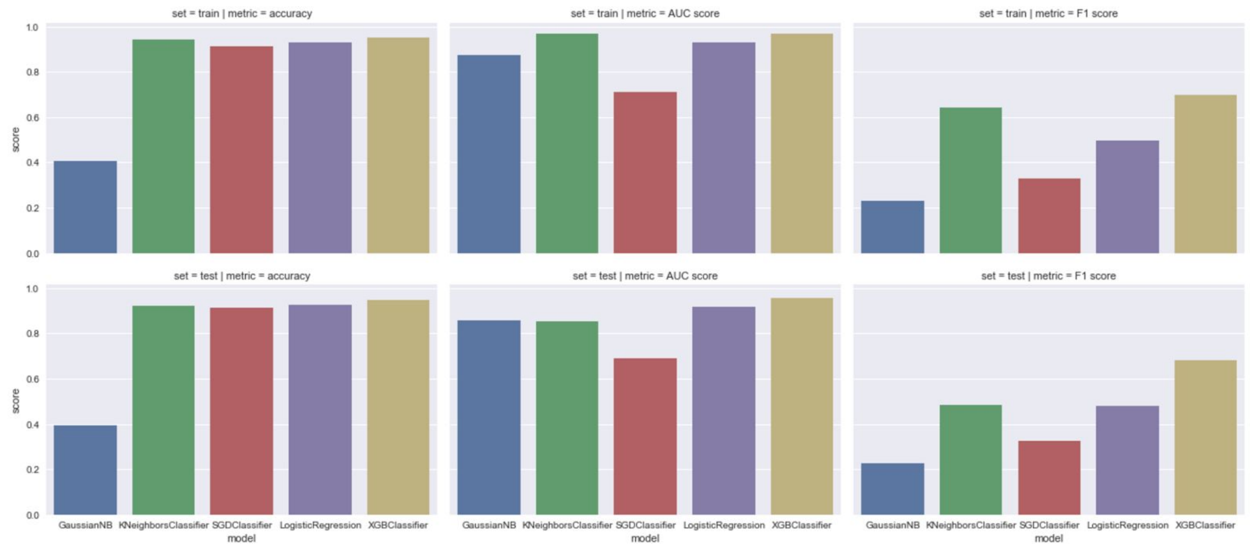
1. First, we defined our focus on **DMND students** that enrolled in any moment **after 2017-04-01** and **before 2017-08-10**.
2. We then gathered **total universe of users**: anyone who are registered and **visited DMND NDOP at least once** in this period.
3. Then, we gathered **everyone who became a Nanodegree student in this period** - the **target feature** we will train our model to predict.
4. Then, we gathered important features, like how old are the accounts, how many times each user enrolled in a webinar or free course, how many times each user navigated to each of our key pages, how many emails each user opened, etc.
5. After combining all this data in a DataFrame, being every column a feature (45 in total) and every row a sample (31,991 in total), we saved in a CSV file to prepare it for modeling/training/testing.
6. Then, we split the data in a matrix containing all features (**X_all**) and a vector containing the target variable to be predicted (**y_all**).
7. Then, we set the number of **training points to 75% of all samples**, and the remainder **25% as testing points**, to be used to **validate the accuracy/performance of our model**.
8. Finally, we shuffled and split the dataset into the number of training and testing points mentioned above.

Implementation

After i) preparing the data for modeling, training and testing, and ii) initializing helper functions used for training and testing the supervised learning models, we runned the models and selected the best ones based on their metrics.

This implementation followed these steps:

1. First, we initialized 5 different models, all with their default parameters:
    a. *Gaussian Naive Bayes*
    b. *K-Nearest Neighbors*
    c. *Stochastic Gradient Descent*
    d. *Logistic Regression*
    e. *Extreme Gradient Boosting*
2. Each of these algorithms were fit on the training data.
3. Then, for each of these algorithms, we made the predictions and calculated the prediction probability for each sample in the training set and on the testing set
4. Using all these results, we calculated the performance/score on the training and the testing sets in 3 different ways: i) accuracy, ii) ROC AUC score, and iii) F1 score.
5. Finally, we plotted all the scores side-by-side facilitate seeing the difference performance of each model, generating the picture below:



Extreme Gradient Boost won all metrics in both training and test datasets. Therefore, we have chosen this algorithm to move forward to parameter tuning/refinement.

Scikit-learn Python library was the main library used to implement the procedure described above, and the implementation was pretty straight forward. The other relevant library used was XGBoost - and as it has a Scikit-learn wrapper interface, it was pretty easy to evaluate this model together with the other 4. Both libraries are pretty well documented, which also helped this phase.

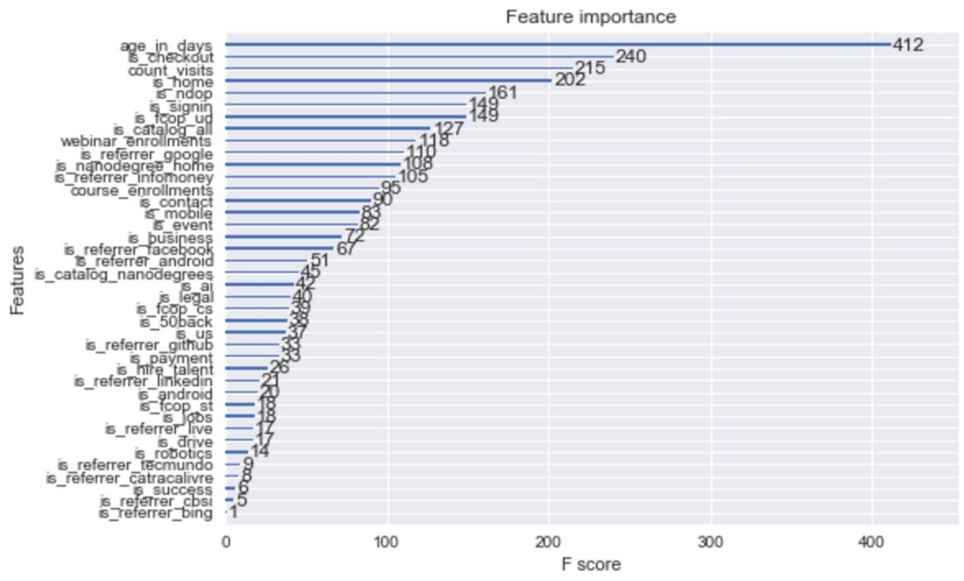
The only watch-out/challenge we faced during implementation was to run our algorithms in multiple threads/processors to speed up our process. Installing XGBoost on Mac OSX with multi-threading was not straightforward: after trying twice, we managed to do it by following a step-by-step approach described in a IBM blog post.

Refinement

To do parameter tuning and model improvement, we used **grid search** (GridSearchCV) library. Also, we followed the step-by-step approach described in Complete Guide to Parameter Tuning in XGBoost article.

Before getting started, we implemented a helper function to enable us to see which were the best parameters and perform **cross-validation** (using the built-in function provided by XGBoost). Also, we **discarded accuracy**, **focusing on AUC score** only, as our **dataset was very imbalanced**.

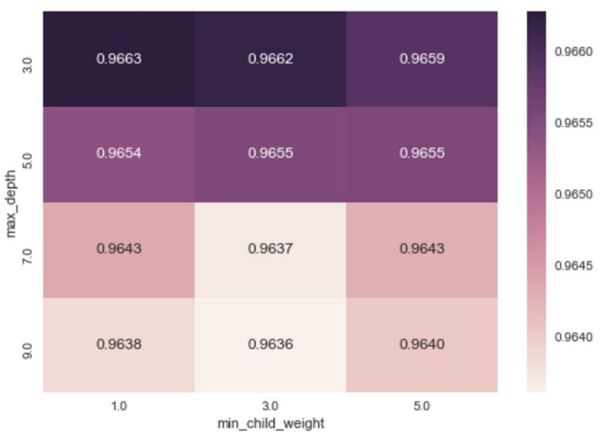
AUC score mean of cross-validation sets, prior to tuning, was **0.9651**. The optimal number of trees was 502, and we were able to get the importance of each feature plotted in the picture below:

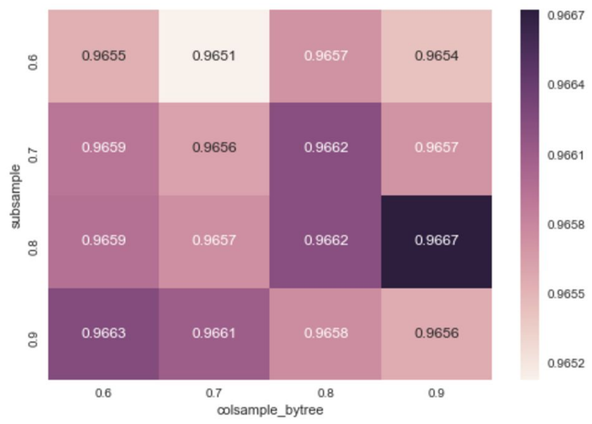


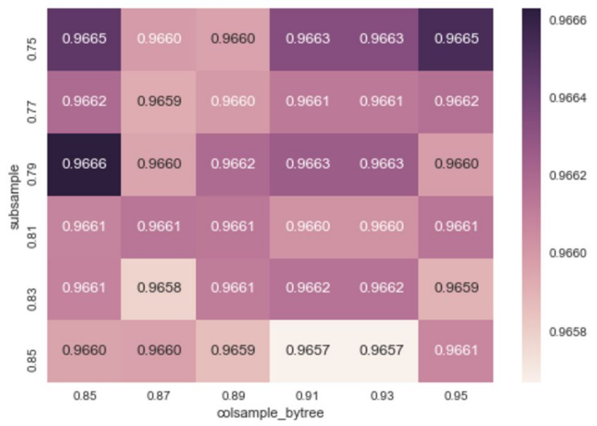
Then, we tuned the parameters in the following order:

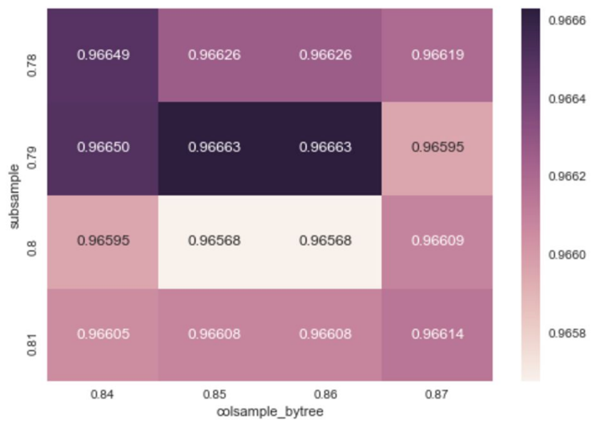
1. **max_depth** and **min_child_weight**
2. **gamma**
3. **subsample** and **colsample_bytree**
4. **reg_alpha** and **reg_lambda**

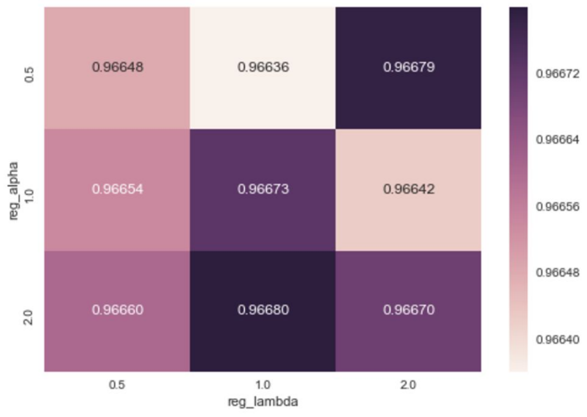
As most of tuning was done varying pairs of parameters, we used several heat maps to visually see the best parameters. Below you can find a sample of the best charts. Please refer to **capstone_final_project.ipynb** for all step-by-step details.

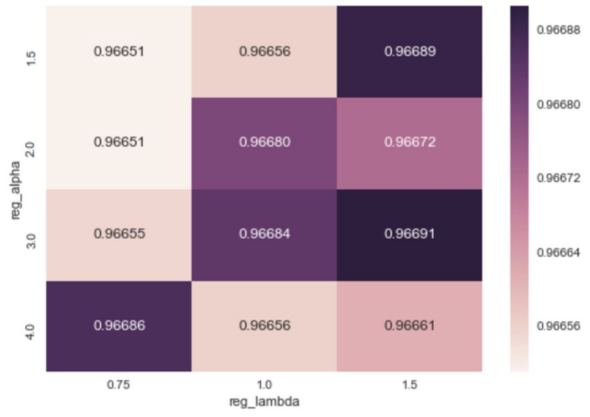








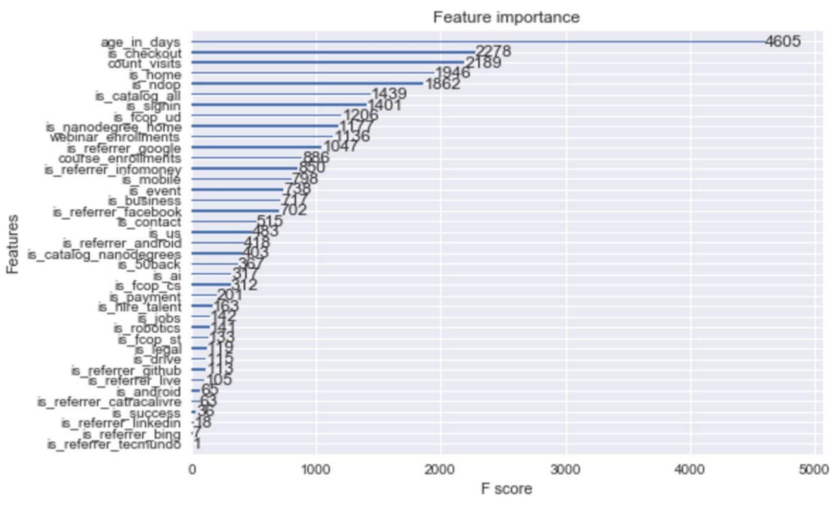




Results

Model Evaluation

After all parameter tuning, we were able to reach an **AUC score** of **0.96691**. Parameter tuning slightly improved ROC AUC score. The high scores in train dataset, CV datasets (mean) and test datasets indicates that we were able to generate a very strong prediction model, without overfitting. Also, we were able to read the most important features - the chart below summarizes the main ones:



Validation

We validated our model using 2 techniques: a) **test and train datasets**, and b) **cross validation**. In the first, we trained the final model using the train dataset, and calculated the performance using the test dataset. These sets were randomly selected using the 75/25 split. As the trained model was not exposed to the test dataset during training, we got very satisfied with **AUC score of 0.9631 achieved using the test dataset**.

To really make sure we achieved great results, we also validated our model with cross validation, which is a more sophisticated approach. This approach uses the entire dataset to train and test the model. It first involves separating the dataset into a number of equally sized groups of instances (called folds). The model is then trained on all folds exception one that was left out and the fitted model is tested on that left out fold. The process is repeated so that each fold gets an opportunity at being left out and acting as the test dataset. Finally, the performance measures are averaged across all folds to estimate the capability of the algorithm on the problem. We used a 5-fold cross validation, which trained and tested our model 5 times:

1. Train on folds 1, 2, 3, 4, test on fold 5
2. Train on folds 1, 2, 3, 5, test on fold 4
3. Train on folds 1, 2, 4, 5, test on fold 3
4. Train on folds 1, 3, 4, 5, test on fold 2
5. Train on folds 2, 3, 4, 5, test on fold 1

We also achieved a high score with cross-validation: **AUC score mean of the 5 sets described above was 0.9664**, which undoubtedly validates our model.

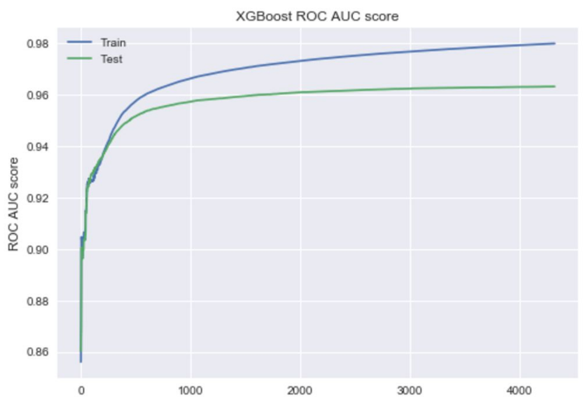
Justification

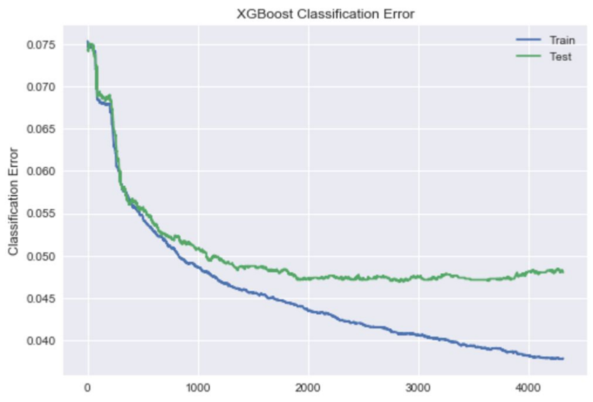
Accuracy of our model was significantly higher than the benchmark used, **0.95** vs. **0.71**. As our dataset was pretty skewed, with only 10% of samples with is_paying_student == 1, we re-did all the exercise undersampling the population of is_paying_student == 0 to reach 1:1 ratio instead of 1:10. The result was still significantly better than the benchmark: we achieved **0.87** vs. **0.71**.

Conclusion

Free-Form Visualization

We retrieved the performance of our model on the evaluation dataset and plotted it to get insight into how learning happened during the training. As we can see in the Learning Curves below, the performance of our model stops improving around 4,000 iterations.





Reflection

In this capstone project, we leveraged everything we learned throughout the Nanodegree program to solve a problem of lead conversion prediction applying machine learning algorithms and techniques. We defined the problem: train an algorithm to successfully predict whether or not a Udacity website visitor becomes a Nanodegree student. We analyzed the problem, visualizing and exploring the data available. Then, we implemented different algorithms and metrics, choosing XGBoost. After this, we documented preprocessed, refined, and postprocessed. Afterwards, we collected the results about the performance of our model and validated/justified these values. Finally, we drew conclusions about our results.

During this whole process, we have chosen to highlight 3 aspects:

1. Gathering high quality data and making sure we build a strong dataset to be used later requires significant amount of work, and is a very important part of the process. Different than the other projects, when the dataset was provided, we had to create the capstone dataset, engineering every feature, one-by-one.
2. The end-to-end process was extremely iterative: we runned the end-to-end process dozens of times, tweaking/sampling the dataset, adding features, removing features, changing the metrics, etc. We only started to really better understand how the process really works after several iterations.

3. Finally, the parameter tuning phase required a lot of patience, as searching the parameters grid after optimal values took a lot of time. However, it paid out: we achieved a high performance model, which can (and will) be tested at Udacity Brazil.

Improvement

We believe the model can be significantly improved by gathering more data in 2 fronts. One would be web scraping visitors' cookies and trying to retrieve LinkedIn data. The other would be collecting and using time spent in each page and each visit.